OpenMMLab

通用视觉框架OpenMMLab

第2讲 图像分类与MMClassification

王若晖

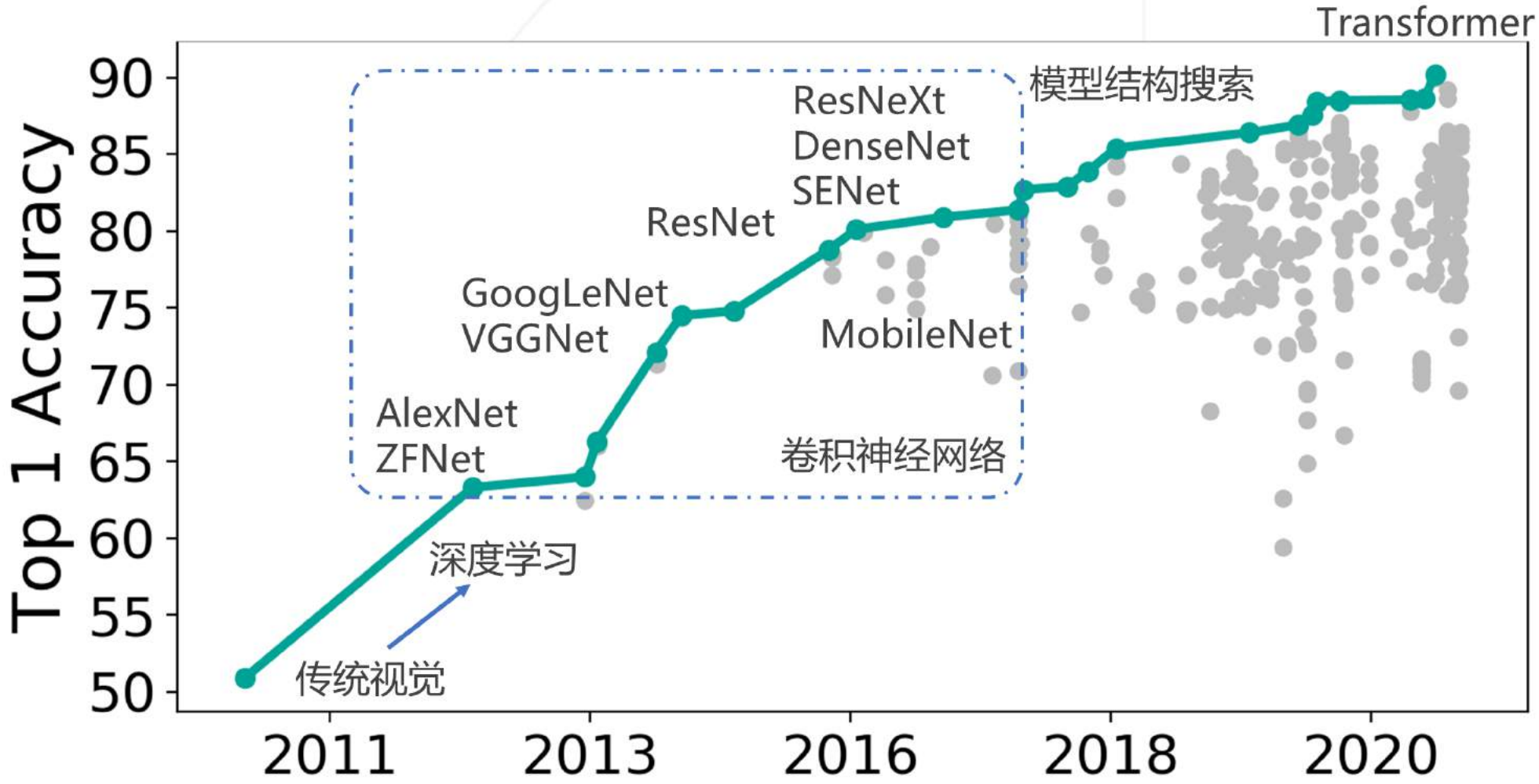
2021 年 4 月

什么是图像分类

任务目标：

给定一张图片，识别图像中的物体是什么。

图像分类算法发展



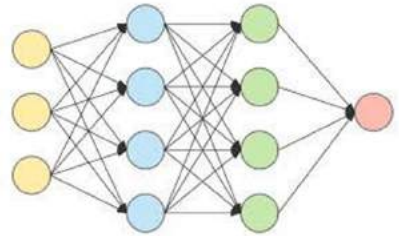
https://paperswithcode.com/sota/image-classification-on-imagenet

课程内容

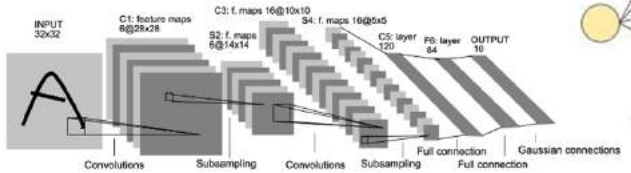
- 机器学习视角下的图像分类

$$\mathcal{H} = \{h(x;\theta) \mid \theta \in \Theta\}$$
$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\, \mathcal{L}(h|\mathcal{D})$$

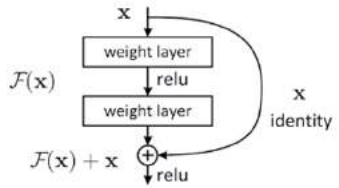
- 多层感知机



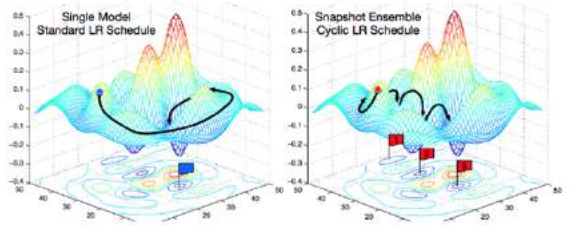
- 卷积神经网络



- **图像分类模型设计**



- **图像分类模型训练**



- PyTorch 介绍

PYTORCH

- **MMClassification 图像分类框架**

MMClassification

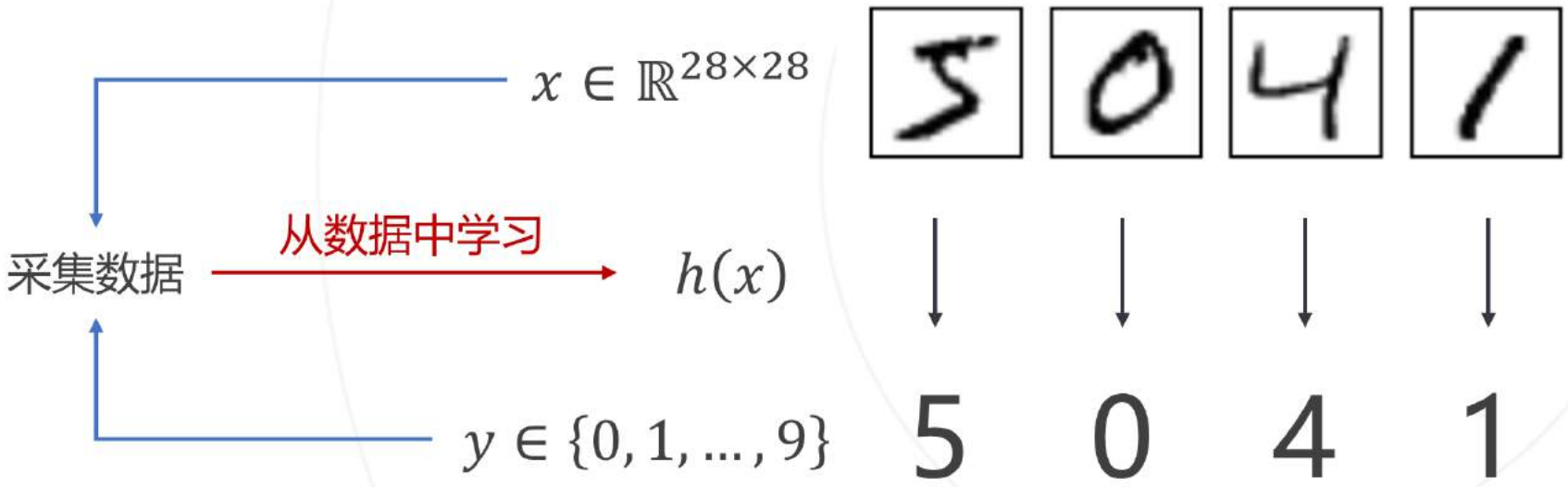
机器学习视角下的图像分类

监督学习中的分类问题

- 从数据集 $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ 中学习一个分类函数

$$y = h(x) \quad 或 \quad P(y|x) = h(x)$$

- 其中，$x \in \mathbb{R}^N$ 为输入模型的数据，$y \in \mathcal{Y}$ 为模型预测的类别，$P(y|x)$ 为模型预测数据属于不同类别的概率。$h(x)$ 也称为**分类器**。



如何学习

1. 构建函数族 $\mathcal{H}$ 包含全部可能的分类器函数 $h(x)$

- 例：线性分类器 $\mathcal{H} = \{\sigma(\theta^T x + \theta_0) \mid \theta \in \mathbb{R}^N, \theta_0 \in \mathbb{R}\}$
- 例：神经网络 $\mathcal{H} = \{\sigma(\theta_2^T \sigma(\theta_1^T x + \theta_{10}) + \theta_{20}) \mid \theta_1 \in \mathbb{R}^{N \times M}, \theta_2 \in \mathbb{R}^M, \theta_{10}, \theta_{20} \in \mathbb{R}\}$
- 例：参数化函数族 $\mathcal{H} = \{h(x;\theta) \mid \theta \in \Theta\}$，对于每一个 θ， $h(x;\theta)$ 是一个分类器

2. 在 $\mathcal{H}$ 中搜索最好的分类器

- 怎么描述“好” -> 错误率越低越好 -> 损失函数=错误率
- 怎么找到这个最好的 -> 调整 θ 降低损失函数 -> 最优化问题

回到图像分类问题

1. 函数族 $\mathcal{H} = \{h(x;\theta) \mid \theta \in \Theta\}$ 应该是什么样的?
   - （确定结构的）卷积神经网络
2. 怎么衡量分类模型的好坏?
   - 交叉熵损失函数
3. 怎么在 $\mathcal{H}$ 中搜索最佳模型?
   - 随机梯度下降为主
   - 各种经验策略辅助

梯度下降算法

目标：寻找损失函数曲面 $L(\theta)$ 的谷点

Step 1：随机选取找一个起始点 $\theta^{(0)}$

Step 2：寻找下降最快的方向，即负梯度方向 $\nabla_\theta L\left(\theta^{(0)}\right)$，并前进一步

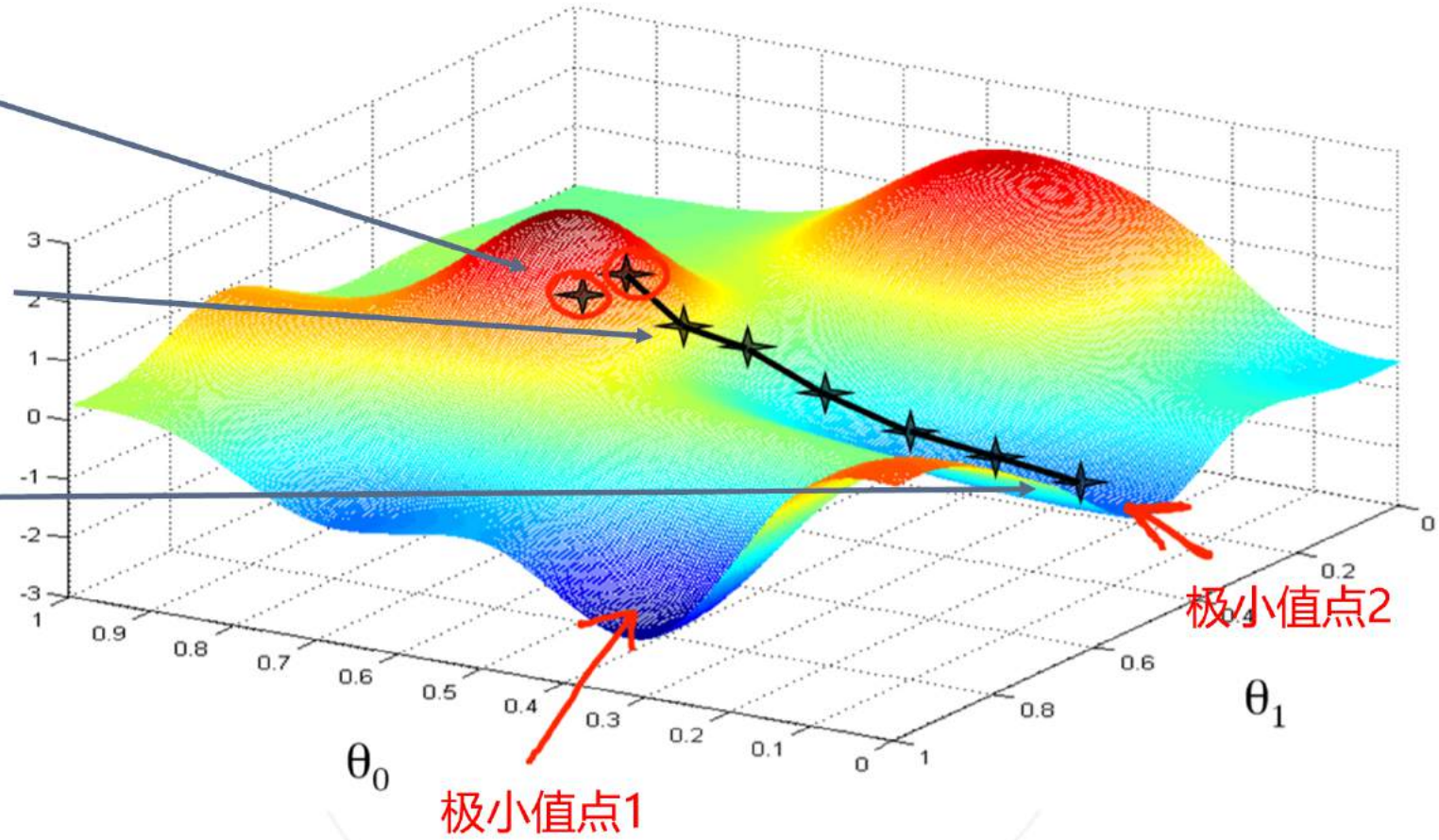
学习率

$$\theta^{(1)} = \theta^{(0)} - \eta \nabla_\theta L\left(\theta^{(0)}\right)$$

Step 3：重复直到不能再下降

凸函数：保证找到最小值点

非凸函数：不能保证，和起始位置、每步的步长有关系

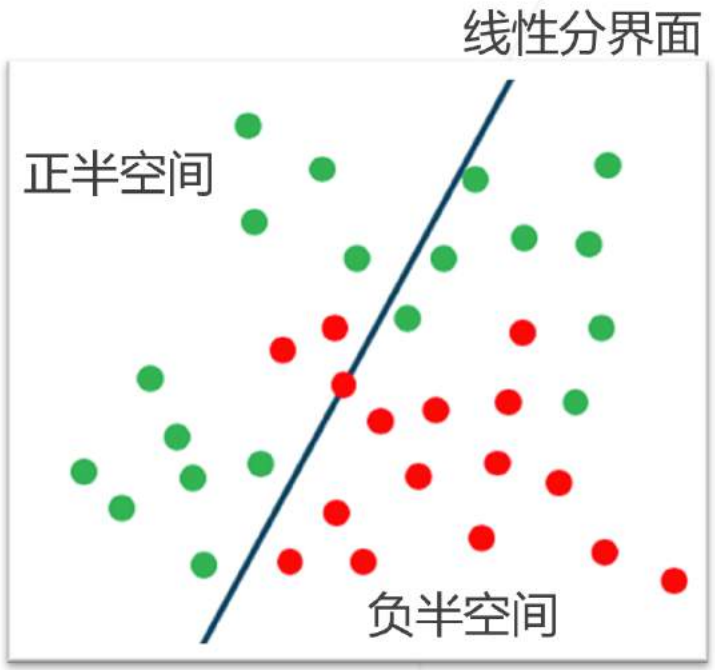


图片来源：https://towardsdatascience.com/an-intuitive-explanation-of-gradient-descent-83adf68c9c33

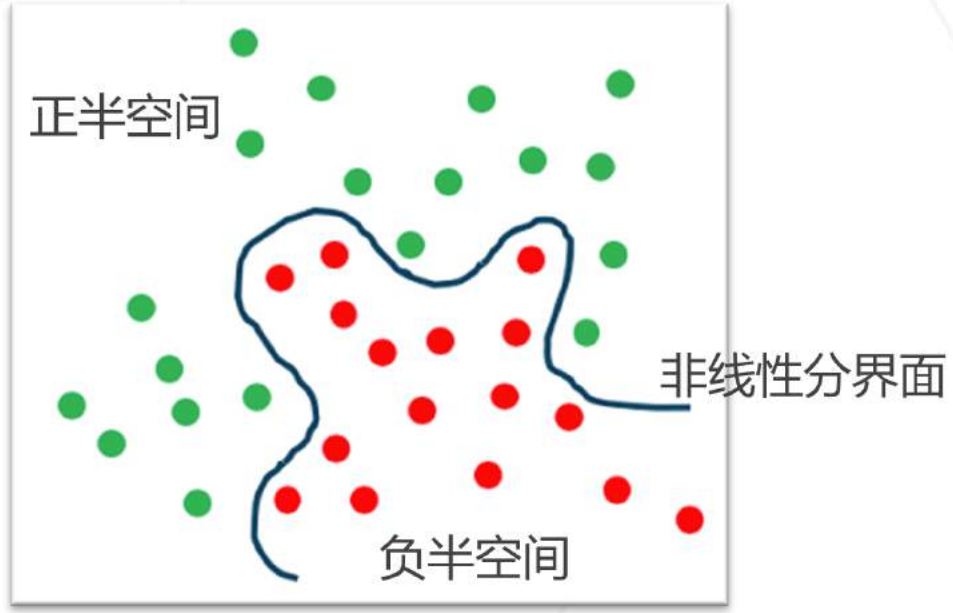
多层感知器

线性分类器与非线性分类器

考虑二分类问题：定义正半空间 $\mathcal{X}^+ = \{x|h(x) = 1\}$，负半空间 $\mathcal{X}^- = \{x|h(x) = 0\}$



数据空间 $\mathbb{R}^2$



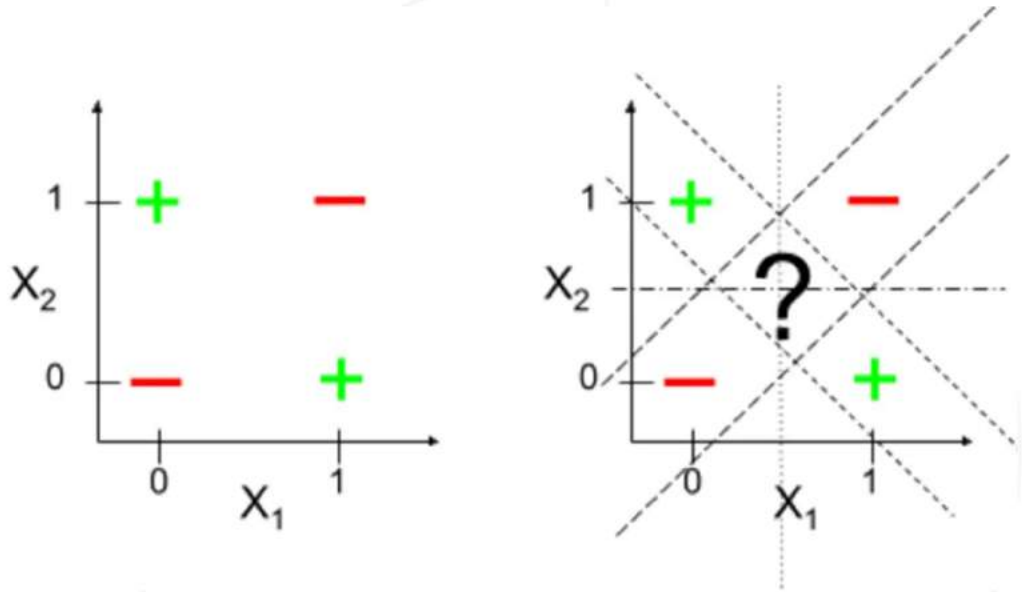
数据空间 $\mathbb{R}^2$

异或问题

异或 (XOR) 是一个基本的布尔函数，是线性不可分的。

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

XOR 的真值表



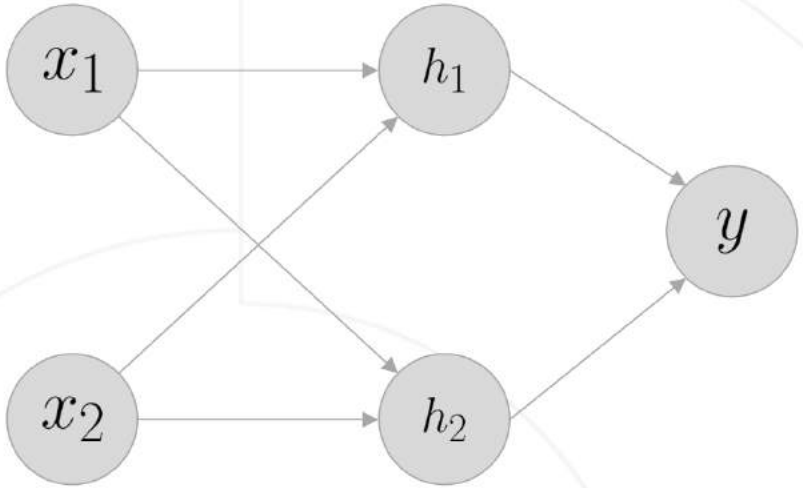
XOR 是线性不可分的

非线性分类器

异或函数可以由一个两层的计算图所表达：

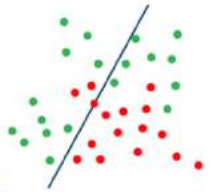
第一层由输入 x_1, x_2 到隐变量 h_1, h_2

第二层由由 h_1, h_2 到输出 y。

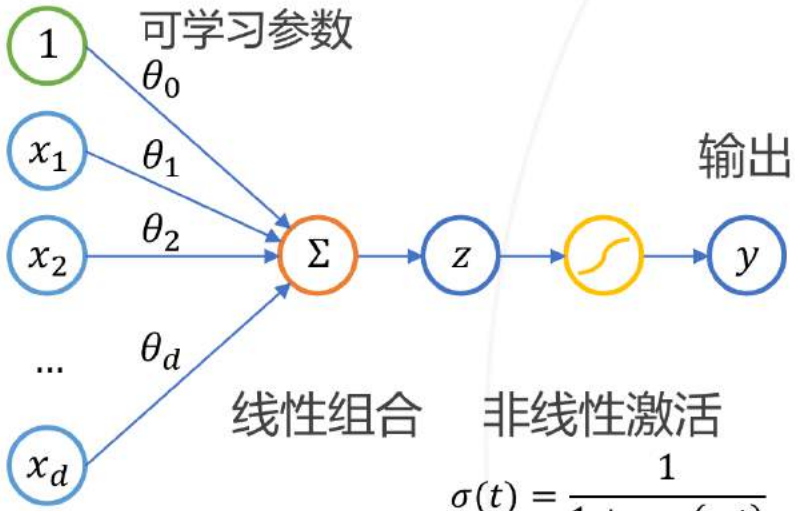


x_1	x_2	$x_1 - x_2$	$x_2 - x_1$	$h_1 = a(x_1 - x_2)$	$h_2 = a(x_2 - x_1)$	$y = h_1 + h_2$
0	0	0	0	0	0	0
0	1	-1	1	0	1	1
1	0	1	-1	1	0	1
1	1	0	0	0	0	0

神经元

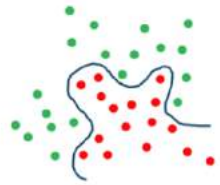


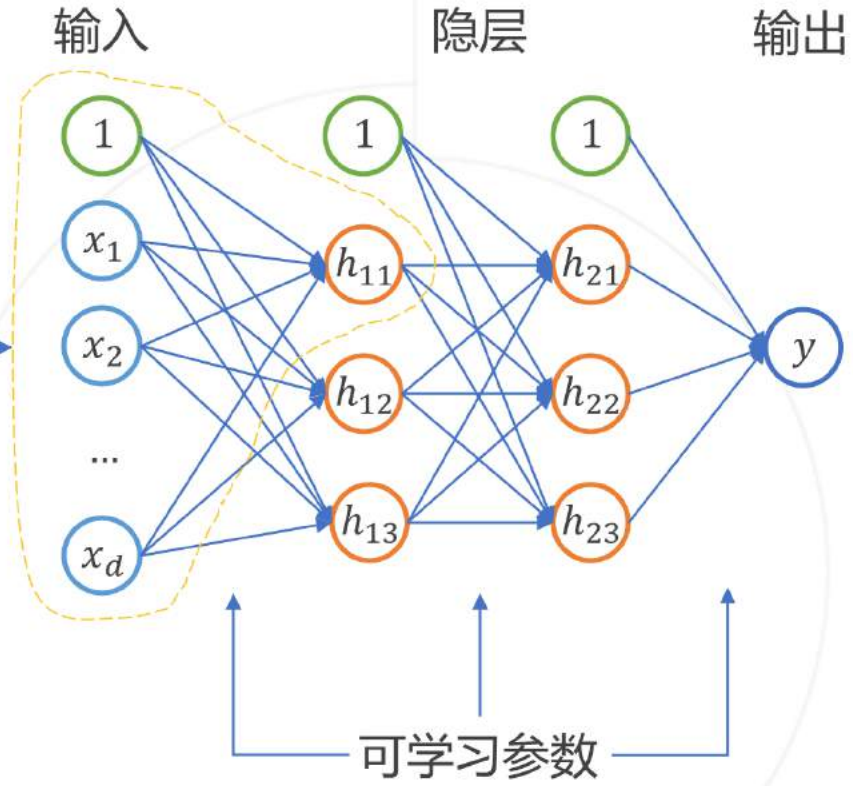
输入



$$y = \sigma(\Theta^{\mathrm{T}} x + \theta_0)$$

多层感知器





$$h_1 = \sigma(\Theta_1^T x + \theta_{10})$$

$$h_2 = \sigma(\Theta_2^T h_1 + \theta_{20})$$

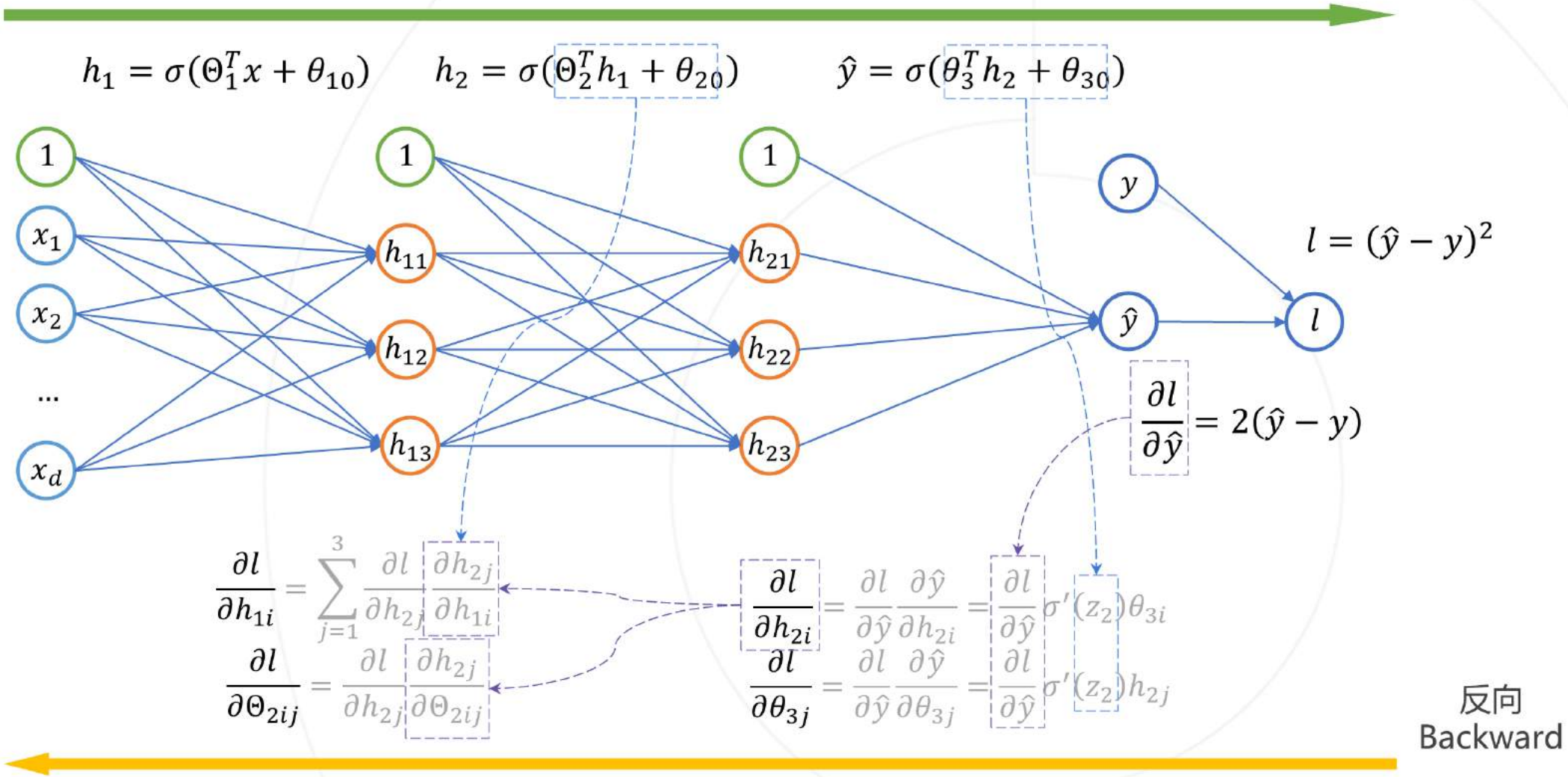
$$y = \sigma(\Theta_3^T h_2 + \theta_{30})$$

1. 确定结构（层数、每层单元数）的多层感知器构成了一个分类器族 $\mathcal{H}$，不同的连接权重对应了族内不同的分类器
2. 基于**梯度下降**寻找最优参数，进而得到最优（准确率最高）的网络

- 如何计算损失函数对于网络参数的梯度？ --> **反向传播算法**

反向传播算法

- 目标：计算损失函数 l 对于所有隐层参数 Θ 的梯度



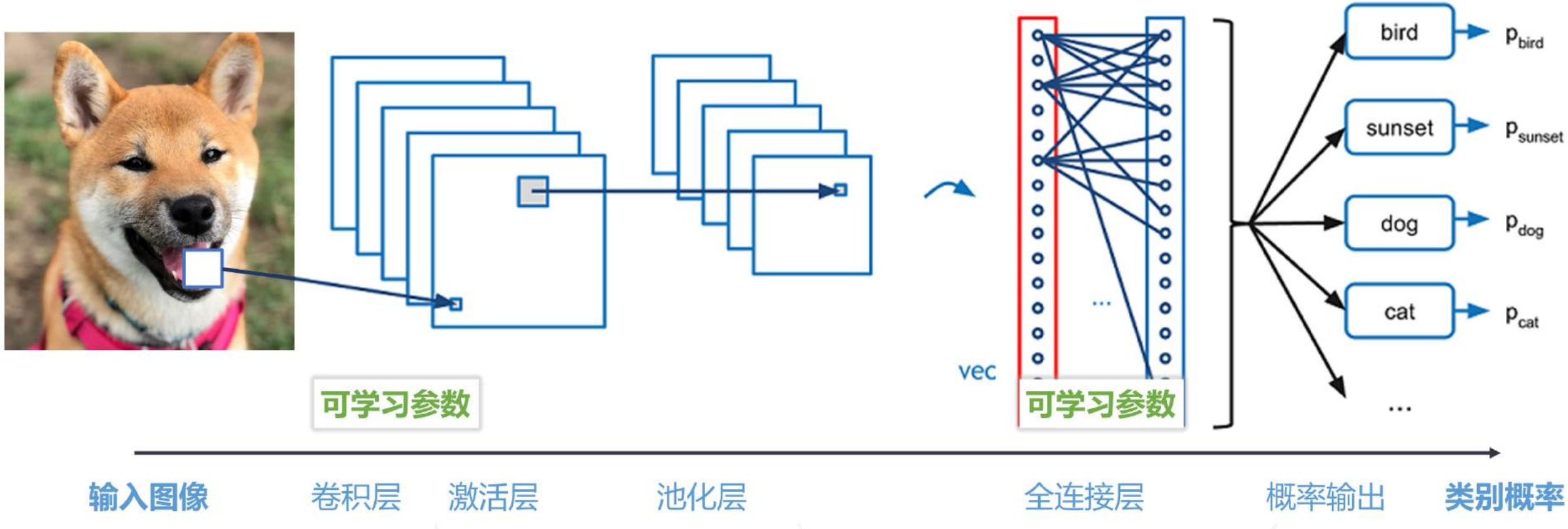
通用近似定理 Universal Approximation Theorem

- 在神经元数目足够的情况下，单隐层的神经网络可以逼近任意连续函数（1990s~2000s）
- 所需的神经元的数目随数据维度指数级增长

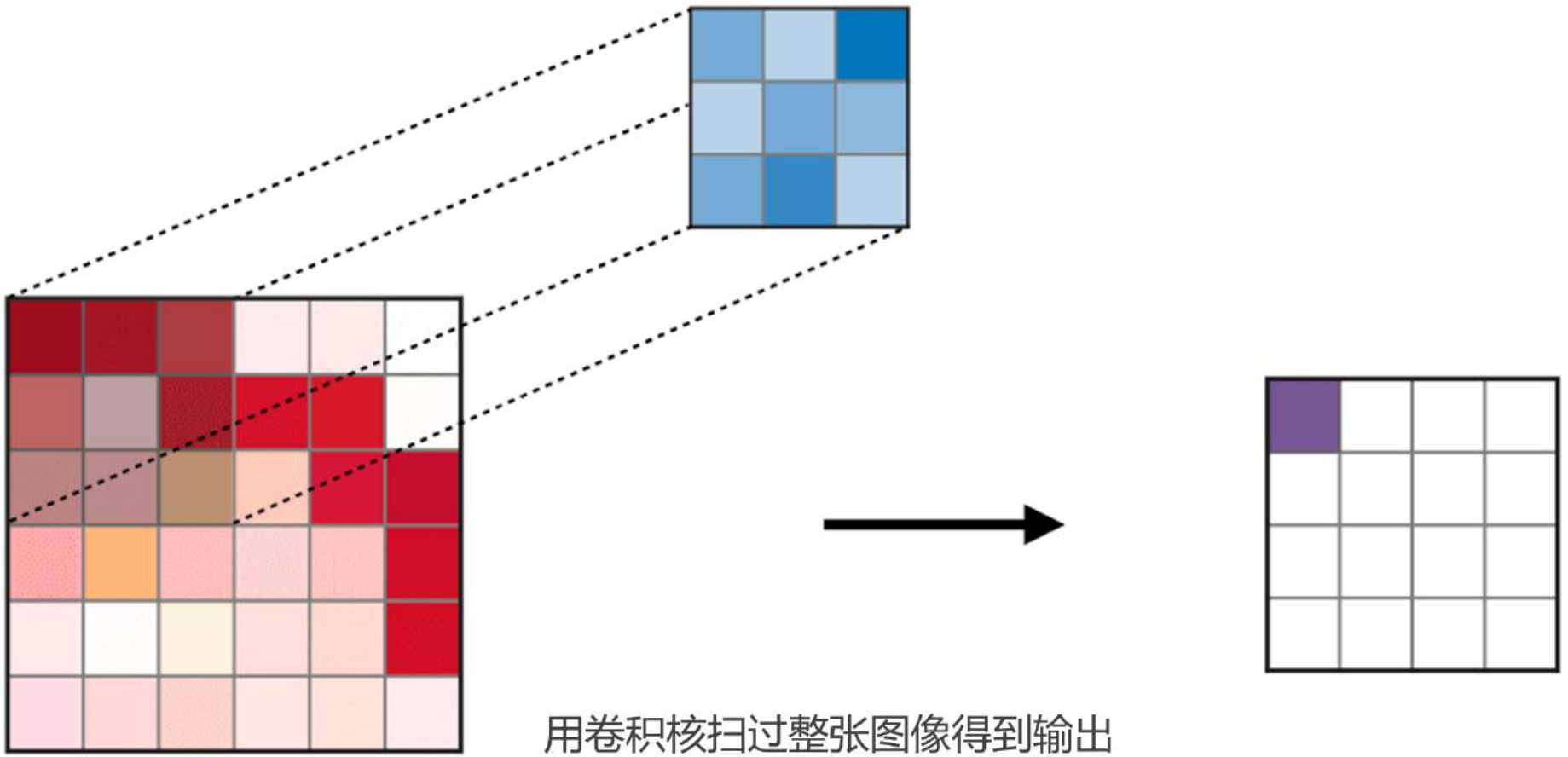
- 图像分类 ≠ 拟合**任意复杂**的分类函数
- 从实用角度讲，需要根据图像的特点设计更高效的模型

卷积神经网络

卷积神经网络的整体结构



二维卷积

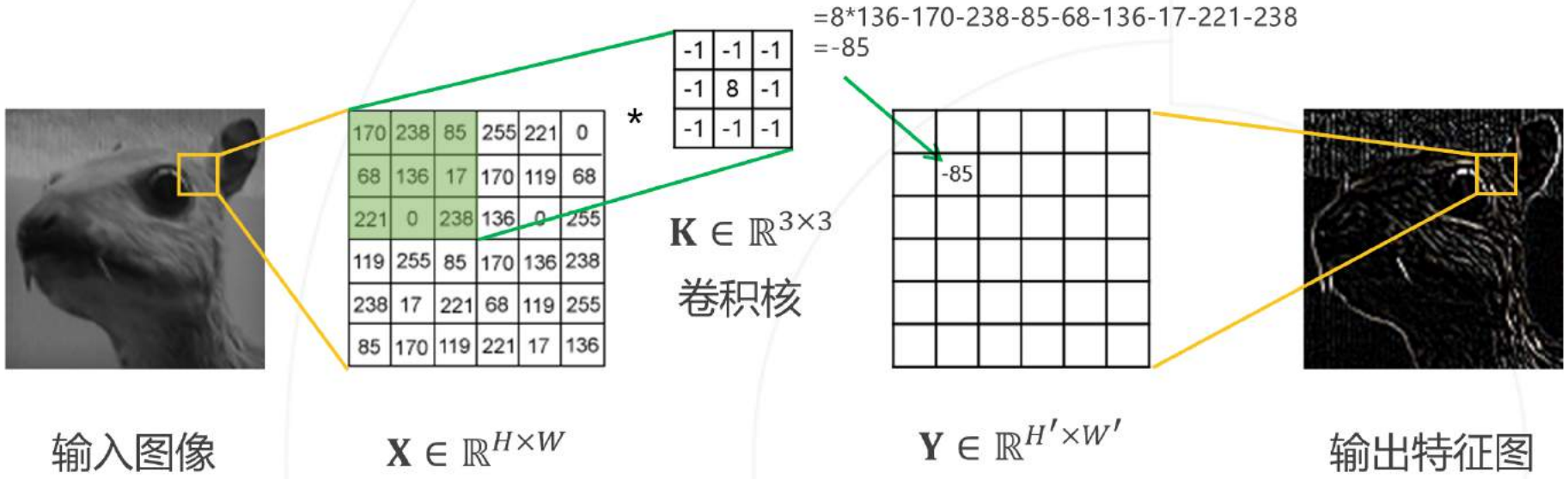


用卷积核扫过整张图像得到输出

$$\mathbf{Y}[h,w] = \sum_{i=-1}^{1}\sum_{j=-1}^{1} \mathbf{X}[h+i,w+j]\mathbf{K}[i,j]$$

图片：CS 230 Cheat Sheet

灰度图像的卷积

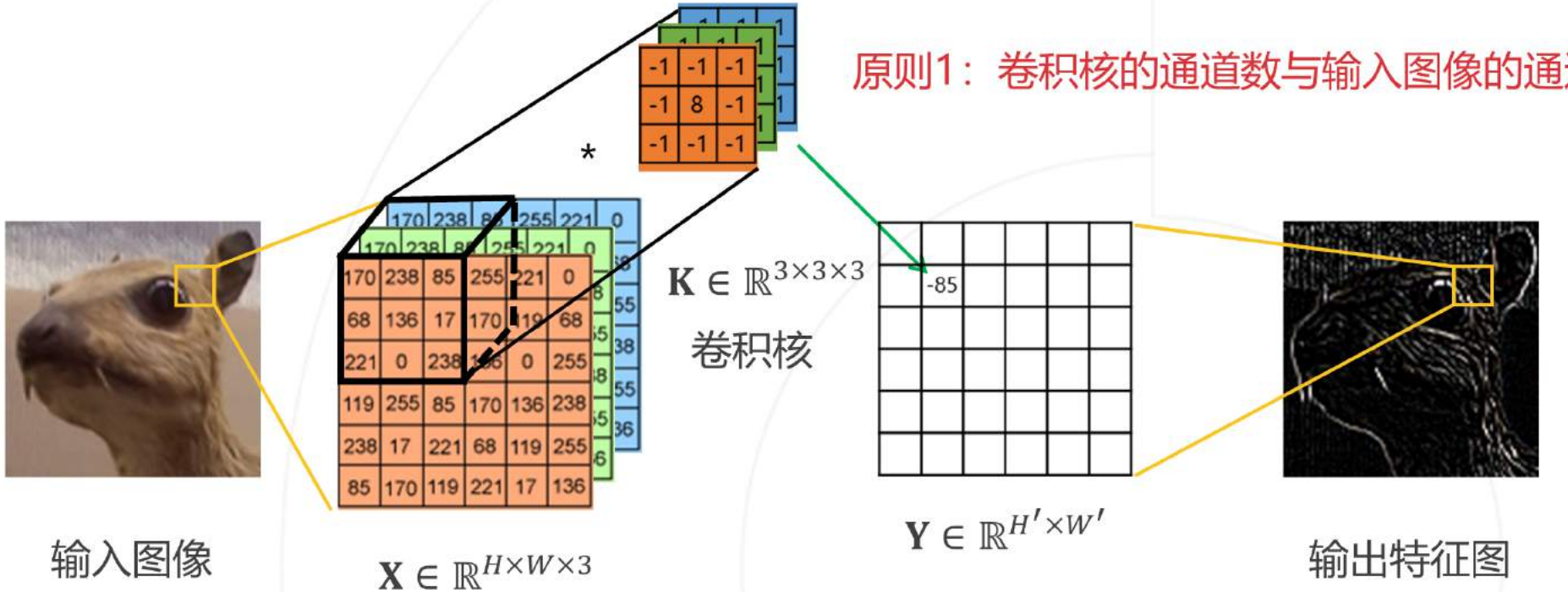


用卷积核扫过整张图像得到输出

$$\mathbf{Y}[h,w] = \sum_{i=-1}^{1}\sum_{j=-1}^{1} \mathbf{X}[h+i, w+j]\mathbf{K}[i,j]$$

图片来源：https://en.wikipedia.org/wiki/Kernel_(image_processing)

彩色图像的卷积

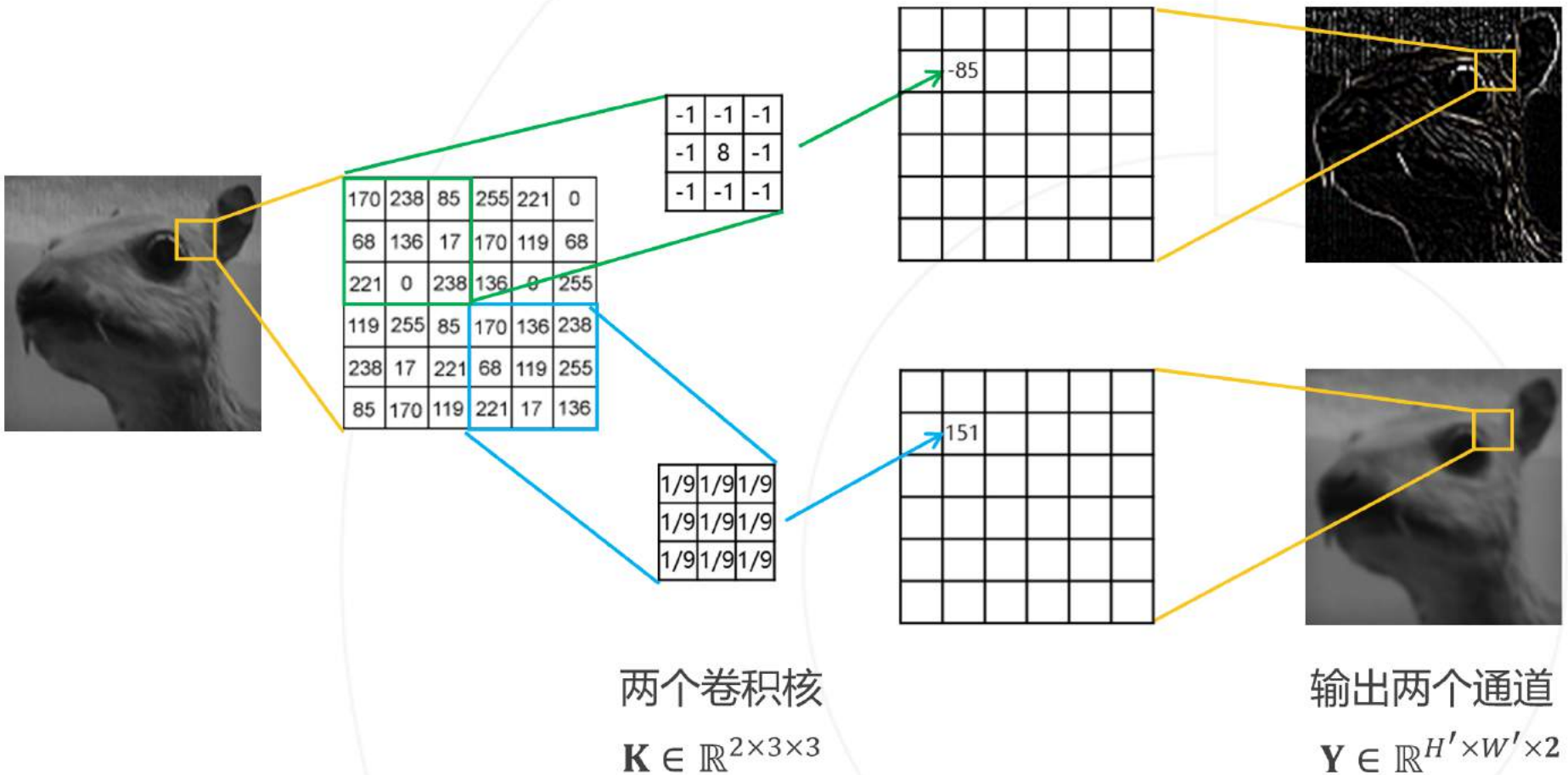


原则1：卷积核的通道数与输入图像的通道数相同

$$\mathbf{Y}[h,w] = \sum_{i=-1}^{1}\sum_{j=-1}^{1}\sum_{k=1}^{3}\mathbf{X}[h+i,w+j,k]\mathbf{K}[i,j,k]$$

图片来源：https://en.wikipedia.org/wiki/Kernel_(image_processing)

使用多个卷积核



两个卷积核

$\mathbf{K} \in \mathbb{R}^{2\times3\times3}$

输出两个通道

$\mathbf{Y} \in \mathbb{R}^{H'\times W'\times 2}$

原则2：输出特征图的通道数与卷积核的个数相同

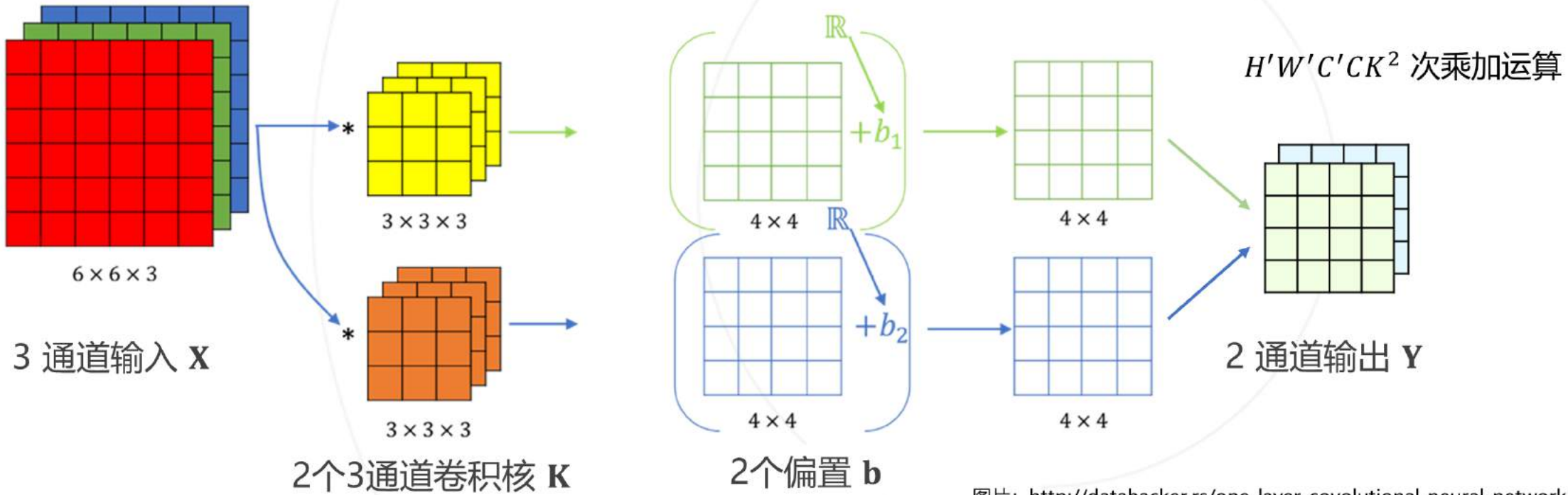
图片来源：https://en.wikipedia.org/wiki/Kernel_(image_processing)

卷积层 Convolutional Layer

卷积层通过卷积运算，将 C 通道的特征图 $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ 变换为 C' 通道的特征图 $\mathbf{Y} \in \mathbb{R}^{H' \times W' \times C'}$，计算公式为：

$$\mathbf{Y}[h, w, c] = \sum_{i=1}^{K} \sum_{j=1}^{K} \sum_{k=1}^{C} \mathbf{X}[h+i, w+j, k] \, \mathbf{K}_c[i, j, k] + \mathbf{b}[c]$$

其中，$\mathbf{K} \in \mathbb{R}^{C' \times K \times K \times C}$ 为 C' 个 C 通道的卷积核，$\mathbf{b} \in \mathbb{R}^{C'}$ 为每个输出通道对应的偏置，二者均为可学习的参数。



$H'W'C'CK^2$ 次乘加运算

图片：http://datahacker.rs/one-layer-covolutional-neural-network/

PyTorch 实现卷积层

```
# Functional API
img = torch.randn(1, 3, 8, 12) # N,C,H,W
weight = torch.randn(6, 3, 3, 3) # C’,C,K,K
bias = torch.randn(6) # C’
out = F.conv2d(img, weight, bias, padding=1)

# Class API
conv = nn.Conv2d(3, 6, kernel_size=(3, 3), padding=1) # C, C’
img = torch.randn(1, 3, 8, 12) # N,C,H,W
y = conv(img)
```

激活层 Activation Layer

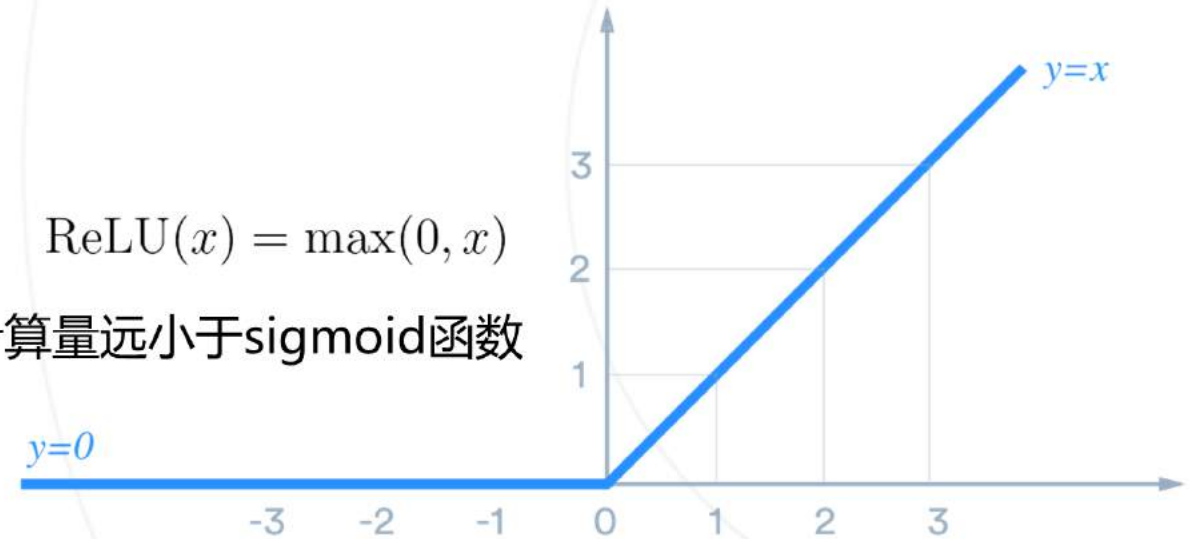
- 激活层基于一个非线性函数 σ: $\mathbb{R} \to \mathbb{R}$ 对输入特征图 $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ 进行逐元素的变换：

$$\mathbf{Y}[h, w, c] = \sigma(\mathbf{X}[h, w, c])$$

- 激活层通常不包含可学习参数。
- ReLU (**Re**ctified **L**inear **U**nit) 是卷积网络中最常用的非线性激活函数。

$$\mathrm{ReLU}(x) = \max(0, x)$$

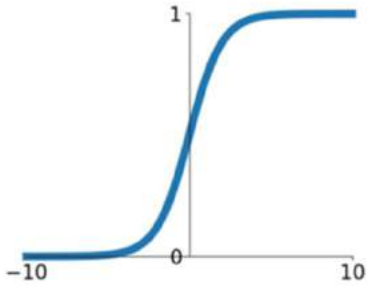
计算量远小于sigmoid函数



常用激活函数

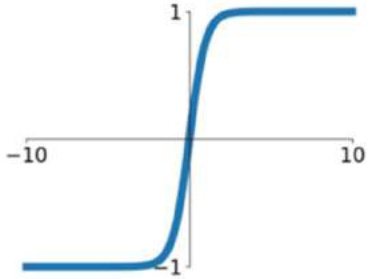
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



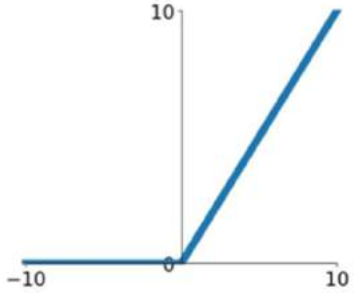
tanh

$$\tanh(x)$$



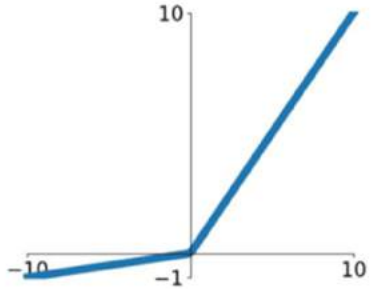
ReLU

$$\max(0, x)$$



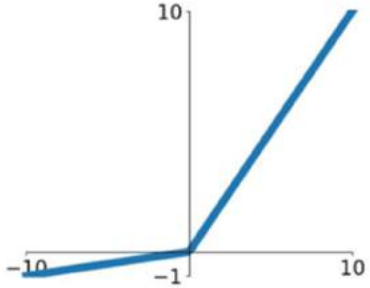
Leaky ReLU

$$\max(0.1x, x)$$



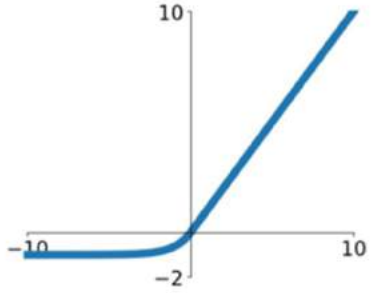
PReLU

$$\max(ax, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



PyTorch 实现激活层

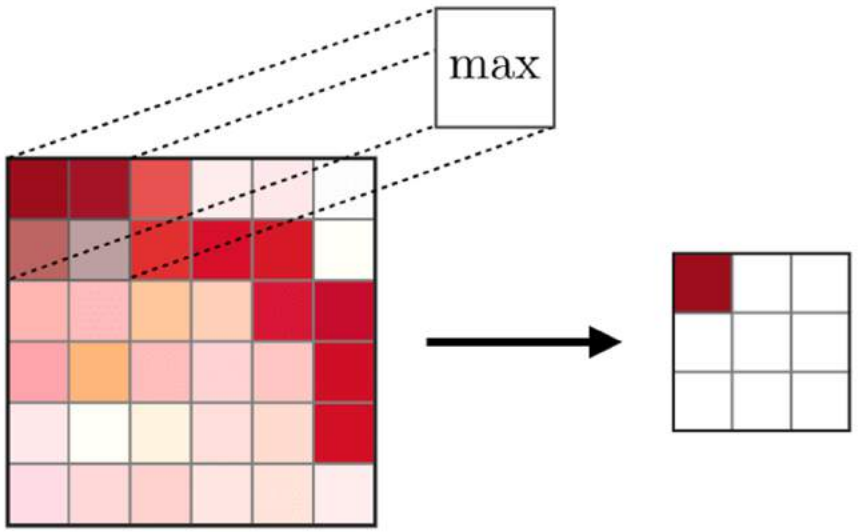
```
x = torch.randn(3, 3)

# Functional API
y = F.relu(x)

# Class API
relu = nn.ReLU()
y = relu(x)
```

池化层 Pooling Layer

池化层在特征图的局部区域内计算最大值或平均值，从而降低特征图分辨率，节省计算量，提高特征的空间鲁棒性。

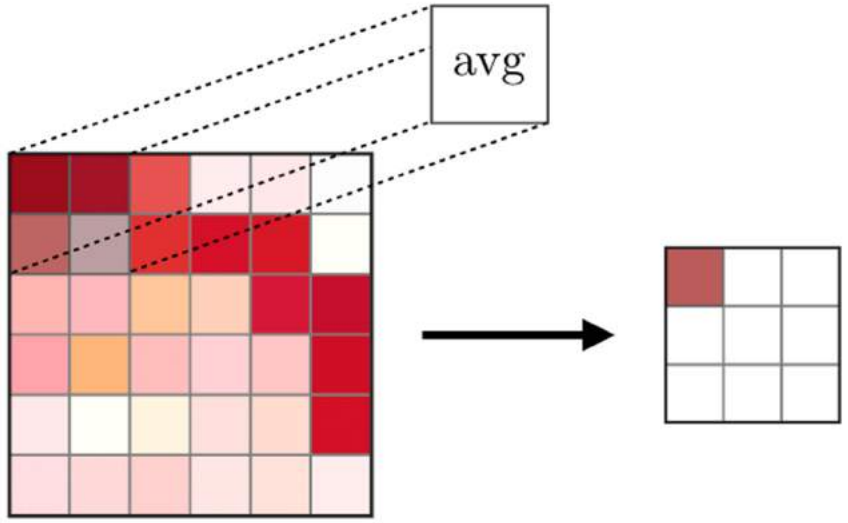
最大池化

$$\mathbf{Y}[h,w,c] = \max_{1\leq i,j\leq k} \mathbf{X}[sh+i, sw+j, c]$$



平均池化

$$\mathbf{Y}[h,w,c] = \frac{1}{k^2} \sum_{1\leq i,j\leq k} \mathbf{X}[sh+i, sw+j, c]$$



图片：CS 230 Cheat Sheet

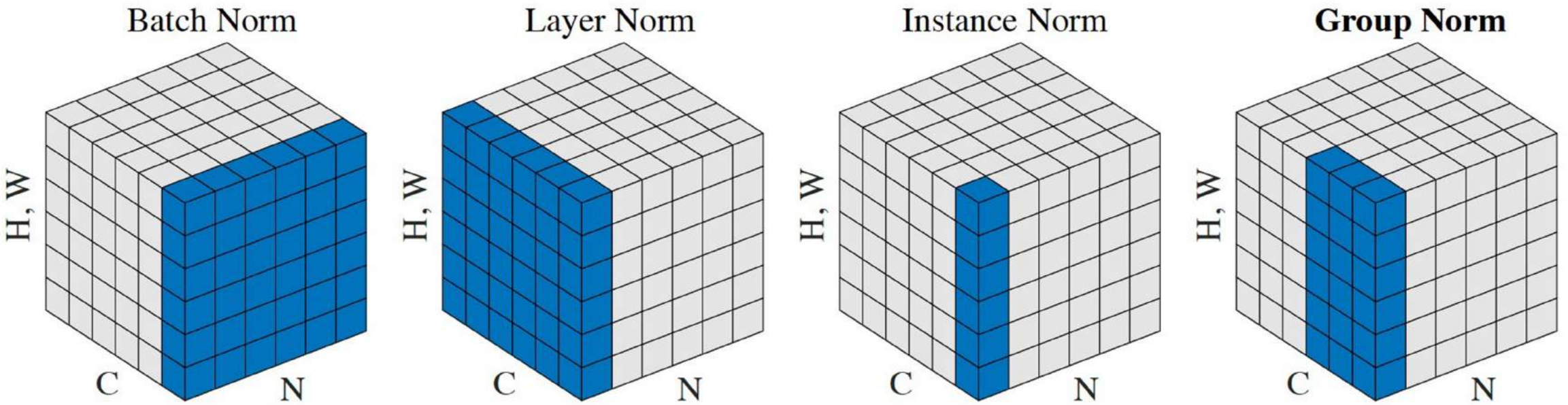
PyTorch 实现最大池化

OpenMMLab

```
x = torch.randn(1, 1, 4, 4)

# Functional API
y = F.max_pool2d(x, kernel_size=2, stride=2)

# Class API
pool = nn.MaxPool2d(kernel_size=(2, 2), stride=2)
y = pool(x)
```

归一化层 Normalization Layer

将特征图中的激活值进行归一化处理，例如0均值1方差，降低训练难度。



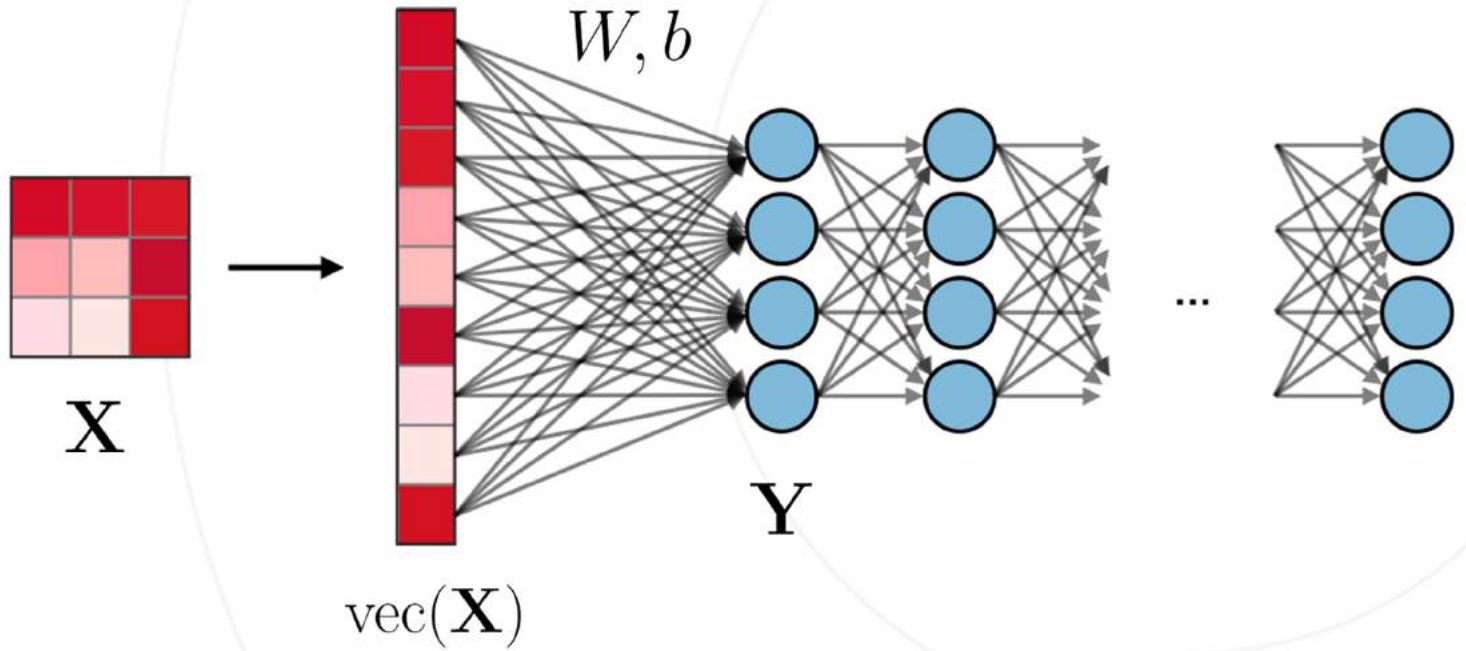
Wu, Yuxin, and Kaiming He. "Group normalization." Proceedings of the European conference on computer vision (ECCV). 2018.

全连接层 Fully Connected Layer

- 全连接层通过矩阵乘法将输入特征 $\mathrm{X} \in \mathbb{R}^N$ 映射为输出特征 $\mathrm{Y} \in \mathbb{R}^M$

$$\mathbf{Y} = W\mathbf{X} + b \quad \text{或} \quad \mathbf{Y} = W\mathrm{vec}(\mathbf{X}) + b$$

- 全连接层包含可学习参数 $W \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M$ 。
- 多层感知器中的层就是全连接层。



图片：CS 230 Cheat Sheet

PyTorch 实现全连接层

OpenMMLab

```
# Vectorize
fmap = torch.randn(1, 2, 2, 2)
x = fmap.view(-1, 8)

# Functional API
weight = torch.randn(6, 8)
bias = torch.randn(6)
y = F.linear(x, weight, bias)

# Class API
linear = nn.Linear(8, 6) # in_features=8, out_features=6
y = linear(x)
```

概率输出层

- 概率输出层 $S\colon \mathbb{R}^K \to [0,1]^K$ 将任意特征向量转换为概率概率向量

$$P(y|x) = S\big(h(x;\Theta)\big)$$

- 二分类，全连接层输出标量 $z \in \mathbb{R}$，通过 Sigmoid 函数计算正类的概率

$$P(y=1|x) = \frac{1}{1+\exp(-z)}$$

- 多分类，全连接层输出 K 维向量 $z \in \mathbb{R}^K$，通过 Softmax 函数计算 K 类的概率

$$P(y=k|x) = \frac{\exp z_k}{\sum_{i=1}^{K} \exp(z_k)}$$

网络

$$\boldsymbol{z} = \begin{bmatrix} -0.5 \\ 0.1 \\ 1 \end{bmatrix}$$

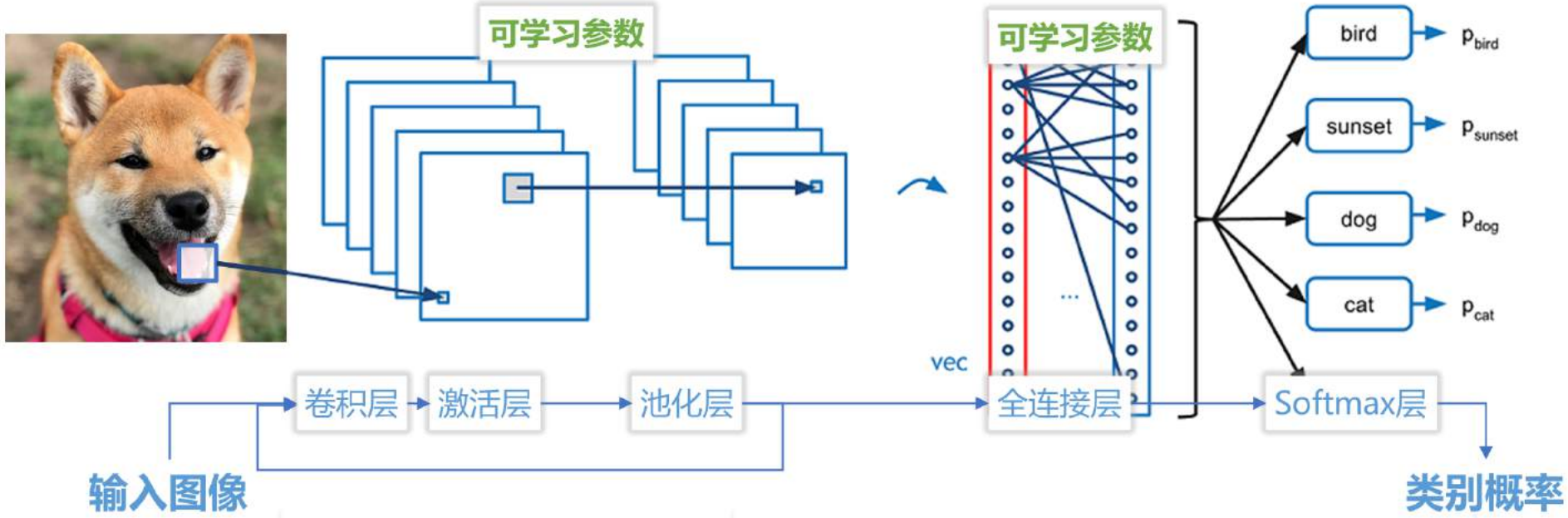
$$\boldsymbol{P} = \begin{bmatrix} 0.14 \\ 0.25 \\ 0.61 \end{bmatrix}$$

PyTorch 实现 Softmax

OpenMMLab

```
z = torch.rand(3)

# Functional API
p = F.softmax(z)

# Class API
s = nn.Softmax()
p = s(z)
```

特征与分类器



一个端到端学习的分类器 $P(y|x) = h(x;\Theta)$

学习一个好的图像特征

简单的分类器完成分类

视觉任务的初衷　　稳健表达图像内容，不受像素外观影响

再谈卷积

全连接层 vs 卷积层

- 二者都是线性层，实现了输入到输出的线性映射

$$\mathbf{Y}[h,w,c] = \sum_{i=1}^{K}\sum_{j=1}^{K}\sum_{k=1}^{C} \mathbf{X}[h+i,w+j,k]\ \mathbf{K}_c[i,j,k] + \mathbf{b}[c]$$

$$\mathbf{Y}[m] = \sum_{i=1}^{N} W_{mi}\mathbf{X}[i] + b[m]$$

- 卷积层输入输出之间的连接仅存在于局部空间，且连接的权重在不同输出单元之间是共享的。

全连接层 vs 卷积层

全连接层：

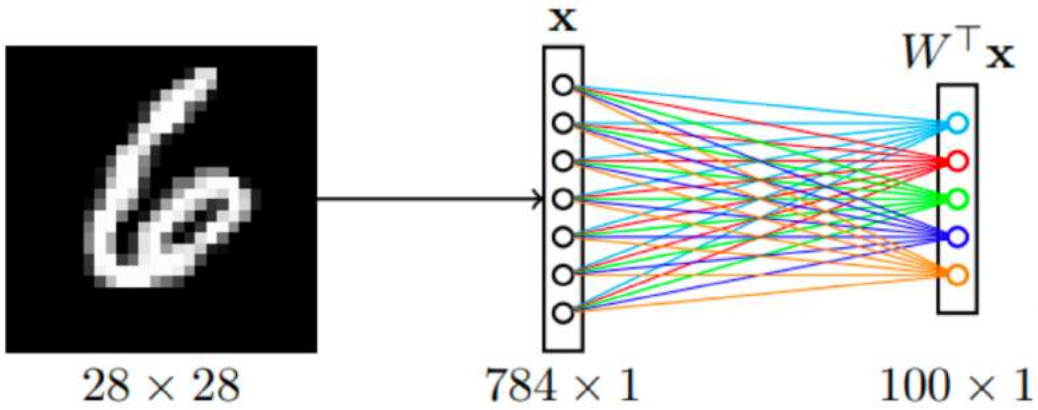
- 密集连接
- 独立权重

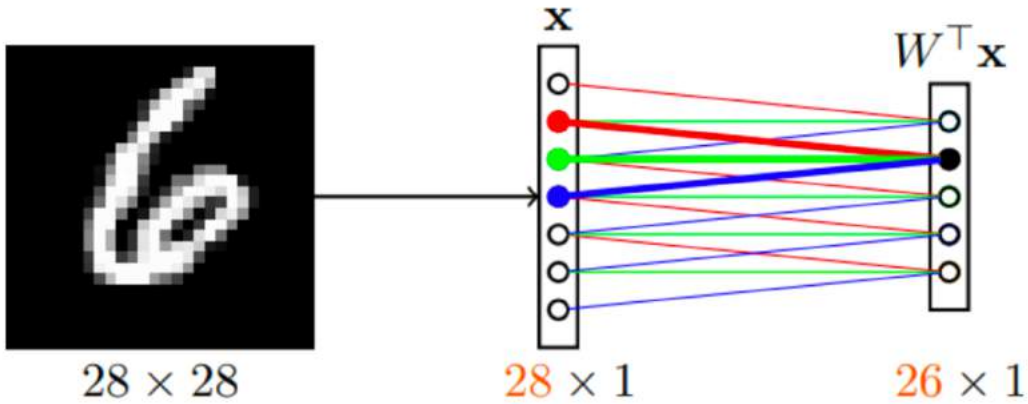
大量冗余参数

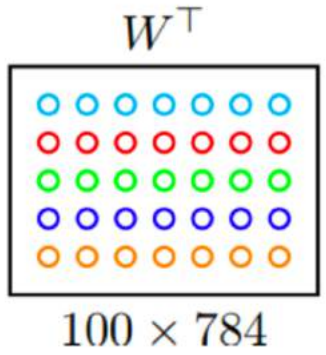
卷积层：

- 局部连接：像素局部相关
- 共享权重：位移不变性

大量节约参数
有效提取图像特征

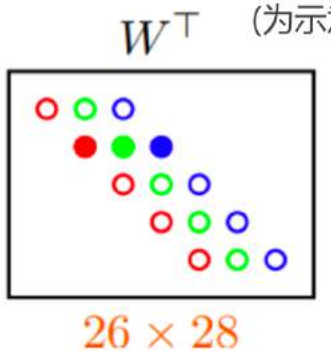






将卷积核中的元素按照向量化输入和输出之间的关系排列成矩阵 W

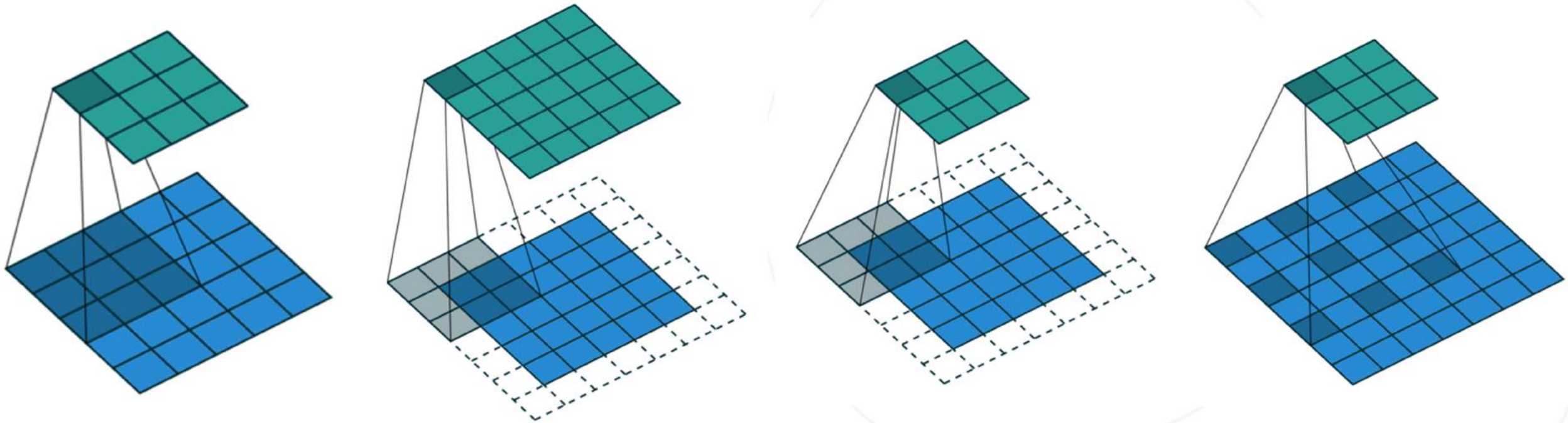
(为示意方便仅展示一维像素)



卷积的变形

卷积层可以增加边缘填充 (padding)、步长 (stride)、和空洞 (dilation)

$$\mathbf{Y}[h, w, c] = \sum_{i=1}^{K} \sum_{j=1}^{K} \sum_{k=1}^{C} \mathbf{X}[sh + di - p, sw + dj - p, k] \, \mathbf{K}_c[i, j, k] + \mathbf{b}[c]$$



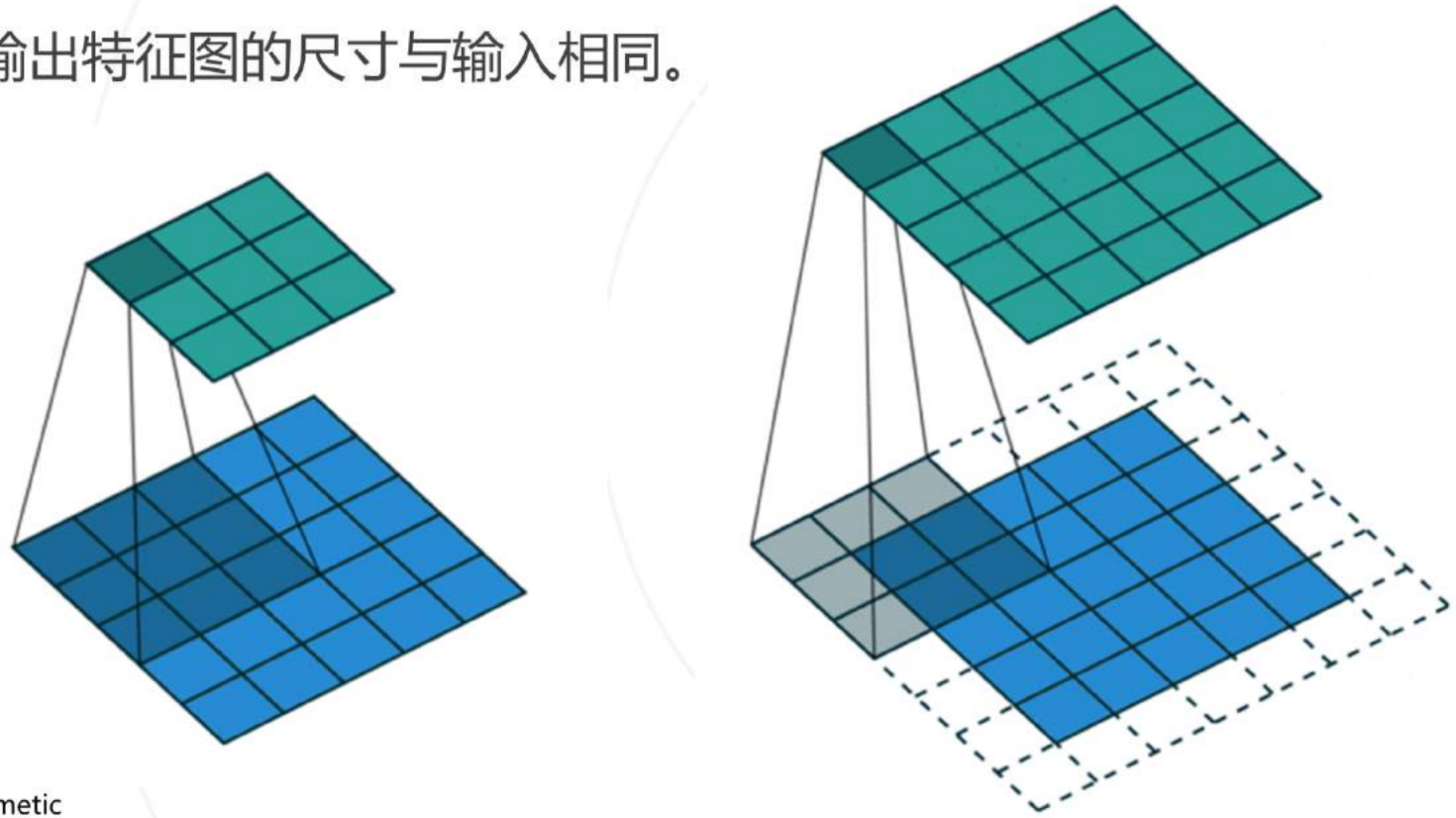
图片：https://github.com/vdumoulin/conv_arithmetic

卷积的边缘填充

- 边缘填充 (padding) 在输入特征图的边界处填补 p 个像素宽度的 0 值，以扩大输入和输出特征图的空间尺寸。

$$H' = H - K + 1 + 2p$$

- 例：当 $p = \frac{F-1}{2}$ 时，输出特征图的尺寸与输入相同。



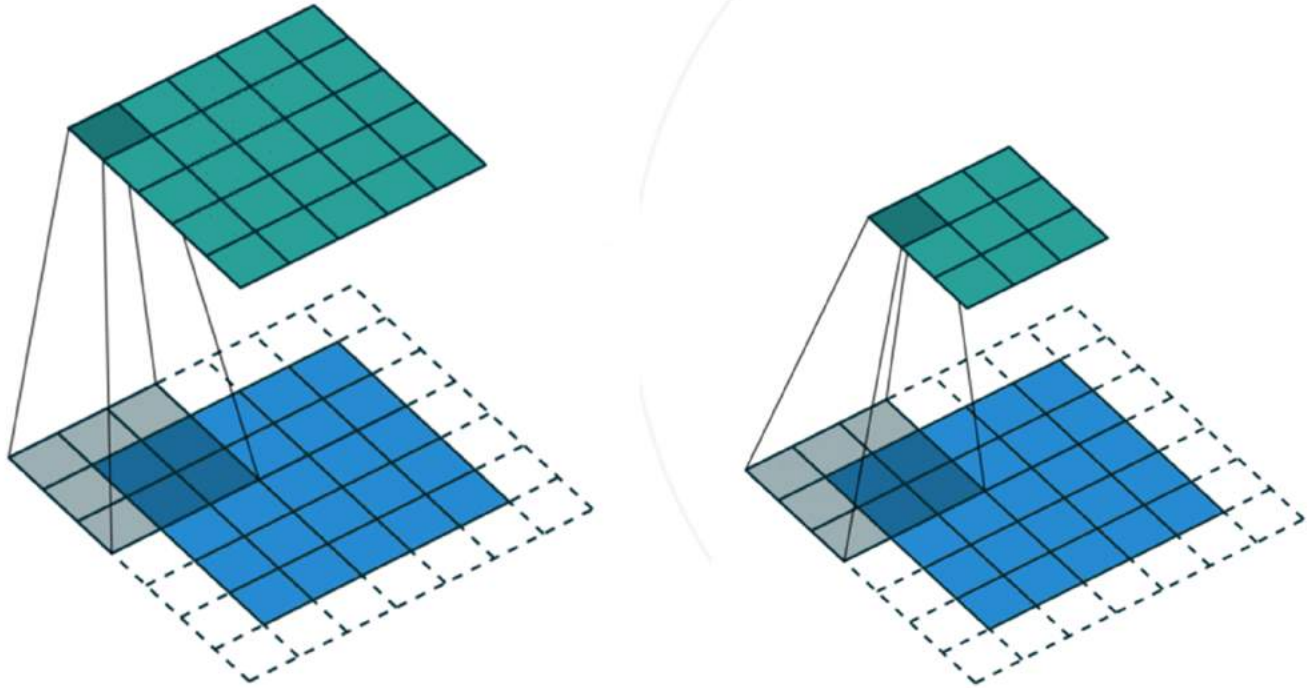
图片：https://github.com/vdumoulin/conv_arithmetic

卷积的步长

- 步长 (padding) 降低卷积计算的空间频率，实现空间降采样。

$$H' = \left\lfloor \frac{H - F + 2p}{S} \right\rfloor + 1$$

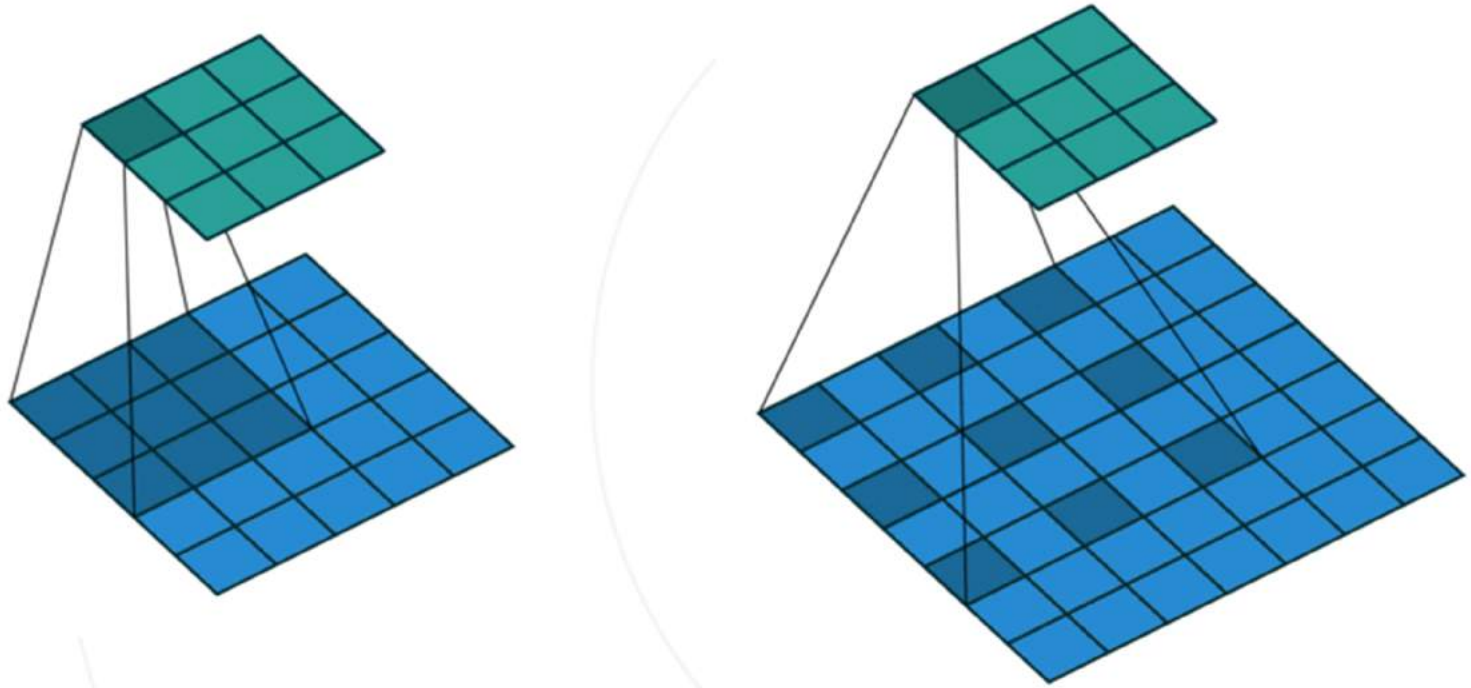
- 例：$F = 3, p = 1, S = 2$ 且输入分辨率 H 为偶数的时候，输出分辨率恰好为输入的一半。模型中广泛使用。



图片：https://github.com/vdumoulin/conv_a...

空洞卷积

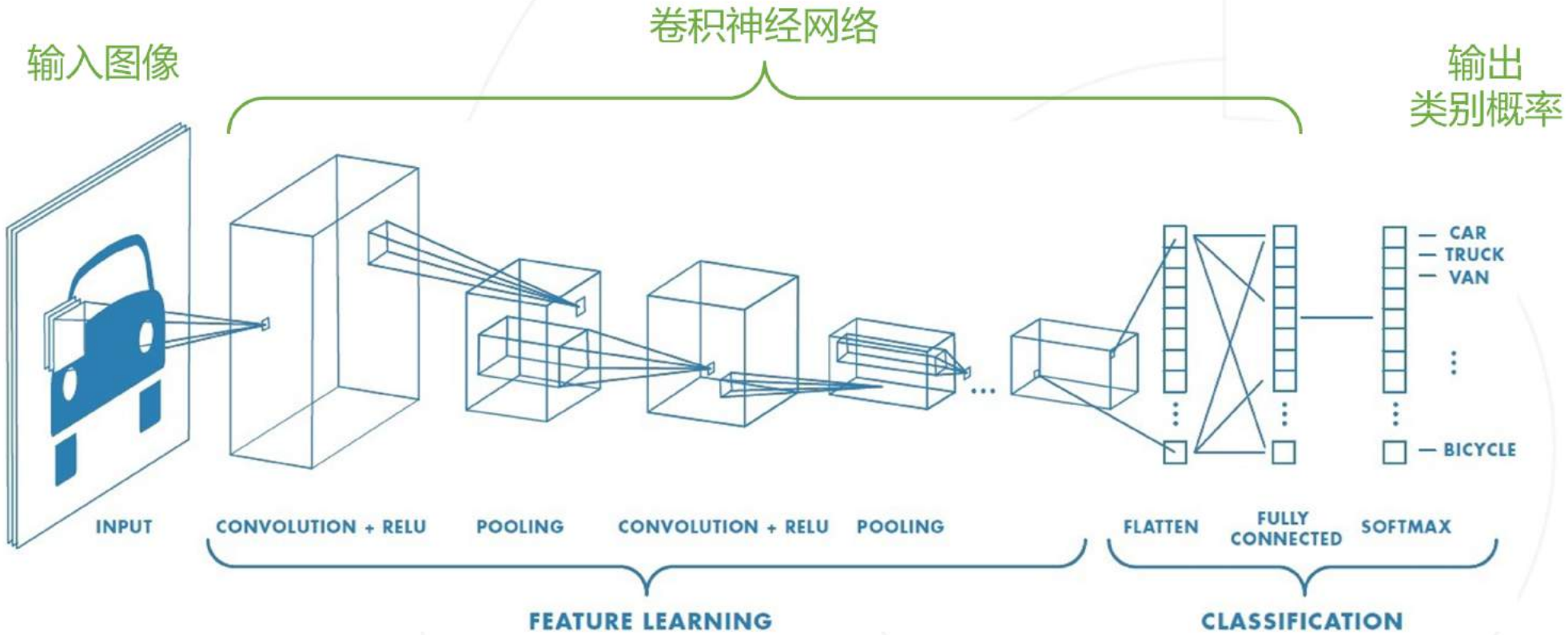
- 空洞卷积 (dilated convolution) 可以在不增加参数的情况下“扩大”卷积核，从而增加卷积层感受野。在语义分割模型中广泛应用。



图像分类模型设计

模型结构

图像分类网络通常可以分为**主干网络**和**分类器**两部分



主干网络产生特征图

分类器预测类别概率

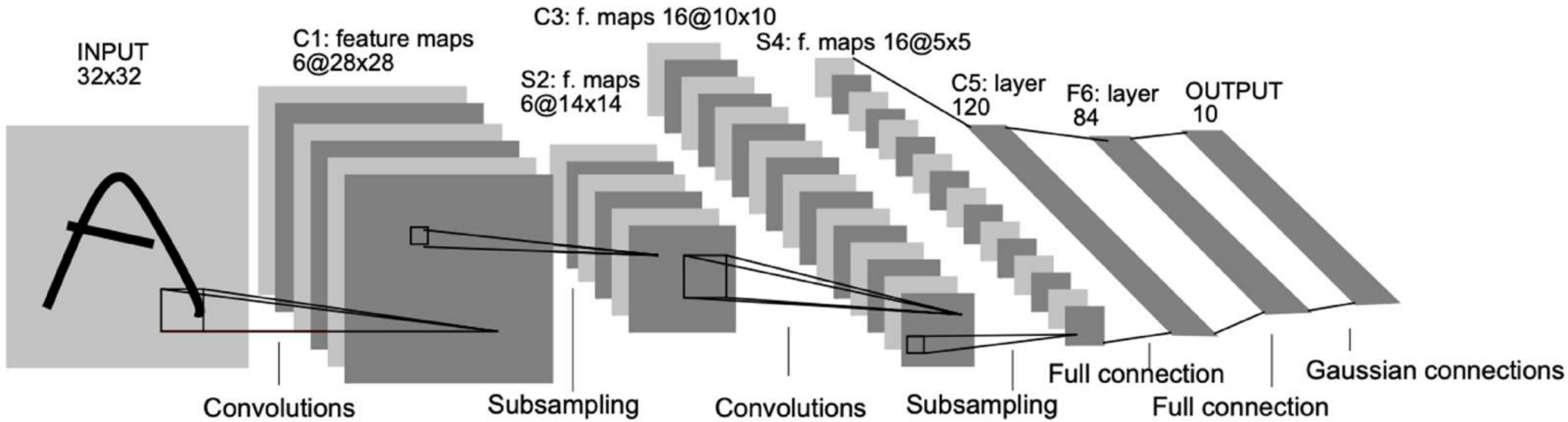
图片：https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

LeNet-5 (1998)

最早的卷积神经网络之一。

LeNet-5 由 7 层网络组成，其中 2 层为卷积层，2 层为下采样层，3 层为全连接层。

60K 个可学习参数。



LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

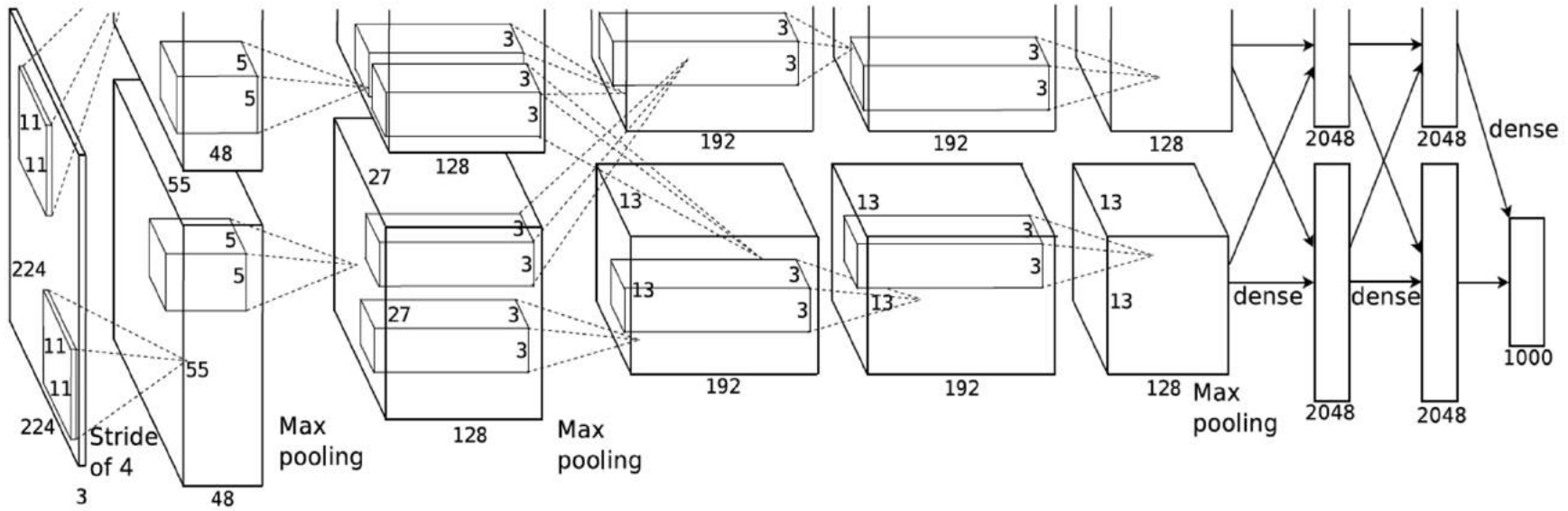
AlexNet (2012)

第一个成功应用于大规模图像分类的卷积神经网络模型。

8 个可学习层，5 个卷积层，3 个全连接层，共有 60M 个可学习参数。

使用 2 个 NVIDIA GTX 580 GPU 训练了 1 周。

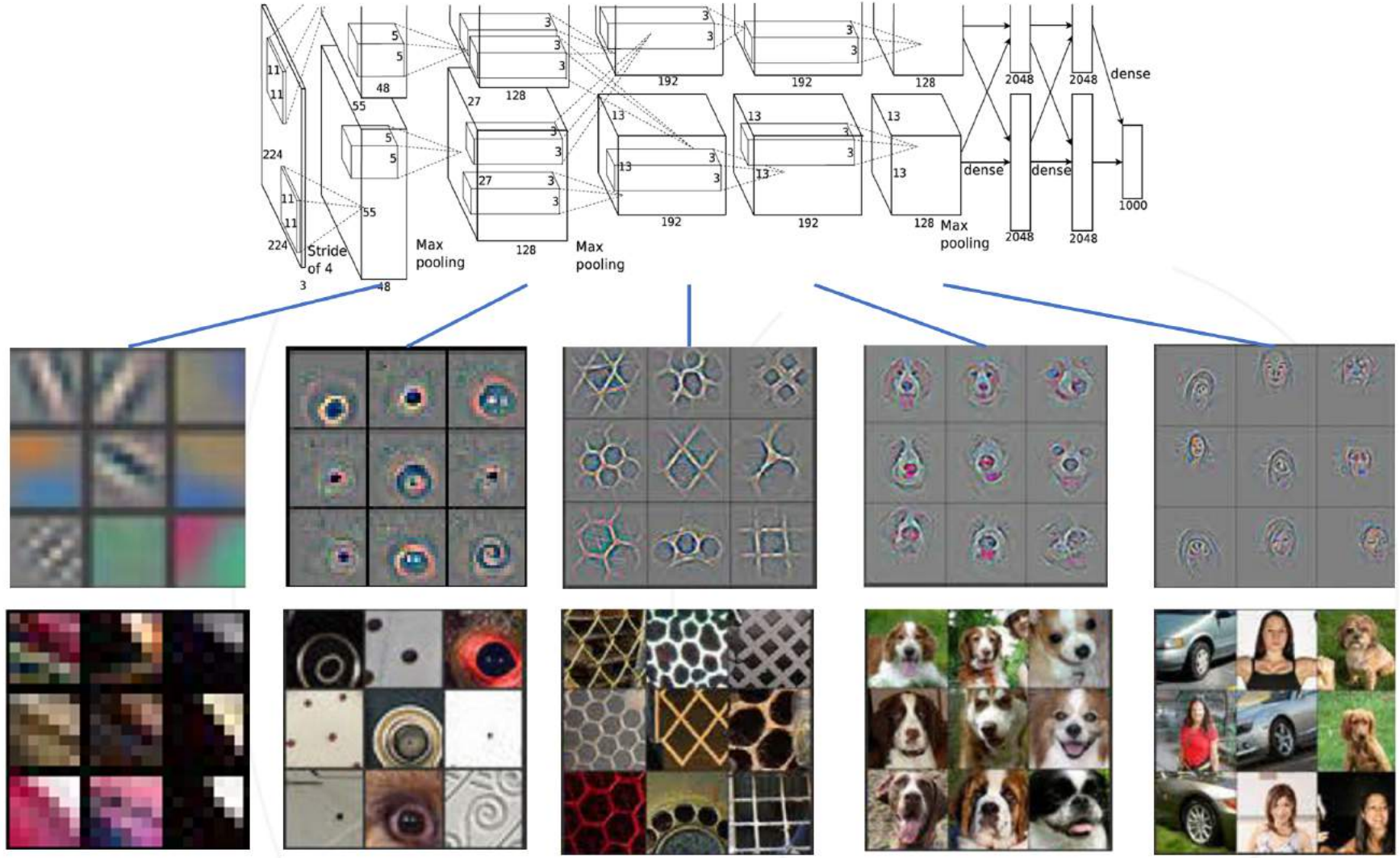
输入图像



1000类
类别概率

Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Advances in neural information processing systems, 2012, 25: 1097-1105.

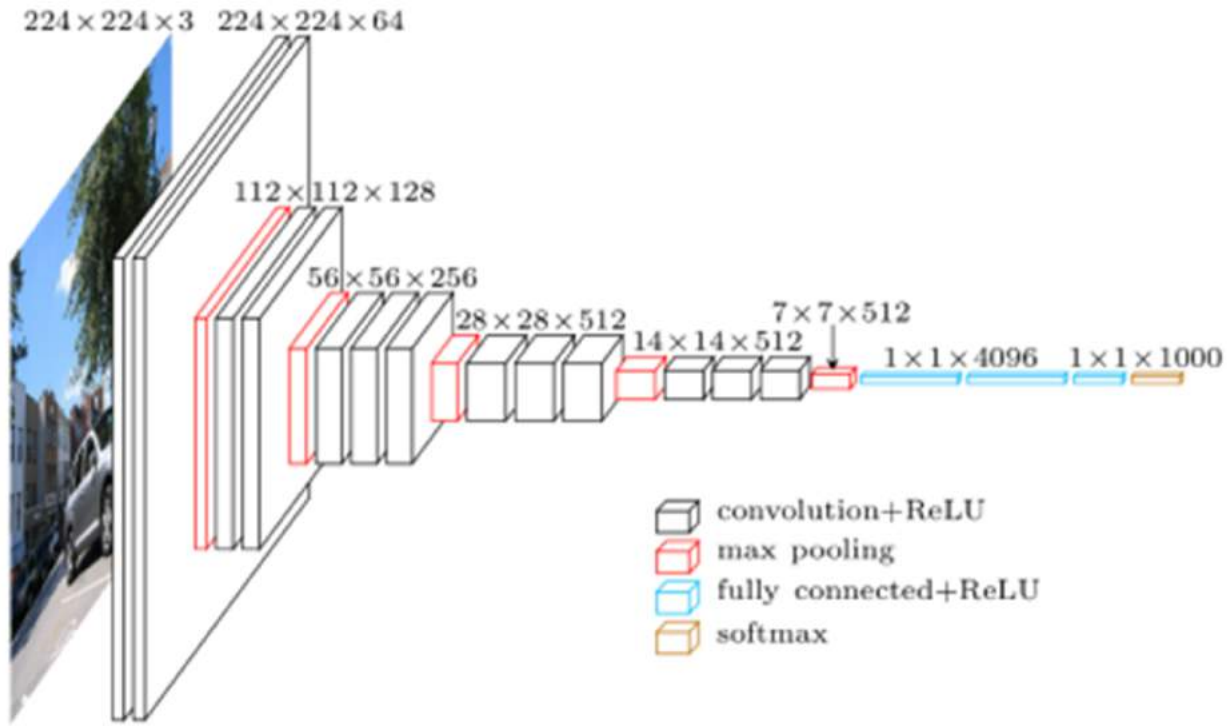
卷积网络中的多层次特征

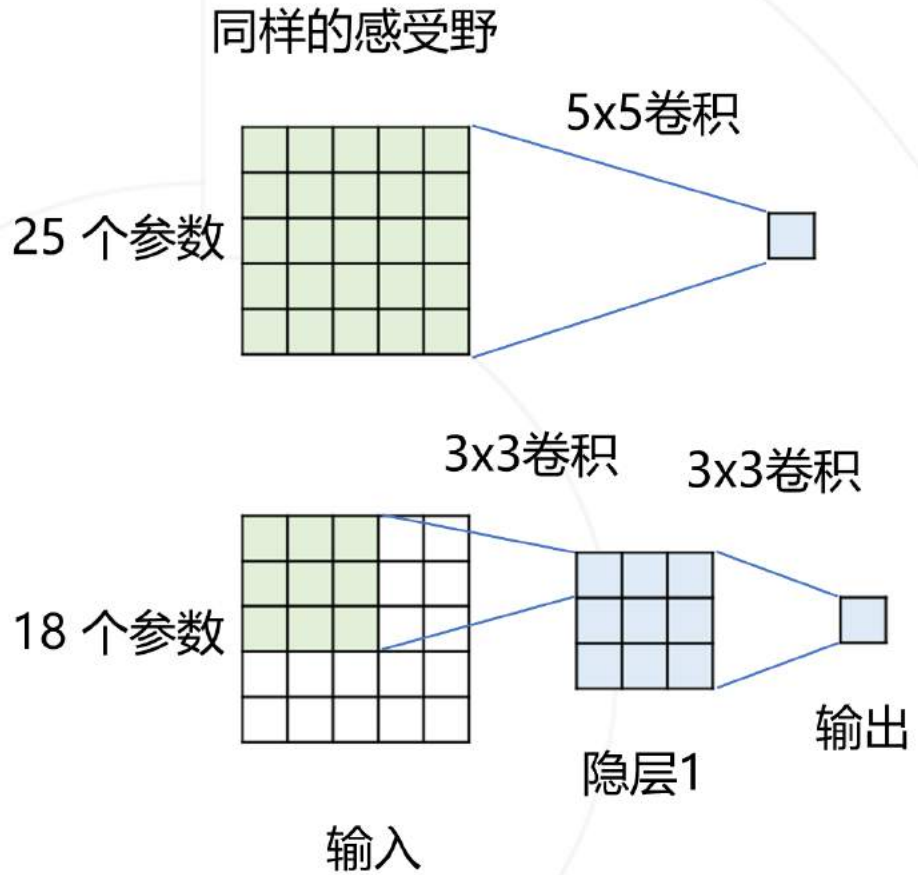


Zeiler M D, Fergus R. Visualizing and understanding convolutional networks[C]//European conference on computer vision. Springer, Cham, 2014: 818-833.

VGGNet (2014)

- 只使用 3x3 的卷积核
- 层数增多：16~19层
- 边界填充1像素，维持空间分辨率
- 倍增通道数的同时减半空间分辨率





Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

VGGNet (2014)

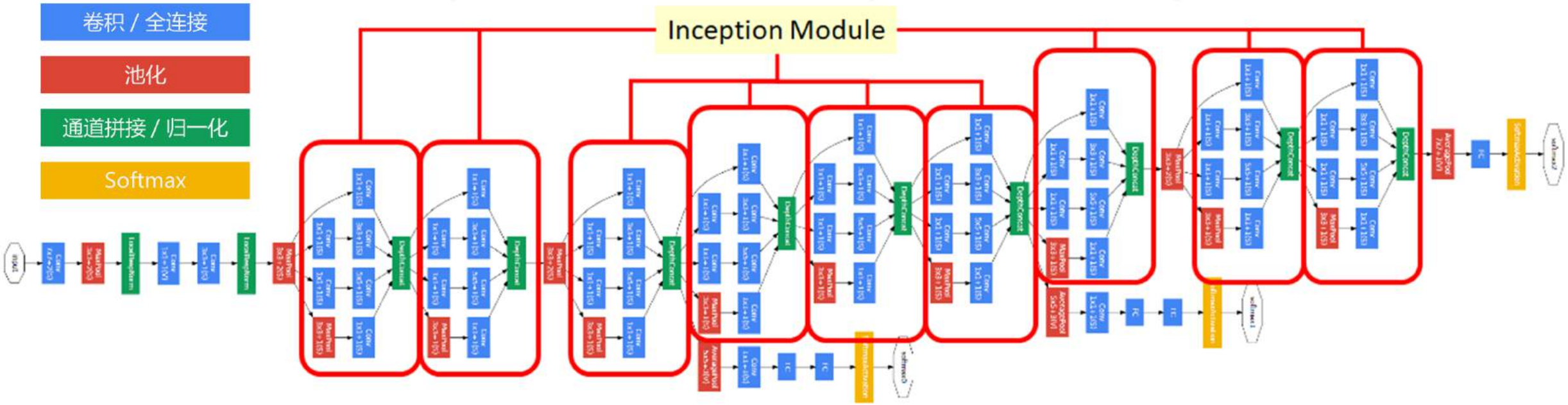
- VGG-16 有 138M 权重参数，其中超过 100M 来自全连接层。

VGG16 - Structural Details													
#	Input Image			output			Layer	Stride	Kernel		in	out	Param
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
Total													138,423,208

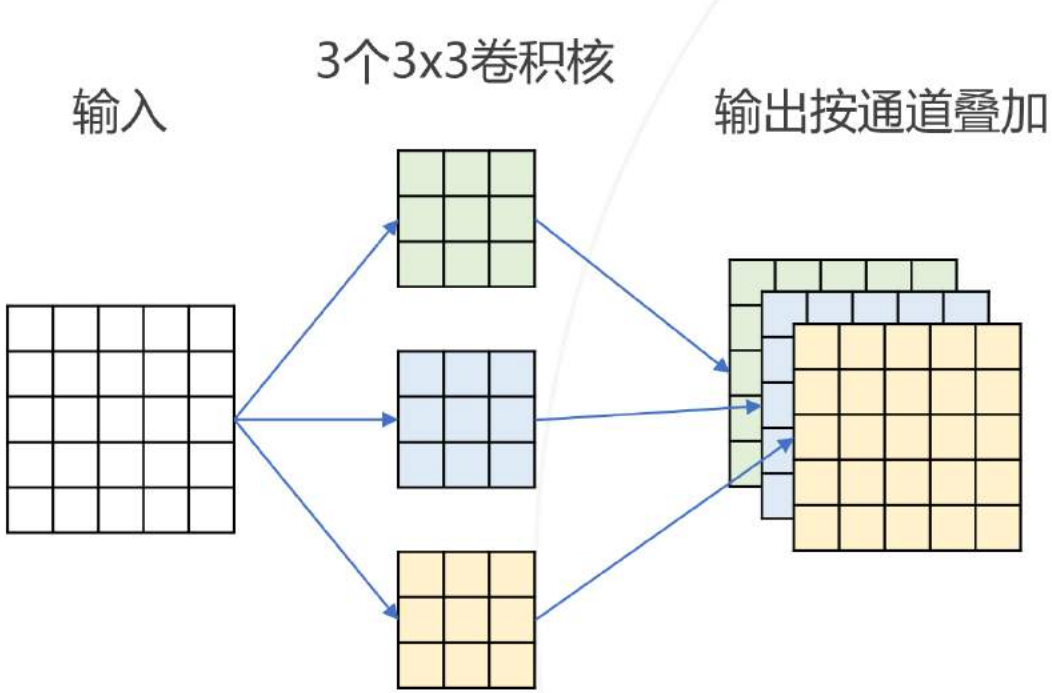
https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

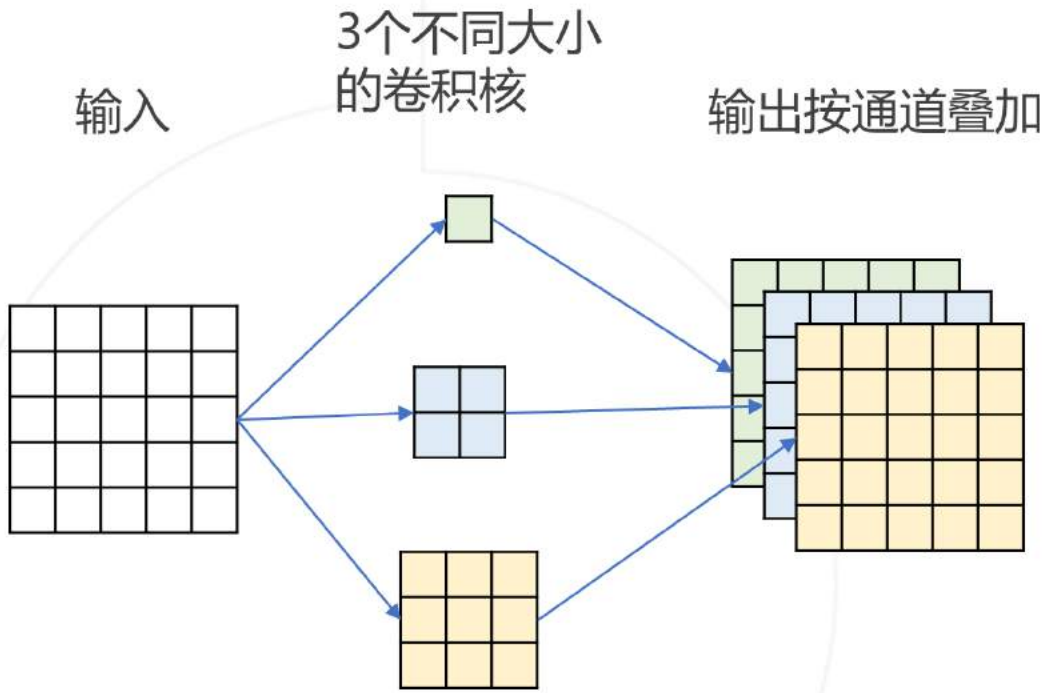
GoogLeNet (Inception v1, 2014)

- 使用 Inception 模块，在同一层使用不同尺寸的卷积核
- 22 个可学习层，仅 7M 权重参数



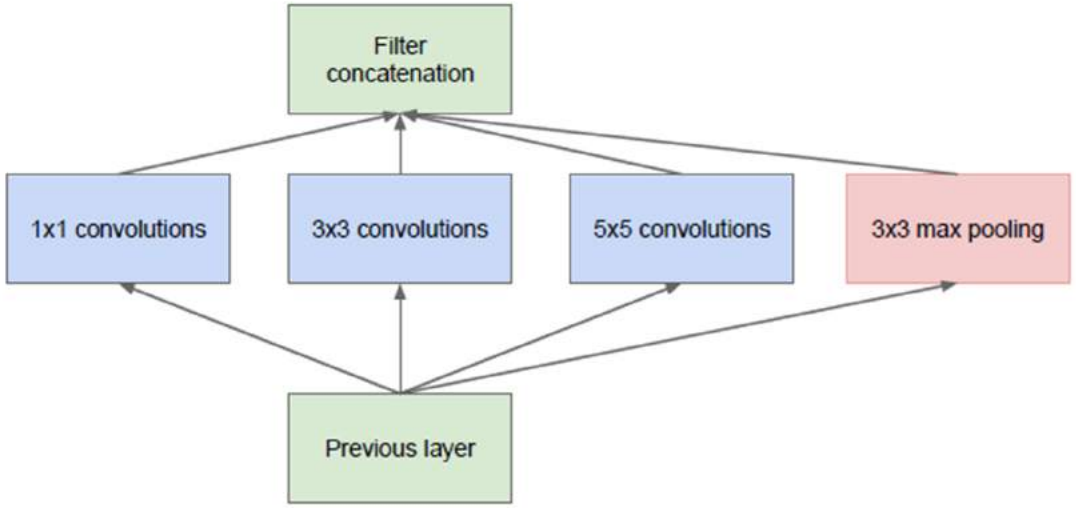
使用不同大小的卷积核



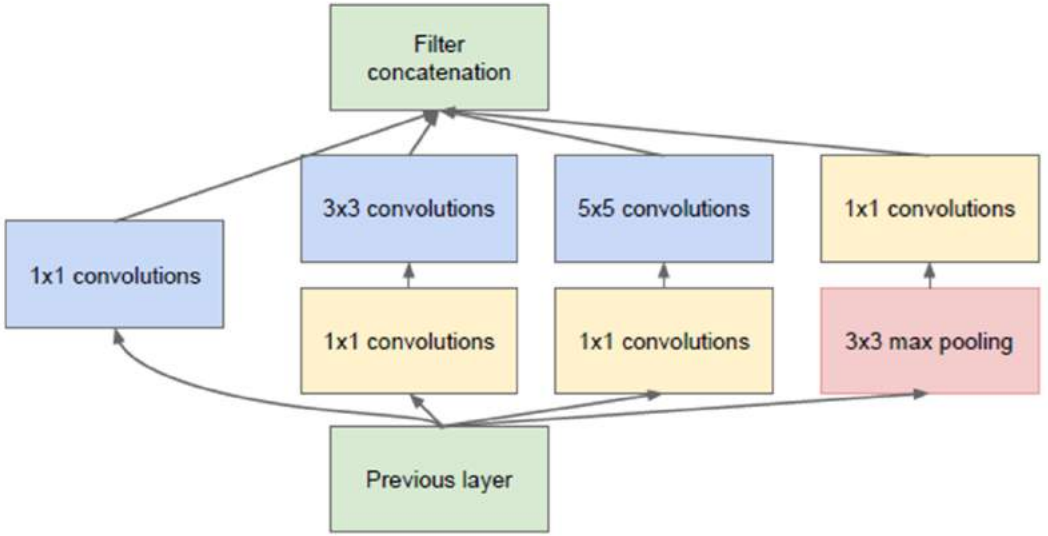


Inception 模块

- 使用不同大小的卷积核，再将特征图沿通道拼接
- 对于 3x3 和 5x5 的卷积核，进一步使用 1x1 的卷积核压缩通道，降低计算量。



(a) Inception module, naïve version

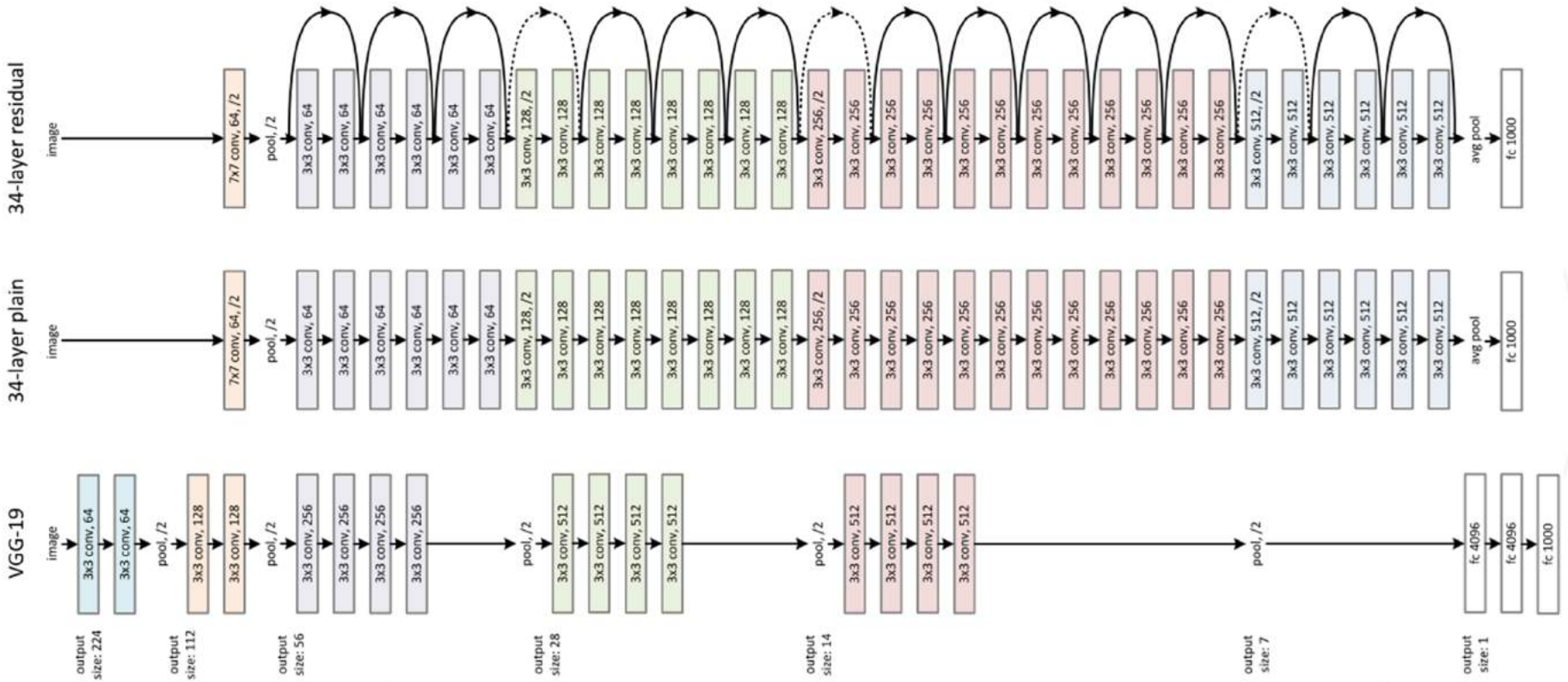


(b) Inception module with dimensionality reduction

Going Deeper With Convolutions

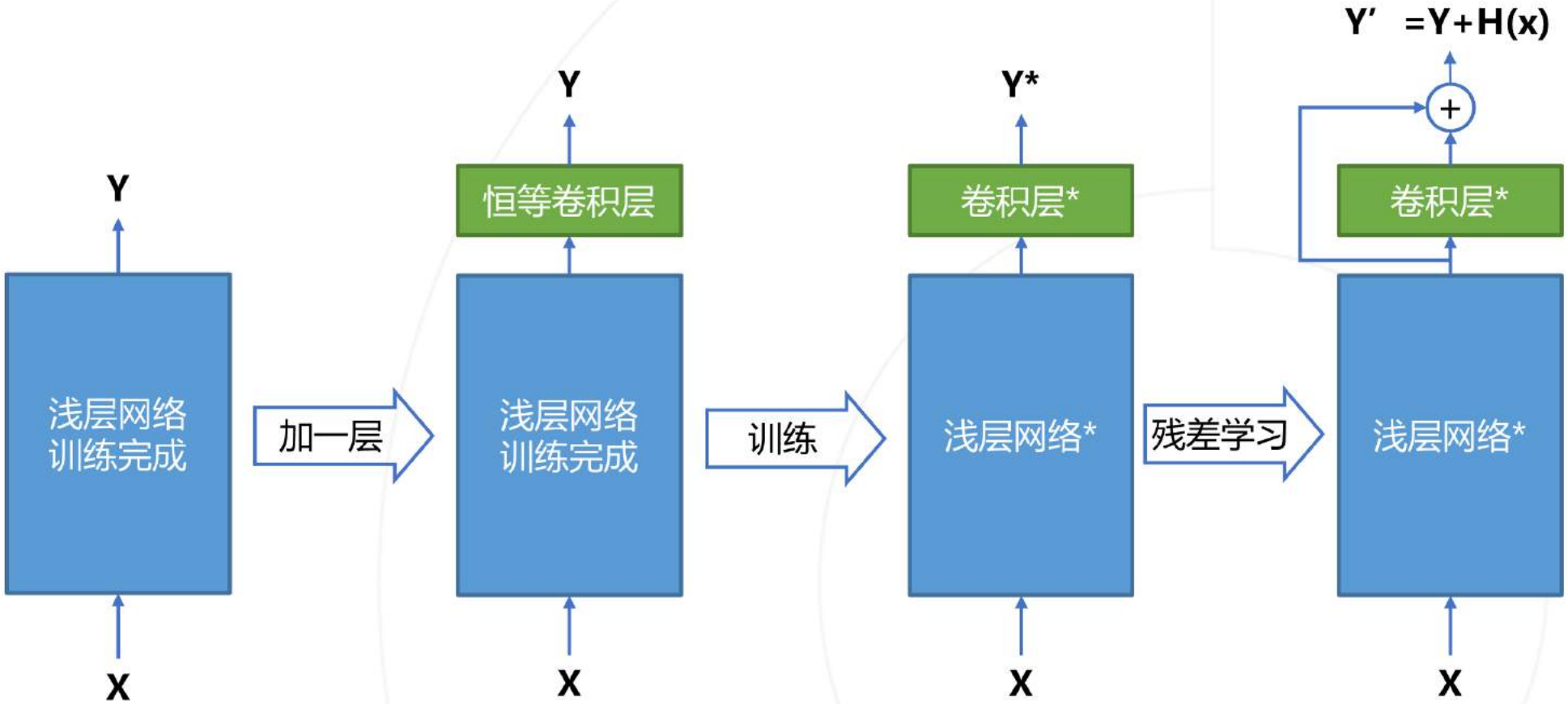
ResNet (2015)

- 迄今为止影响力最大、使用最广泛的模型结构之一，获得 CVPR 2016 最佳论文奖。
- **残差学习**：引入了跨层连接，将网络深度提高到了上百层。



He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

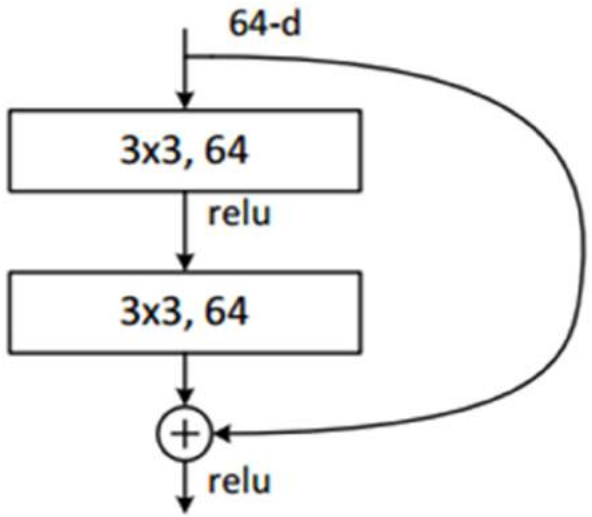
残差学习源起



深层网络应该具有不低于浅层网络的精度

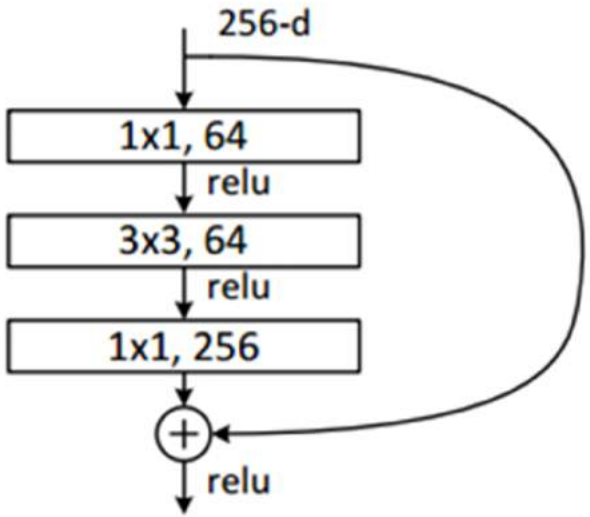
实验：精度下降 -> 退化问题

残差模块



Basic block

用于 ResNet-18 和 34



Bottleneck block

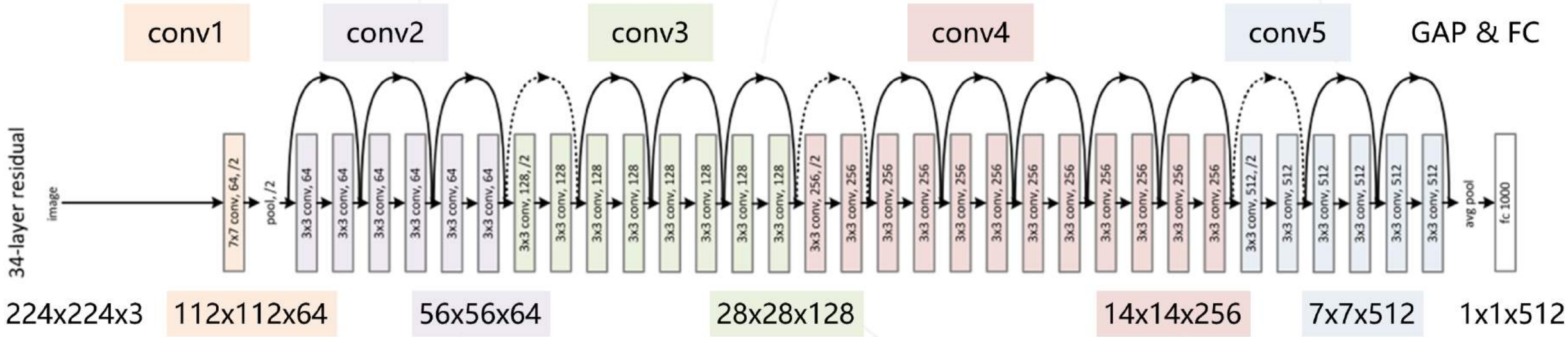
用于 ResNet-50、101 和 152

1x1 卷积用于压缩通道，降低计算开销

He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

ResNet 模型结构

- 5 级，每级包含若干残差模块。不同残差模块个数 → 不同 ResNet 结构。
- 每级输出分辨率减半，通道倍增
- 全局平均池化压缩空间维度
- 单层全连接层产生类别概率

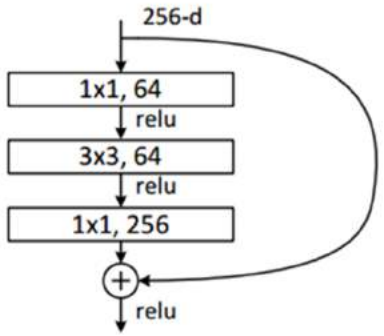


He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

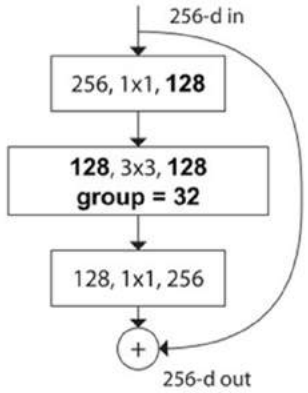
ResNeXt (2017)

- ResNeXt 将 ResNet 的 Bottleneck block 中 3x3 的卷积改为分组卷积，降低模型计算量。

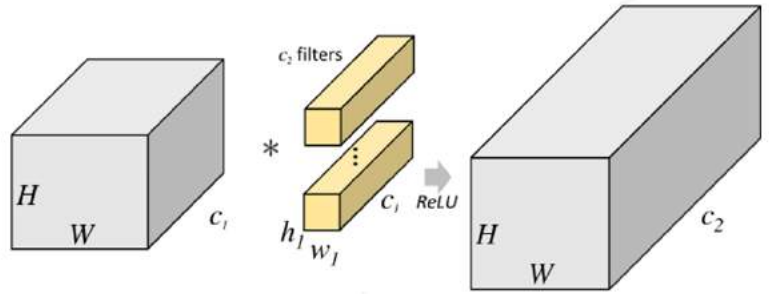
ResNet Bottleneck block



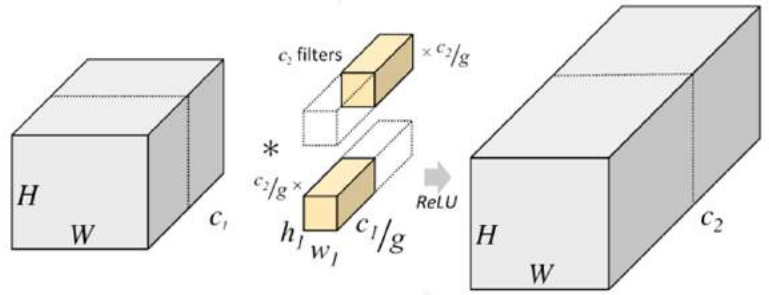
ResNeXt Bottleneck block

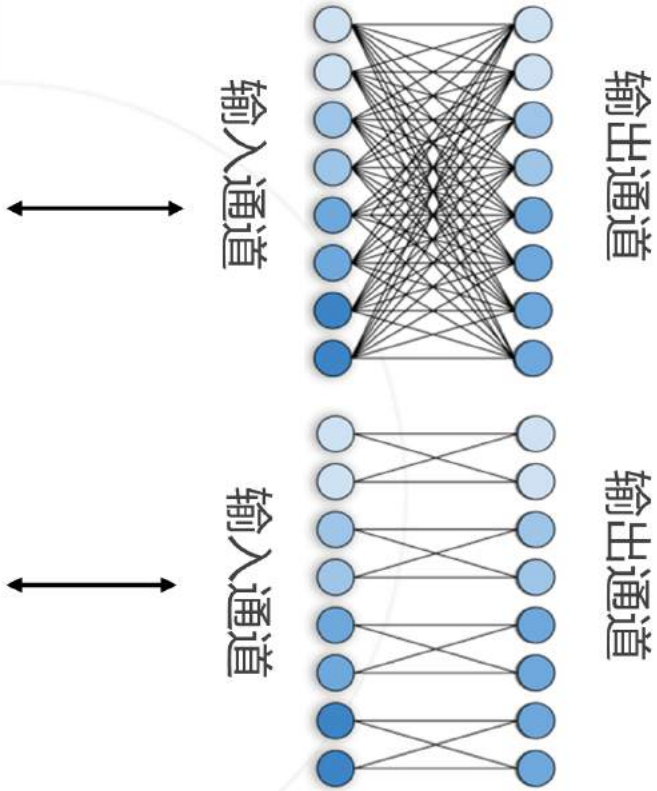


每个卷积核与全部输入通道进行内积运算



通道分组，仅与组内通道参与内积运算，计算量降为 1/g

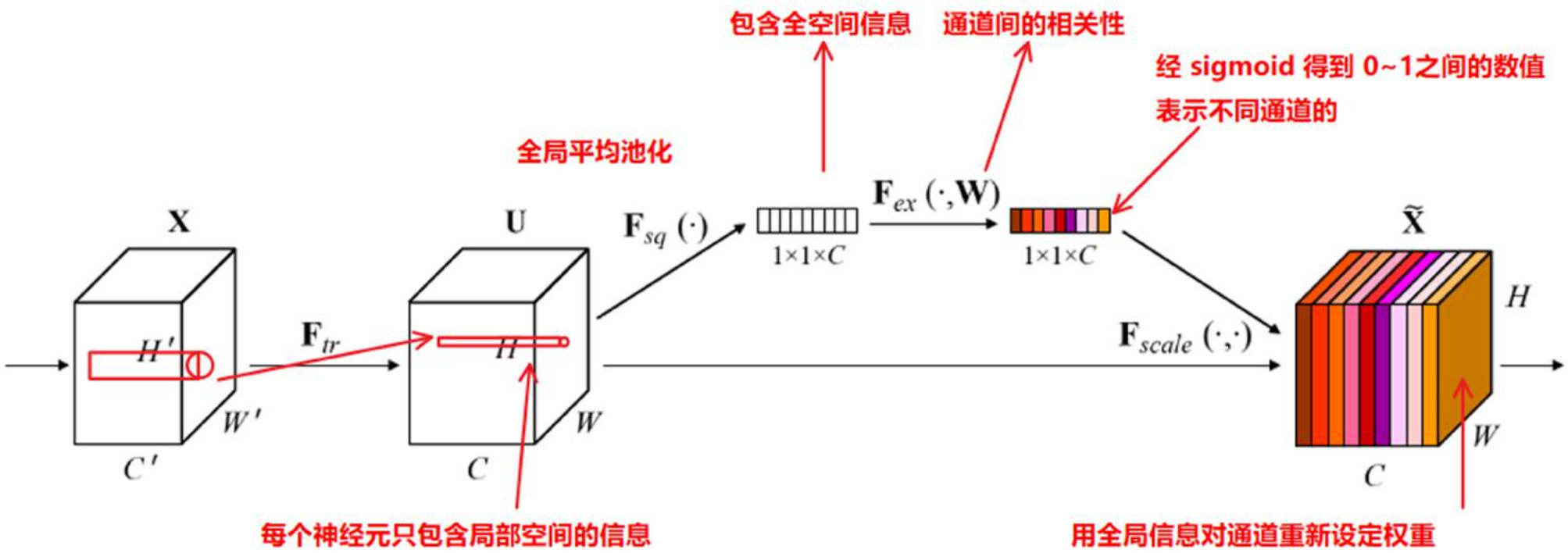




https://blog.yani.ai/filter-group-tutorial/

SENet (2017)

- 引入注意力机制，显式建模不同通道的重要性。
- 即插即用模块，可以插入到任意模型中：SE + ResNet = SENet

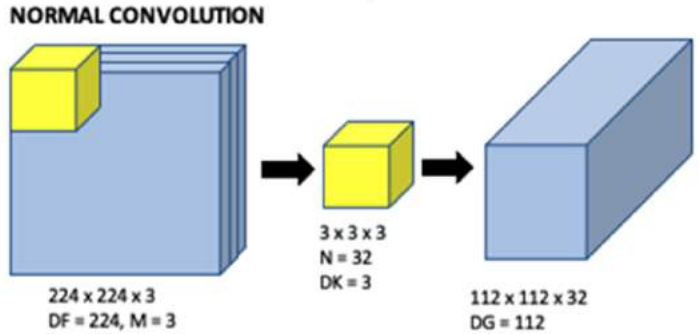


Hu J, Shen L, Sun G. Squeeze-and-excitation networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 7132-7141.

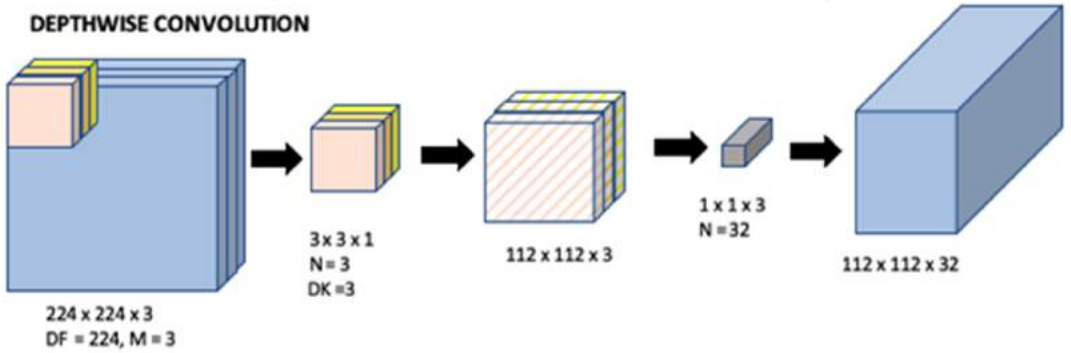
MobileNet (2017)

- 将常规卷积分解为逐层卷积和逐点卷积，以降低参数量和计算量
- 全尺寸的 MobileNet 只有 4.2M参数

常规卷积



逐层卷积
逐点卷积



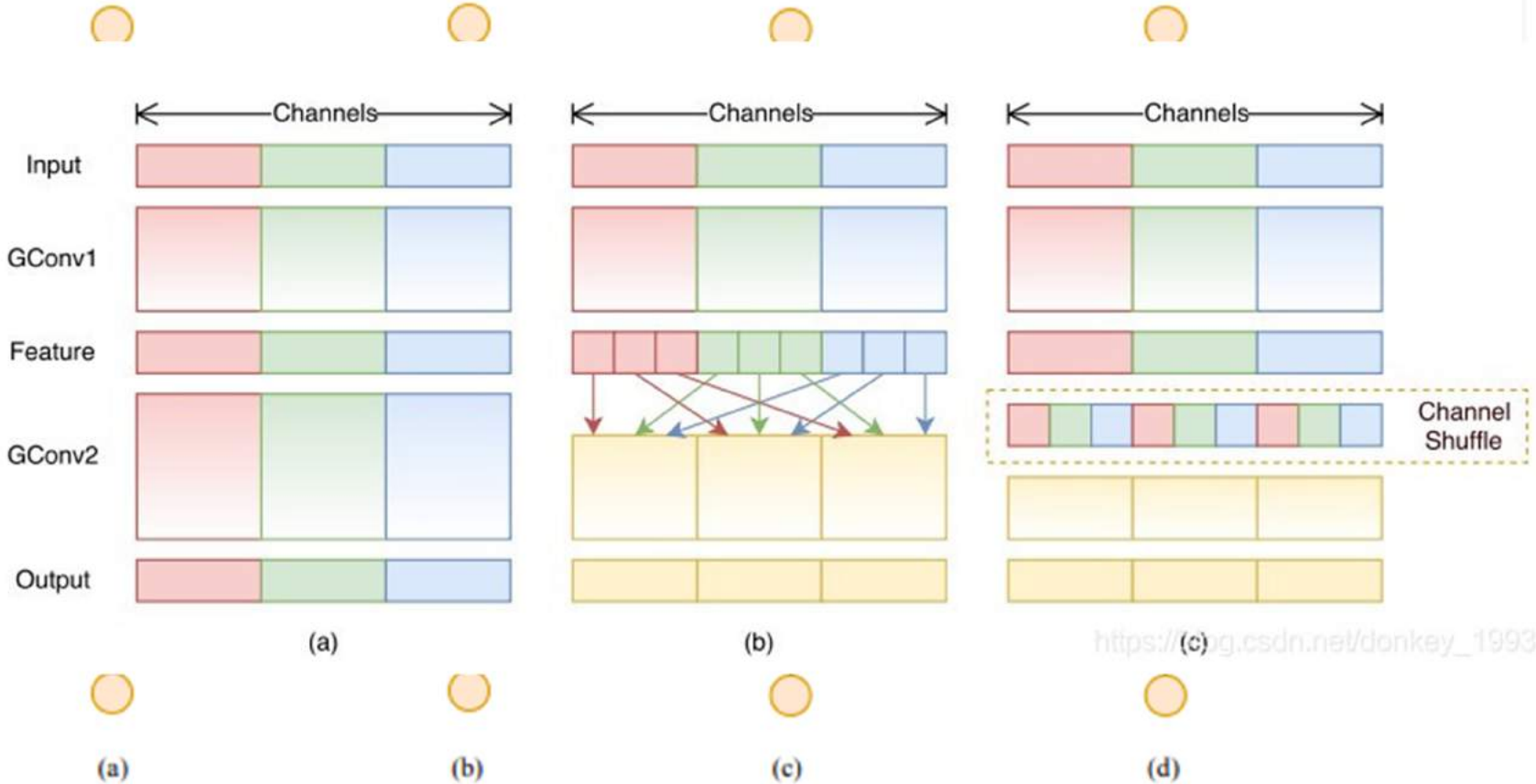
参数量：$D_K^2 NM$
计算量：$D_K^2 NM D_G^2$

$$\frac{\text{参数量：}D_K^2 NM \quad \text{计算量：}D_K^2 NM D_G^2}{\text{参数量：}D_K^2 M + MN \quad \text{计算量：}(D_K^2 M + MN) D_G^2} = \frac{1}{\frac{1}{N} + \frac{1}{D_K^2}}$$

- MobileNet v2/v3 在 v1 的基础上加入了残差模块和 SE 模块

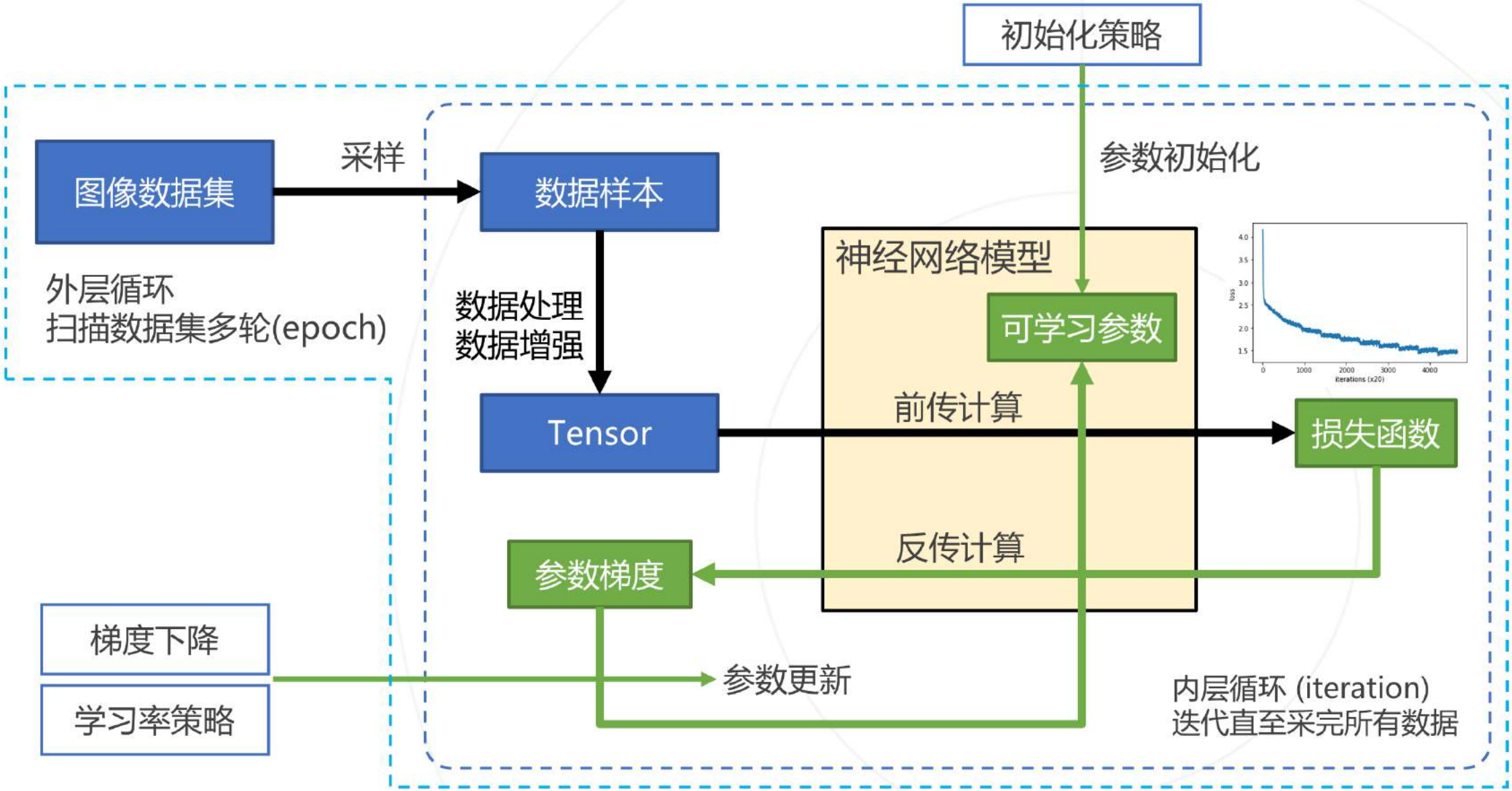
https://subscription.packtpub.com/book/data/9781838827069/15/ch15lvl1sec82/overview-of-mobilenet
MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

ShuffleNet (2017)



图像分类模型训练

整体流程


初始化策略
参数初始化
图像数据集
采样
数据样本
外层循环
扫描数据集多轮(epoch)
数据处理
数据增强
神经网络模型
可学习参数
Tensor
前传计算
损失函数
反传计算
参数梯度
梯度下降
学习率策略
参数更新
内层循环 (iteration)
迭代直至采完所有数据

交叉熵损失函数 Cross-entropy Loss

对于预测类别概率概率 $P \in [0,1]^K$ 和真值 y，定义交叉熵损失为：

$$l(P, y) = -\log P_y$$

P_y 的减函数，$P_y = 1$ 时取最小

此时 $P = [0, 0, \dots, 1, \dots, 0]$，中只有第 y 项为 1，$l(P, y) = 0$

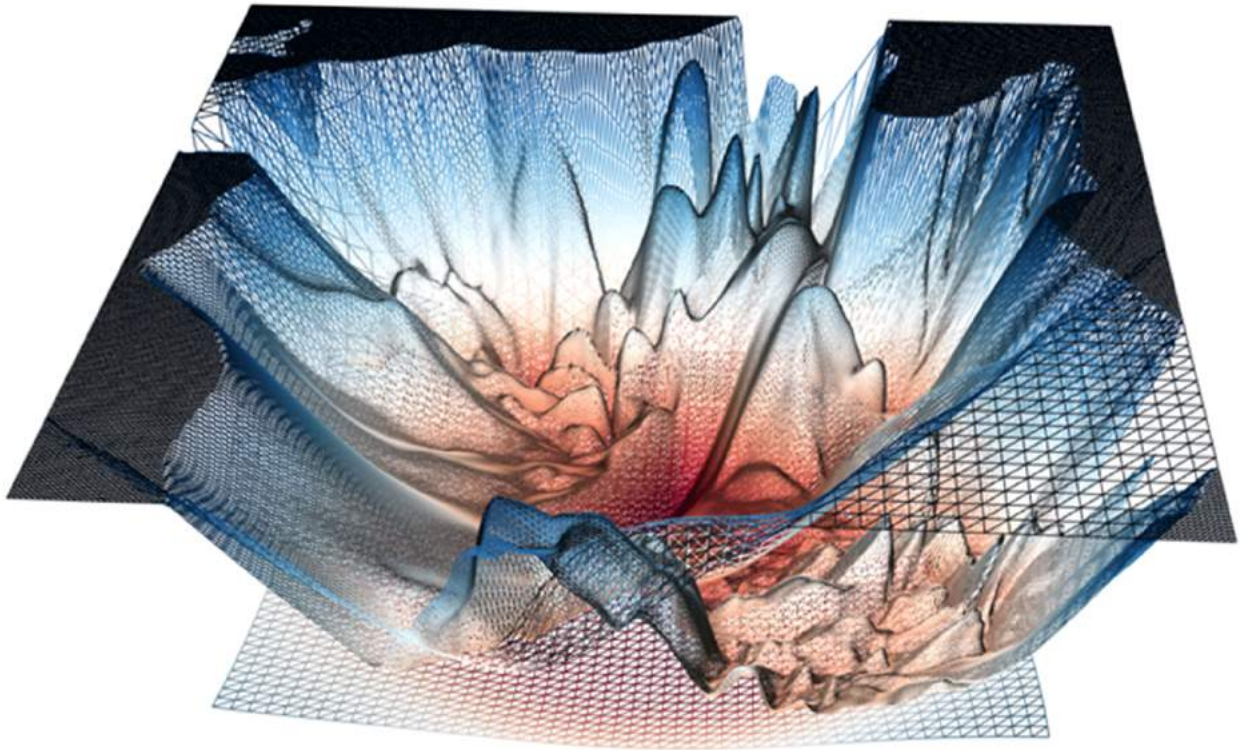
训练深层神经网络的困难

1. 数据量大、计算量大

➢ 例：在 ImageNet 数据集上训练 AlexNet，$N \approx 1\,000\,000$，$\dim(\Theta) \approx 60\,000\,000$

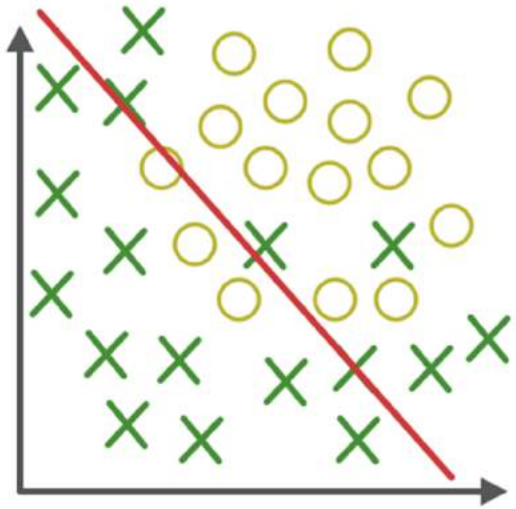
2. 损失函数高度复杂：高维、非凸

3. 模型复杂度高，容易过拟合



图片出处：https://www.telesens.co/2019/01/16/neural-network-loss-visualization/

欠拟合与过拟合

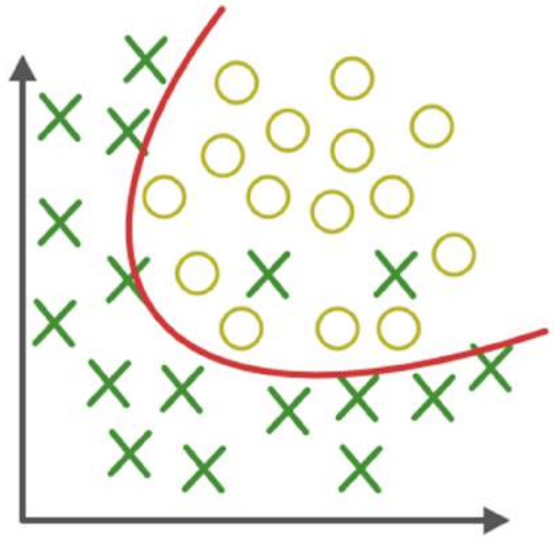


欠拟合

没有捕捉数据中的规律

不能准确预测未来数据

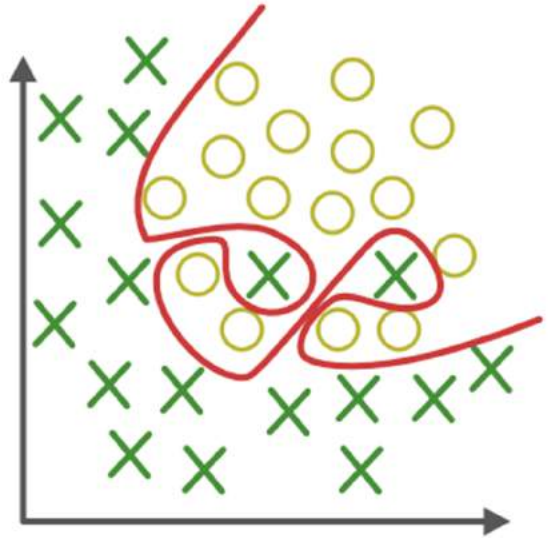
模型过于简单



拟合

捕捉到数据中的规律

可以准确预测未来数据



过拟合

过度拟合到数据中的**噪声**

不能准确预测未来数据

模型过于复杂、且数据不够

图片来源：https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

梯度下降算法

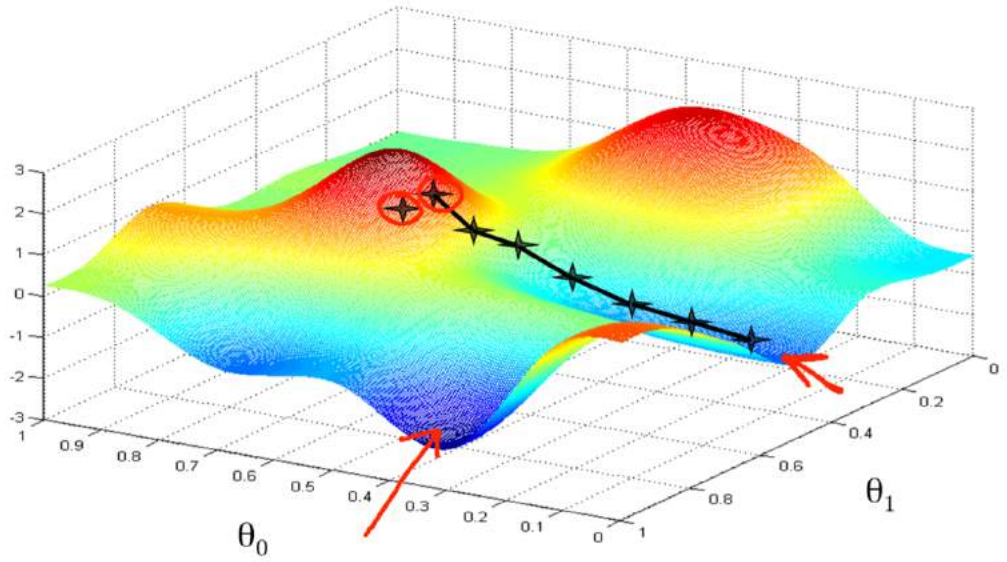
- 给定数据集 $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$，模型 $h(x;\Theta)$ 和损失函数 $l(P, y)$，

很大

$$L(\Theta|\mathcal{D}) \coloneqq \frac{1}{N}\sum_{i=1}^{N} l(h(x_i;\theta), y_i)$$

- 随机初始化参数 $\Theta^{(0)}$
- 迭代直至收敛：
  - 前向+反向传播计算梯度 $\nabla_\Theta L\left(\Theta^{(t-1)}\right)$
  - 参数更新 $\Theta^{(t)} = \Theta^{(t-1)} - \eta\nabla_\Theta L\left(\Theta^{(t-1)}\right)$

学习率

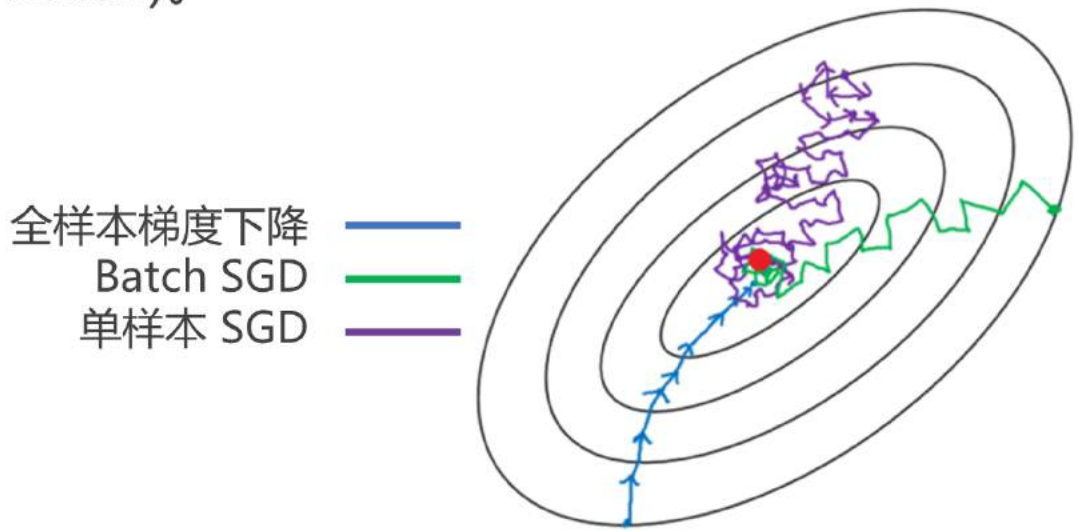


随机梯度下降 Stochastic Gradient Descent

- 每次迭代随机选取指标集 $\mathcal{B}^{(t)} = \{i_1, i_2, \dots, i_B\}$ 近似计算损失函数的梯度

$$L_{\mathcal{B}}(\Theta|\mathcal{D}) = \frac{1}{B}\sum_{i=1}^{B} l\left(h\left(x_{i_j};\theta\right), y_{i_j}\right) \approx \frac{1}{N}\sum_{i=1}^{N} l(h(x_i;\theta), y_i)$$

- 对应的样本子集 $D_B = \left\{x_{i_j}, y_{i_j}\right\}_{j=1}^{B} \subset D$ 称为批 (Mini Batch)。



https://suniljangirblog.wordpress.com/2018/12/13/variants-of-gradient-descent/

动量 Momentum SGD

- 将前次的移动延续到本次

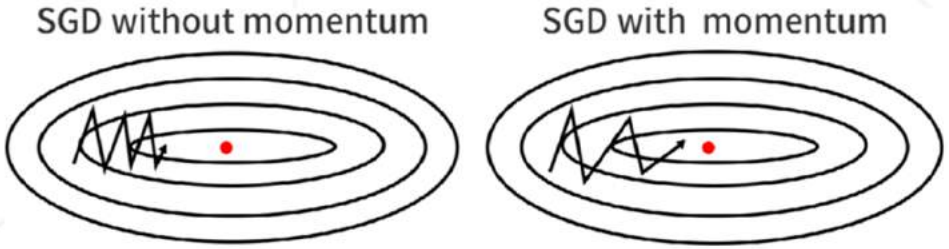
以往轮次的梯度的积累　　当前轮次的梯度

$$\Delta^{(0)} := 0$$

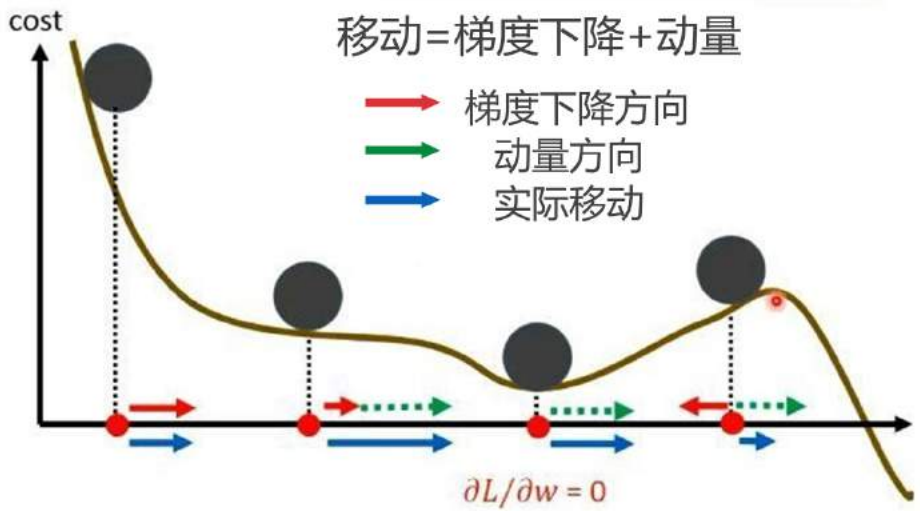
指数滑动平均　　$$\Delta^{(t)} := \alpha\Delta^{(t-1)} + \nabla_w L^{(t)}$$

按照积累值移动　　$$w^{(t)} := w^{(t-1)} - \eta\Delta^{(t)}$$

- 缓解随机梯度下降的波动

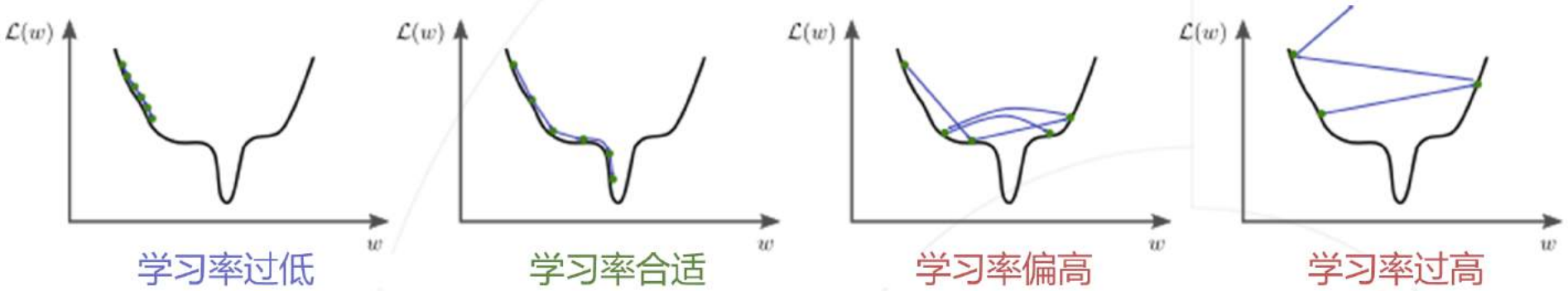


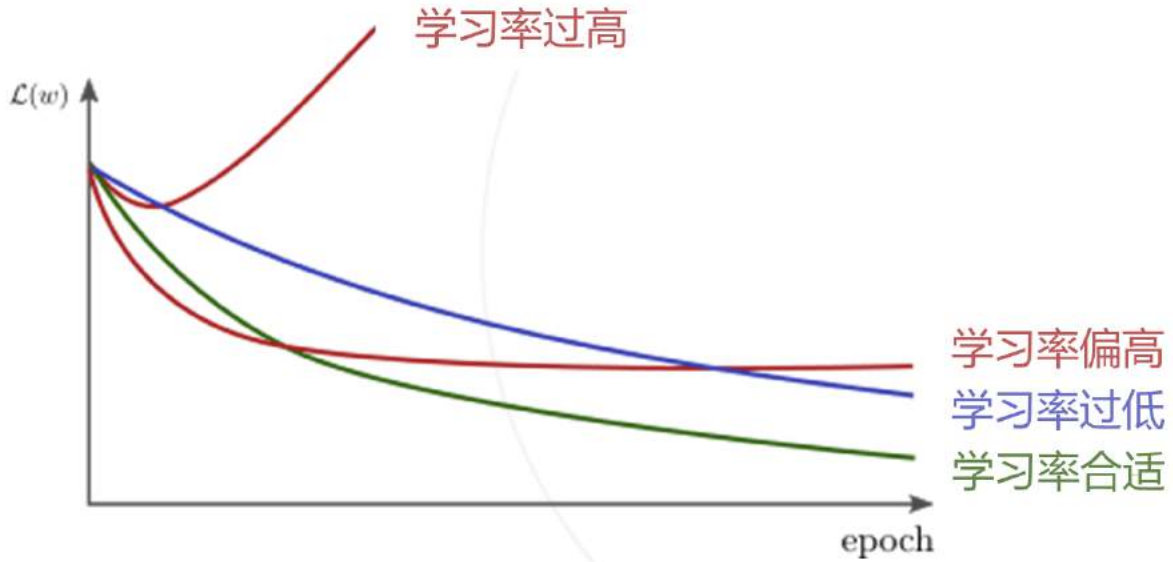
- 帮助逃离局部极小值和鞍点



https://medium.com/ai%C2%B3-theory-practice-business/hyper-parameter-momentum-dc7a7336166e

学习率对训练的影响



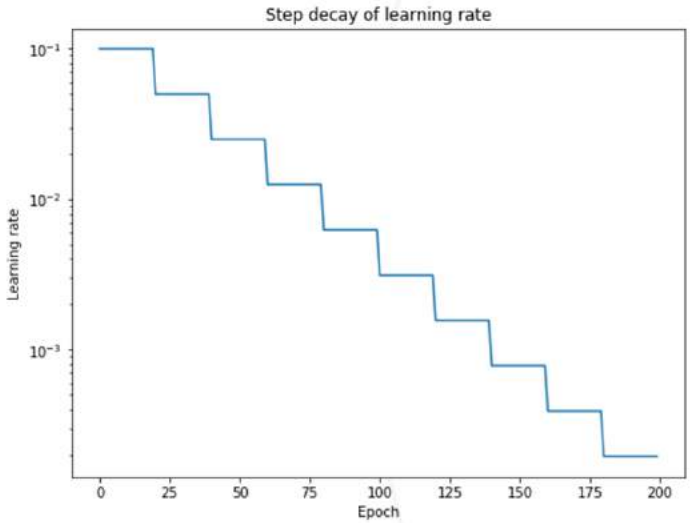


图片出处：http://www.bdhammel.com/learning-rates/

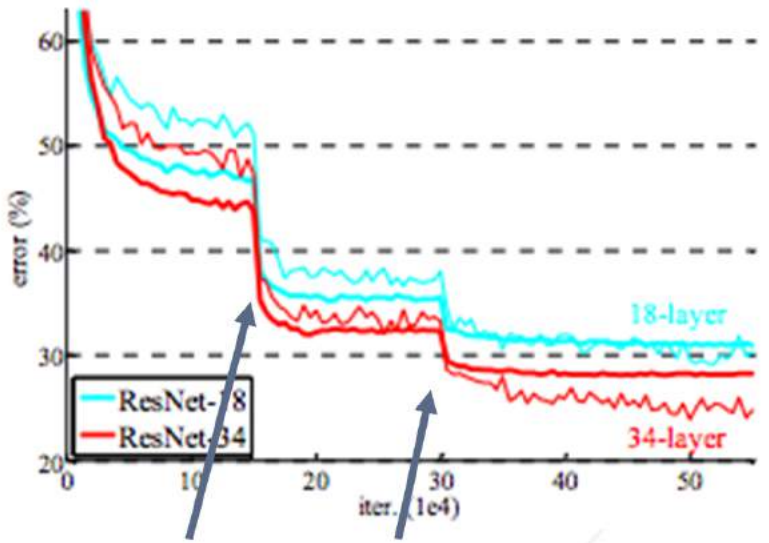
学习率退火

在训练初始阶段使用较大的学习率，损失函数稳定后下降学习率：

- 按步长下降
- 按比例下降 $\eta(t) = \eta_0 e^{-kt}$
- 按倒数下降 $\eta(t) = \frac{\eta_0}{\epsilon + t}$



每 20 轮降低学习率至 1/3



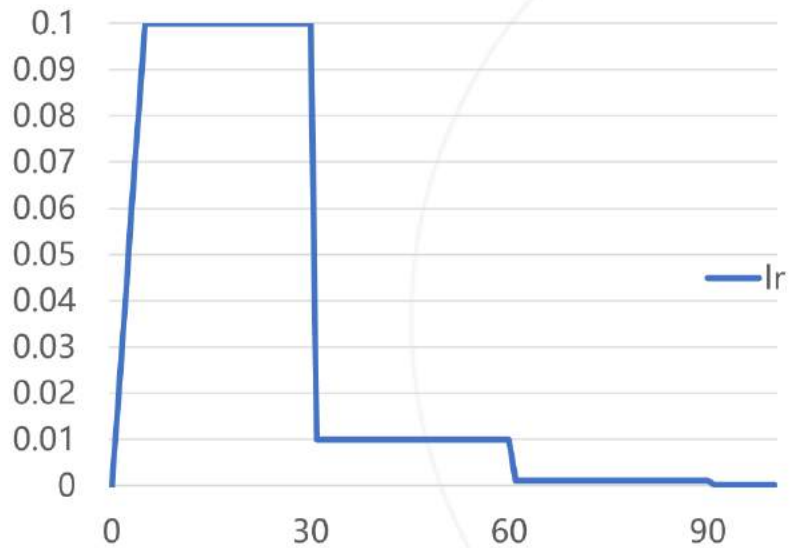
降低学习率后损失函数继续下降

https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

学习率升温 Warmup

在训练前几轮学习率逐渐上升，直到预设的学习率，以稳定训练的初始阶段

- 线性上升 $\eta(t) = \frac{t}{T_0}\eta_0, t \leq T_0$



自适应梯度算法

- 不同梯度需要不同的学习率
- 根据梯度的幅度自动调整学习率

Adagrad

用梯度的历史幅度
对当前梯度的幅度
进行归一化

$$G_i^{(t)} := \sum_{\tau=0}^{t} \left(g_i^{(\tau)} \right)^2$$

$$w_i^{(t)} := w_i^{(t-1)} - \eta \frac{g_i^{(t)}}{\sqrt{G_i^{(t)}}}$$

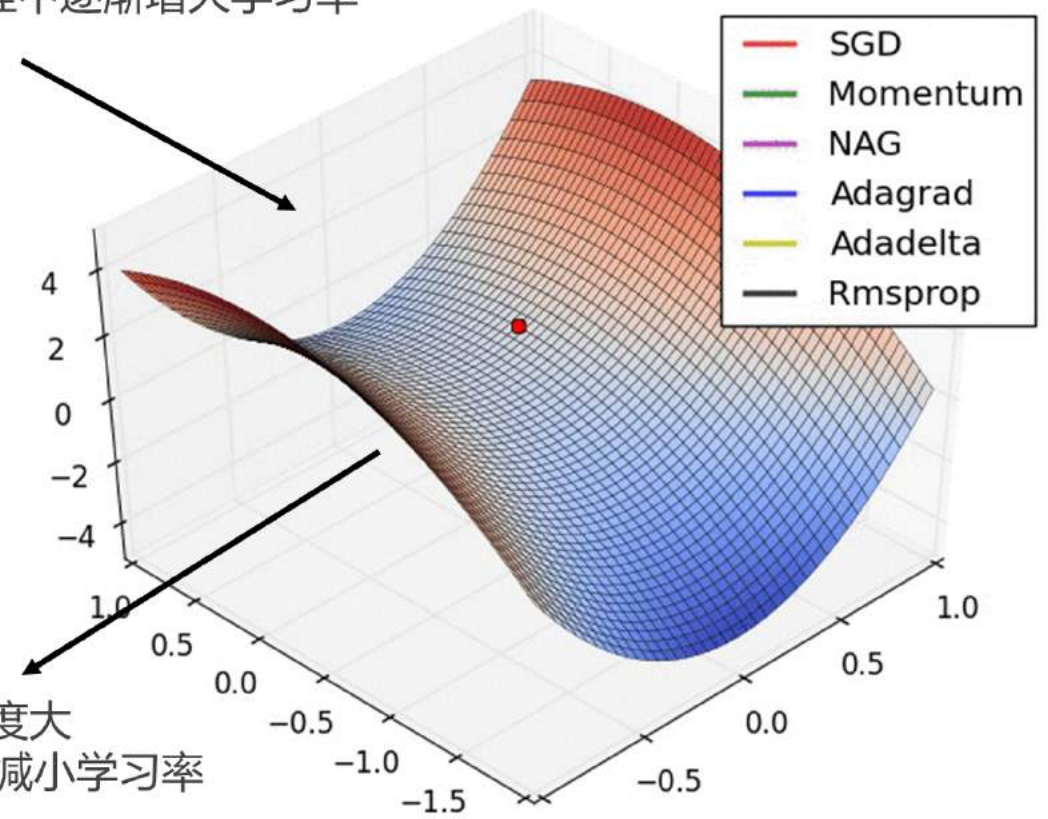
Adadelta

RMSProp

ADAM

.....

X 方向，初始梯度小
训练过程中逐渐增大学习率



Y 方向，初始梯度大
训练过程中逐渐减小学习率

权重初始化

针对卷积层和全连接层，初始化连接权重 W 和偏置 b

- 随机初始化

1. 朴素方法：依照均匀分布或高斯分布

$$W_{ij}, b_i \sim U(-a, a) \quad \text{or} \quad W_{ij}, b_i \sim \mathcal{N}(0, \sigma^2)$$

$a=\sqrt{3}\sigma \Rightarrow$ 方差相同

2. Xavier 方法 (2010)：前传时维持激活值的方差，反传维持梯度的方差

$$a = \sqrt{6/(\text{fan}_{\text{in}} + \text{fan}_{\text{out}})} \quad \text{or} \quad \sigma = \sqrt{2/(\text{fan}_{\text{in}} + \text{fan}_{\text{out}})}$$

3. Kaiming 方法 (2015)：同上，但针对 ReLU 激活函数

$$a = \sqrt{6/n} \quad \text{or} \quad \sigma = \sqrt{2/n} \quad , \text{here } n = \text{fan}_{\text{in}} \text{ or } \text{fan}_{\text{out}}$$

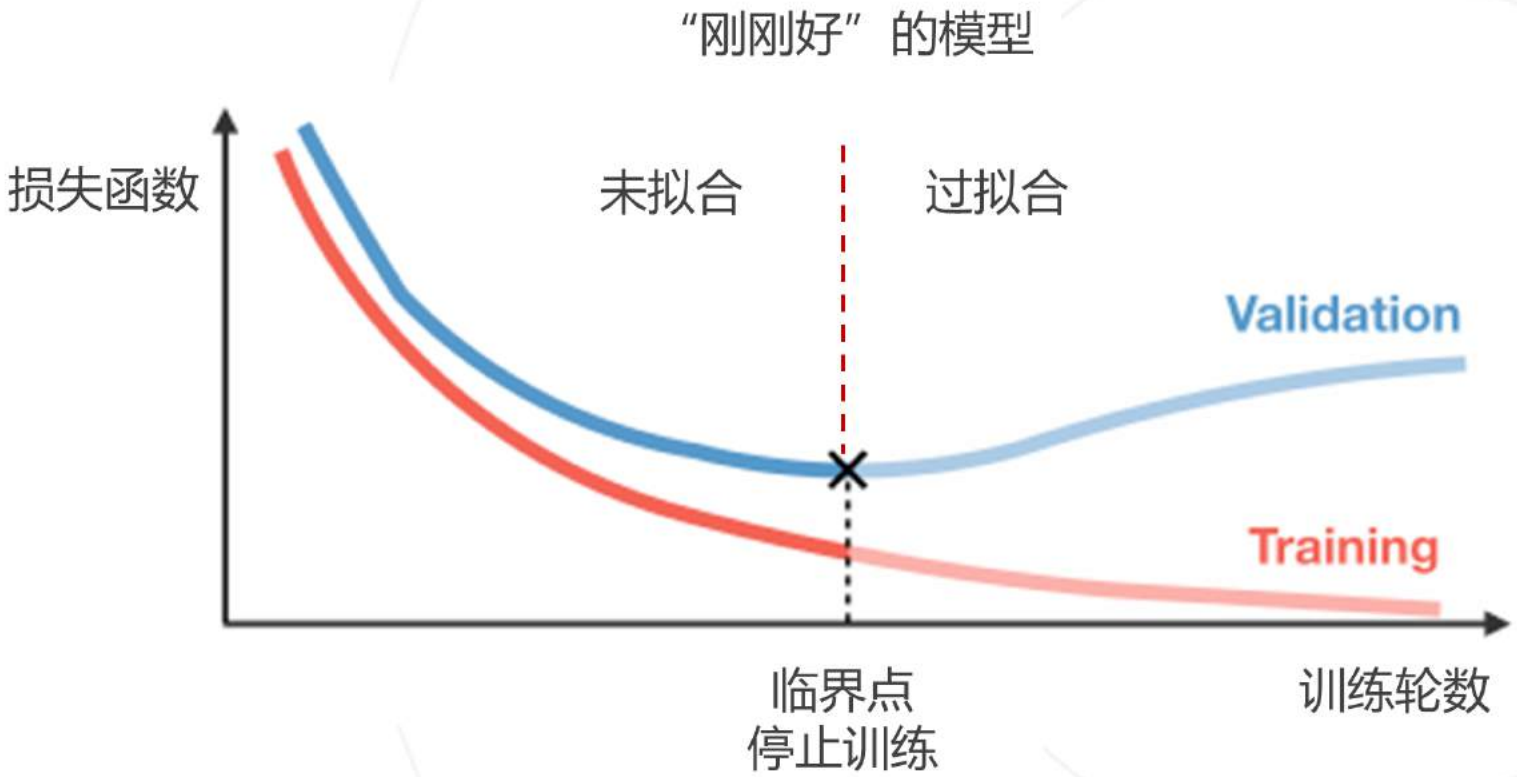
权重初始化

- 用训练好的模型（通常基于ImageNet数据集）进行权重初始化
  - 替换预训练模型的分类头，进行微调训练 (finetune)

早停 Early Stopping

将训练数据集划分为训练集和验证集，在训练集上训练，周期性在验证集上测试。

当验证集的 loss 不降反升时，停止训练，防止过拟合。



正则化与权重衰减 Weight Decay

- 在损失函数中引入正则化项，以鼓励训练出相对简单的模型：

$$R(\Theta) = L(\Theta|\mathcal{D}) + \frac{1}{2}\lambda\|\Theta\|_2^2$$

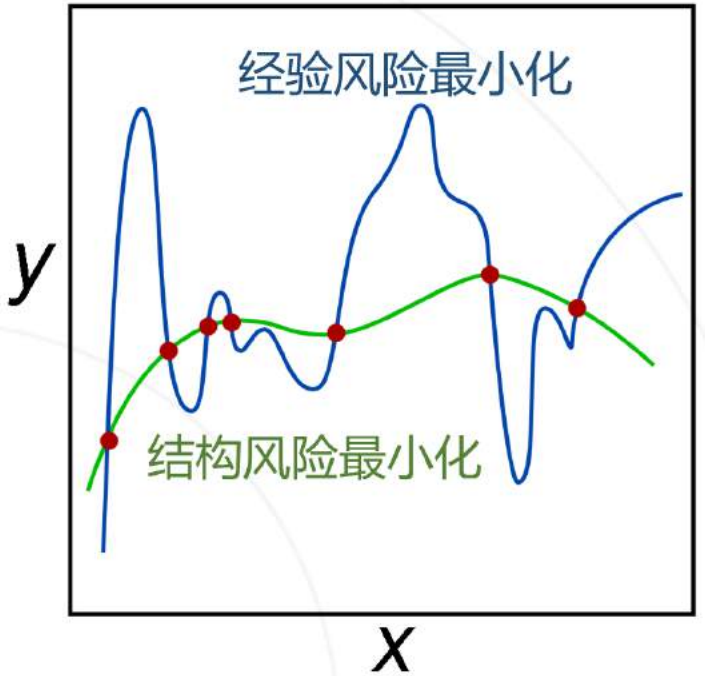
- 结构风险的梯度：

$$\nabla_\Theta R = \nabla_\Theta L + \lambda\Theta$$

- 梯度更新策略：

$$\begin{aligned}\Theta^{(t)} &= \Theta^{(t-1)} - \eta\nabla_\Theta R\left(\Theta^{(t-1)}\right) \\ &= \Theta^{(t-1)} - \eta\nabla_\Theta L\left(\Theta^{(t-1)}\right) - \lambda\eta\Theta^{(t-1)} \\ &= \boxed{(1-\lambda\eta)}\Theta^{(t-1)} - \boxed{\eta\nabla_\Theta L\left(\Theta^{(t-1)}\right)}\end{aligned}$$

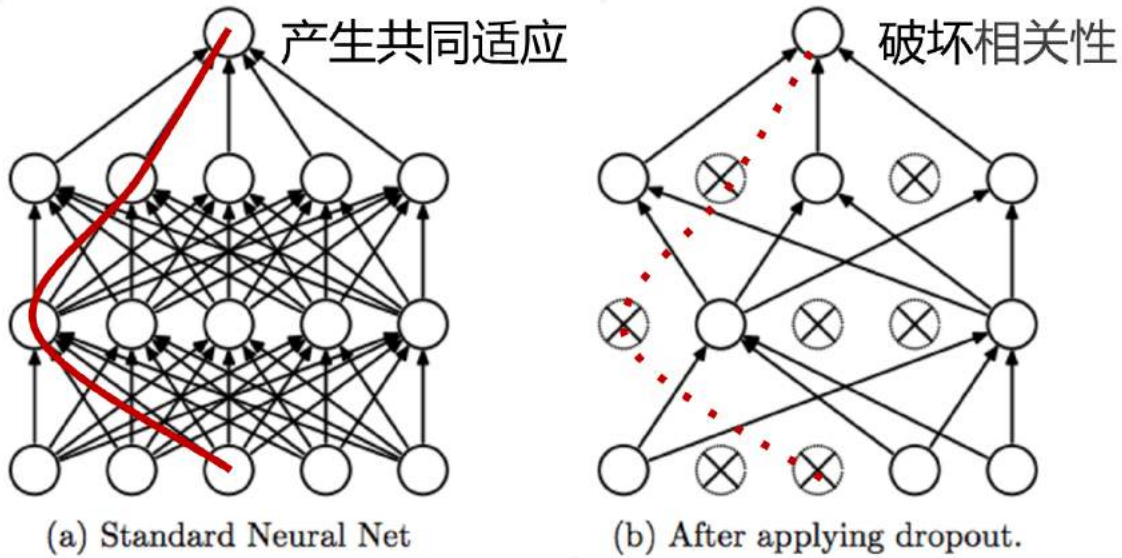
权重衰减　　常规梯度下降



图片：https://en.wikipedia.org/wiki/Regularization_(mathematics)

丢弃层 Dropout

- 神经网络在训练时会出现共适应现象 (co-adaption)，神经元之间产生高度关联，导致过拟合
- 训练时随机丢弃一些连接，破坏神经元之间的关联，鼓励学习独立的特征
- 推理时使用全部连接
- 常用于全连接层



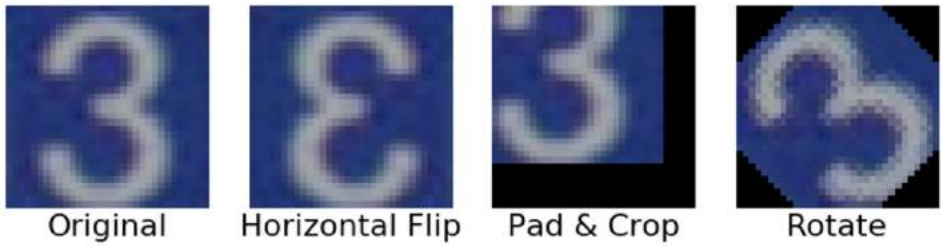
Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. The journal of machine learning research, 2014, 15(1): 1929-1958.

图像增广 Data Augmentation

- 泛化性好的模型 ← 大量多样化的数据
- 数据的采集标注是有成本的

- **数据增广**
- 利用简单的随机变换，从一张图片扩充出多张图片

几何变换



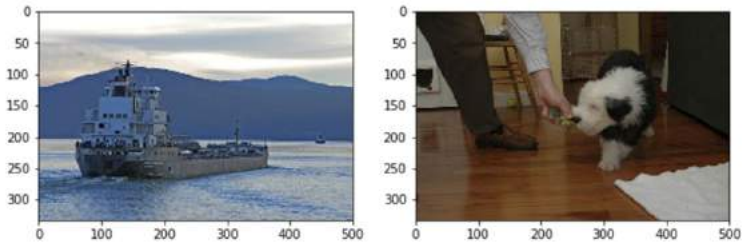
色彩变换

随机遮挡

Albumentations: fast and flexible image augmentationsRandom Erasing Data Augmentation

Mixup Training

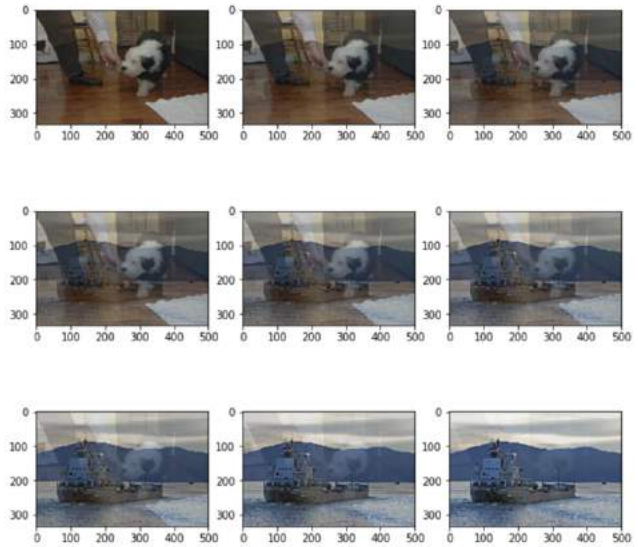
将数据样本的线性组合作为数据进行训练，让网络看到更多的“数据”。



$$\tilde{x} = \lambda x_i + (1-\lambda) x_j$$

$$\tilde{y} = \lambda y_i + (1-\lambda) y_j$$

λ 取值从 0.1 到 0.9.



Zhang H, Cisse M, Dauphin Y N, et al. mixup: Beyond empirical risk minimization[J]. arXiv preprint arXiv:1710.09412, 2017.

批归一化 Batch Normalization (2015)

- 在一个 Batch 中，对神经元响应进行归一化，使每层的输出有相对稳定的数值分布。从而降低网络的学习难度，提高训练的稳定性，并允许使用更大的学习率。

m 为 batch size

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \ , \ \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2$$

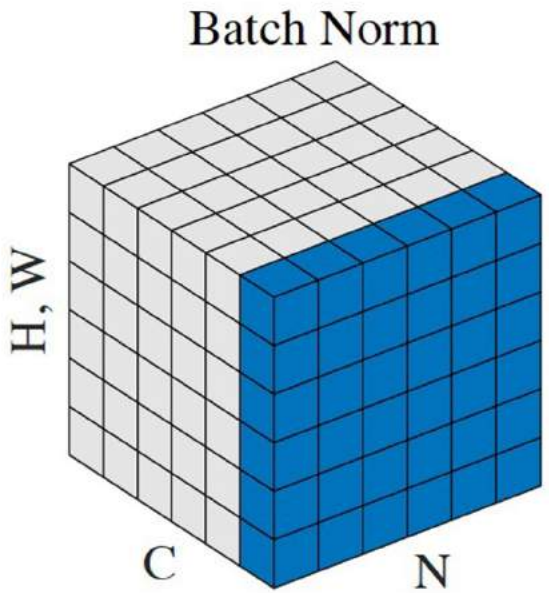
$$\hat{x}_i = \frac{x_i - \mu}{\sigma + \epsilon} \ , \ y_i = \gamma \hat{x}_i + \beta$$

- 在归一化之后进行仿射变换扩大数值范围，方向和幅度的分解。
- 推理时没有 batch → 使用统计量的平均值。

Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C] International conference on machine learning. PMLR, 2015: 448-456.

批归一化 Batch Normalization (2015)

- BN 一般用于卷积层：将同一通道内、不同空间位置、来自不同样本的所有响应值归为一组进行归一化
- 应用 BN 的卷积层一般不需要bias



$$\mu_{\mathcal{B}}[c] = \frac{1}{m \times H \times W} \sum_{i=1}^{m} \sum_{h=1}^{H} \sum_{w=1}^{W} \mathbf{Z}_i[h, w, c]$$

$$\sigma_{\mathcal{B}}^2[c] = \frac{1}{m \times H \times W} \sum_{i=1}^{m} \sum_{h=1}^{H} \sum_{w=1}^{W} (\mathbf{Z}_i[h, w, c] - \mu_{\mathcal{B}}[c])^2$$

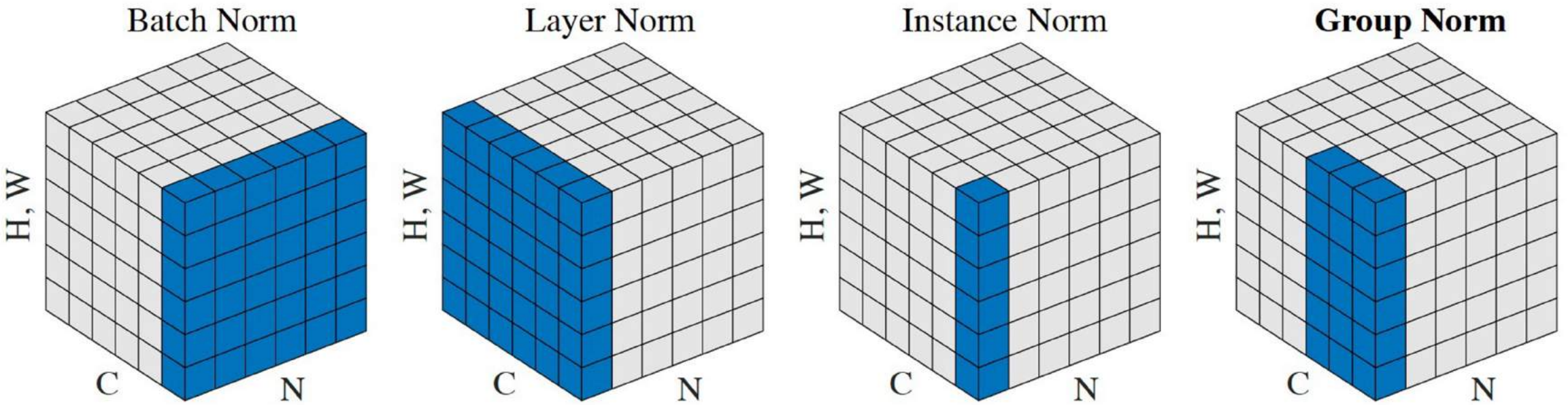
$$\hat{\mathbf{Z}}_i[h, w, c] = \frac{\mathbf{Z}_i[h, w, c] - \mu_{\mathcal{B}}[c]}{\sigma_{\mathcal{B}}[c]}$$

$$\mathbf{Y}_i[h, w, c] = \gamma[c] \cdot \hat{\mathbf{Z}}_i[h, w, c] + \beta[c]$$

z	前层输出
i	样本序号
h,w	空间坐标
c	通道序号

Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C] International conference on machine learning. PMLR, 2015: 448-456.

不同的归一化层



Wu, Yuxin, and Kaiming He. "Group normalization." Proceedings of the European conference on computer vision (ECCV). 2018.

总结

- 随机梯度下降为主，各种经验策略辅助
- 损失函数高度不规则，非凸
  - 权重初始化：kaiming init，预训练模型
  - 优化器改进：动量 SGD、自适应梯度算法
  - 学习率策略：学习率退火、学习率升温
- 防止过拟合
  - 数据增广、早停、Dropout
- Batch Normalization：稳定特征数据分布，降低训练难度

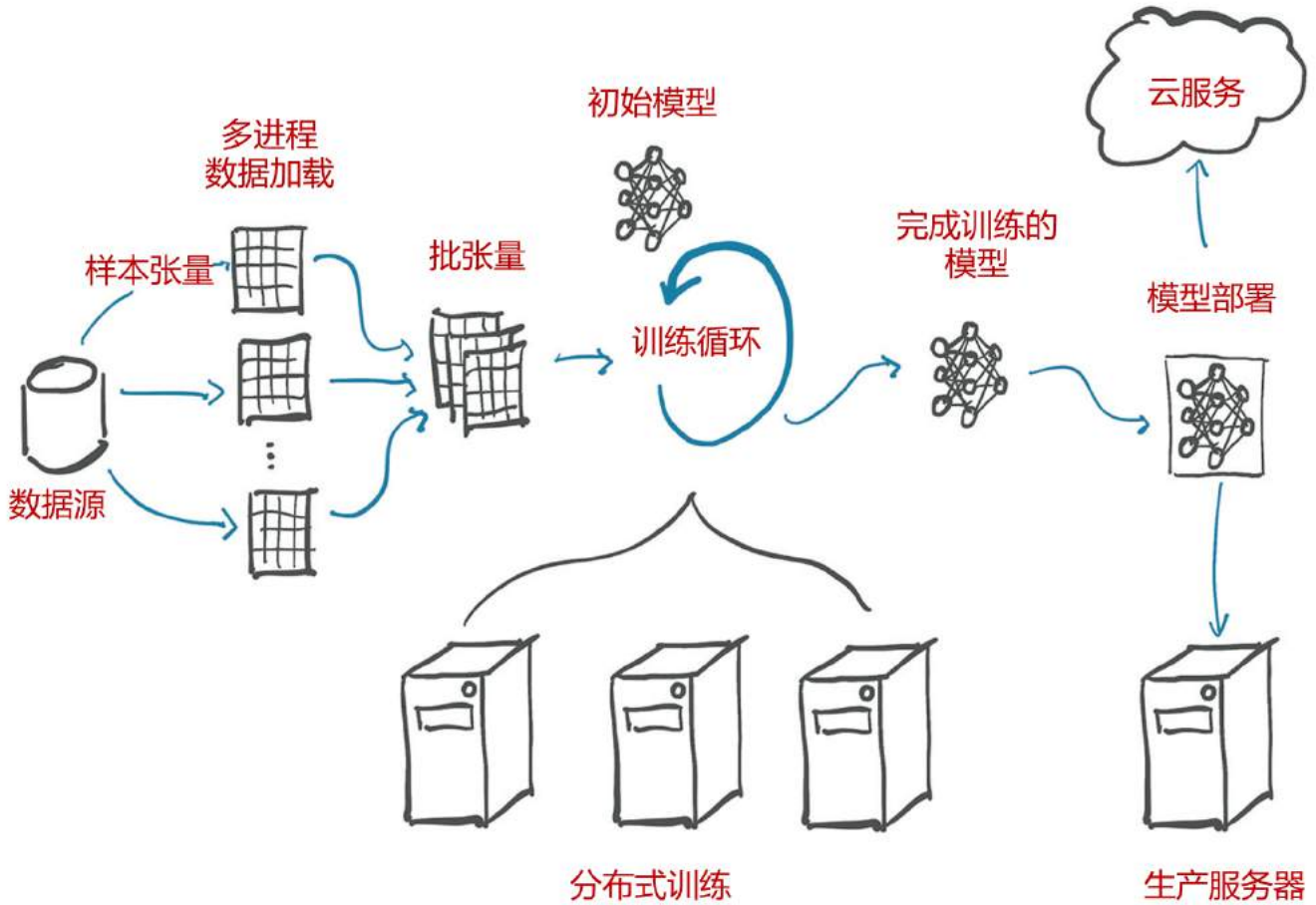
PyTorch 简介

PYTORCH

PyTorch 是由 Facebook 开发的开源深度学习框架。

PyTorch 提供了完整的工具链用于构建、训练和部署深度学习模型。

https://pytorch.org/

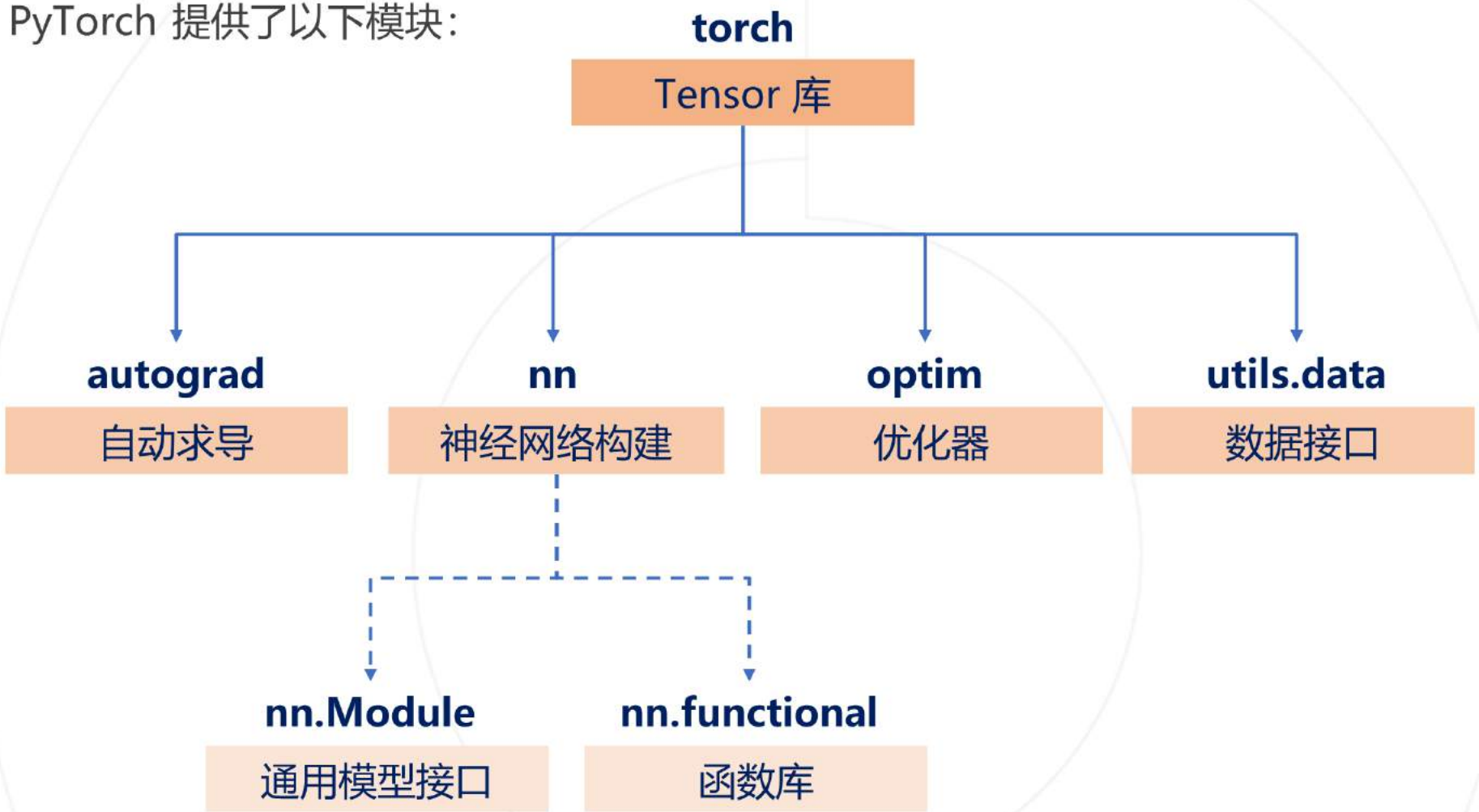


使用 PyTorch 进行模型生产的流程

图片来源 Eli Stevens Luca Antiga Thomas Viehmann《Deep Learning with PyTorch》

PyTorch 的基本模块

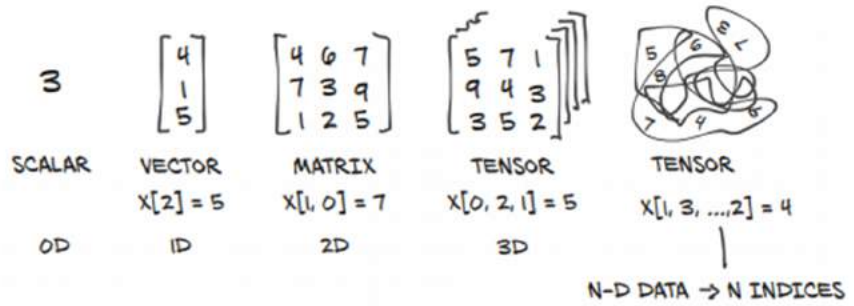
针对深度学习模型构建与训练，PyTorch 提供了以下模块：

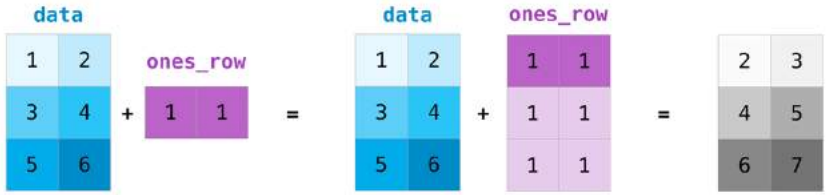


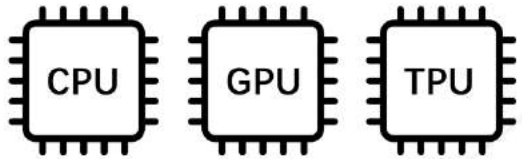
数值计算库 torch

torch 库

- 多维数组的数据结构 Tensor
- 多维数组的运算
- 多计算后端支持







图片来源 Eli Stevens Luca Antiga Thomas Viehmann《Deep Learning with PyTorch》

自动求导 torch.autograd

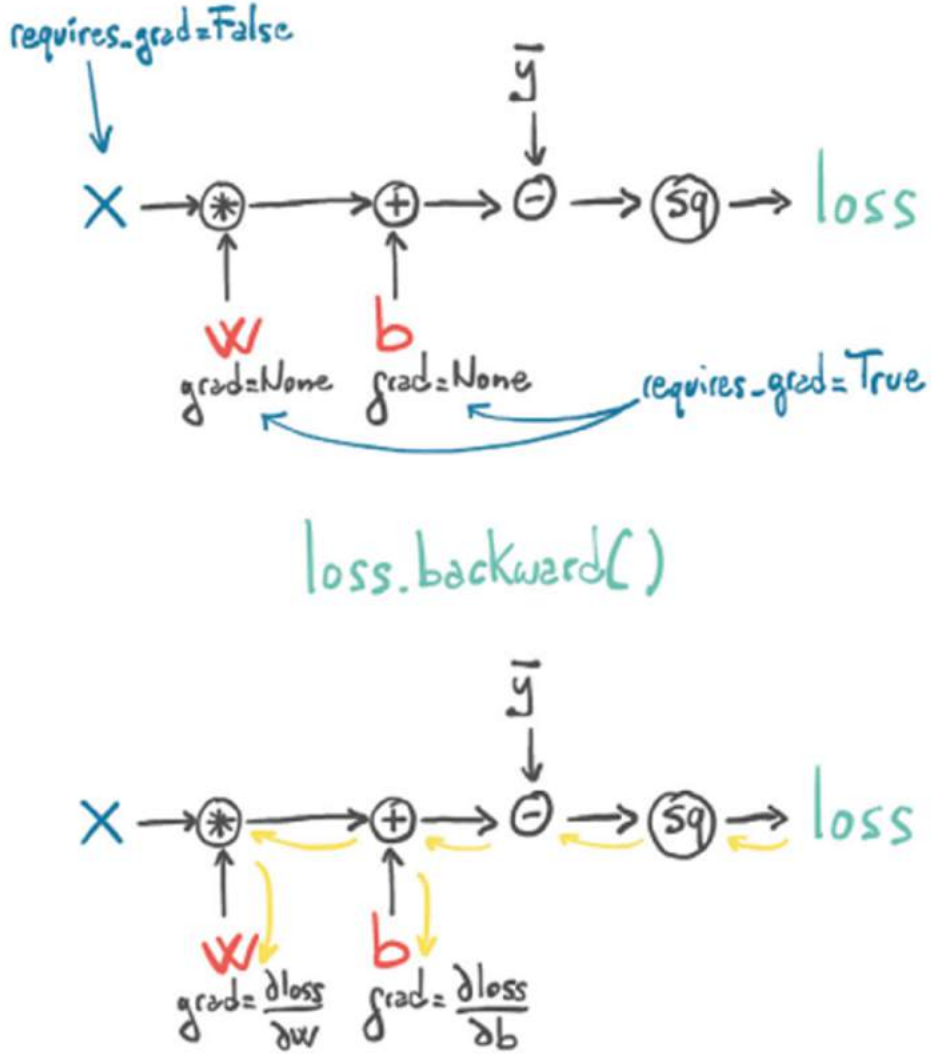
例：梯度下降求解线性回归：

$$L = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - (w^T x_i + b)\right)^2$$

借助 PyTorch 自动求导：

```
loss = 0.5 * (Y - X.T @ w - b) ** 2
loss = loss.sum()
loss.backward()
```

loss 对 w 和 b 的导数存储在 w.grad 和 b.grad 中



图片来源 Eli Stevens Luca Antiga Thomas Viehmann《Deep Learning with PyTorch》

函数库 torch.nn.functional

OpenMMLab

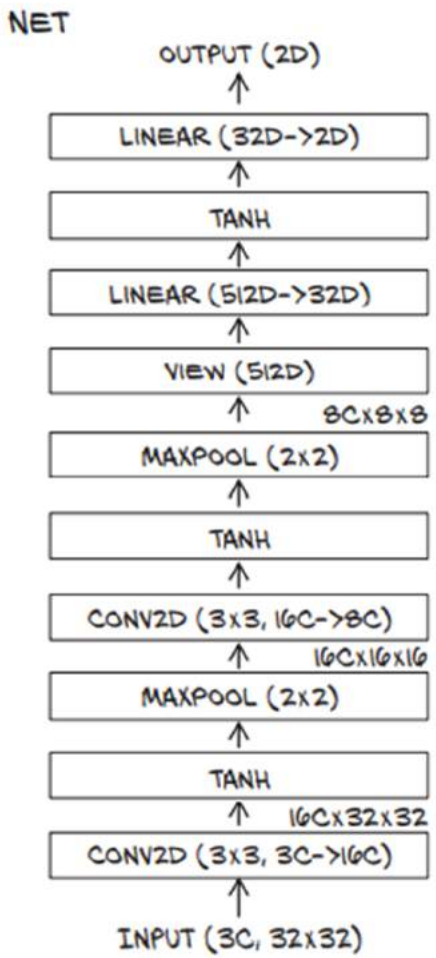
nn.functional 提供了构建神经网络所需的计算函数：

- 线性函数
- 卷积
- 池化
- 非线性激活
- 归一化
- ……

```python
import torch
import torch.nn.functional as F

# ReLU non-linear activation
x = torch.randn(3, 3)
y = F.relu(x)

# Max Pooling
x = torch.randn(1, 1, 4, 4)
y = F.max_pool2d(x, kernel_size=2, stride=2)

# Convolution
img = torch.randn(1, 3, 8, 12)
weight = torch.randn(6, 3, 3, 3)
bias = torch.randn(6)
out = F.conv2d(img, weight, bias, padding=1)
```

通用模型封装 torch.nn.Module

torch.nn.Module 定义了神经网络模型的抽象接口




```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.tanh1 = nn.Tanh()
        self.pool1 = nn.MaxPool2d(kernel_size=(2,2))
        self.conv2 = nn.Conv2d(16, 8, kernel_size=3, padding=1)
        self.tanh2 = nn.Tanh()
        self.pool2 = nn.MaxPool2d(kernel_size=(2,2))
        self.fc1 = nn.Linear(8*8*8, 32)
        self.tanh3 = nn.Tanh()
        self.fc2 = nn.Linear(32, 2)

    def forward(self, x):
        out = self.pool1(self.tanh1(self.conv1(x)))
        out = self.pool2(self.tanh2(self.conv2(out)))
        out = out.view(-1, 8*8*8)
        out = self.tanh3(self.fc1(out))
        out = self.fc2(out)
        return out
```

可调用对象 Callable object

```python
net = Net()
pred = net(img_batch)
```

图片来源 Eli Stevens Luca Antiga Thomas Viehmann《Deep Learning with PyTorch》

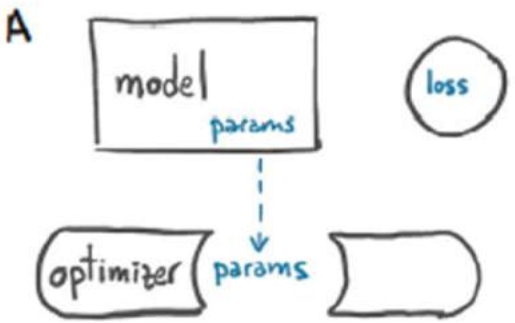
优化器 torch.optim

torch.optim 支持常用的优化算法

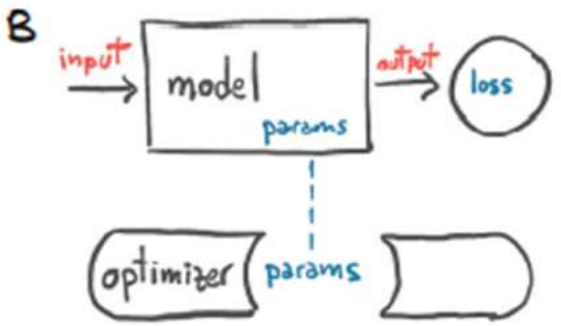
- SGD
- Adam
- RMSprop
- ……

以及常用的学习率策略

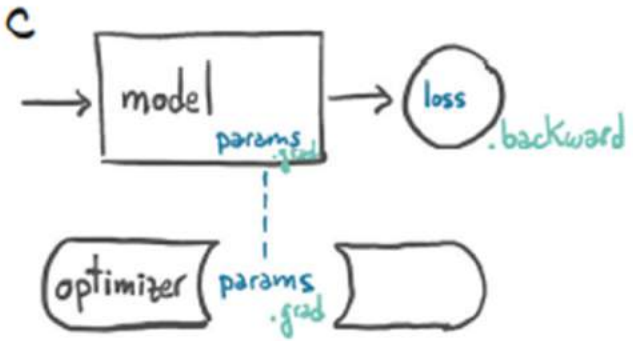
- 步长衰减
- 指数衰减
- 学习率循环
- ……



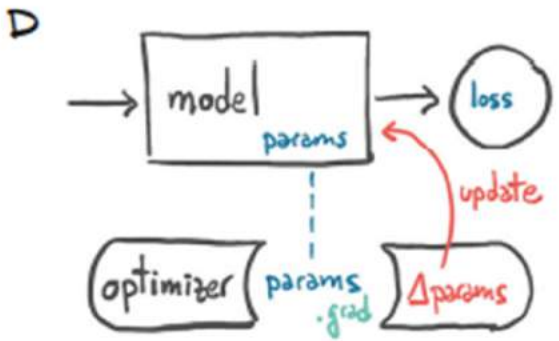

```
param = net.parameters()
opt = SGD(param, lr = 0.1)
```




```
pred = net(img_batch)
loss = loss_fn(pred, label)
```




```
loss.backward()
```



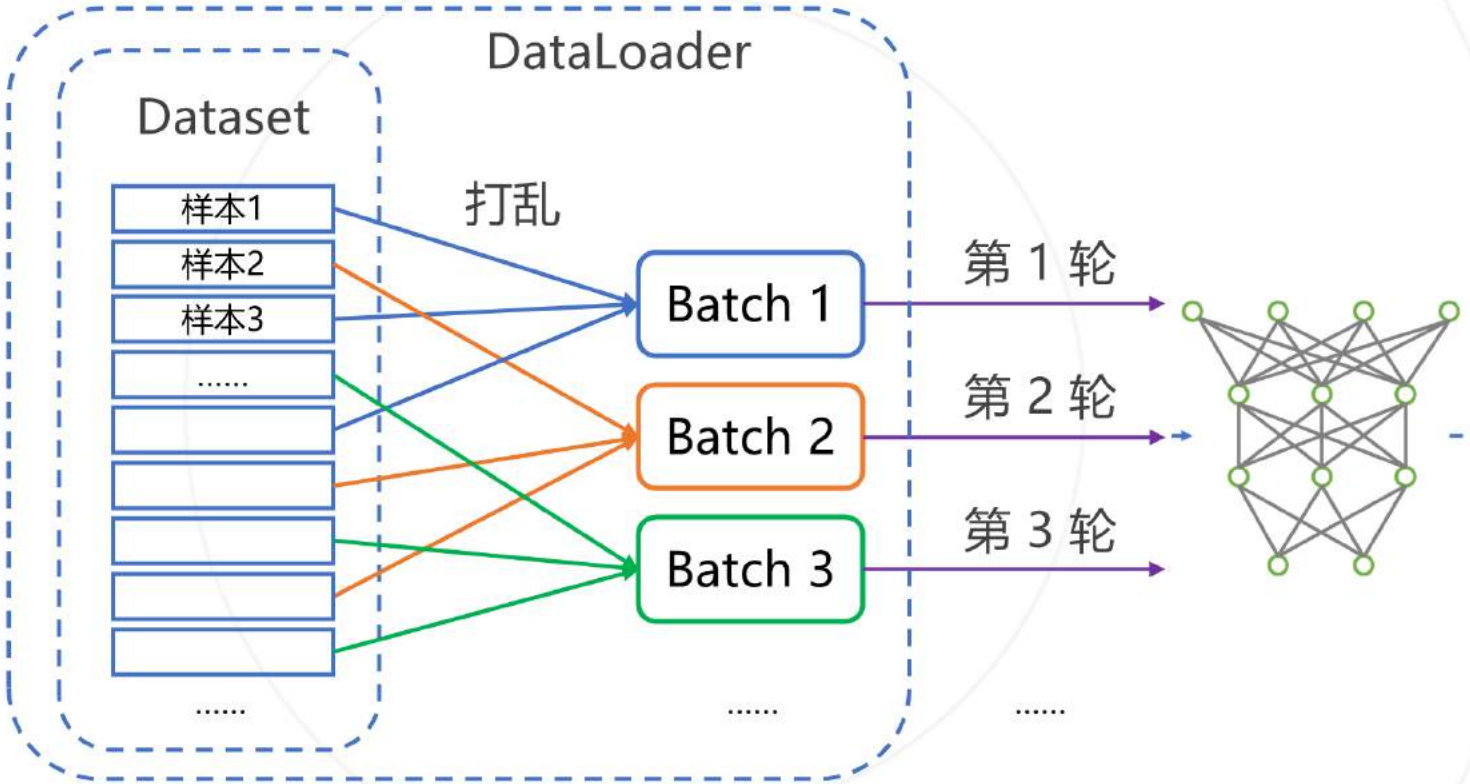

```
opt.step()
```

图片来源 Eli Stevens Luca Antiga Thomas Viehmann《Deep Learning with PyTorch》

数据工具 torch.utils.data

torch.utils.data 统一方便的数据集模型 Dataset
以及支持多线程预读的数据加载器 DataLoader 。

```
for i, data_batch in enumerate(dataloader):
    # process data_batch
    data, label = data_batch
    out = model(data)
    loss = loss_fn(out, label)
    loss.backward()
    opt.step()
```



MMClassification

OpenMMLab

MMClassification

丰富的模型

VGG VGG-BN	ResNet ResNet V1D	ResNeXt ShuffleNet	SE-ResNet ShuffleNet V2 MobileNet V2	ResNeSt	ViT
2014	2015	2017	2018	2020	2021

丰富的数据集支持

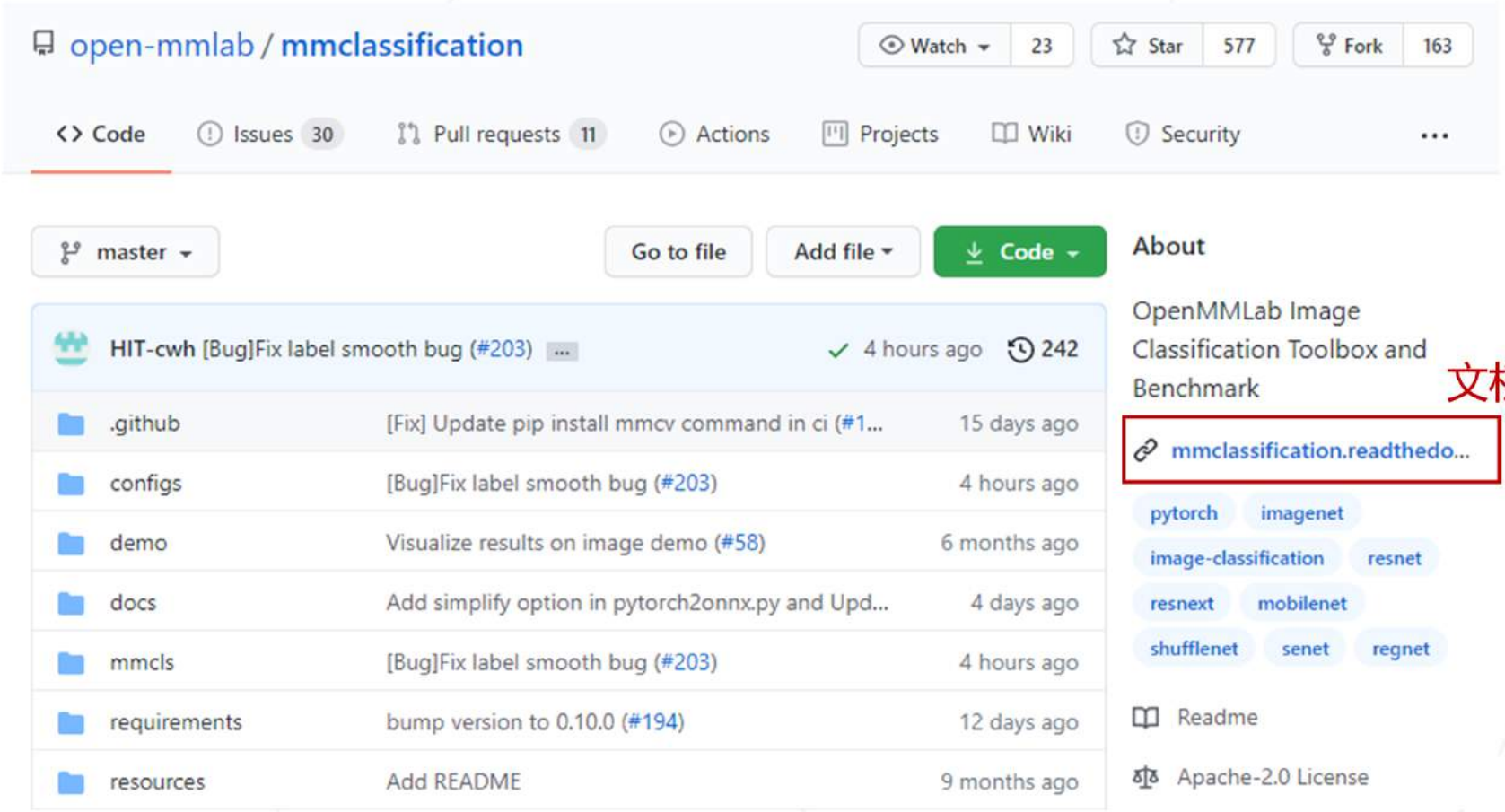
- CIFAR10
- ImageNet
- ……

丰富的训练技巧与策略

- 数据增强策略
- 优化器和学习率策略

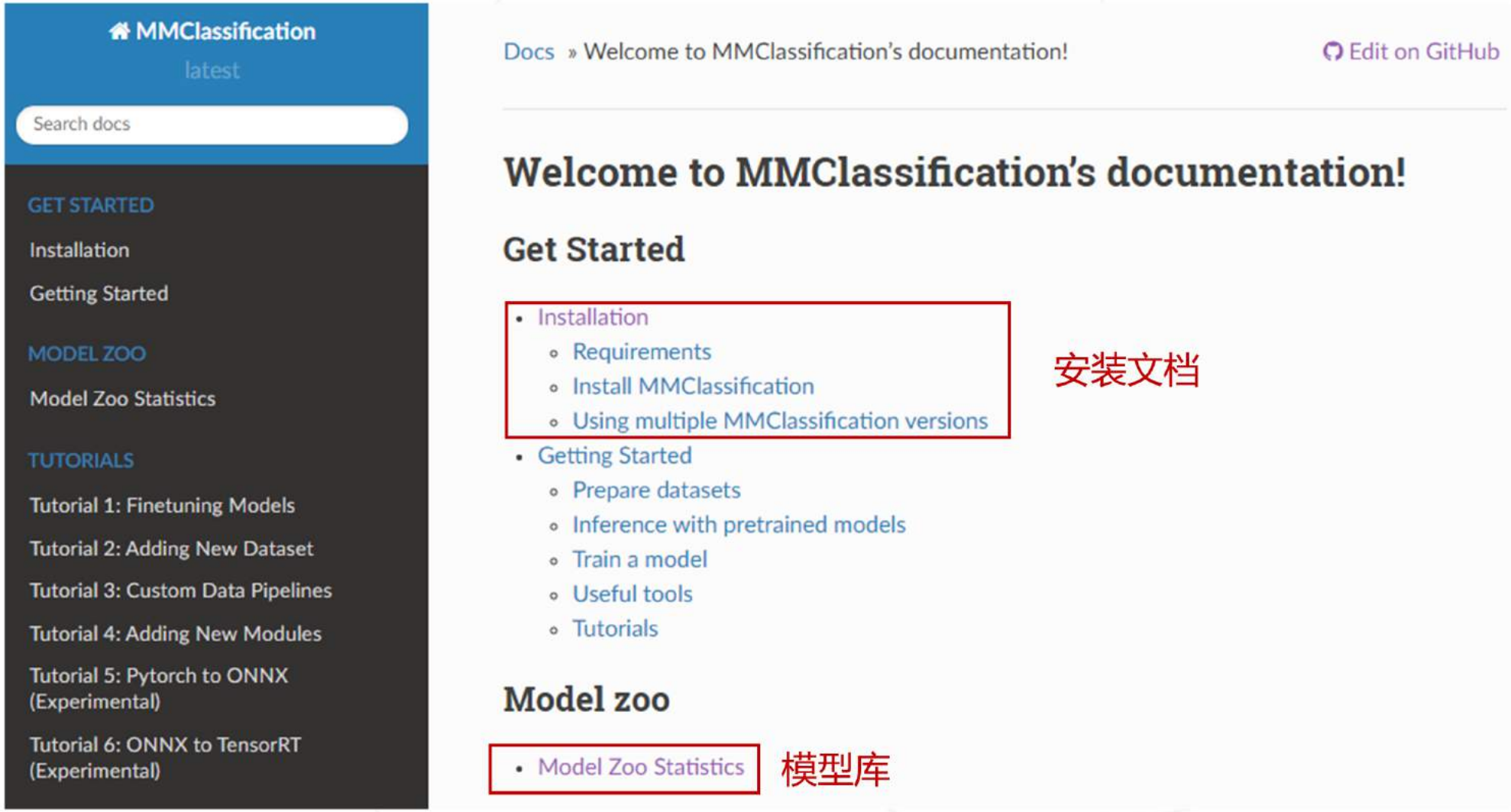
视频演示

https://github.com/open-mmlab/mmclassification

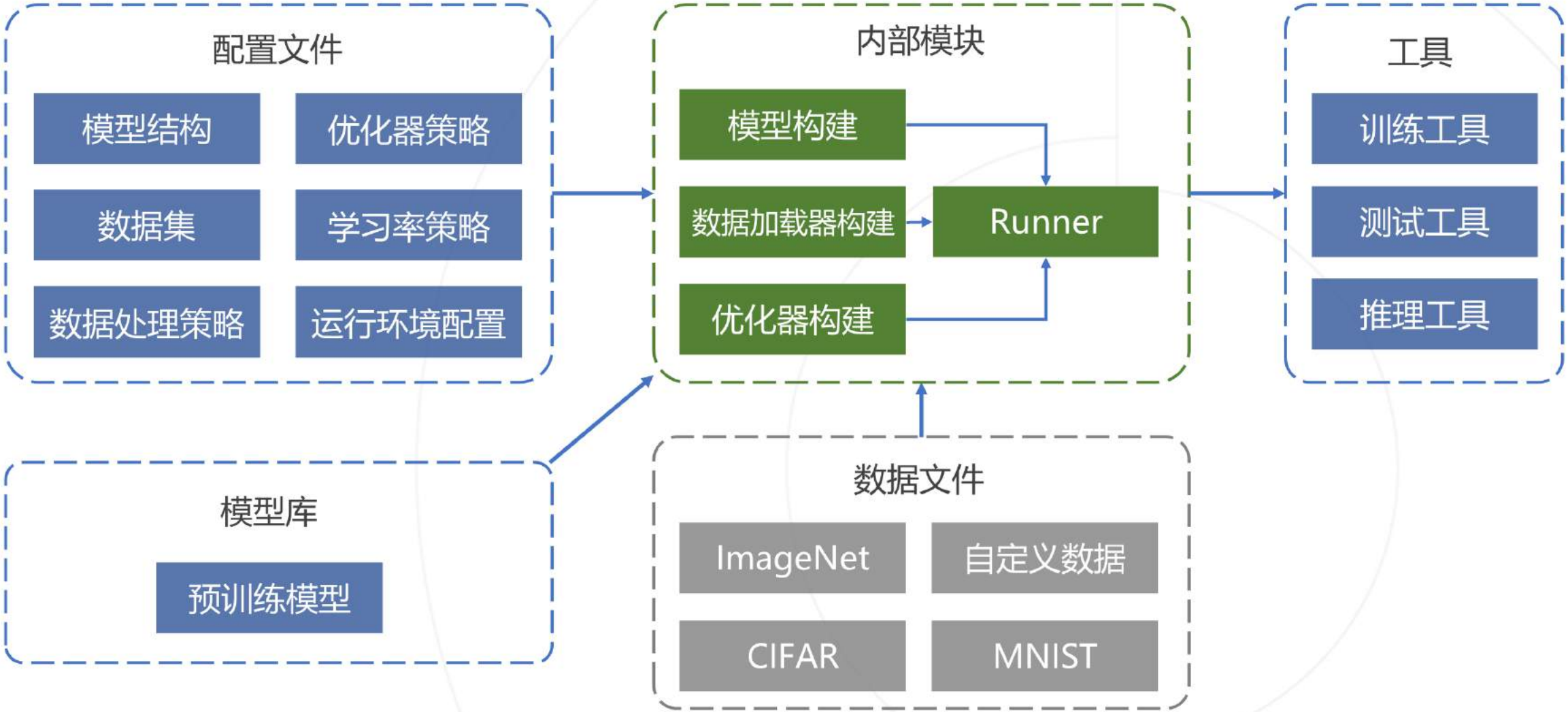


视频演示

https://mmclassification.readthedocs.io/en/latest/

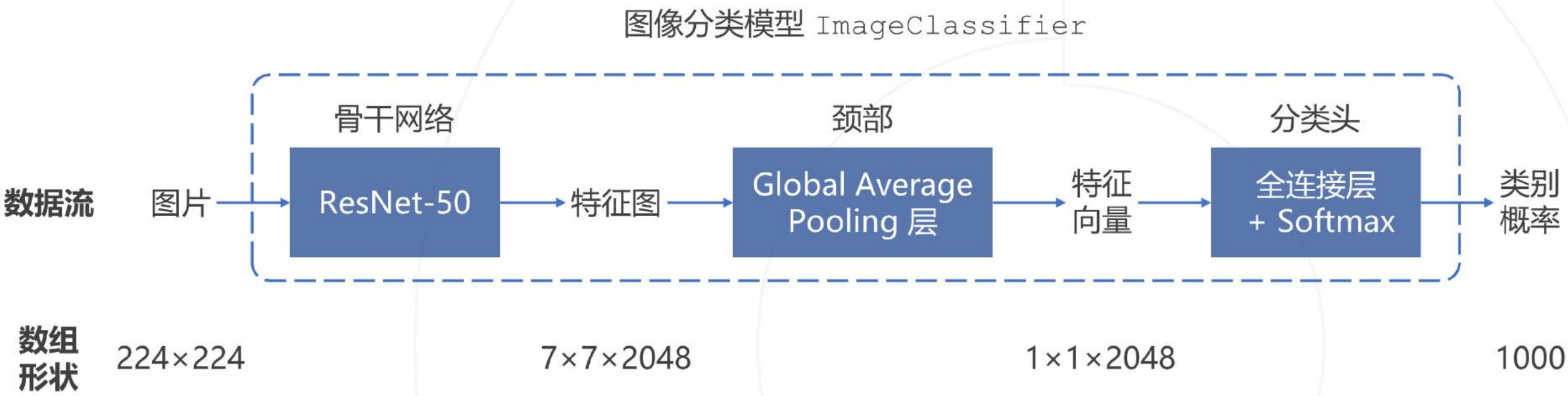


MMClassification 的项目结构

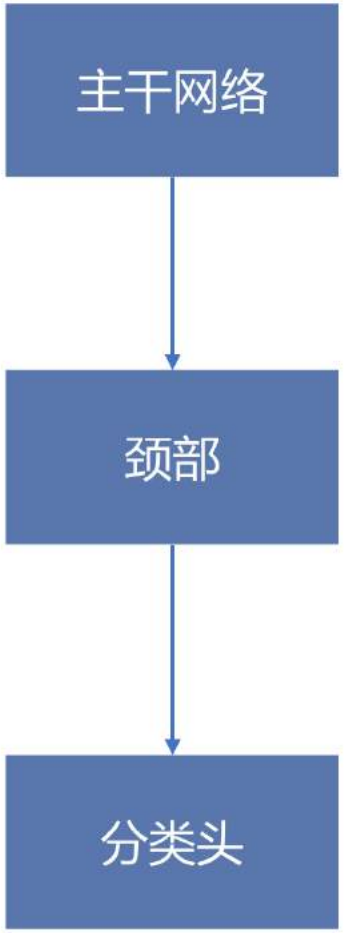


也适用于其他 OpenMMLab 工具包

图像分类模型的构成


图像分类模型 ImageClassifier
骨干网络
颈部
分类头
数据流
图片
ResNet-50
特征图
Global Average Pooling 层
特征向量
全连接层 + Softmax
类别概率
数组形状
224×224
7×7×2048
1×1×2048
1000

图像分类模型构建



模型

ResNet 50 模型

主干网络

使用全部4组卷积层

并输出最后一个卷积层的特征

使用v1b结构变体

全局平均池化

颈部

分类头

单层线性分类头

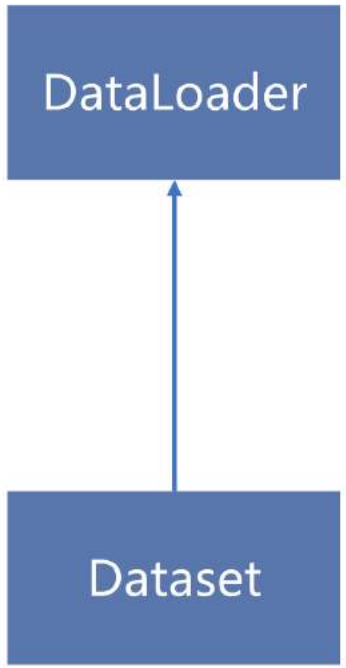
目标类别1000类

输入特征维度2048

使用交叉熵损失函数

```
model = dict(
    type='ImageClassifier',
    backbone=dict(
        type='ResNet',
        depth=50,
        num_stages=4,
        out_indices=(3, ),
        style='pytorch'),
    neck=dict(type='GlobalAveragePooling'),
    head=dict(
        type='LinearClsHead',
        num_classes=1000,
        in_channels=2048,
        loss=dict(type='CrossEntropyLoss',
loss_weight=1.0)),
    )
```

数据集构建



数据

设置 batch size

设置数据加载进程数

定义训练数据子集

ImageNet数据积类型

指定数据文件的路径

指定数据加载流水线

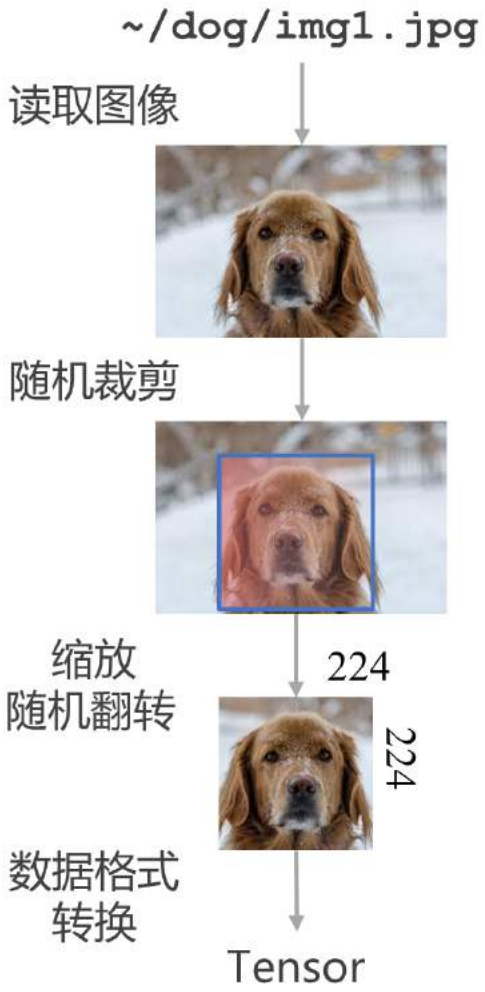
类似的方式

定义验证子集

和测试子集

```
data = dict(
    samples_per_gpu=32,
    workers_per_gpu=2,
    train=dict(
        type='ImageNet',
        data_prefix='data/imagenet/train',
        pipeline=train_pipeline),
    val=dict(
        # similar to train
    ),
    test=dict(
        # similar to val
    ),
)
```

定义数据加载流水线



像素归一化的均值和标准差

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395,
57.12, 57.375], to_rgb=True)
```

定义数据加载流水线

从文件中读取图像

随机裁剪与缩放

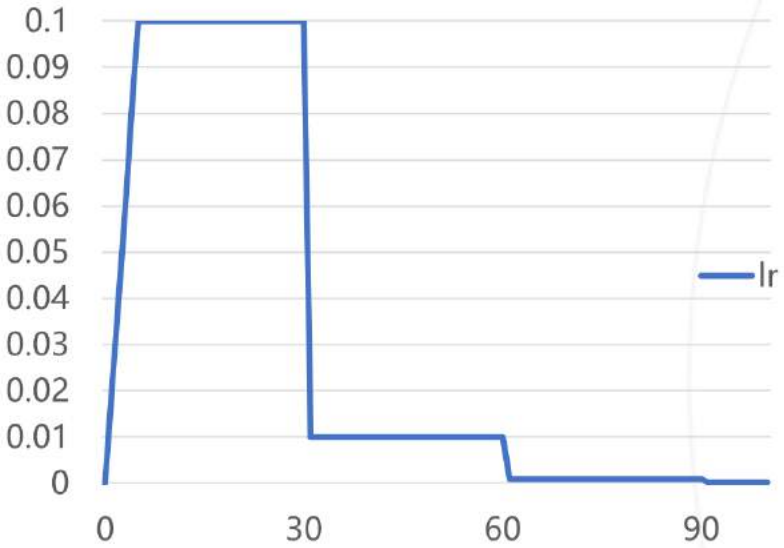
随机水平翻转

像素值归一化

将数据转换为 PyTorch Tensor

```
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', size=224),
    dict(type='RandomFlip', flip_prob=0.5,
direction='horizontal'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label'])
]
```

配置学习策略



优化器配置

使用 SGD 优化器

初始学习率 = 0.1

动量 = 0.9

权重衰减 = 0.0001

```
optimizer = dict(
    type='SGD',
    lr=0.1,
    momentum=0.9,
    weight_decay=0.0001)
```

学习率策略

使用步长下降策略

在特定周期降低学习率至 1/10

使用线性 warmup 策略

5个周期的 warmup

```
lr_config = dict(
    policy='step',
    step=[30, 60, 90],
    by_epoch=True,
    warmup='Linear',
    warmup_iters=5
    )
```

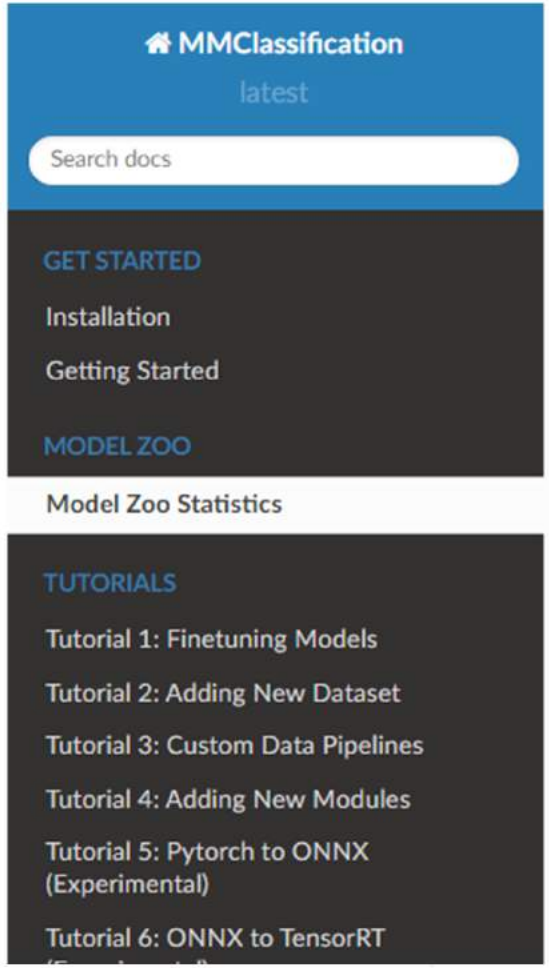
运行器 Runner 设置

基于轮数的运行器

最多训练 100 轮

```
runner = dict(
    type='EpochBasedRunner',
    max_epochs=100
    )
```

https://mmclassification.readthedocs.io/en/latest/modelzoo_statistics.html



Docs » Model Zoo Statistics

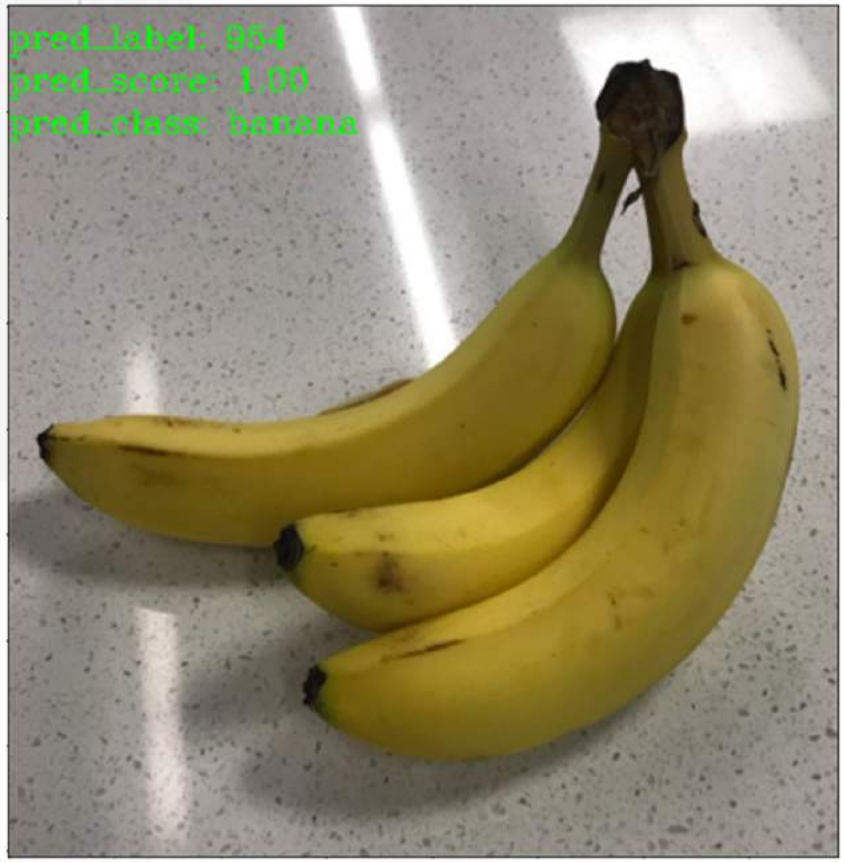
Edit on GitHub

Model Zoo Statistics

- Number of papers: 8
  - ALGORITHM: 7
  - OTHERS: 1
- Number of checkpoints: 31
  - [OTHERS] Mixed Precision Training (1 ckpts)
  - [ALGORITHM] MobileNetV2: Inverted Residuals and Linear Bottlenecks (1 ckpts)
  - [ALGORITHM] Deep Residual Learning for Image Recognition (13 ckpts)
  - [ALGORITHM] Aggregated Residual Transformations for Deep Neural Networks (4 ckpts)
  - [ALGORITHM] Squeeze-and-Excitation Networks (2 ckpts)
  - [ALGORITHM] ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices (1 ckpts)
  - [ALGORITHM] Shufflenet v2: Practical guidelines for efficient cnn architecture design (1 ckpts)
  - [ALGORITHM] Very Deep Convolutional Networks for Large-Scale Image Recognition (8 ckpts)

Python 推理 API

```python
from mmcls.apis import inference_model, init_model, show_result_pyplot

# Specify the path to config file and checkpoint file
config_file = 'configs/mobilenet_v2/mobilenet_v2_b32x8_imagenet.py'
checkpoint_file = 'checkpoints/mobilenet_v2_batch256_imagenet_20200708-3b2dc3af.pth'
# Specify the device. You may also use cpu by `device='cpu'`.
device = 'cuda:0'
# Build the model from a config file and a checkpoint file
model = init_model(config_file, checkpoint_file, device=device)
# Test a single image
img = 'demo/banana.png'
result = inference_model(model, img)
# Show the results
show_result_pyplot(model, img, result)
```




```
{'pred_class': 'banana',
 'pred_label': 954,
 'pred_score': 0.9999284744262695}
```

推理工具

单张图像推理

```
python demo/image_demo.py ${IMAGE_FILE} ${CONFIG_FILE} ${CHECKPOINT_FILE}
```

整个数据集推理

- 单机单卡

```
python tools/inference.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${RESULT_FILE}]
```

- 分布式推理

```
./tools/dist_inference.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [--out ${RESULT_FILE}]
```

举例：

假设预训练模型已经存放至checkpoints文件夹，基于ResNet50推理ImageNet的验证集的示例如下：

```
python tools/inference.py configs/imagenet/resnet50_batch256.py checkpoints/xxx.pth
```

单卡训练

```
python tools/train.py ${CONFIG_FILE} [optional arguments]
```

分布式训练

```
./tools/dist_train.sh ${CONFIG_FILE} ${GPU_NUM} [optional arguments]
```

支持任务调度器 Slurm

在通过slurm管理的集群上训练，可以使用slurm_train.sh脚本：

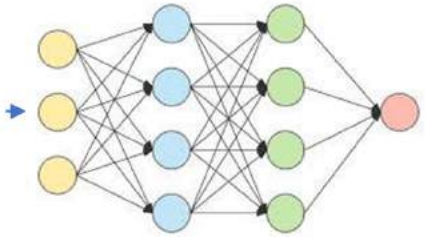
```
[GPUS=${GPUS}] ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${WORK_DIR}
```

课程内容

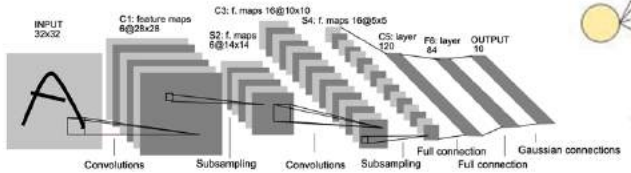
- 机器学习视角下的图像分类

$$\mathcal{H} = \{h(x;\theta) \mid \theta \in \Theta\}$$
$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\, \mathcal{L}(h|\mathcal{D})$$

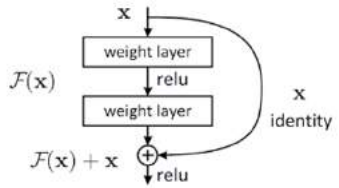
- 多层感知机



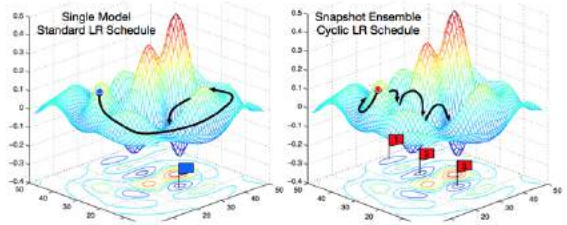
- 卷积神经网络



- **图像分类模型设计**



- **图像分类模型训练**



- PyTorch 介绍

PYTORCH

- **MMClassification 图像分类框架**

MMClassification

谢谢大家

OpenMMLab

通用视觉框架OpenMMLab

实践2 MMClassification

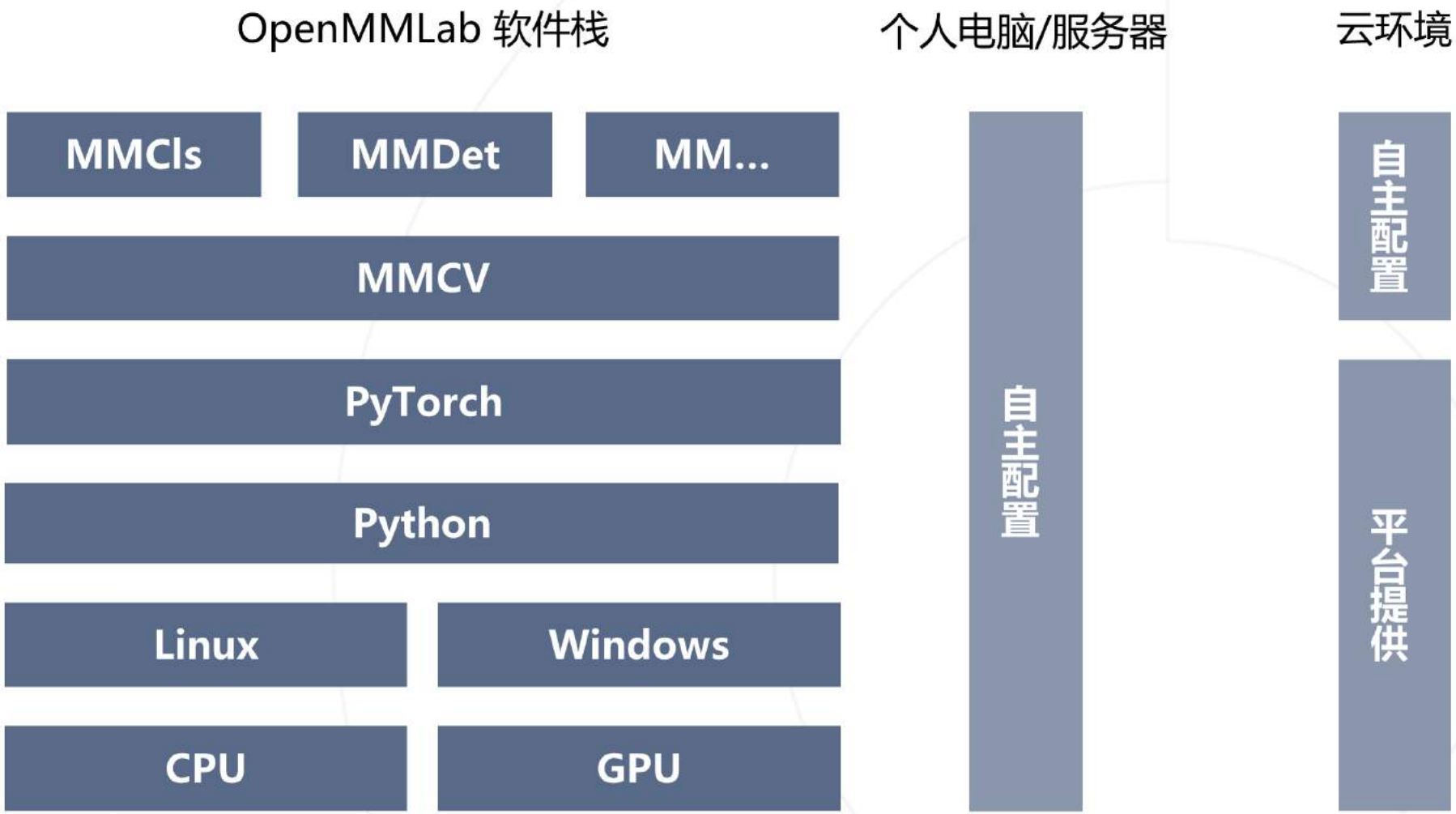
王若晖

2021年4月

内容

- MMClassification 运行环境搭建
- 使用预训练模型完成图像分类
- 训练自己的图像分类模型

OpenMMLab 环境依赖



总结

- MMClassification 运行环境搭建
  - Python 和 PyTorch 环境配置 （或使用云平台提供的环境）
  - MMCV 安装
  - MMClassification 安装
- 使用预训练模型完成图像分类
  - 配置文件
  - 预训练模型
  - Python API
- 训练自己的图像分类模型
  - 修改配置文件
  - 使用预训练模型

现状

1 架构

- 所有项目基于一致架构开发

15+ 研究方向

- 涵盖多个研究热点方向，算法覆盖完善

140+ 算法

- 包括 140+ 先进算法，性能领先

1200+ 预训练模型

- 拥有超过 1200 个预训练模型，真正实现开箱即用

答疑群

更多内容可参见 GitHub 上的代码文档以及教程 。
如有问题欢迎加入我们的QQ群进行提问。

谢谢大家

MMClassification

OpenMMLab

MMClassification

丰富的模型

VGG VGG-BN	ResNet ResNet V1D	ResNeXt ShuffleNet	SE-ResNet ShuffleNet V2 MobileNet V2	ResNeSt	ViT
2014	2015	2017	2018	2020	2021

丰富的数据集支持

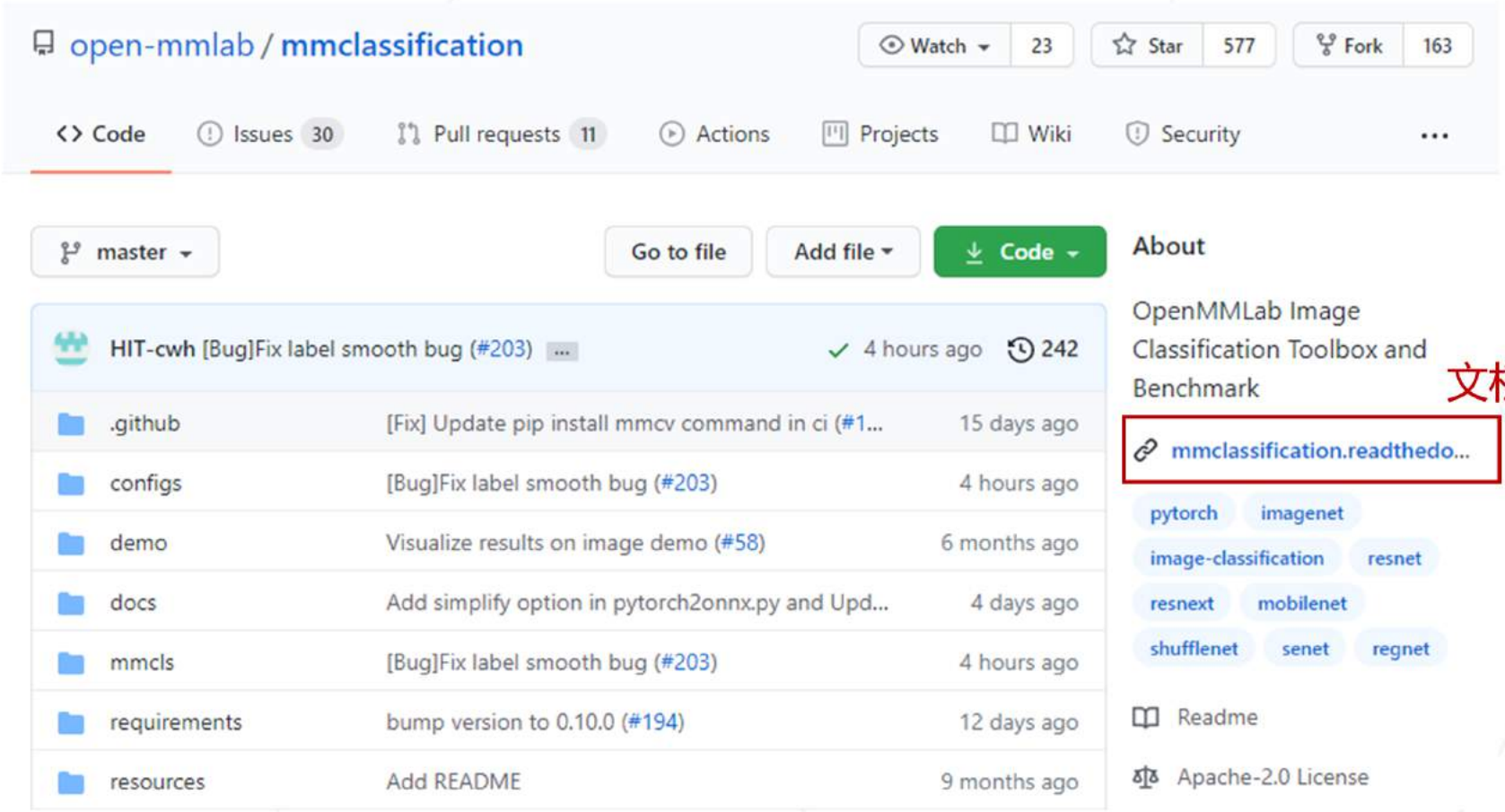
- CIFAR10
- ImageNet
- ……

丰富的训练技巧与策略

- 数据增强策略
- 优化器和学习率策略

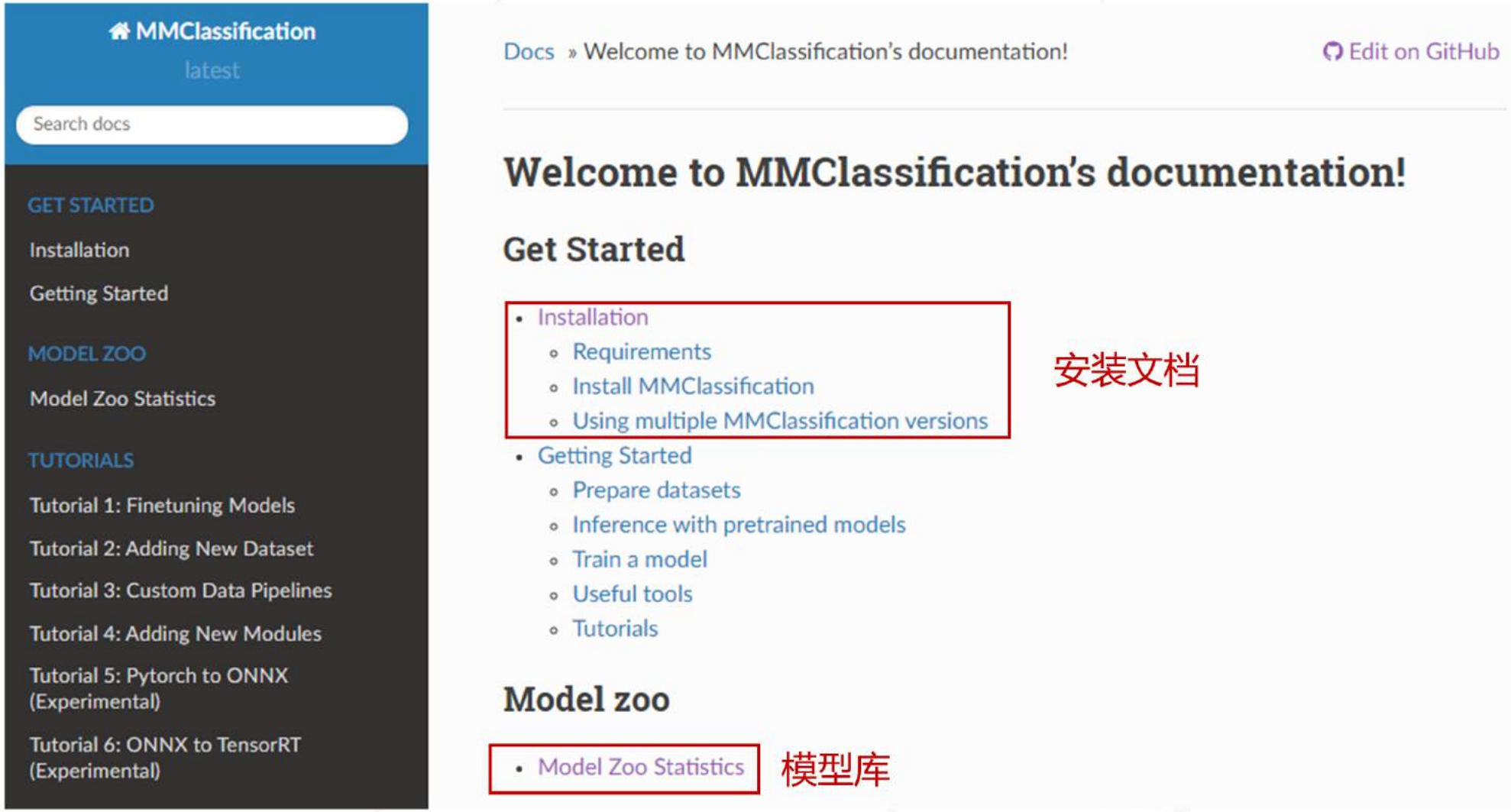
视频演示

https://github.com/open-mmlab/mmclassification

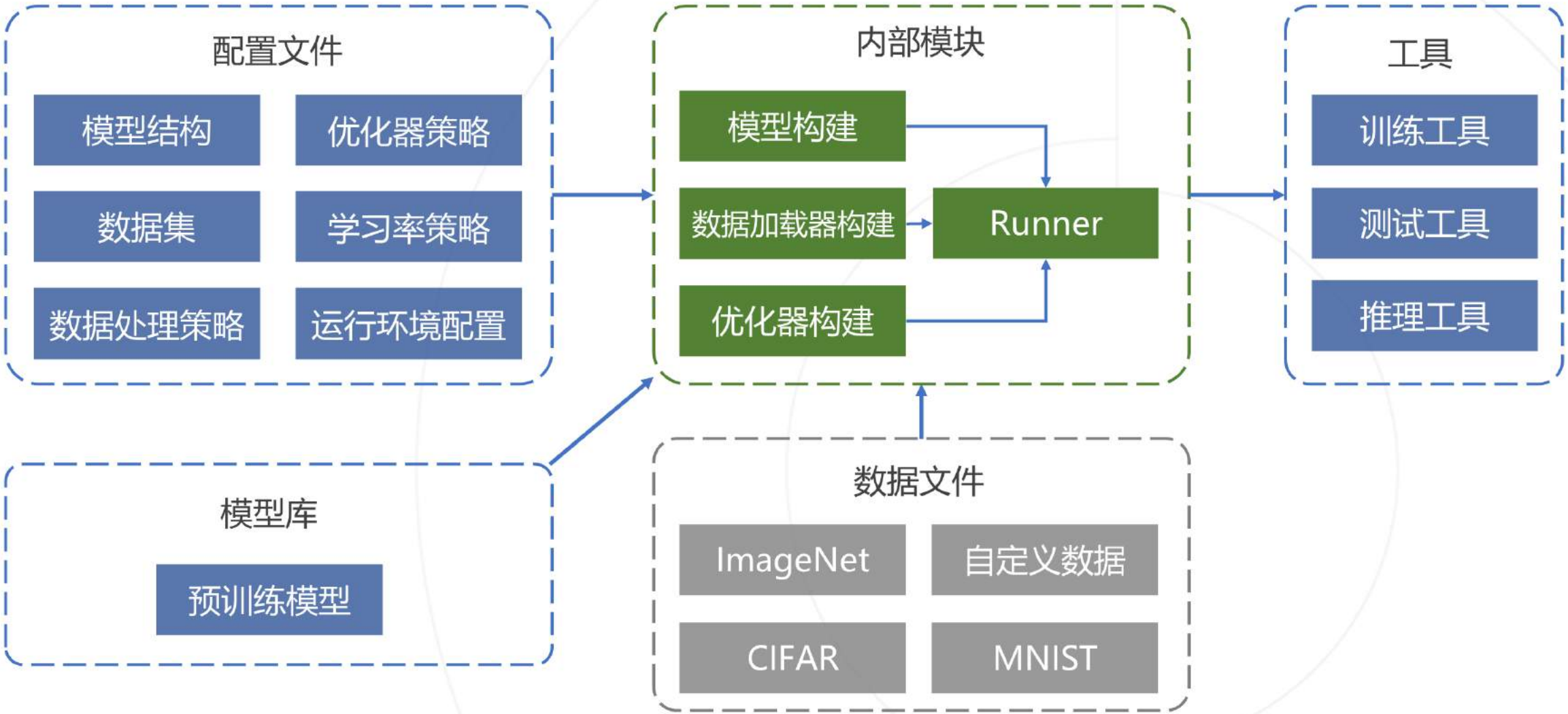


视频演示

https://mmclassification.readthedocs.io/en/latest/

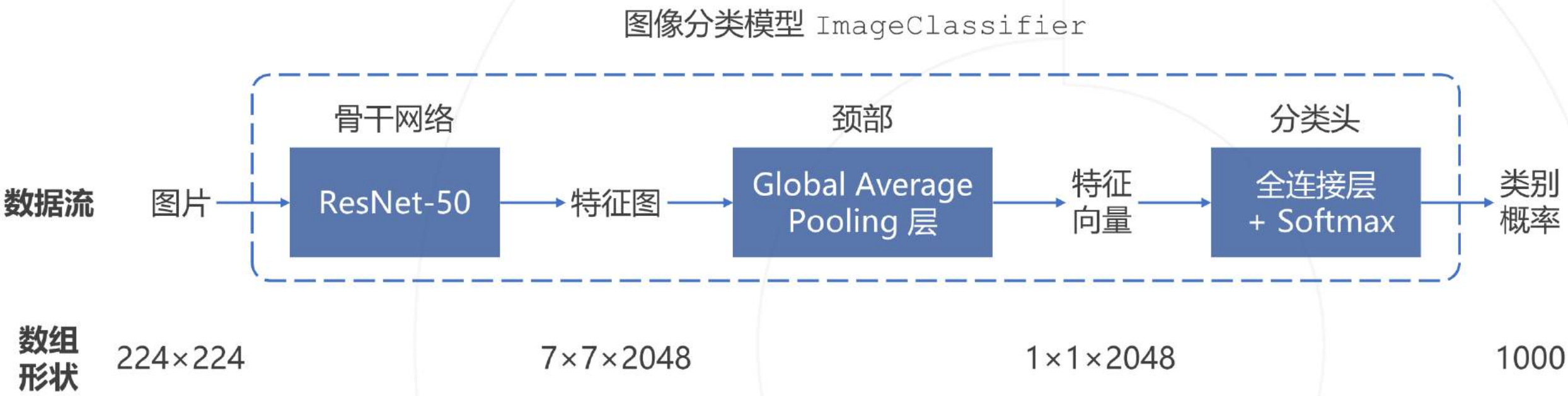


MMClassification 的项目结构

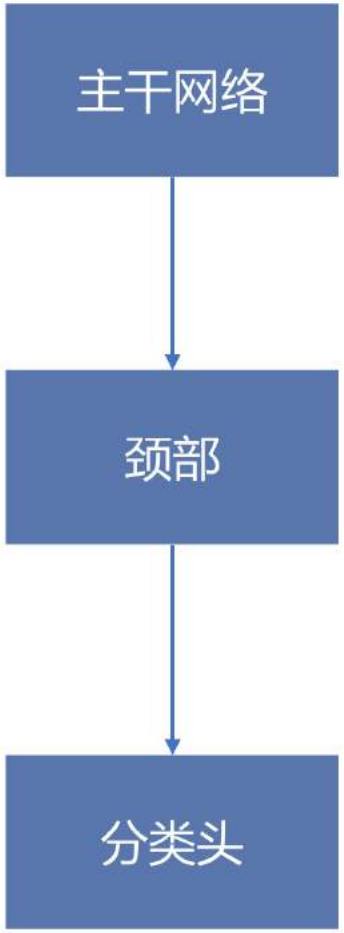


也适用于其他 OpenMMLab 工具包

图像分类模型的构成


图像分类模型 ImageClassifier
骨干网络
颈部
分类头
数据流
图片
ResNet-50
特征图
Global Average Pooling 层
特征向量
全连接层 + Softmax
类别概率
数组形状
224×224
7×7×2048
1×1×2048
1000

图像分类模型构建



模型

ResNet 50 模型

使用全部4组卷积层

并输出最后一个卷积层的特征

使用v1b结构变体

全局平均池化

颈部

分类头

单层线性分类头

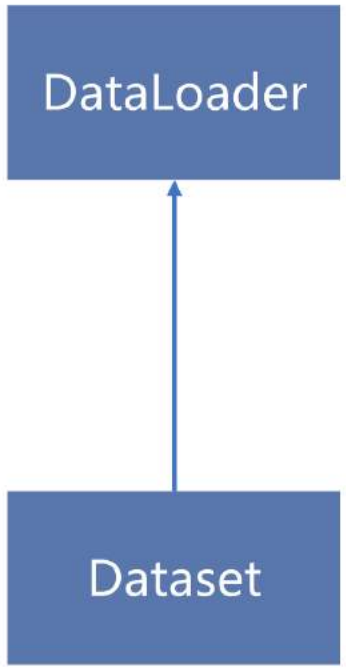
目标类别1000类

输入特征维度2048

使用交叉熵损失函数

```
model = dict(
    type='ImageClassifier',
    backbone=dict(
        type='ResNet',
        depth=50,
        num_stages=4,
        out_indices=(3, ),
        style='pytorch'),
    neck=dict(type='GlobalAveragePooling'),
    head=dict(
        type='LinearClsHead',
        num_classes=1000,
        in_channels=2048,
        loss=dict(type='CrossEntropyLoss',
loss_weight=1.0)),
    )
```

数据集构建



数据

设置 batch size

设置数据加载进程数

定义训练数据子集

ImageNet数据积类型

指定数据文件的路径

指定数据加载流水线

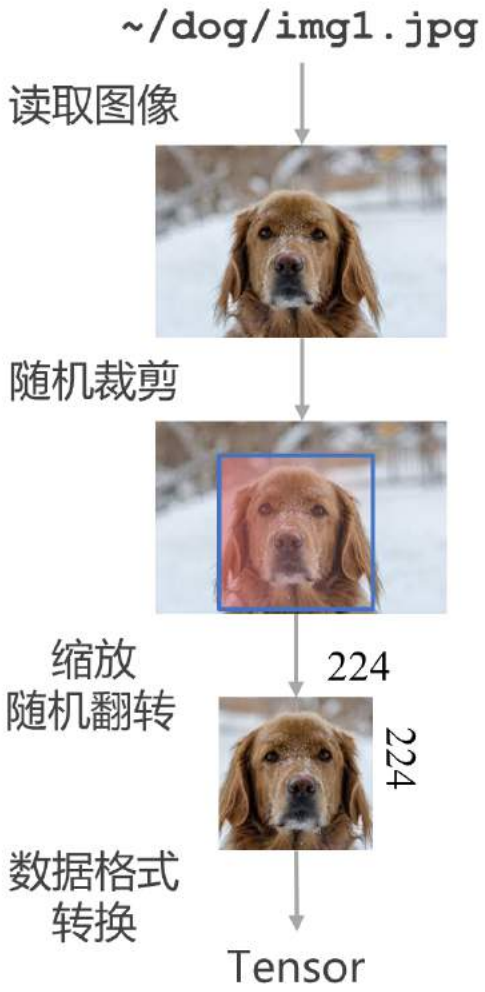
类似的方式

定义验证子集

和测试子集

```
data = dict(
    samples_per_gpu=32,
    workers_per_gpu=2,
    train=dict(
        type='ImageNet',
        data_prefix='data/imagenet/train',
        pipeline=train_pipeline),
    val=dict(
        # similar to train
    ),
    test=dict(
        # similar to val
    ),
)
```

定义数据加载流水线



像素归一化的均值和标准差

```
img_norm_cfg = dict(
    mean=[123.675, 116.28, 103.53], std=[58.395,
57.12, 57.375], to_rgb=True)
```

定义数据加载流水线

从文件中读取图像

随机裁剪与缩放

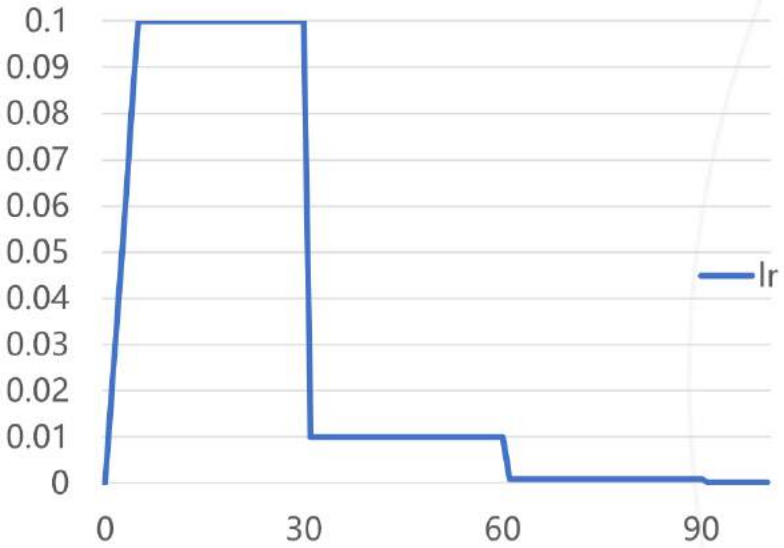
随机水平翻转

像素值归一化

将数据转换为 PyTorch Tensor

```
train_pipeline = [
    dict(type='LoadImageFromFile'),
    dict(type='RandomResizedCrop', size=224),
    dict(type='RandomFlip', flip_prob=0.5,
direction='horizontal'),
    dict(type='Normalize', **img_norm_cfg),
    dict(type='ImageToTensor', keys=['img']),
    dict(type='ToTensor', keys=['gt_label']),
    dict(type='Collect', keys=['img', 'gt_label'])
]
```

配置学习策略



优化器配置

使用 SGD 优化器

初始学习率 = 0.1

动量 = 0.9

权重衰减 = 0.0001

```
optimizer = dict(
    type='SGD',
    lr=0.1,
    momentum=0.9,
    weight_decay=0.0001)
```

学习率策略

使用步长下降策略

在特定周期降低学习率至 1/10

使用线性 warmup 策略

5个周期的 warmup

```
lr_config = dict(
    policy='step',
    step=[30, 60, 90],
    by_epoch=True,
    warmup='Linear',
    warmup_iters=5
    )
```

运行器 Runner 设置

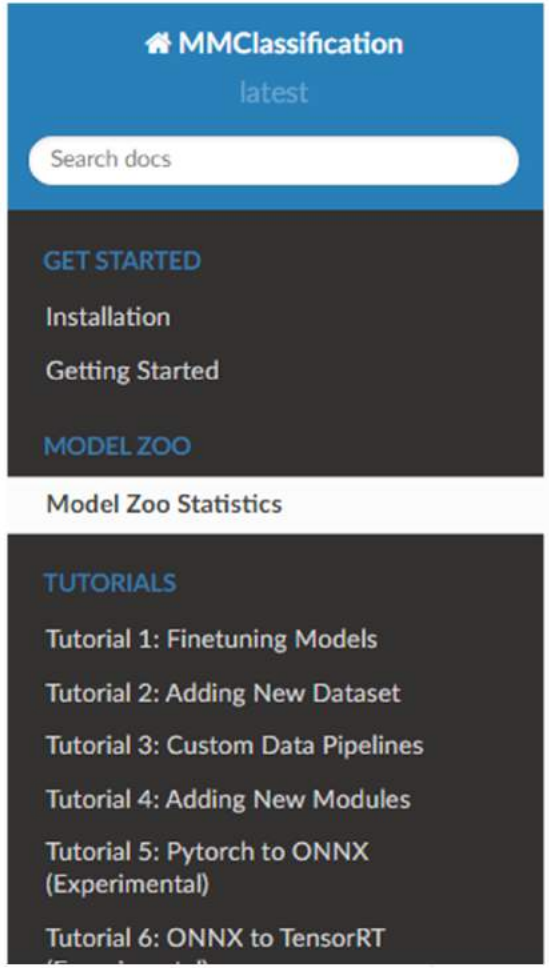
基于轮数的运行器

最多训练 100 轮

```
runner = dict(
    type='EpochBasedRunner',
    max_epochs=100
    )
```

预训练模型库

https://mmclassification.readthedocs.io/en/latest/modelzoo_statistics.html



Docs » Model Zoo Statistics

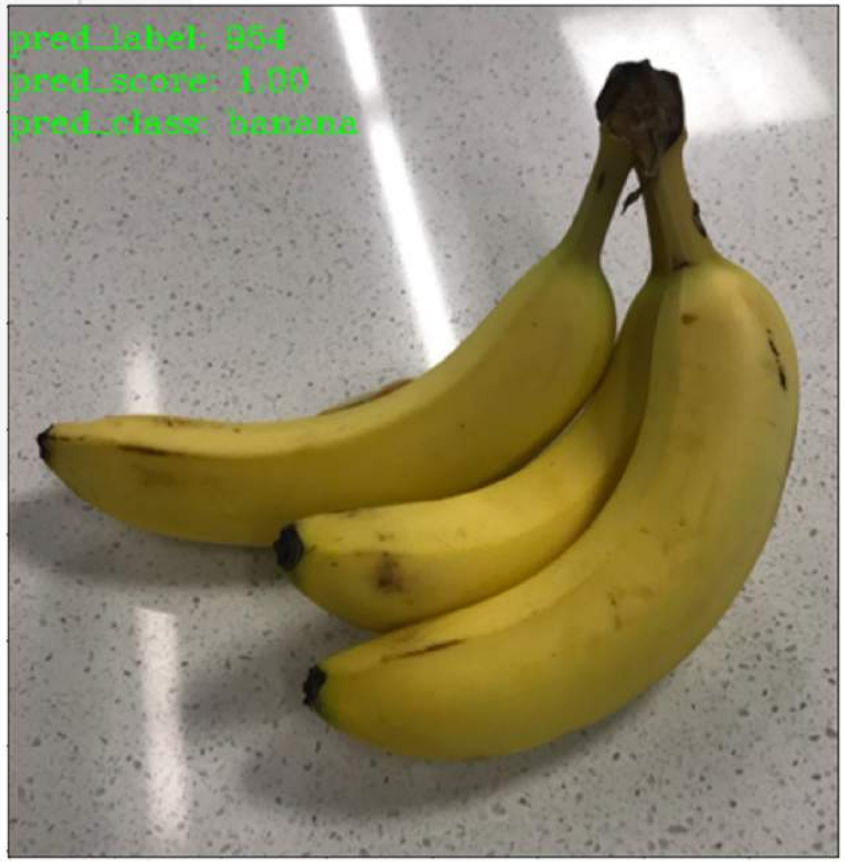
Edit on GitHub

Model Zoo Statistics

- Number of papers: 8
  - ALGORITHM: 7
  - OTHERS: 1
- Number of checkpoints: 31
  - [OTHERS] Mixed Precision Training (1 ckpts)
  - [ALGORITHM] MobileNetV2: Inverted Residuals and Linear Bottlenecks (1 ckpts)
  - [ALGORITHM] Deep Residual Learning for Image Recognition (13 ckpts)
  - [ALGORITHM] Aggregated Residual Transformations for Deep Neural Networks (4 ckpts)
  - [ALGORITHM] Squeeze-and-Excitation Networks (2 ckpts)
  - [ALGORITHM] ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices (1 ckpts)
  - [ALGORITHM] Shufflenet v2: Practical guidelines for efficient cnn architecture design (1 ckpts)
  - [ALGORITHM] Very Deep Convolutional Networks for Large-Scale Image Recognition (8 ckpts)

Python 推理 API

```python
from mmcls.apis import inference_model, init_model, show_result_pyplot

# Specify the path to config file and checkpoint file
config_file = 'configs/mobilenet_v2/mobilenet_v2_b32x8_imagenet.py'
checkpoint_file = 'checkpoints/mobilenet_v2_batch256_imagenet_20200708-3b2dc3af.pth'
# Specify the device. You may also use cpu by `device='cpu'`.
device = 'cuda:0'
# Build the model from a config file and a checkpoint file
model = init_model(config_file, checkpoint_file, device=device)
# Test a single image
img = 'demo/banana.png'
result = inference_model(model, img)
# Show the results
show_result_pyplot(model, img, result)
```




```
{'pred_class': 'banana',
 'pred_label': 954,
 'pred_score': 0.9999284744262695}
```

推理工具

单张图像推理

```
python demo/image_demo.py ${IMAGE_FILE} ${CONFIG_FILE} ${CHECKPOINT_FILE}
```

整个数据集推理

- 单机单卡

```
python tools/inference.py ${CONFIG_FILE} ${CHECKPOINT_FILE} [--out ${RESULT_FILE}]
```

- 分布式推理

```
./tools/dist_inference.sh ${CONFIG_FILE} ${CHECKPOINT_FILE} ${GPU_NUM} [--out ${RESULT_FILE}]
```

举例：

假设预训练模型已经存放至checkpoints文件夹，基于ResNet50推理ImageNet的验证集的示例如下：

```
python tools/inference.py configs/imagenet/resnet50_batch256.py checkpoints/xxx.pth
```

训练工具

单卡训练

```
python tools/train.py ${CONFIG_FILE} [optional arguments]
```

分布式训练

```
./tools/dist_train.sh ${CONFIG_FILE} ${GPU_NUM} [optional arguments]
```

支持任务调度器 Slurm

在通过slurm管理的集群上训练，可以使用slurm_train.sh脚本：

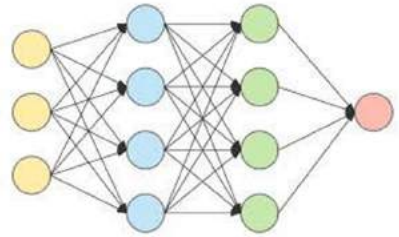
```
[GPUS=${GPUS}] ./tools/slurm_train.sh ${PARTITION} ${JOB_NAME} ${CONFIG_FILE} ${WORK_DIR}
```

课程内容

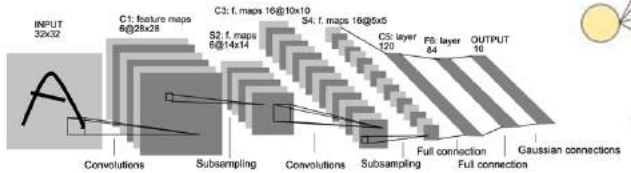
- 机器学习视角下的图像分类

$$\mathcal{H} = \{h(x;\theta) \mid \theta \in \Theta\}$$
$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\, \mathcal{L}(h|\mathcal{D})$$

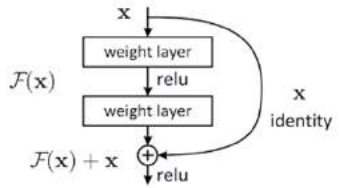
- 多层感知机



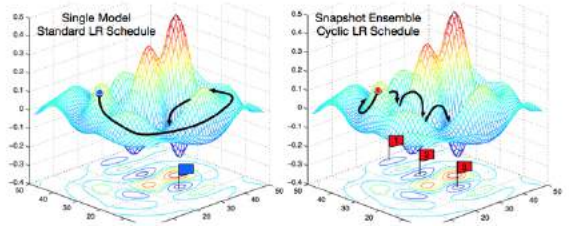
- 卷积神经网络



- **图像分类模型设计**



- **图像分类模型训练**



- PyTorch 介绍

PYTORCH

- **MMClassification 图像分类框架**

MMClassification