AI 安全入门

—— 以 针对目标检测模型的基于对抗补丁的攻击方法 为例

冯佳男

目 录

CONTENTS

1 深度学习简介

- 人工智能、机器学习、深度学习三者的关系
- 深度学习
- 卷积神经网络
  - 卷积层
  - 非线性层
  - 池化层
  - 全连接层
- 神经网络训练和测试

人工智能

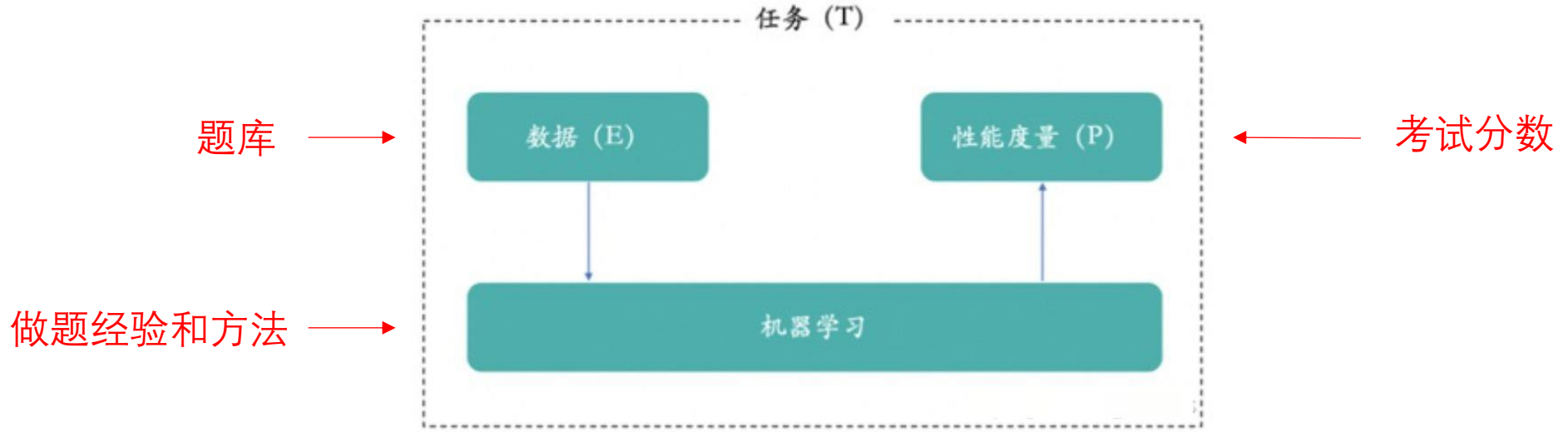
- 在介绍深度学习之前，先向大家介绍一下人工智能和机器学习，理清人工智能、机器学习、深度学习三者之间的关系。

- 1956年8月，在美国汉诺斯小镇的达特茅斯学院中，几位科学家在会议上正式提出**“人工智能”**(Artificial Intelligence，AI)这一概念。

- 当时人类已经制造出各类各样的机器如汽车、飞机等，但这些机器都需要经过人来操作使用，无法自己具备操作的能力。
- 科学家探讨能不能制造出一个可以像人类大脑的一样思考的机器，拥有人类的智慧，这就是人工智能。

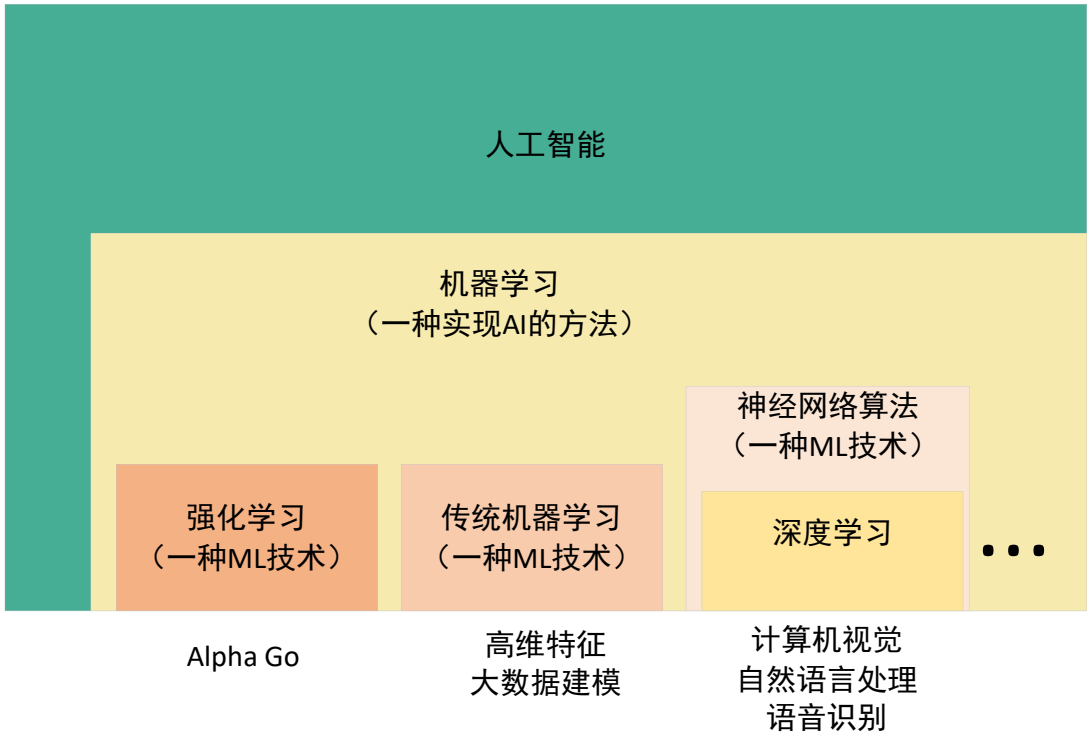
机器学习

- 1956年的美国达特茅斯会议上，IBM的工程师Arthur Samuel正式提出“Machine Learning”这个概念。
- 将实现人工智能的方法统称为“**机器学习**”。《西瓜书》里面如此介绍机器学习：机器学习是机器从历史数据中学习规律，来提升系统的某个性能度量。
- 例如：大家在学习数学的时候刷过大量的**题库**，老师会强调我们要学会总结，从之前做过的题目中，**总结经验和方法**（可以理解为就是机器学习产出的模型）。然后我们再做数学题，利用之前总结的经验和方法就可以**考更高的分**。



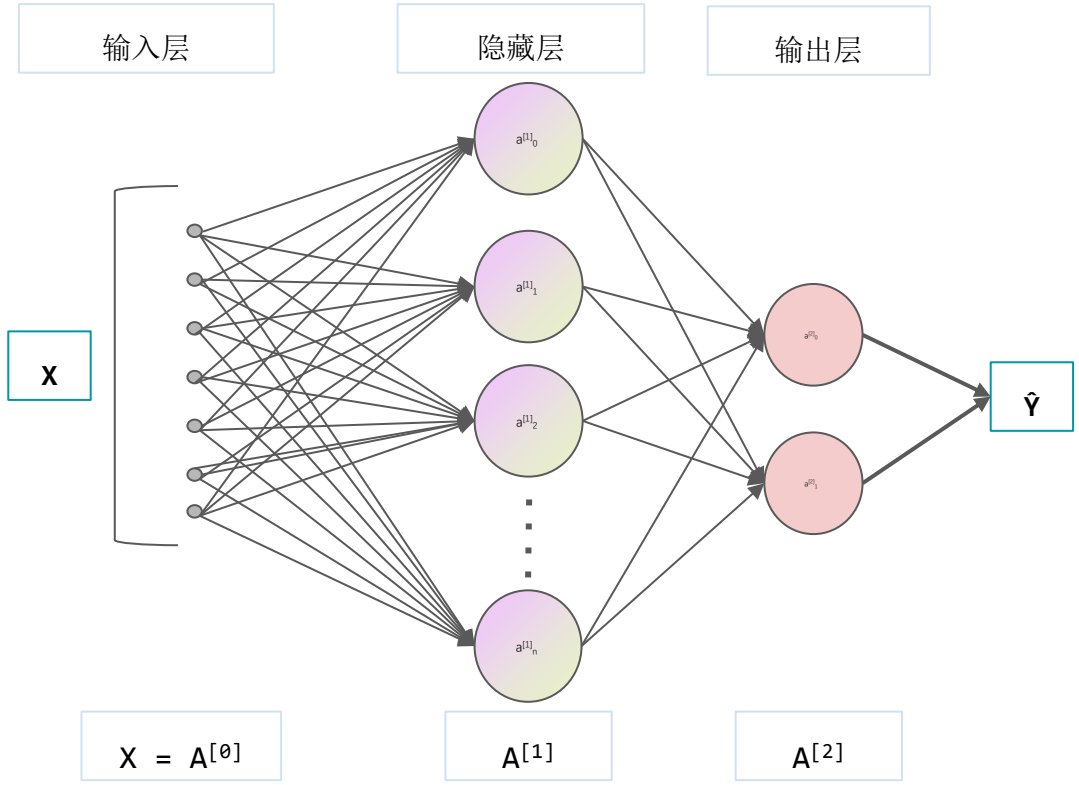
人工智能、机器学习、深度学习的关系

- 人工智能、机器学习、深度学习的关系可以通过下图进行描述：

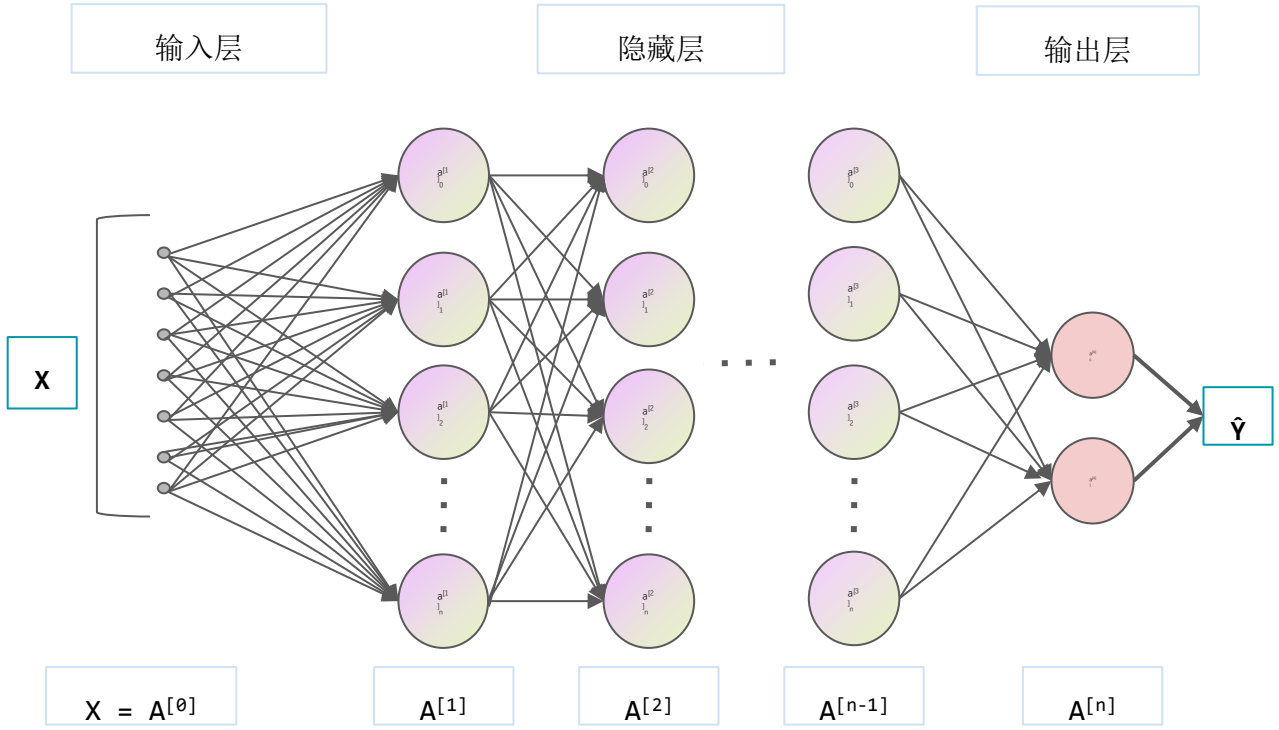


- 让机器实现AI是人类的一个美好愿景，而机器学习是实现AI的一种方法论，深度学习是该方法论下一种新的技术，在图像识别、语义理解和语音识别等领域具有不错的效果。

神经网络



神经网络

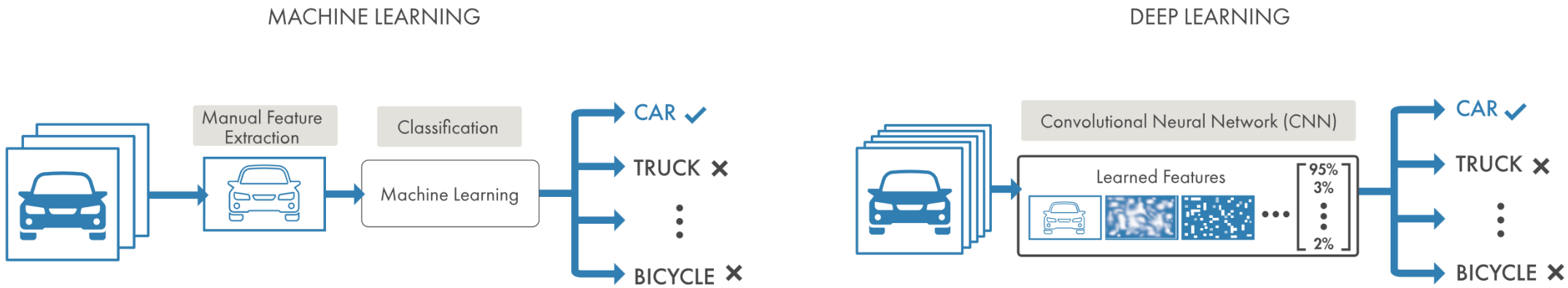


深度神经网络

大多数深度学习方法使用**神经网络**架构，这也是深度学习模型通常被称为**深度神经网络**的原因。

深度学习

- 以图像分类任务为例，在传统机器学习中（左图），**手动**选择用于对图像进行分类的特征和分类器。
- 在深度学习中（右图），特征提取和建模步骤是**自动化**的。



- 卷积神经网络CNN是最流行的深度神经网络类型之一

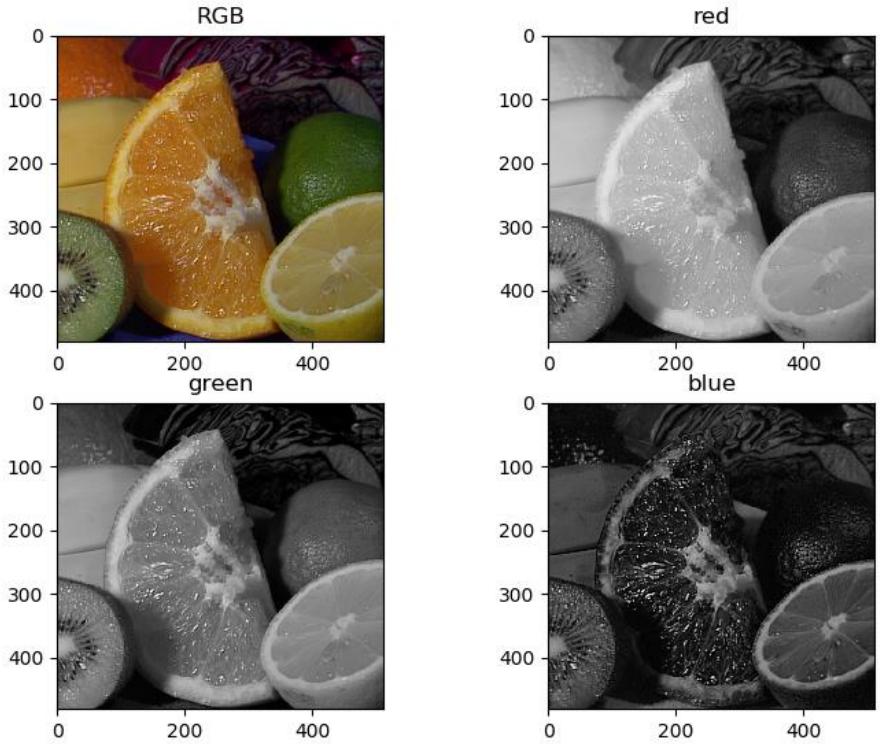
卷积神经网络

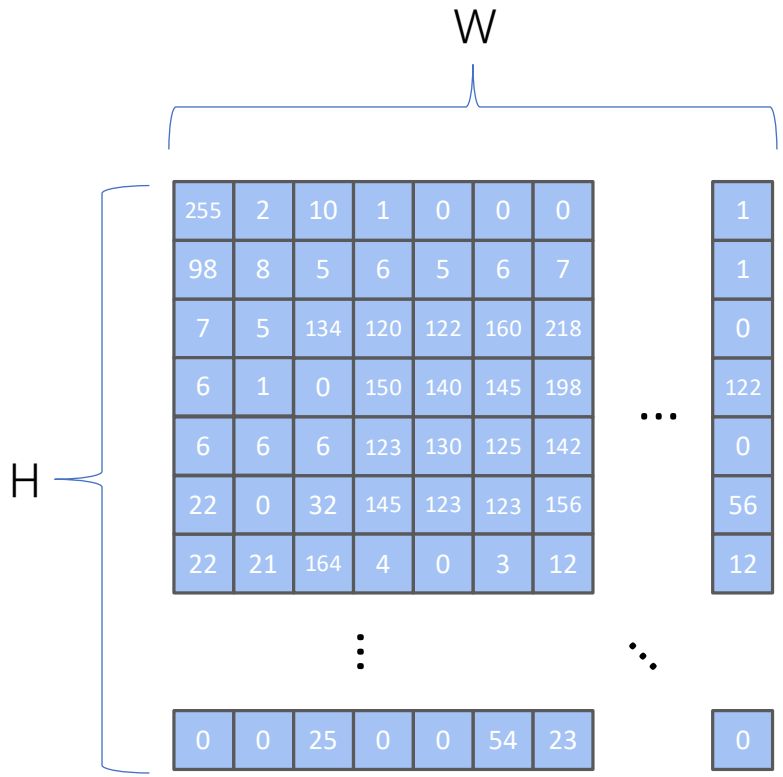
- 卷积神经网络是一类包含卷积计算且具有深度结构的神经网络，是深度学习的代表算法之一。在图像任务中具有非常好的效果。

- 卷积神经网络主要包含：
  - 输入层：对原始图像数据进行预处理，如去均值、归一化等。
  - 卷积层：神经网络中最重要的结构。
  - 非线性层：对卷积层输出做非线性映射，如ReLU、tanh、Sigmoid等。
  - 池化层：夹在在连续的卷积层中间，用于压缩数据，如最大、平均池化等。
  - 全连接层：传统的神经网络神经元的连接方式。

卷积神经网络 - 输入层

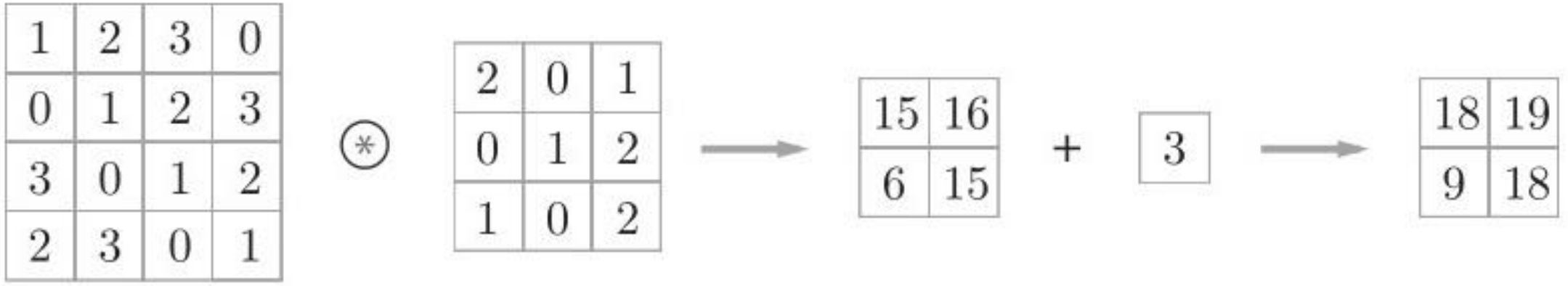
- 图像 = 矩阵。 [480, 512, 3] 代表一张高为480, 宽为512的RGB图像。矩阵中的每个值表示像素值（0-255），像素值的大小代表图像的明暗程度。
- 对原始图像数据进行预处理，如去均值、归一化（像素值都除以255）等





卷积神经网络 – 卷积层

- 在CNN中，将卷积层的输入和输出数据称为输入特征图（input feature map）和输出特征图（output feature map）
- 卷积层进行的操作相当于图像处理中的“滤波器运算”
- $output = input \cdot w + b$　　$\cdot$表示卷积运算



输入特征图
$input$ (4, 4)

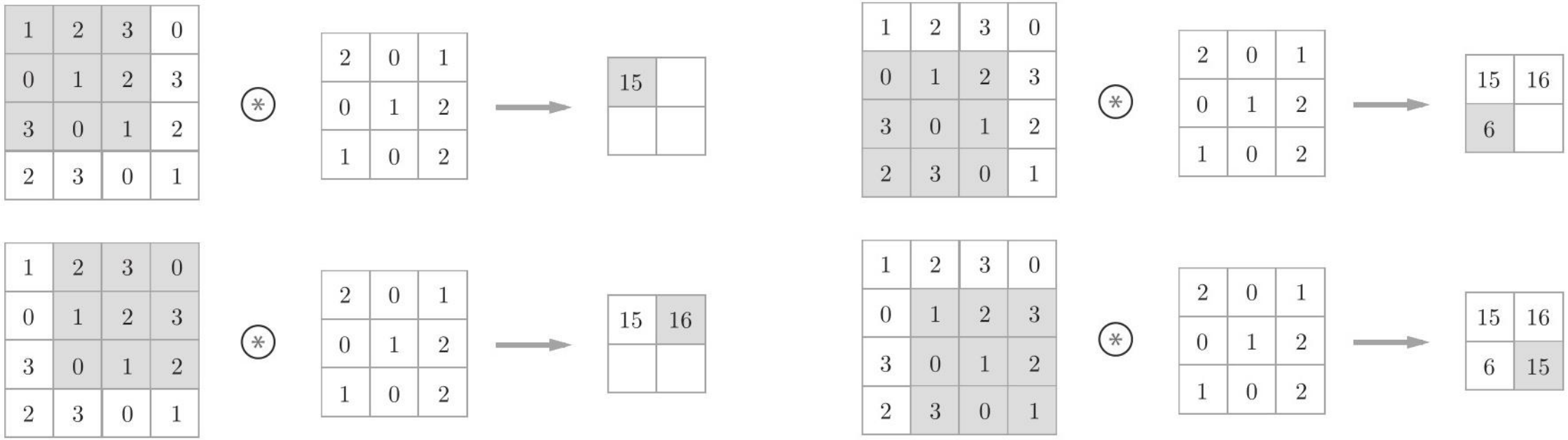
卷积核
w (3, 3)

偏置
b (1)

输入特征图
$output$ (2, 2)

卷积神经网络 – 卷积层

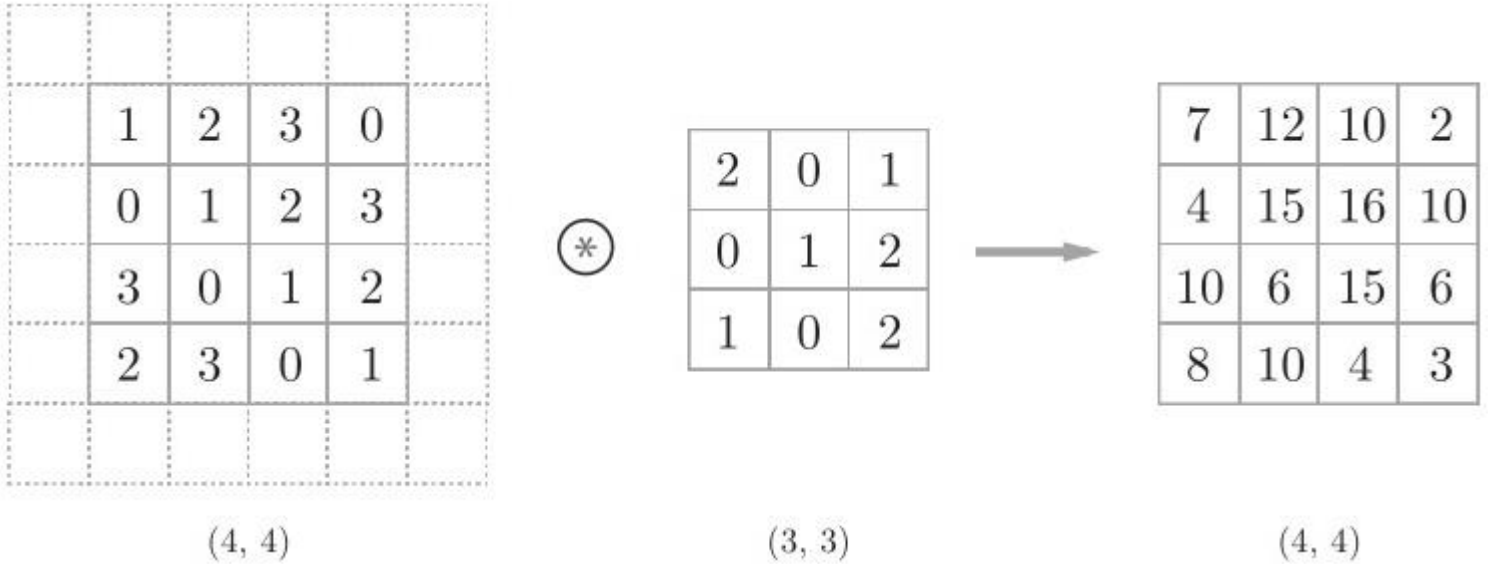
- 卷积运算的顺序:



- 对大小为（4，4）的输入数据应用（3，3）的卷积核时，输出大小变为（2，2），相当于高和宽各缩小了2。如果每次卷积后都缩小，那么在某个时刻输出大小就有可能变为1，导致无法再进行卷积操作。为避免出现这样的情况，使用填充操作。

卷积神经网络 – 卷积层（填充 padding）

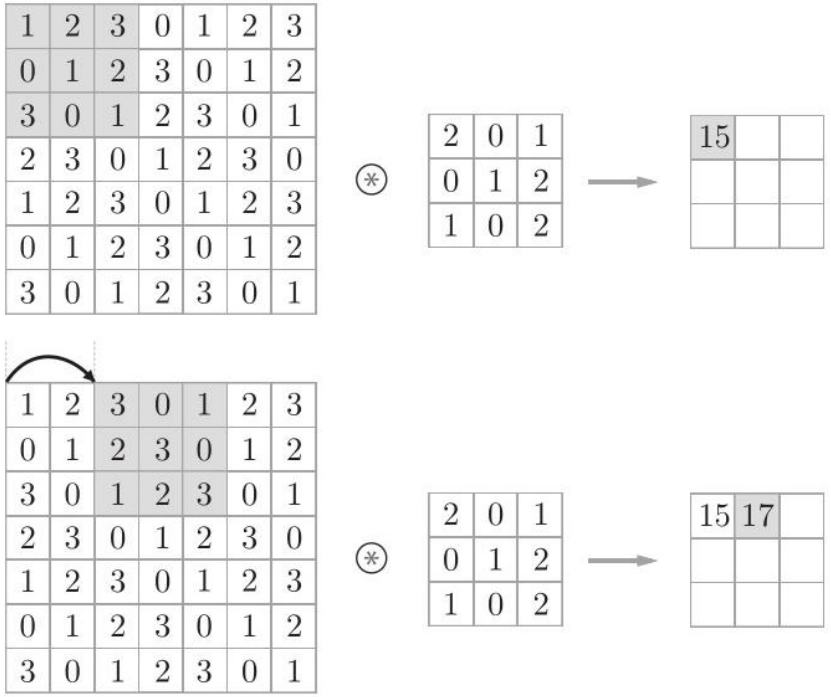
- 向输入数据的周围填入0（图中用虚线框表示填充，并省略了填充的内容“0”）
- 此时向外填充的范围padding为1



- 这样可以使得输出特征和输入特征保持一样的形状

卷积神经网络 – 卷积层（步长 stride）

- 将卷积核每次移动的距离称为**步长**
- 当卷积核移动的步长为2时：



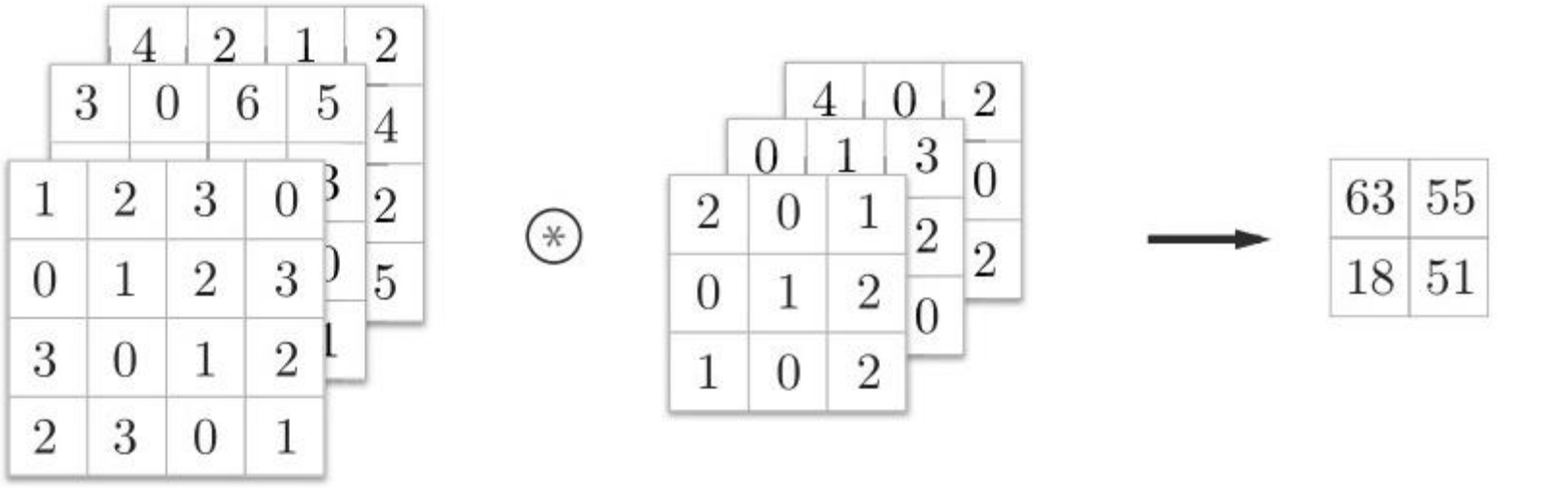
假设输入大小为 (H, W),
卷积核大小为(FH, FW),
填充为P，步长为S,
此时输出大小(OH, OW)可以通过下方公式进行计算：

$$OH = (H + 2 * P - FH)/S + 1$$
$$OW = (W + 2 * P - FW)/S + 1$$

$$OH = (7 + 2 * 0 - 3)/2 + 1 = 3$$

卷积神经网络 – 卷积层

- 3维数据的卷积运算例子（暂时忽略偏置）：
- 使用多维数组表示3维数据时，书写顺序为(Channel, H, W)



输入特征图
$input$ (3, 4, 4)

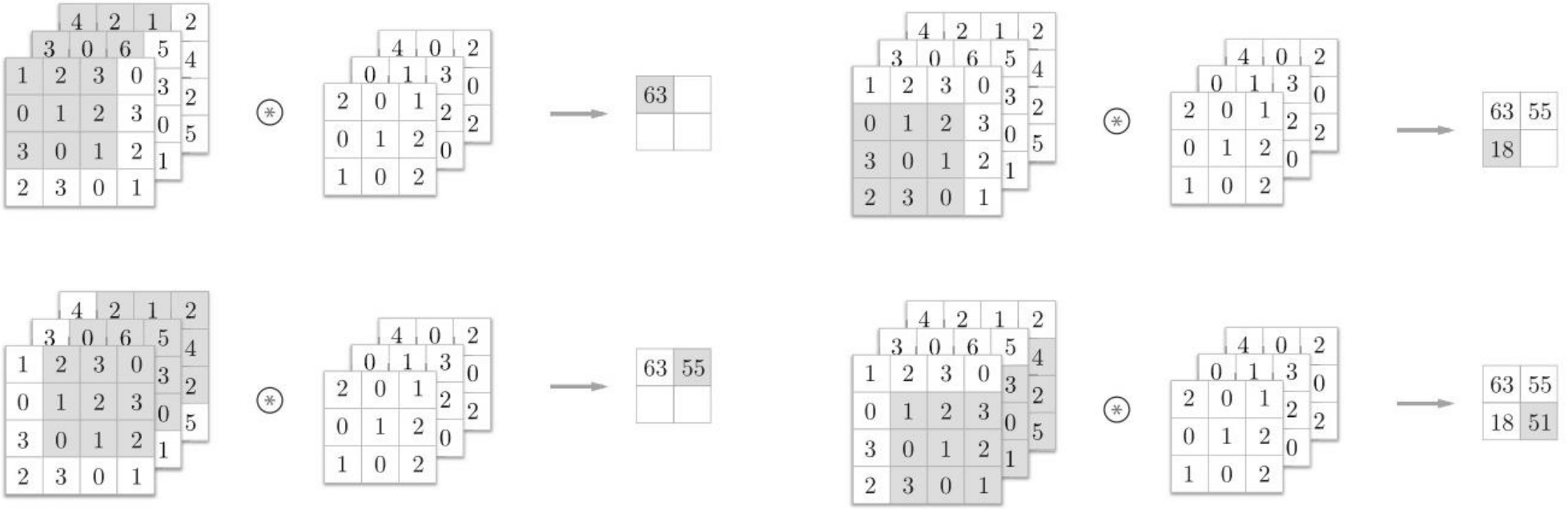
卷积核
w (3, 3, 3)

卷积后的结果
(2, 2)

- 一组卷积核的个数和输入特征图的通道数是对应的

卷积神经网络 – 卷积层

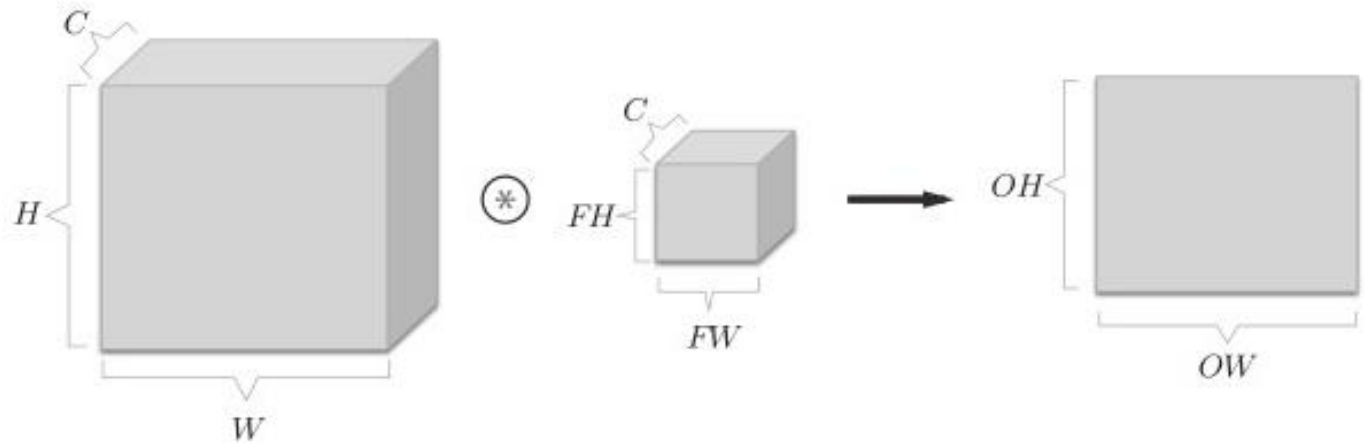
- 3维数据的卷积运算的顺序:



- 通道方向上有多个特征图时，会按通道进行输入数据和卷积核的卷积运算，并将结果相加，从而得到输出

卷积神经网络 – 卷积层

- 将数据和卷积核结合方块来考虑，帮助理解三维数据的卷积运算



输入特征图
$input$ (C, H, W)

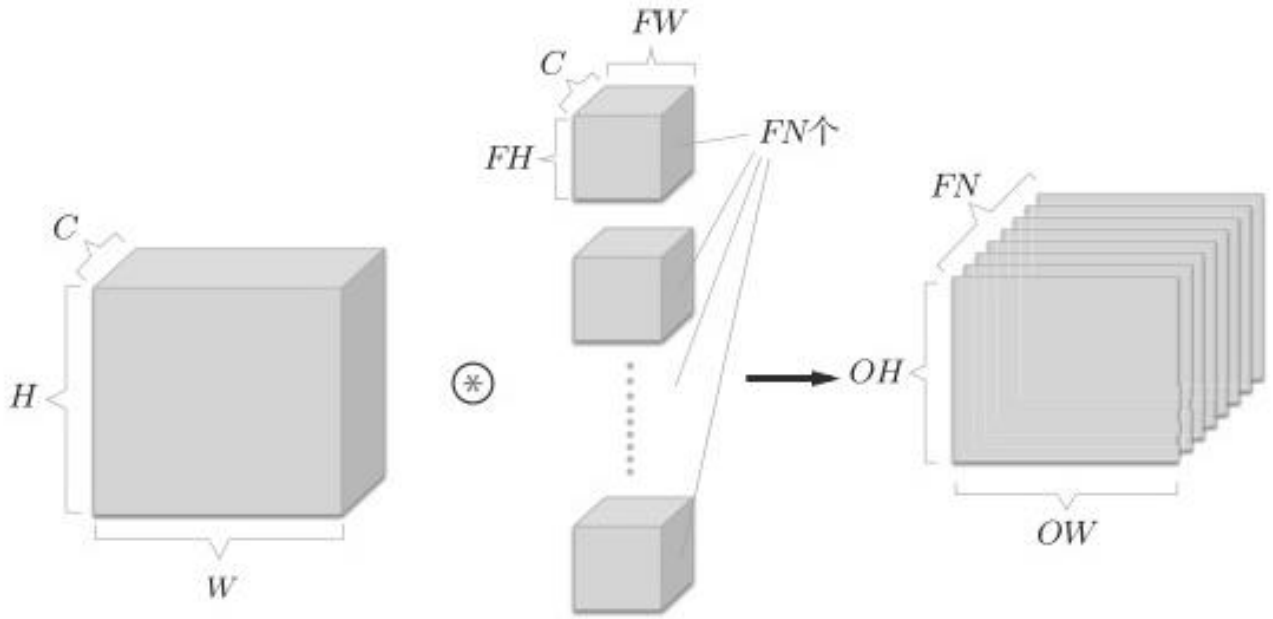
卷积核
w (C, FH, FW)

卷积后结果
(1, OH, OW)

- 按照上述方式获得的输出特征图的通道数为1，如何获得通道数为n的特征图？

卷积神经网络 – 卷积层

- 通过设置**FN**组卷积核，来获得通道数为**FN**的输出特征图



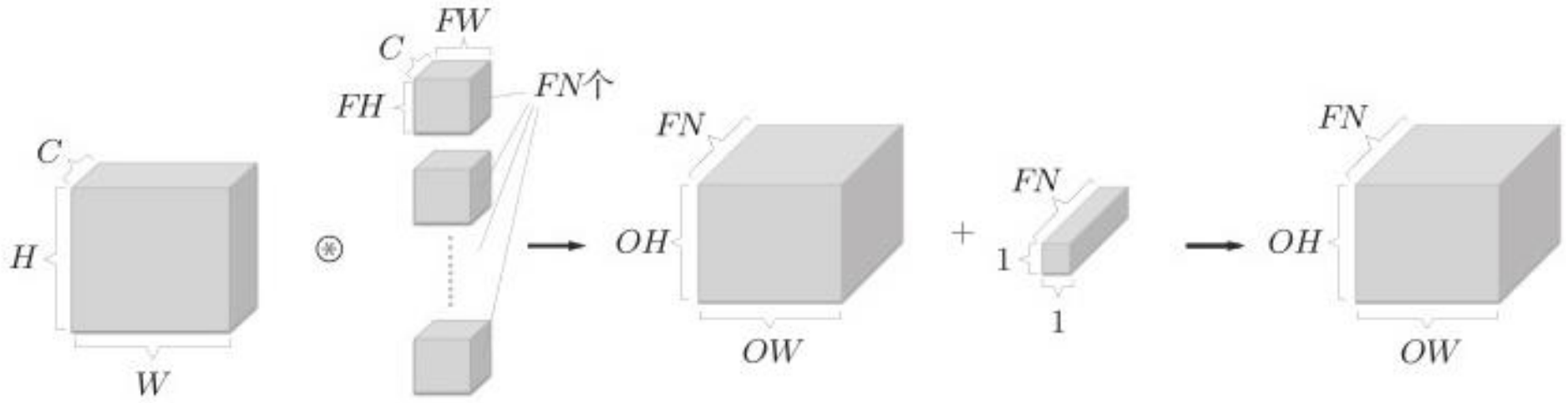
输入特征图
$input$ (C, H, W)

卷积核
w (**FN,** C, FH, FW)

卷积后结果
(**FN**, OH, OW)

卷积神经网络 – 卷积层

- 完整的3维数据的卷积运算:



输入特征图
$input$ (C, H, W)

卷积核
w (**FN,** C, FH, FW)

偏置
(**FN**, 1, 1)

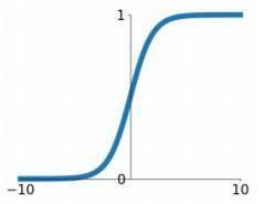
卷积后结果
(**FN**, OH, OW)

卷积神经网络 – 非线性层（激活层）

- 非线性层：对卷积层输出做非线性映射，如ReLU、tanh、Sigmoid等。
- 特点：不改变特征图的形状；一般卷积层后都会跟着非线性层。

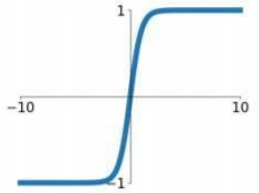
Sigmoid

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



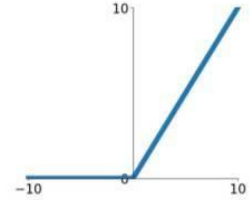
tanh

$$\tanh(x)$$



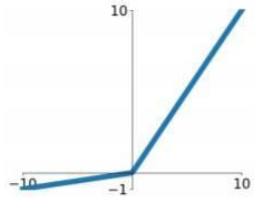
ReLU

$$\max(0, x)$$



Leaky ReLU

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

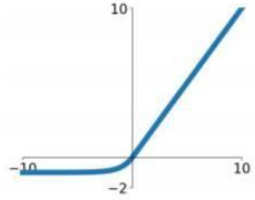


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

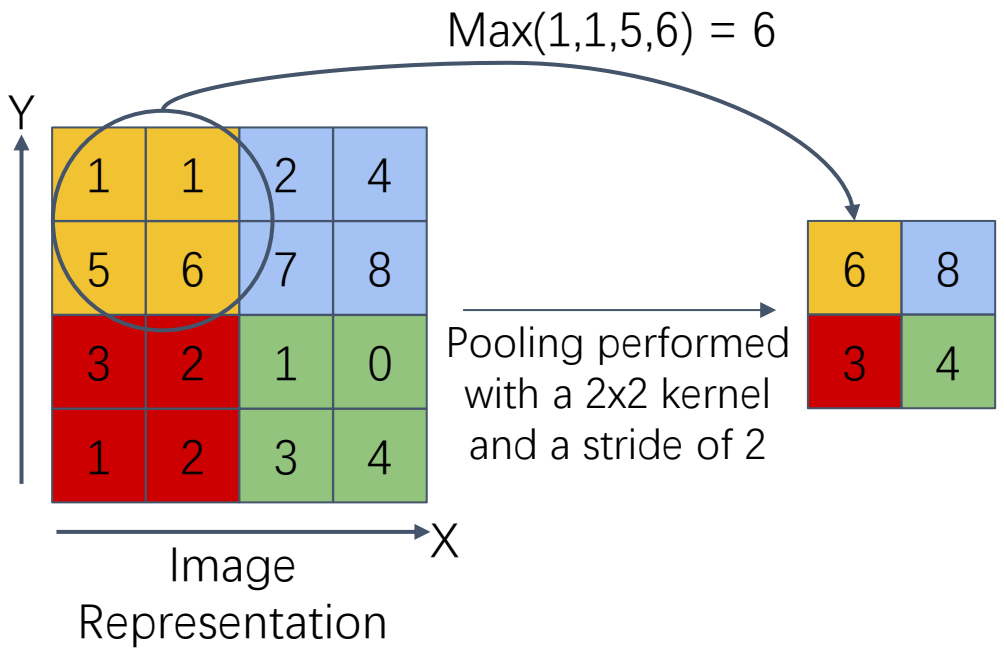
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



卷积神经网络 – 池化层

- 夹在在连续的卷积层中间，用于压缩数据(宽和高或者通道)
- 右图所示的是步长为2的（2，2）最大池化操作
- 还有平均池化操作（求平均值）、

通道池化（在通道维度上进行池化操作）

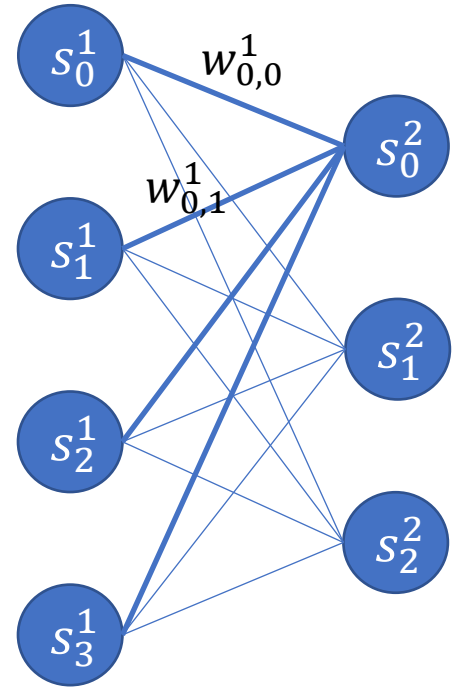


卷积神经网络 – 全连接层

- 全连接层中，输出的第i个神经元节点等于前一层神经元乘以对应的权重，求和再加上偏置。

- $s_0^2 = s_0^1 * w_{0,0}^1 + s_1^1 * w_{0,1}^1 + s_2^1 * w_{0,2}^1 + s_3^1 * w_{0,3}^1 + B_0^1;$

- 决定网络最终输出的类别数。



神经网络训练和测试

训练：

- 准备数据　　准备题库（题目+正确答案）
- 准备模型　　学生准备
- 循环一定的次数：
  - 遍历所有数据：　　从题库中抽取一定的题目和正确答案
    - （前向传播）将输入数据喂给模型，获得输出　　学生做题
    - 比较模型输出与真实标签，根据损失函数计算损失（距离）　　将做的结果与正确答案进行比较
    - （反向传播）根据损失更新模型　　让学生学习正确的思路

- 测试：　　让刷过题库的学生参加考试（未做过的题目），给出分数
  - 准备训练完成的模型，给定测试数据，获得输出，与真实标签进行比较

2 目标检测简介

- **目标检测（Object Detection）**是通过训练好的神经网络模型，从一张图像或视频帧中找出若干特定目标的方法。

目标检测结果

目标检测模型应用在视频监控中
监控该场景中的人

YoloV2

H × W × 3

Resize

608 × 608 × 3

Input

CBL
MaxPooling

304 × 304 × 32

CBL
MaxPooling

152 × 152 × 64

CBL * 3
MaxPooling

152 × 152 × 128

CBL * 3
MaxPooling

76 ×76 ×256

CBL * 5
MaxPooling

38 ×38 ×512

CBL * 5
MaxPooling

19 ×19 ×1024

CBL * 2

CBL
MaxPooling

38 ×38 ×64

Reorg

19 ×19 ×256

Concat

19 ×19 ×1280

CBL

19 ×19 ×1024

CBL

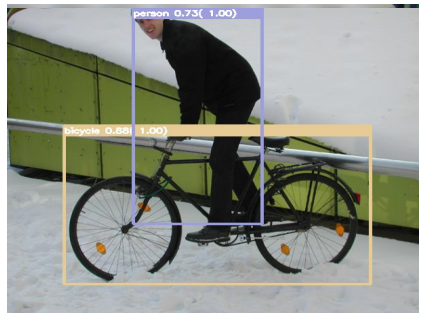
19 ×19 ×425

Detecion
Results

get_region_boxes

NMS

```
[[0.510 0.644 0.742 0.519 0.884  1.    ]
 [0.462 0.353 0.312 0.706 0.733 0.    ]]
```

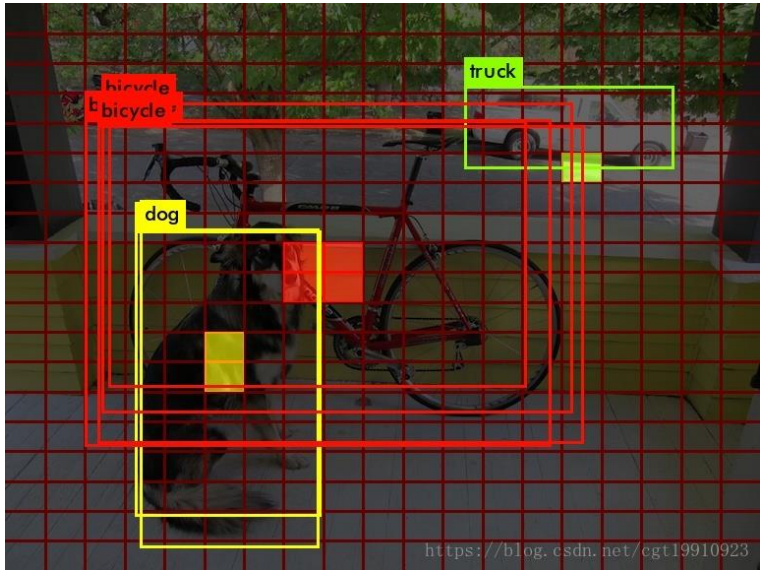


CBL

conv → Batch Normal → Leaky ReLU

YoloV2

- Yolov2的模型输出 19×19×425 的结果
  - 19×19代表整幅图像的不同区域
  - 每个区域输出5个形状不同的anchor box结果
  - 每个anchor box结果包含以下信息：
    - 位置信息(4)： x, y, w, h
    - 目标置信度(1)： $Object_{score}$
    - 分类信息(80)：属于哪个类别
  - 5*(4+1+80)=425

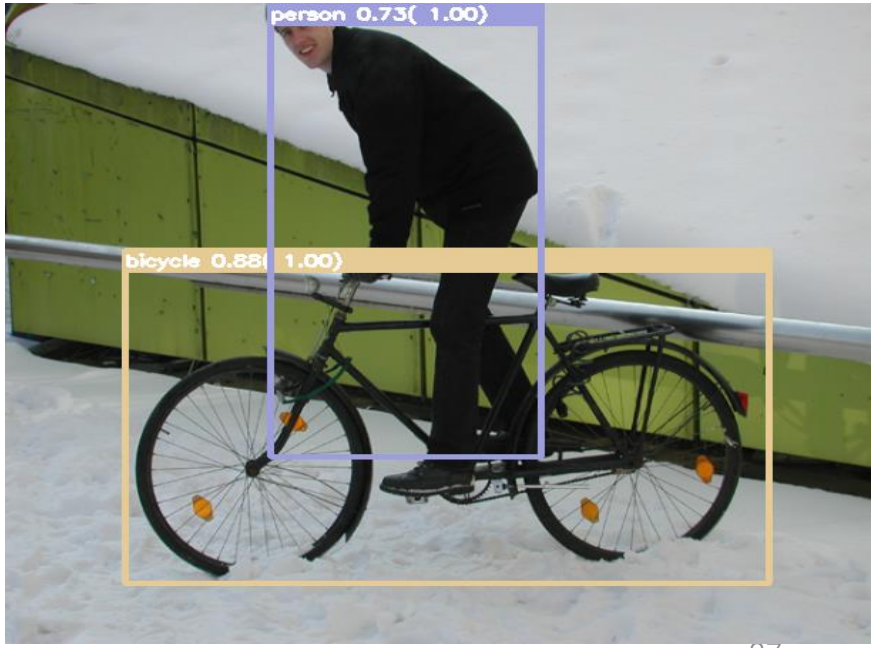


- 对模型结果进行筛选：
  - （1）get_region_boxes：当$Object_{score}$大于设定的阈值时，表示该物体被检测到
  - （2） NMS：经过非极大值抑制操作，移除重复的边框

YoloV2

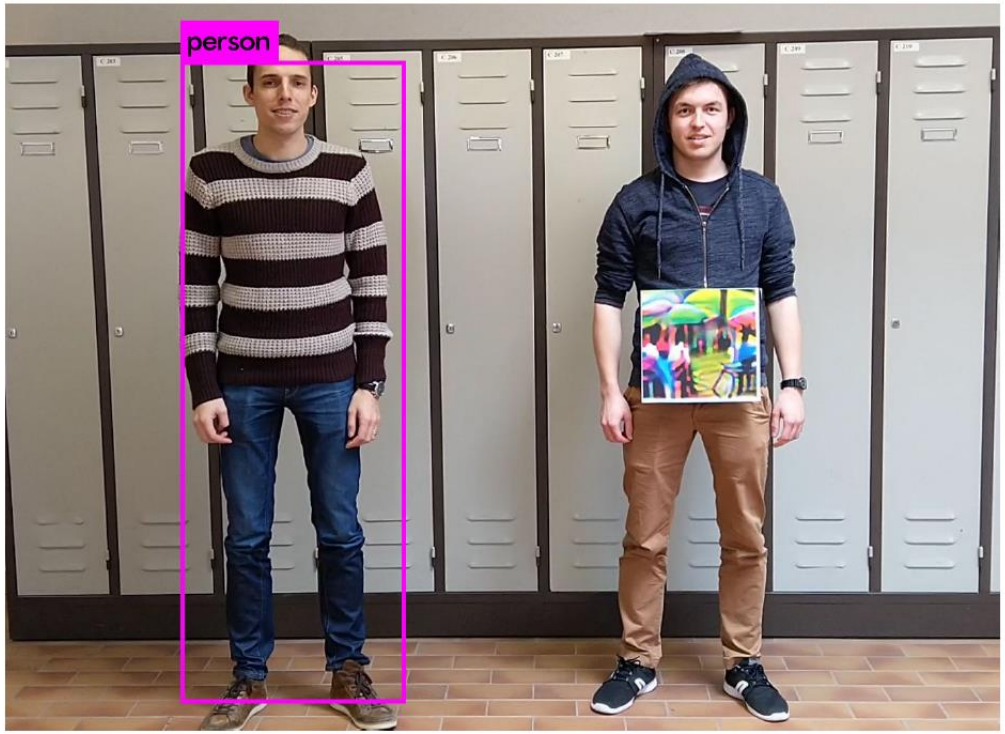
- 预测(x, y, w, h, $Object_{score}$, $Class$)
- x, y, w, h表示目标中心坐标以及宽高
- $Object_{score}$表示认为该目标存在的概率
- $Class$表示该目标的类别

```
[[0.510 0.644 0.742 0.519 0.884  1.       ]
 [0.462 0.353 0.312 0.706 0.733 0.       ]]
```



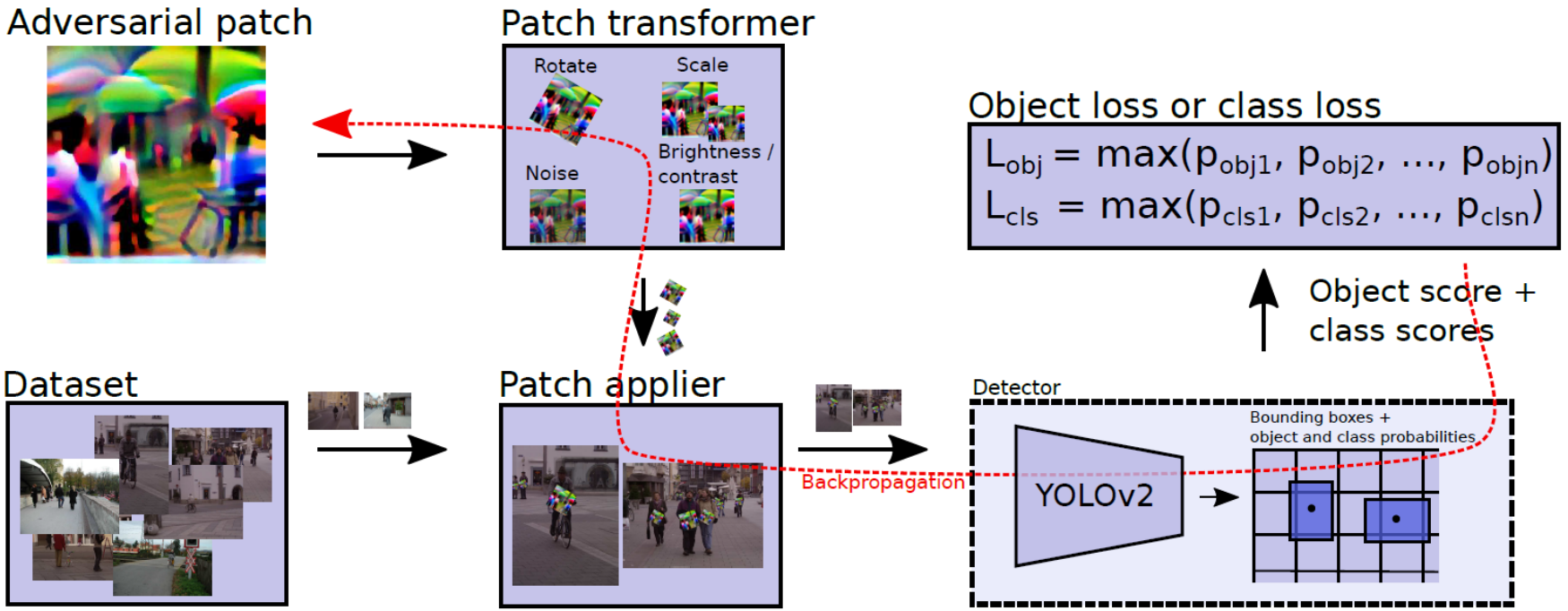
3 基于补丁的对抗攻击

- Fooling automated surveillance cameras: adversarial patches to attack person detection (CVPRW 2019)
- 目的：使得目标检测网络检测不到携带补丁的人



基于补丁的对抗攻击

- 1. 使用随机旋转、缩放、添加噪声、改变亮度和对比度四种图像处理手段对patch 进行处理，模拟真实情况，增强patch的泛化性
- 2. 将增强后的patch“贴”在图片的“人”（提前通过yolov2确定“人”的位置）中央，送入yolov2，根据损失函数计算损失更新patch



基于补丁的对抗攻击

- 损失函数：$L = \alpha L_{nps} + \beta L_{tv} + L_{obj}$
- （1）不可打印性分数 $L_{nps} = \sum_{p_{\text{patch}} \in p} \min_{c_{\text{print}} \in C} |p_{\text{patch}} - c_{\text{print}}|$
- p_{patch}是补丁p中的一个像素，c_{print}是一组可打印颜色C中的一种颜色。
- 使生成的Patch与打印机的可打印颜色非常接近。
- （2）图像全变化分数 $L_{tv} = \sum_{i,j} \sqrt{((p_{i,j} - p_{i+1,j})^2 + (p_{i,j} - p_{i,j+1})^2}$
- 如果相邻像素相似则得分较低，不同则得分较高。该损失可以让补丁的颜色较为平滑，防止补丁出现噪声。
- （3）最大目标置信度分数 $L_{obj} = max(p_{obj1}, \quad p_{obj2}, \dots, \quad p_{objn})$
- 最小化模型输出的目标“人”

基于补丁的对抗攻击

原检测结果　　贴上补丁后的检测结果

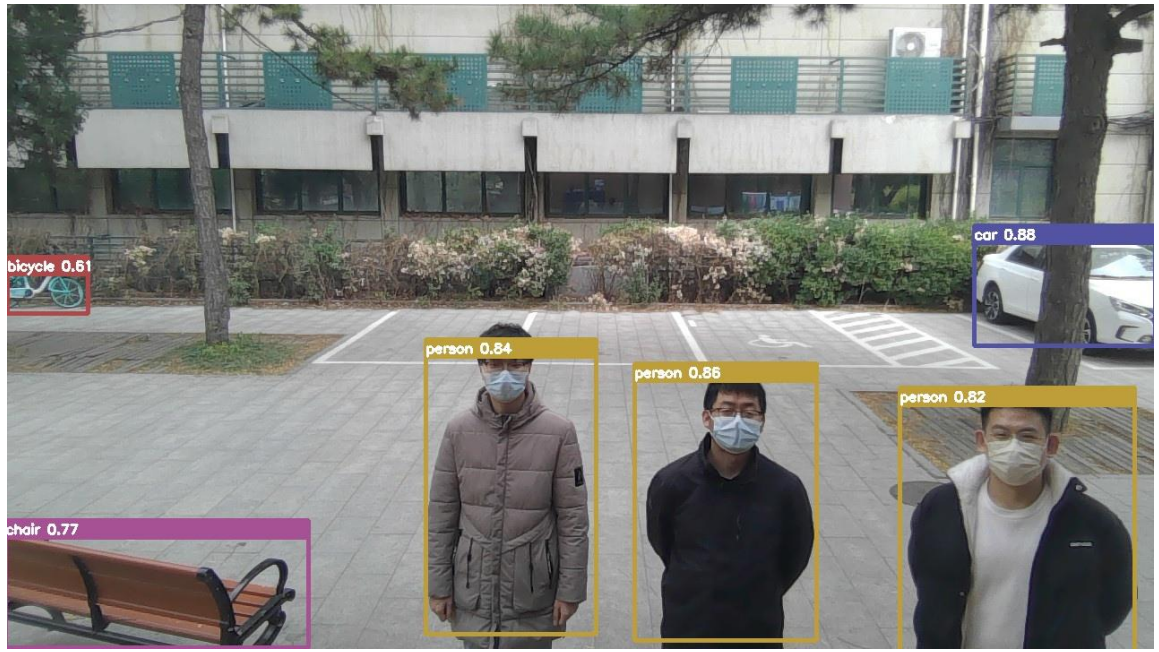
基于补丁的对抗攻击

原检测结果

贴上补丁后的检测结果

基于补丁的对抗攻击

- 在真实场景中进行“隐身”



基于补丁的对抗攻击

训练 100 epoch 得到的补丁

训练 500 epoch 得到的补丁

4 环境搭建 + 实验

- 实验使用的编程语言为Python，深度学习框架为PyTorch框架
- 配置环境：
  - 使用 Docker
  - 或者 安装Cuda+CuDNN+Anaconda

- 代码地址： https://gitlab.com/EAVISE/adversarial-yolo
- 下载后按照 README.md 进行操作

针对图像分类的对抗补丁攻击入门

苗长青

梯度

- 梯度 --> 求偏导数，带入点值
- $C(x,y)=\frac{3}{2}x^2+\frac{1}{2}y^2, X=(x,y)$
- $\nabla C(x=1,y=1)=[\frac{\partial C}{\partial x}(x=1),\frac{\partial C}{\partial y}(y=1)]$
- $=[(3x)(x=1),(y)(y=1)]$
- $=[3,1]$
- 意义：在当前条件下，输入的每个维度对输出的影响力。
- 我们就可以参考该影响力修改输入，从而达到我们的目标。

损失函数

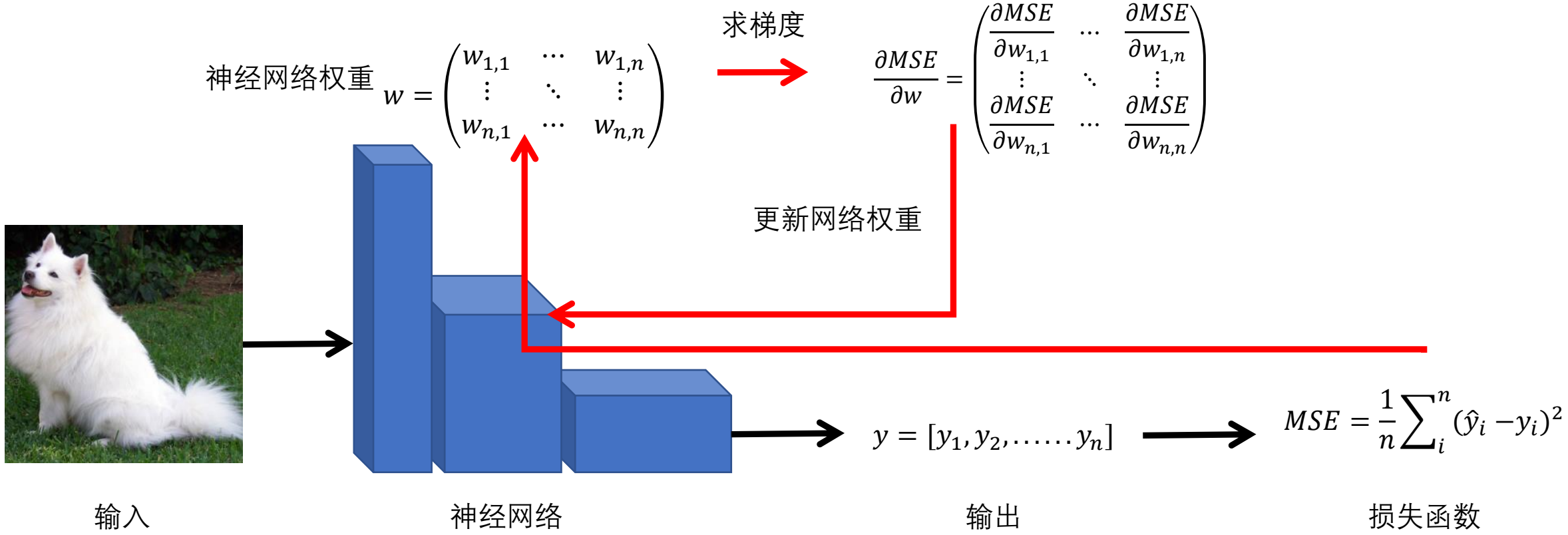
- 很多时候，模型的输出不能直接用来表达我们的目标。这时候就需要用到损失函数来表达我们的目标。
- 例如：均方误差

$$MSE = \frac{1}{n}\sum_{i}^{n}(\hat{y}_i - y_i)^2$$

- 其中，$\hat{y}_i$为我们想要得到的输出，y_i为我们实际得到的输出。
- 当MSE的值越大时，说明我们距离目标越远。

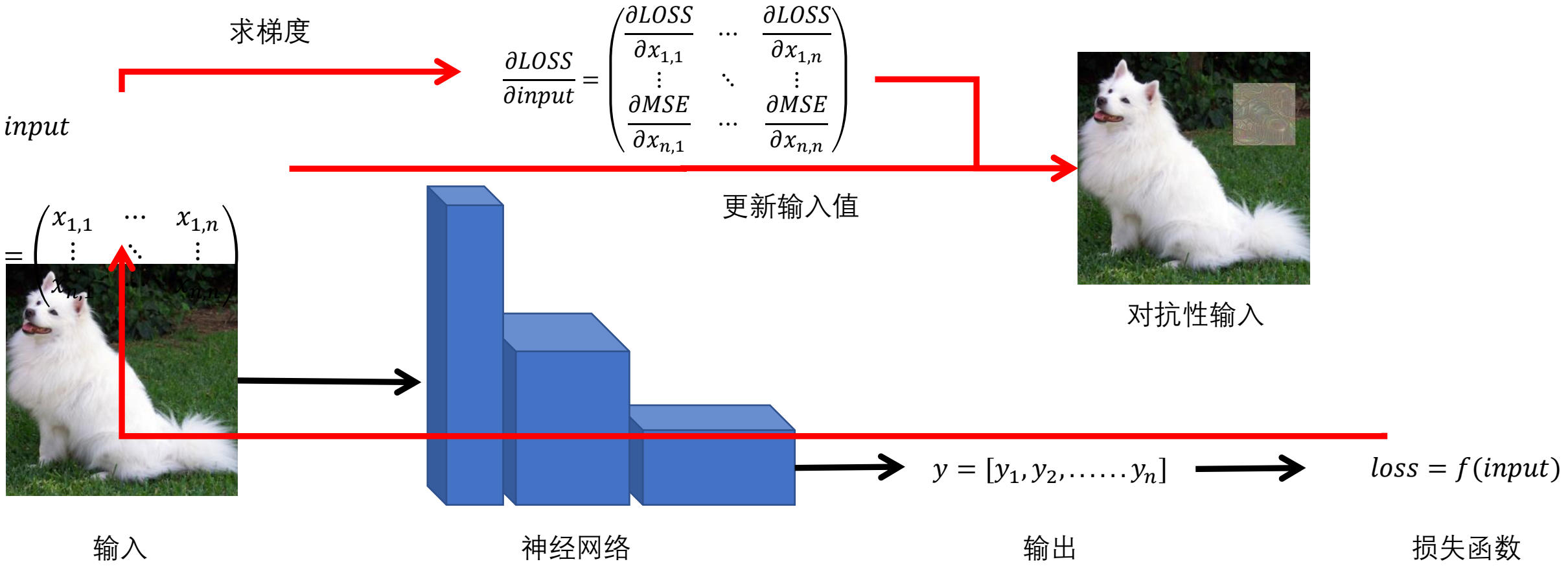
反向传播：模型训练

- 梯度和损失函数用于模型的训练：
- 计算损失函数关于训练目标的梯度，通过梯度指导训练目标的改变。



反向传播：模型训练

- 同理，我们可以求损失函数关于输入的梯度，并用其指导输入的改变。



对抗补丁攻击：损失函数设定

- 我们所要攻击的图像分类器的分类标签为输出向量中值最大维度对应的标签。
- 目标：让我们的目标类别对应的维度变大，同时使当前标签对应的值变小。
- 设定损失函数：
- 当target类不是得分最高的类时：

$$loss = y_{target} - max(y_{all})$$

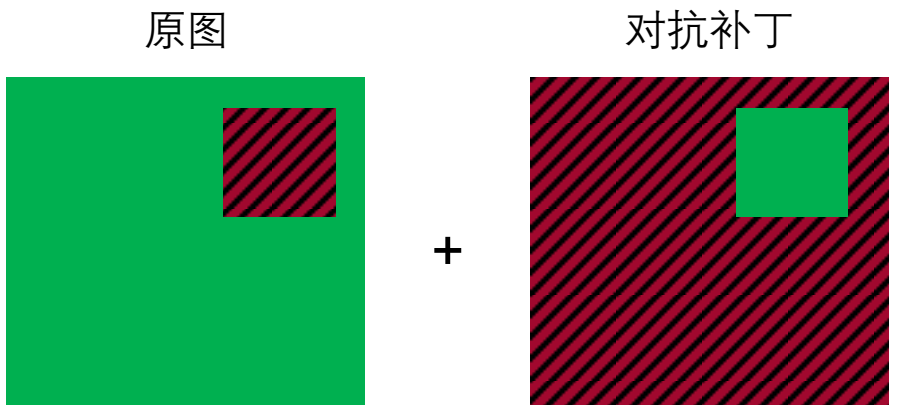
- 当target类是得分最高的类时：

$$loss = y_{target}$$

- loss越大越接近目标。

对抗补丁攻击：针对单张图片的攻击

- 初始化对抗补丁位置选择mask：与输入等大，放补丁的地方是1，其余是0。
- 初始化对抗补丁：全为0的向量，与输入等大。
- 初始化输入变量：原图中放补丁的地方放置随机数。
- 1. 将输入变量送入分类器，得到输出分类得分。
- 2. 使用分类得分计算loss。
- 3. 求得loss关于输入的梯度。
- 4. 使用梯度更新补丁：**新补丁=旧补丁+梯度*步长**。对补丁进行折叠（使得输入合法）
- 5. 放置补丁：**新输入变量 = 旧输入变量 *（1-mask）+ 补丁 * mask**
- 6. 判断是否达成攻击，若是，则输出，若否，则返回步骤1：
  - 强成功：目标类别分类概率在90%以上。
  - 弱成功：目标类别分类概率最大。



对抗补丁攻击：针对单张图片的攻击

85.76% 萨摩耶

92.44% 吸蜜鹦鹉

对抗补丁攻击：针对多张图片的攻击

- 不同点：在一个集合中随机挑选图片作为输入，随机选择位置作为放置补丁位置。

- 1. 将输入变量送入分类器，得到输出分类得分。
- 2. 使用分类得分计算loss。
- 3. 求得loss关于输入的梯度。
- 4. 使用梯度更新补丁：新补丁=旧补丁+梯度*步长。对补丁进行折叠（使得输入合法）
- **5. 随机挑选输入和补丁位置，使用新的补丁位置更新mask。**
- 6. 放置补丁：新输入变量 = **随机挑选输入** *（1-**mask**）+ 补丁 * **mask**
- 7. 判断是否达成攻击，若是，则输出，若否，则返回步骤1：
  - 强成功：目标类别分类概率在90%以上。
  - 弱成功：目标类别分类概率最大。

期待感兴趣的同学一同探讨交流！