### 一、Pytorch使用

#### 1. 生成指定形状张量

Torch.rand(3,3) 生成3\*3矩阵，矩阵元素服从[0,1)上均匀分布

Torch.randn(2,3,4) 生成2\*3\*4张量，张量元素服从标准正态分布

Torch.zeros(2,2,2) 生成2\*2\*2张量，张量元素全为0

Torch.ones(1,2,3) 生成1\*2\*3的张量，张量元素全为一

Torch.eye(3) 生成3\*3的单位矩阵

Torch.randint(0,10,(3,3)) 生成[0,10)之间的均匀分布整数的3\*3矩阵

Torch.rand\_like(t) 生成一个与张量t相同的均匀分布张量

Torch.randn\_like(t) 生成一个与张量t相同的正态分布张量

#### 2. 张量形状相关函数

t.ndimension() 获取该张量的元素数目

t.size() 获取张量每个维度的总元素数目，调用方法

t.shape() 获取张量每个维度的大小，访问属性

t.size(0) 获取张量维度0的大小，调用方法

t.view(3,4) 向量改变形状为3\*4的矩阵

#### 3. 张量函数运算

Torch.sqrt(t) 张量函数方法，不改变张量的值

t.sqrt() 张量内部方法，不改变张量的值

t.sqrt\_() 张量原地操作，会改变张量的值

torch.sum(t) 对张量所有元素求和

torch.sum(t,[0,1]) 对第0维和第1维的元素进行求和

t.mean() 求平均值

#### 4. 张量拼接和分割

Torch.stack 张量堆叠

Torch.cat 张量拼接

Torch.split 张量分割

Torch.chunk 张量分割

#### 5. 张量的扩增和压缩

t.unsqueeze() 扩增维度

t.squeeze() 压缩维度

### 二．Opencv使用

#### 1.创建和显示窗口

* namedWindow()
* imshow()
* destoryAllWindows()
* resizeWindow

#### 2. 加载图片

im = image

imread(path)

#### 3.保存图片

imwrite(name,img)

#### 4.视频采集

* VideoCapture() 0为摄像头设备，也可以直接输入视频路径
* cap.read():返回两个值，第一个为状态值，成功读到为TRUE，第二个值为视频帧
* cap.release()

#### 5.视频录制

* VideoWriter():

参数一为输出文件，参数二为多媒体文件格式，第三个参数是帧率，第四个参数是分辨率

* write()
* release()
* isOpened():判断摄像头是否已经打开

#### 6.设置鼠标回调函数

* setMouseCallback(winname, callback, userdata):  
   第一个参数是窗口名字，第二个是回调函数，第三个是给回调函数传参
* callback(event,x,y,flags,userdata):

第一个参数是事件（左键右键），第二三是鼠标坐标，第四个是组合键，第五个是上面函数传入的参数

#### 7. 设置Trackbar

* createTrackbar(trackbarname,winname,value,count,callback,userdata):

第一个参数是控件名字，第二个是窗口的名字,第三个是trackbar当前值,第四个设置最大值，第五个设置回调函数，第六个传入参数

* getTrackbarPos(trackbarname,winname):

第一个参数是控件名字，第二个参数是窗口名字，函数的输出结果是当前值

#### 8.Numpy创建矩阵

* array():创建矩阵
* zeros()/ ones():创建全0数组
* full():创建全值矩阵
* identify/eye():创建单元数组，eye()创建的是m\*n矩阵从i开始斜线为1

#### 9.图像属性

* .shape: shape属性包括高度长度和通道数
* .size:图像占用空间
* .dtype:图像每个元素的位深

#### 10.通道的分割与合并

* split(mat):分离为多个通道
* merge((ch1,ch2…)):把多个通道合并

#### 11. 绘制图形

* line(img,开始点，结束点，颜色，粗度，线型，缩放比例):画线
* ellipse(img, 中心点，(长轴,半轴)，倾角，开始角度，结束角度…)
* polylines(img, 点集, 是否闭环，颜色…)
* fillPoly(img,点集,颜色)
* putText(img, 字符串，起始点，字体，字号…)

#### 12. 图像算术操作

* add():返回一个矩阵
* substract(A,B): A-B
* multiply(A,B)
* divide(A,B)
* addWighted(A,alpha,B,beta,gamma):alpha和beta是权重，gamma是静态权重

#### 13. 图像位运算

* bitwise\_not()
* bitwise\_and(img1,img2)
* bitwise\_or(img1,img2)
* bitwise\_xor(img1,img2)

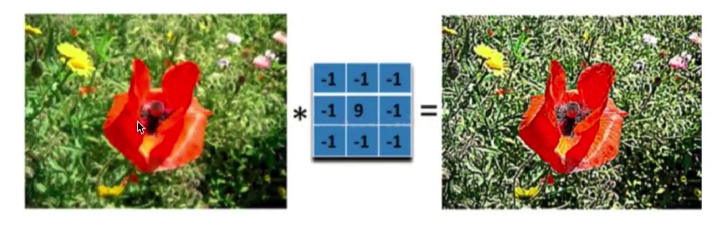
#### 14. 图像基本变换

* resize(src,dst,dsize,fx,fy,interpolation):fx是x轴的缩放因子，fy是y的缩放因子
  + 缩放算法：
    - INTER\_NEAREST,临近算法，速度快，效果差
    - INTER\_LINEAR,双线性插值，原图中的4个点
    - INTER\_CUBIC,三次插值，原图中的16个点
    - INTER\_AREA,效果最好
* Flip(img,flipCode):
  + flipCode==0,上下反转
  + flipCode<0,上下+左右
  + flipCode>0,左右
* rotate(img, rotateCode):
  + ROTATE\_90\_CLOCKWISE
  + ROTATE\_180
  + ROTATE\_90\_COUNTERCLOCKWISE
* warpAffine(src,M,size,flags,mode,value)
  + M:变换矩阵
  + Dsize:变换后的大小，输出尺寸
  + Flag:与resize中的插值算法一致
  + Mode:边界外推法标志
  + Value:填充边界的值
* getRotationMatrix2D(center,angle,scale):
  + center中心点
  + angle角度(倒角，按逆时针旋转)
  + scale缩放比例
* getAffineTransform(src[],dst[])
  + 通过三点确定变换的位置
* warpPerspective(img,M,dsize,…):透视变换
  + M变换矩阵
  + dsize变换大小
* getPerspectiveTransform(src[],dst[])
  + 指定区域的四个点（四个角）

### 三、 滤波器

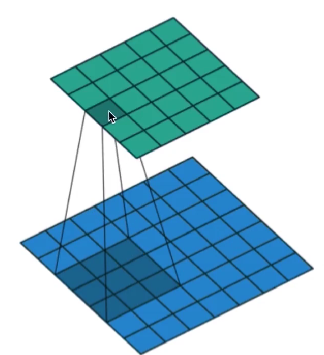
#### (1) 滤波的作用

一幅图像通过滤波器得到另一幅图像，其中滤波器又称为卷积核，滤波的过程称为卷积(本质是矩阵与小矩阵相乘得到)



#### (2) 卷积的过程

通常情况下，通过卷积处理得到的结果一定是比原来小的



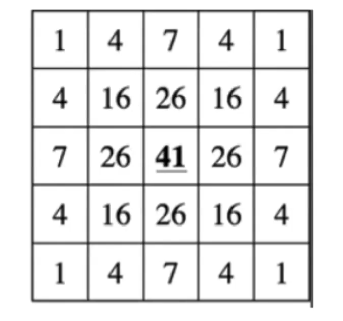
#### (3) 卷积的基本概念

a. 卷积核的大小：

1. 卷积核一般为奇数，如3\*3,5\*5,7\*7,原因是一方面增加padding,一方面保证锚点在中间，防止位置发生偏移

2. 卷积核越大，看到的信息（感受野越多），提取的特征越好，同时计算量越大

b. 锚点：防止信息的偏差



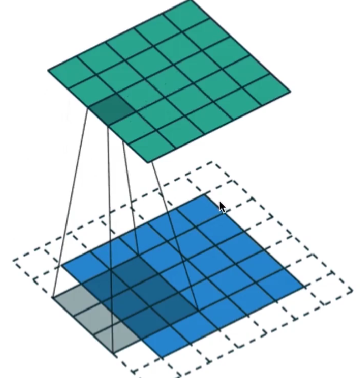
c. 边界扩充:

当卷积核大于1，且不进行边界扩充，输出尺寸将相应缩小；

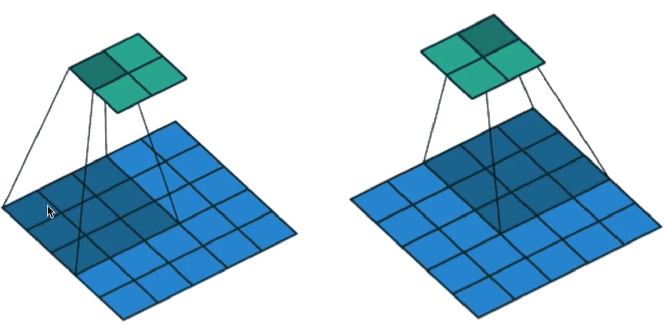
当卷积核以标准方式进行边界扩充，则输出数据的空间尺寸与输入数据相同

计算公式：N=(W-F+2P)/S+1

N输出图像大小，W原图大小，F卷积核大小，P扩充尺寸，S步长大小



d. 步长:



#### (4) 低通滤波与高通滤波

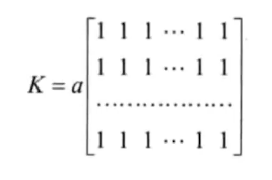
低通滤波可以去除噪音或平滑图像

高通滤波可以帮助查找图像的边缘

* filter2D(src,ddepth,kernel,anchor,delta,borderType)
  + kernal核
  + anchor锚点，可以不设
  + delta

#### (5) 方盒滤波与均值滤波

方盒滤波卷积核



参数a的作用：

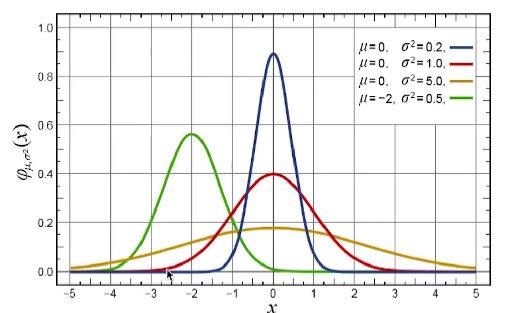
normalize = true, a = 1/w\*H

normalize = false, a =1

当normalize == true时，方盒滤波等于均值滤波

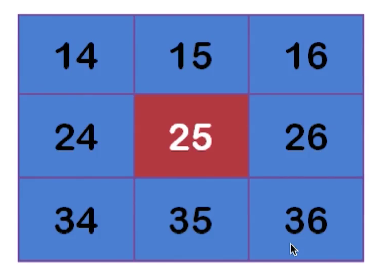
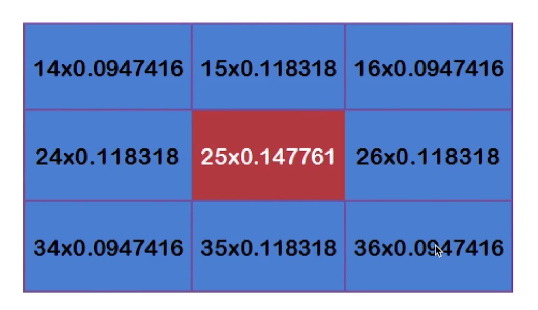
* boxFilter(src, ddepth,ksize,anchor,normalize,borderType)
* blur(src,ksize,anchor,borderType)

#### (6) 高斯滤波（中型滤波）



原理：

越接近卷积核中心，比重越高，但值不一定是最高的

* GaussianBlur(img,kernal,sigmaX,sigmaY,…)

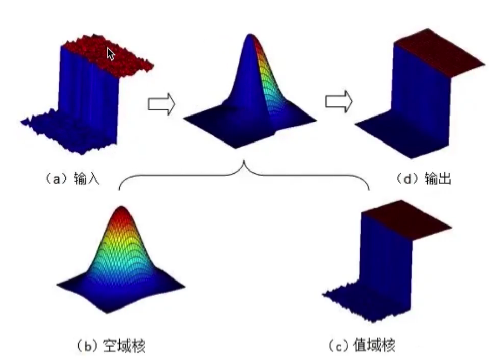
#### (7) 中值滤波

取数组中值作为卷积后的结果值

* medianBlur(img,ksize)
  + ksize:数组核大小

#### (8) 双边滤波

主要功能是进行美颜



* bilateralFilter(img, d,sigmaColor,sigmaSpace,...)
  + d:核的大小
  + Color:int
  + Space:int

#### (9) 常见的高通滤波

1. Sobel索贝尔算子(对噪音不敏感，内部使用高斯噪音，对噪音进行过滤，有强抗噪性)

* + 先对x方向求导
  + 然后在y方向求导
  + 最终结果: |G| = |Gx|+|Gy|
  + Sobel(src,ddepth,dx,dy,ksize=3)

2. Scharr沙尔算子(比起Sobel可以检测更细的噪音，但是不能像Sobel一样更改卷积核大小)

* Scharr(src,ddepth,dx,dy,scale=1,delta=0,borderType)

Tip：Sobel和Scharr都是对x求，再对y求，所以还需要加合

3. Laplacian拉普拉斯算子(对噪音敏感，内部没有降噪，需要提前降噪)

可以同时求两个方向的边缘，对噪音敏感，一般先去噪在使用拉普拉斯算子

#### (10) Canny边缘检测

1. 使用5\*5高斯滤波消除噪声

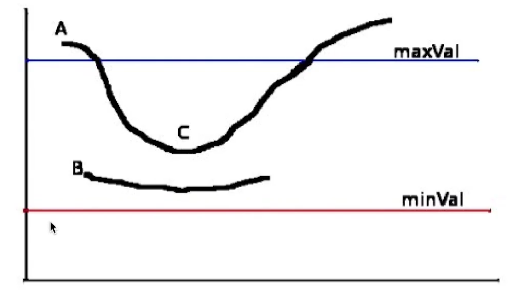
2. 计算图像梯度的方向（0,35,90,135）

3. 取局部最大值

4. 阈值计算

* Canny(img,minVal,maxVal,...)

原理：大于阈值，一定是边缘，位于之间，通过计算核实是否是边缘



### 四、形态学

#### 1. 二值化

将图像的每个像素变成两个值，如0,255

全局二值化

局部二值化

* threshold(img,thresh,maxVal,type)
  + img图像，最好是灰度图
  + thresh阈值
  + maxVal:超过阈值，替换成maxVal
  + threshodType:
    - THRESH\_BINARY THRESH\_BINARY\_INY
    - THRESH\_TRUNC
    - THRESH\_TOZERO THRESH\_TOZERO\_INV

#### 2. 自适应阈值

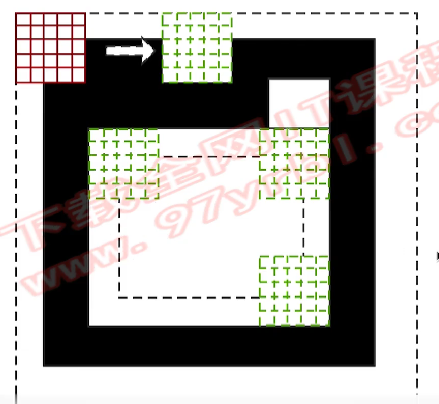
由于光照不均匀以及阴影的存在，只有一个阈值会在阴影处生成白色

* adaptiveThreshold(img,maxVal,adaptiveMethodtype,blockSize,C)
  + adaptiveMethod:计算阈值的方法
    - ADAPTICE\_THRESH\_MEAN\_C:计算临近区域平均值
    - ADAPTIVE\_THRESH\_GAUSSIAN\_C:计算高斯窗口加权平均值
  + blockSize:对指定大小的区域单独进行阈值计算
  + C：常量

#### 3.腐蚀

* erode(img,kernel,iterations=1)
  + kernel,核大小越大，腐蚀效果越好

原理：



效果：



#### 4. 获得卷积核

* getStructuringElement(type,size)
  + type:
    - MORPH\_RECT
    - MORPH\_ELLIPSE
    - MORPH\_CROSS
  + size:一般设为(3,3),(5,5),(7,7)

#### 5. 膨胀

* dilate(img,kernal,iterations=1)

#### 6. 开操作

* morphologyEx(img,MORPH\_OPEN,kernel)

先腐蚀再膨胀，可以把字符外面的噪点去除

#### 7. 闭运算

* morphologyEx(img,MORPH\_CLOSE,kernel)

先膨胀再腐蚀，可以把字符里面的噪点去除

#### 8. 形态学梯度

梯度 = 原图 – 腐蚀

* morphologyEx(img,MORPH\_GRADIENT,kernel)

#### 9. 顶帽运算

顶帽 = 原图 – 开运算（求大区域外的噪点）

* morphologyEx(img,MORPH\_TOPHAT,kernel)

#### 10. 黑帽运算

黑帽 = 原图 – 闭运算 （求大区域里面的小噪点）

* morphologyEx(img,MORPH\_BLACKHAT,kernel)

#### 11. 轮廓查找

* findContours(img, mode, ApproximationMode…) 返回值contours和hierarchy
  + mode
    - mode=0(RETR\_EXTERNAL=0) 只检测外轮廓
    - mode=1(RETR\_LIST=0) 检测的轮廓不建立等级关系
    - mode=2(RETR\_CCOMP=0) 每层最多两级
    - mode=3(RETR\_TREE=0) 按树形存储轮廓
  + approximation
    - CHAIN\_APPROX\_NONE 保存轮廓上的点
    - CHAIN\_APPROX\_SIMPLE 只保存角点

#### 12. 轮廓绘制

drawContours(img, countours, contourIdx, color, thickness)

#### 13. 轮廓计算

* contourArea(contour) 面积计算
* arclength(contour) 周长计算

#### 14. 多边形逼近与凸包



* approxPolyDP(curve, epsilon, closed)
  + curve:轮廓
  + epsilon:精度
  + closed:闭合
* convexHull(points)

#### 15. 外接矩形

最小外接矩阵

* minAreaRect(points) 返回值RotatedRect

最大外接矩阵

* boundingRect (array) 返回值Rect



#### 16. 去除背景

* createBackgorundSubtractorMOG()
  + bgsub = createBackgroundSubtractorMOG()
  + bgsub.apply(去除背景的对象) 返回值为去除后的对象
  + 使用高斯去噪的时候，一般在去背景前去噪，去噪前先进行图像灰度化

### 五、 特征点检测与匹配

#### 1. Harris角点检测

* cornerHarris(img, dst, blockSize, ksize, k)
  + blockSize: 检测窗口大小
  + ksize: Sobel的卷积核
  + k: 权重系数，经验值，一般取0.02~0.04之间

#### 2. Shi-Tomasi角点检测

* goodFeaturesToTrack(img, maxCorners,…)
  + maxCorners: 角点的最大数，值为0表示无限制
  + quailtyLevel: 小于1.0的正数，一般在0.01~0.1之间
  + minDistance: 角之间最小欧式距离，忽略小于此距离的点
  + mask: 感兴趣的区域

#### 3. SIFT（与缩放无关的特征转换）

步骤

创建SIFT对象

进行检测，kp= sift.detect(img, …)

绘制关键点, drawKeypoints(gray, kp, img)

#### 4. SIFT关键点和描述子

关键点: 位置，大小和方向

关键点描述子：记录了关键点周围对其有贡献的像素点的一组向量值，其不受仿射变换、光照变换等影响

* kp, des = sift.compute(img, kp) 计算描述子

计算描述子作用是进行特征匹配

* kp, des = sift.detectAndCompute(img, ...)
  + mask: 指明对img中哪个区域进行计算

#### 5. SURF（加速鲁棒性检测）

* surf = cv2.xfeatures2d.SURF\_create()
* kp, def = surf.detectAndCompute(img, mask)

#### 6. ORB() oriented and rorated BRIEF

可以实时监测

* orb = cv2.ORB\_create()
* kp, des = orb.detectAndCompute(img, mask)

#### 7. 暴力特征匹配

用第一组中的每个特征的描述子与第二组的所有特征描述子进行匹配，计算差距，将最接近的返回

* BFMatcher(normType, crossCheck) 创建匹配器
  + normType:NORM\_L1(SIFT), NORM\_L2(SURF), HAMMING1(ORB)
  + crossCheck:是否进行交叉匹配，默认为false
* bf.match(des1, des2) 进行特征匹配,返回匹配结果
* cv2.drawMatches(img1, kp1, img2, k2) 绘制匹配点

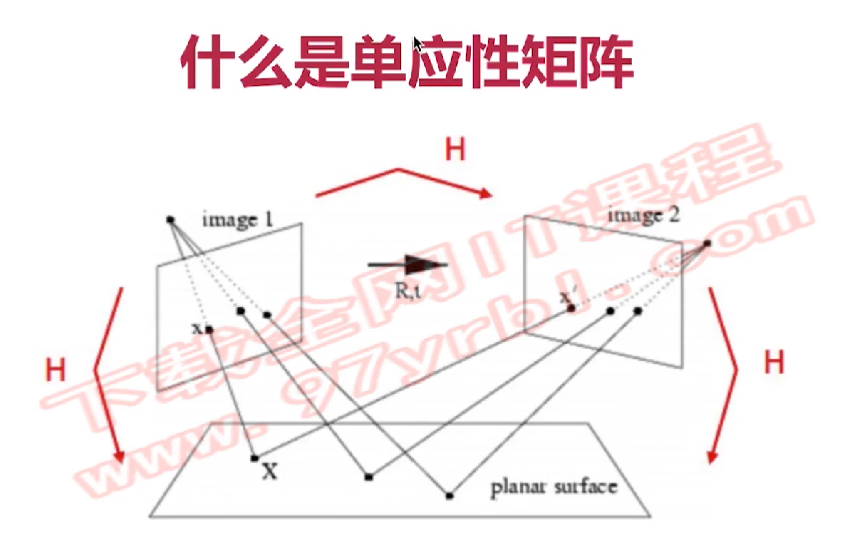
#### 8. FLANN特征匹配

批量特征匹配时速度更快，使用的是临近近似值，精确度较差

* flannBasedMatcher(…)
  + index\_params: KDTREE(SIFT,SURF),LSH（ORB）匹配算法,不对应会报错
    - index\_params=dict(algorithm=FLANN\_INDEX\_KDTREE,tree=5)
  + search\_params(指定KDTREE算法中的遍历树次数，经验值为5，次数为50)
    - search\_params=dict(checks=50)
* flann.match/knnMatch()
  + knnMatch参数为SIFT/SURF/ORB等计算的描述子，k表示取欧氏距离最近的前k个关键点，返回结果是DMatch对象
    - DMatch内容
      * distance: 描述子之间的距离，值越低越好
      * queryIdx, 第一个图像的描述子索引值
      * trainIdx, 第二个图的描述子索引值
      * imgIdx, 第二个图的索引值
* cv2.drawMatch
* drawMarchesKnn
  + 搜索img, kp
  + 匹配图img, kp
  + Match方法返回的匹配结果

9. 单应性矩阵

单应性矩阵，又称作Homography Matrix，是计算机视觉中重要的矩阵之一。它是一个3x3的矩阵，用于描述两个平面之间的基本变换关系，如旋转、平移、缩放等。



单应性矩阵的应用：图像转正，图像替换

### 六、下游任务

#### 1. 图像的分割与修复

1) 分水岭法

* **处理步骤**
  + **标记背景**
  + **标记前景**
  + **标记未知域**
  + **进行分割**
* **watershed**

2) GrabCut 通过交互的方式获得前景物体

3) meanshift

* pyrMeanShiftFiltering(img, sp, sr, maxLevel, termcrit=TermCriteria…)
  + sp:半径
  + sr:色彩幅值

4) MOG去背景

* createBackgroundSubtractorMOG(history=200,nmixtures=5,backgroundRatio=0.7,noiseSigma=0)
  + history参考帧的时长
  + nmixture高斯范围值
  + backgroundRatio背景比率
  + noiseSigma=0自动降噪
* createBackgroundSubtractorMOG2(…):对亮度产生的阴影有更好的识别

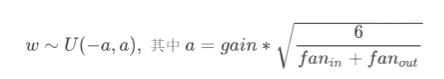
5) 图像修复

* inpaint(img, mask, inpaintRadius,flags)
  + inpaintRadius:每个点的圆形邻域半径
  + flags: INPAINT\_NS, INPAINT\_TELEA

### 七、深度学习基础

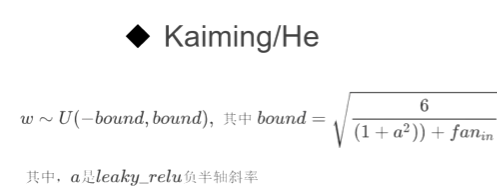
#### 1. 初始化模型参数

**Xavier**



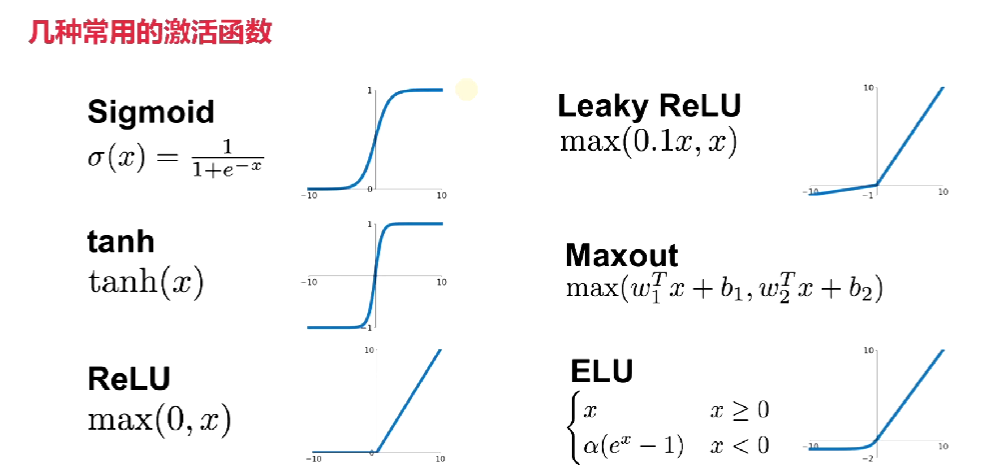
* nn.init.xavier\_uniform\_(model.weight, gain = nn.init.calculate\_gain(‘tanh’))

Kaiming

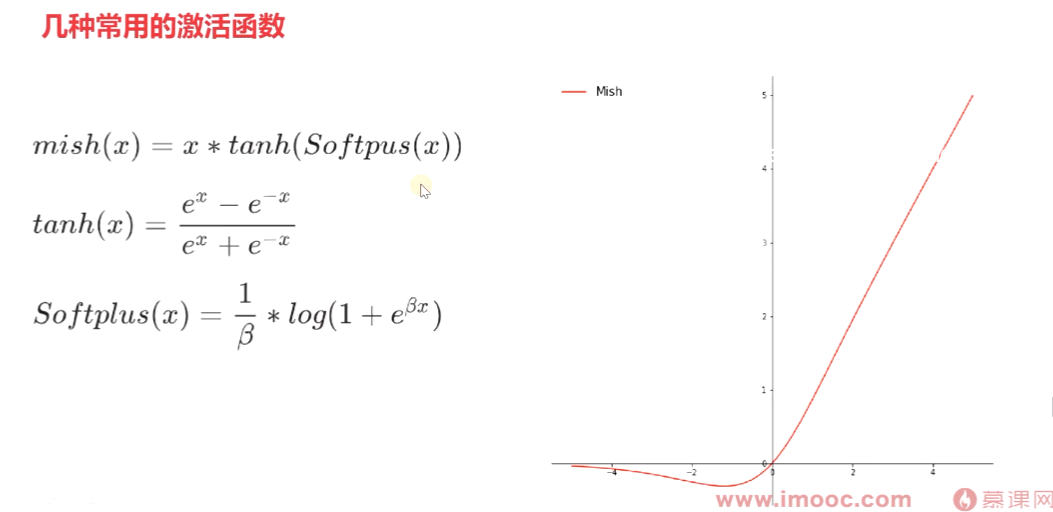


* nn.init.kaiming\_uniform\_(model.weight, a=1, mode=”fan\_in”, nonlinearity=”leaky\_relu”)

#### 2. 激活函数



当不知道怎么选的时候，选mish(x)激活函数



* import torch.nn.functional as F
  + F.sigmoid()
  + F.relu()
  + F.leaky\_relu()
  + F.mish()

3. 优化器

* optimizer.zero\_grad()

清空模型参数的梯度，以确保每次迭代的梯度计算都是基于当前小批量数据的，而不会受之前迭代的影响。这是为了避免在优化过程中梯度的不正确累积。

* optimizer.step()

根据优化器的配置和计算得到的梯度来更新模型参数。具体地，它执行了参数更新的操作，通常是将当前参数减去学习率乘以参数的梯度。学习率是一个超参数，用于控制参数更新的步长大小。通过不断地执行 optimizer.step()，模型的参数将逐渐调整，以减小损失函数。

* loss.backward()

根据当前的损失值（通常是一个标量）计算模型参数的梯度。它会从损失开始，然后通过模型的各个层反向传播梯度，最终计算出每个参数相对于损失的梯度。这些梯度将被用于后续的参数更新，以减小损失。