数据标注常用工具

2021-08-11

funykatebird

Revised 2021-08-13

funykatebird

Revised 2021-08-20

Funykatebird

Revised 2025-03-30

funykatebird

目录

[一、 标注工具 3](#_Toc80375399)

[1.1 Labelme介绍与安装 3](#_Toc80375400)

[1) Labelme 是什么？ 3](#_Toc80375401)

[2) Labelme 安装 3](#_Toc80375402)

[3) Labelme 使用教程 5](#_Toc80375403)

[4) 标注类型 5](#_Toc80375404)

[1.2 LabelImg介绍与安装 6](#_Toc80375405)

[1) Label介绍 6](#_Toc80375406)

[2) LabelImg安装 7](#_Toc80375407)

[3) LabelImg的使用 8](#_Toc80375408)

[4) 常见问题 12](#_Toc80375409)

[1.3 VOC数据标签和格式和YOLO数据标签格式说明 12](#_Toc80375410)

[1) VOC数据格式 12](#_Toc80375411)

[2) YOLO数据格式 14](#_Toc80375412)

[二、 数据集格式 15](#_Toc80375413)

[2.1 Voc格式 15](#_Toc80375414)

[2.2 Coco格式 15](#_Toc80375415)

[1) Object Instance 类型的标注格式 17](#_Toc80375416)

[2) Object Keypoint 类型的标注格式 19](#_Toc80375417)

[3) Image Caption的标注格式 21](#_Toc80375418)

[2.3 Labelme格式 22](#_Toc80375419)

[2.4 格式转换 24](#_Toc80375420)

[1) voc转换为yolo格式计算 24](#_Toc80375421)

[2) yolo转换为voc格式计算 26](#_Toc80375422)

[三、 深度学基本概念 27](#_Toc80375423)

[padding 27](#_Toc80375424)

[卷积步长 (Stride) 31](#_Toc80375425)

[四、 参考资料 32](#_Toc80375426)

# 标注工具

## Labelme介绍与安装

### Labelme 是什么？

Labelme 是一个图形界面的图像标注软件。其的设计灵感来自于 <http://labelme.csail.mit.edu/> 。它是用 Python 语言编写的，图形界面使用的是 Qt（PyQt）。

Labelme 能干啥？

对图像进行多边形，矩形，圆形，多段线，线段，点形式的标注（可用于目标检测，图像分割，等任务）。对图像进行进行 flag 形式的标注（可用于图像分类 和 清理 任务）。视频标注生成 VOC 格式的数据集（for semantic / instance segmentation）生成 COCO 格式的数据集（for instance segmentation）

<https://github.com/wkentaro/labelme/tree/master/examples>

### Labelme 安装

Ubuntu / macOS / Windows

Python2 / Python3

PyQt4 / PyQt5 / PySide2

Labelme 安装方法大体可分为两大类：

通用安装方法（各平台都适用）：Anaconda，Docker。

各平台上的安装方法：Ubuntu，macOS，Windows。

Anaconda

首先安装 Anaconda，然后运行下列命令：

##################

## for Python 2 ##

##################

conda create --name=labelme python=2.7

source activate labelme

# conda install -c conda-forge pyside2

conda install pyqt

pip install labelme

# 如果想安装最新版本，请使用下列命令安装：

# pip install git+https://github.com/wkentaro/labelme.git

##################

## for Python 3 ##

##################

conda create --name=labelme python=3.6

source activate labelme

# conda install -c conda-forge pyside2

# conda install pyqt

pip install pyqt5 # pyqt5 can be installed via pip on python3

pip install labelme

Docker

首先安装 Docker，然后运行下列命令：

wget https://raw.githubusercontent.com/wkentaro/labelme/master/labelme/cli/on\_docker.py -O labelme\_on\_docker

chmod u+x labelme\_on\_docker

# Maybe you need http://sourabhbajaj.com/blog/2017/02/07/gui-applications-docker-mac/ on macOS

./labelme\_on\_docker examples/tutorial/apc2016\_obj3.jpg -O examples/tutorial/apc2016\_obj3.json

./labelme\_on\_docker examples/semantic\_segmentation/data\_annotated

Ubuntu

# Ubuntu 14.04 / Ubuntu 16.04

# Python2

# sudo apt-get install python-qt4 # PyQt4

sudo apt-get install python-pyqt5 # PyQt5

sudo pip install labelme

# Python3

sudo apt-get install python3-pyqt5 # PyQt5

sudo pip3 install labelme

Windows

首先按照4.1的操作安装，然后进行如下操作：

# Windows 上的 Pillow5 会导致 dll 加载错误，所以请安装 Pillow4。

# 详情见：https://github.com/wkentaro/labelme/pull/174

conda install pillow=4.0.0

### Labelme 使用教程

Labelme 能够进行多种形式的图像数据标注。Labelme 以 JSON 文件存储标注信息。下面介绍一些 labelme 软件的基本操作。

labelme # 打开labelme软件

labelme apc2016\_obj3.jpg # 指定图像文件

labelme apc2016\_obj3.jpg -O apc2016\_obj3.json # 保存后关闭labelme

labelme apc2016\_obj3.jpg --nodata # JSON文件不包含图像数据，而包含图像的相对路径

labelme apc2016\_obj3.jpg \

--labels highland\_6539\_self\_stick\_notes,mead\_index\_cards,kong\_air\_dog\_squeakair\_tennis\_ball # 指定 label list

labelme data\_annotated/ # 指定图像文件夹

labelme data\_annotated/ --labels labels.txt # 使用文件指定 label list

labelme 常用的命令行参数：

--flags： comma separated list of flags 或者 file containing flags

--labels：comma separated list of labels 或者 file containing labels

--nodata：stop storing image data to JSON file

--nosortlabels：stop sorting labels

--output：指定输出文件夹

### 标注类型

分类标注

使用 labelme 进行图像分类标注的教程详见

目标检测标注

使用 labelme 进行目标检测标注的教程详见

场景分割标注

使用 labelme 进行场景分割标注的教程详见

实例分割标注

使用 labelme 进行实例分割标注的教程详见

视频标注

使用 labelme 进行视频标注的教程详见：

<https://github.com/wkentaro/labelme/tree/master/examples>

其它形式的标注

Labelme 除了能进行上面形式的标注，还能进行下面形式的标注：

多边形

矩形

圆形

多段线

线段

点

命令行工具

1. labelme\_draw\_json：

使用该命令可以快速查看JSON格式的标注。

2. labelme\_json\_to\_dataset：

使用该命令可以将JSON文件转为一组图像和标签文本文件。

3. labelme\_draw\_label\_png：

将label文本文件以图例的形式绘制到PNG格式的标签上，并显示出来。

## LabelImg介绍与安装

### Label介绍

github是目标检测数据标注工具，可以标注标注两种格式：

VOC标签格式，标注的标签存储在xml文件

YOLO标签格式，标注的标签存储在txt文件中

LabelImg的github主页地址：

<https://github.com/tzutalin/labelImg>

### LabelImg安装

使用pip工具进行安装LabelImg （推荐）

使用pip安装LabelImg安装时最简单的方式，首推，安装完之后会自动把LabelImg添加到环境变量中，这样你就可以在命令行中输入：labelimg直接打开该工具——windows用户强烈推荐此方法

使用pip安装labelimg

pip install labelimg -i https://pypi.tuna.tsinghua.edu.cn/simple

从源码开始安装LabelImg，一般比较麻烦，不推荐

**在Ubuntu Linux上安装LabelImg**

1、Python 2 + Qt4

Python2的环境使用如下命令安装

sudo apt-get install pyqt4-dev-tools

sudo pip install lxml

make qt4py2

python labelImg.py

python labelImg.py [IMAGE\_PATH] [PRE-DEFINED CLASS FILE]

2、Python 3 + Qt5 (Recommended)

Python3的环境使用如下命令安装

sudo apt-get install pyqt5-dev-tools

sudo pip3 install -r requirements/requirements-linux-python3.txt

make qt5py3

python3 labelImg.py

python3 labelImg.py [IMAGE\_PATH] [PRE-DEFINED CLASS FILE]

**在macOS上安装LabelImg**

1、Python 2 + Qt4

Python2的环境使用如下命令安装

brew install qt qt4

brew install libxml2

make qt4py2

python labelImg.py

python labelImg.py [IMAGE\_PATH] [PRE-DEFINED CLASS FILE]

2、Python 3 + Qt5 (Recommended)

Python3的环境使用如下命令安装

brew install qt # Install qt-5.x.x by Homebrew

brew install libxml2

or using pip

pip3 install pyqt5 lxml # Install qt and lxml by pip

make qt5py3

python3 labelImg.py

python3 labelImg.py [IMAGE\_PATH] [PRE-DEFINED CLASS FILE]

**在Virtualenv虚拟环境中安装labelimg**

使用virtualenv创建虚拟环境可以避免python版本和QT版本带来的一些问题，当然你也可以用conda创建虚拟环境，道理是一样的

brew install python3

pip3 install pipenv

pipenv run pip install pyqt5==5.13.2 lxml

pipenv run make qt5py3

python3 labelImg.py

[Optional] rm -rf build dist; python setup.py py2app -A;mv "dist/labelImg.app" /Applications

**其他安装方法**

作者还给出了很多的其他安装方法，如docker安装方法等，大家可以去作者的github主页进行参考，这里我就不一一列举了！

### LabelImg的使用

安装好之后，就是在命令行中打开LabeImg，在命令行中输入（因为安装完之后会自动添加到）：

LabelImg 或 labelimg

打开Labelimg

我们在要标定数据的时候，一般是已经制定好了要标注物体的类别，这样在打开LabelImg之后，我们只要框定目标之后，选择要对应的标签即可。

labelimg JPEGImage predefined\_classes.txt

说明：

JPEGImage 目录：存储的是图片的名字

Annotation 目录：用于存放标注图片的标签文件

predefined\_classes.txt 文件：定义自己要标注的所有类别

例如：我的predefined\_classes.txt中定义的类别如下：

circle\_red

circle\_gray

rectangle\_red

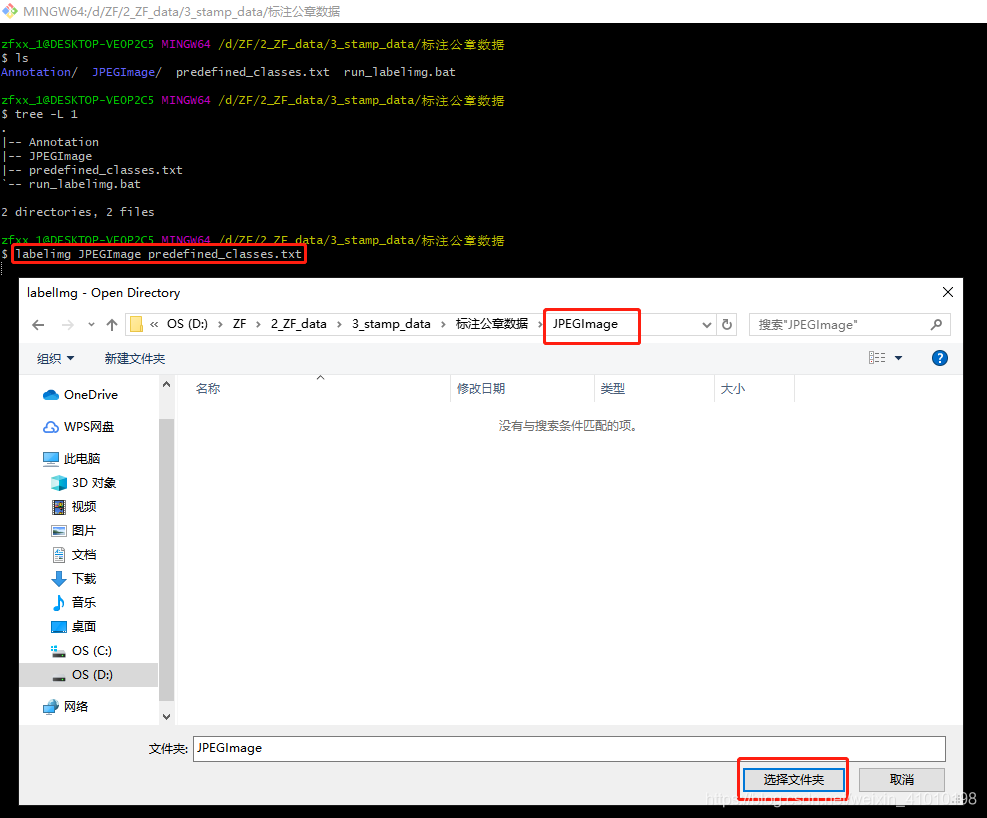
rectangle\_gray

fingeprint\_red

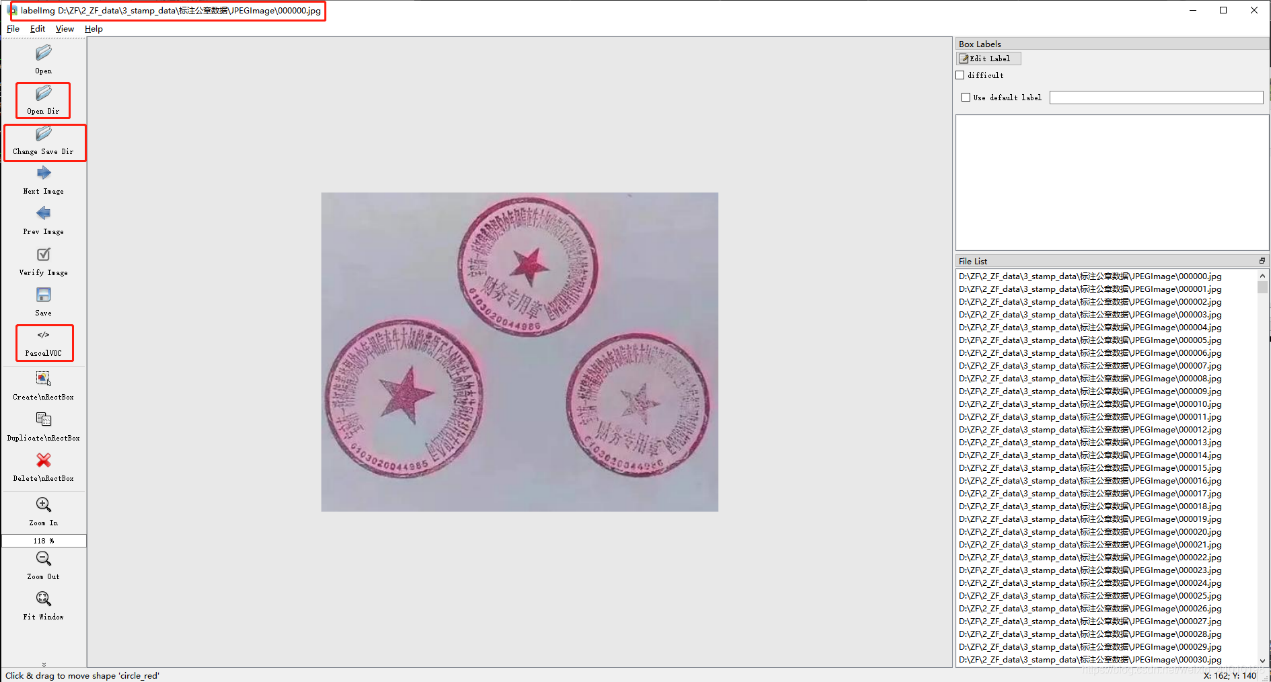
fingeprint\_gray

other

执行命令，然后选择JPEGImage待标注图片数据文件夹，然后点选择文件夹，如下图：



选择好标注数据文件夹之后，即进入到了LabelImg的界面，如下图：



最上方显示的是当前标注图片的路径

Open Dir：待标注图片数据的路径文件夹，即选择JPEGImage文件夹

Change Save Dir：保存类别标签的路径文件夹，即选择Annotation文件夹

PascalVOC：标注的标签保存成VOC格式，在鼠标点一下就变成YOLO，即此时就会把标注的标签变成YOLO格式

标注前先进行一些设置，点击View显示如下图，然后把如下的几个选项勾上：

Auto Save mode：当你切换到下一张图片时，就会自动把上一张标注的图片标签自动保存下来，这样就不用每标注一样图片都按Ctrl+S保存一下了

Display Labels：标注好图片之后，会把框和标签都显示出来

Advanced Mode：这样标注的十字架就会一直悬浮在窗口，不用每次标完一个目标，再按一次W快捷键，调出标注的十字架。



标注常用的快捷键

W：调出标注的十字架，开始标注

A：切换到上一张图片

D：切换到下一张图片

Ctrl+S：保存标注好的标签

del：删除标注的矩形框

Ctrl+鼠标滚轮：按住Ctrl，然后滚动鼠标滚轮，可以调整标注图片的显示大小

Ctrl+u：选择要标注图片的文件夹

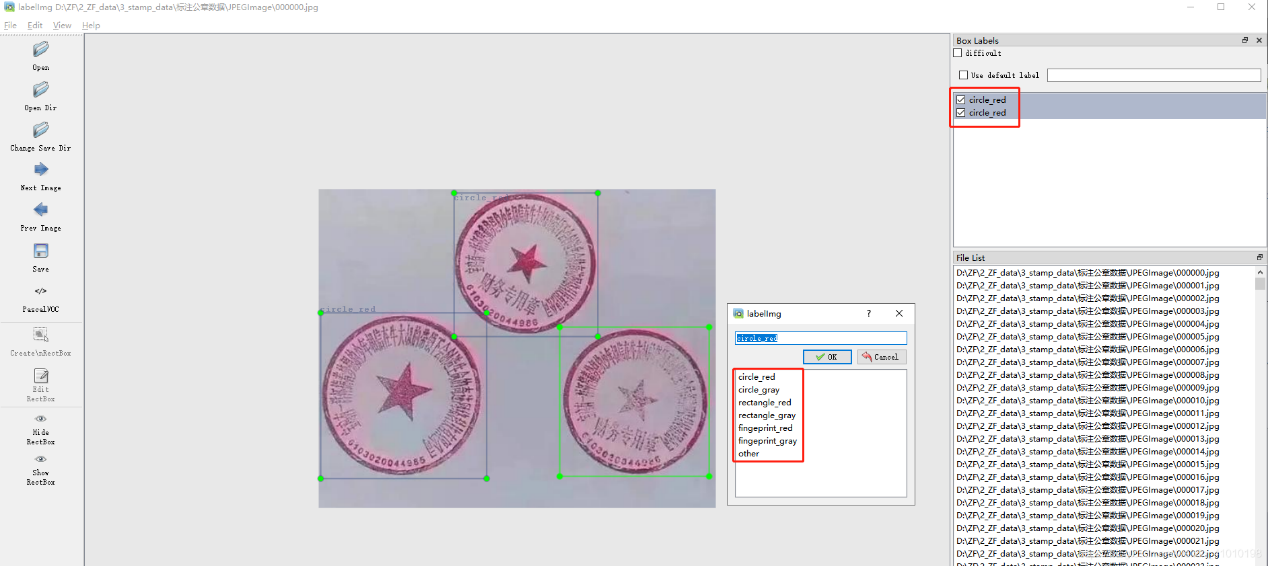
Ctrl+r：选择标注好的label标签存放的文件夹

↑→↓←：移动标注的矩形框的位置

使用标注的十字架，框住要标注的目标即可，如下图，是我对公章数据进行标注：

标注的时候，会把predefined\_classes.txt预定义的标加载出来，然后我们只要选择对应的标签即可

右上角会显示已经标注目标的标签



### 常见问题

每次打开labelimg都莫名奇妙出现很多标签

如下图，命名没有设置这些标签，但是莫名其妙出现这些标签：

labelImg目录下全局搜索一下发现在data目录下有一个predefined\_classes.txt文件，这个文件在克隆的时候就保留的，后面删除

## VOC数据标签和格式和YOLO数据标签格式说明

### VOC数据格式

VOC数据格式，会直接把每张图片标注的标签信息保存到一个xml文件中

例如：我们上面标注的JPEGImage/000001.jpg图片，标注的标签信息会保存到Annotation/000001.xml文件中，000001.xml中的信息如下：

<annotation>

<folder>JPEGImage</folder>

<filename>000000.jpg</filename>

<path>D:\ZF\2\_ZF\_data\3\_stamp\_data\标注公章数据\JPEGImage\000000.jpg</path>

<source>

<database>Unknown</database>

</source>

<size>

<width>500</width>

<height>402</height>

<depth>3</depth>

</size>

<segmented>0</segmented>

<object>

<name>circle\_red</name>

<pose>Unspecified</pose>

<truncated>0</truncated>

<difficult>0</difficult>

<bndbox>

<xmin>168</xmin>

<ymin>2</ymin>

<xmax>355</xmax>

<ymax>186</ymax>

</bndbox>

</object>

<object>

<name>circle\_red</name>

<pose>Unspecified</pose>

<truncated>0</truncated>

<difficult>0</difficult>

<bndbox>

<xmin>2</xmin>

<ymin>154</ymin>

<xmax>208</xmax>

<ymax>367</ymax>

</bndbox>

</object>

<object>

<name>circle\_red</name>

<pose>Unspecified</pose>

<truncated>0</truncated>

<difficult>0</difficult>

<bndbox>

<xmin>305</xmin>

<ymin>174</ymin>

<xmax>493</xmax>

<ymax>364</ymax>

</bndbox>

</object>

</annotation>

xml中的关键信息说明：

* 图片的名字
* 每个目标的标定框坐标：即左上角的坐标和右下角的坐标
  + xmin
  + ymin
  + xmax
  + ymax

### YOLO数据格式

YOLO数据格式，会直接把每张图片标注的标签信息保存到一个txt文件中

例如：上面标注的JPEGImage/000001.jpg图片，标注的标签信息会保存到Annotation/000001.txt文件中（同时会生成一个classes.txt文件，也保存到Annotation/classes.txt），000001.txt中的信息如下：

0 0.521000 0.235075 0.362000 0.450249

0 0.213000 0.645522 0.418000 0.519900

0 0.794000 0.665423 0.376000 0.470149

txt中信息说明：

每一行代表标注的一个目标

第一个数代表标注目标的标签，第一目标circle\_red，对应数字就是0

后面的四个数代表标注框的中心坐标和标注框的相对宽和高（进行了归一化，如何归一化可以参考我的这篇博客中的介绍）

同时会生成一个Annotation/classes.txt实际类别文件classes.txt，里面的内容如下：

circle\_red

circle\_gray

rectangle\_red

rectangle\_gray

fingeprint\_red

fingeprint\_gray

other

# 数据集格式

## Voc格式

标准的voc数据格式如下：

VOC2007/

Annotations/

0d4c5e4f-fc3c-4d5a-906c-105.xml

0ddfc5aea-fcdac-421-92dad-144/xml

...

ImageSets/

Main/

train.txt

test.txt

val.txt

trainval.txt

JPEGImages/

0d4c5e4f-fc3c-4d5a-906c-105.jpg

0ddfc5aea-fcdac-421-92dad-144.jpg

...

<https://github.com/jiachen0212/voc2coco-pattern>

## Coco格式

COCO的 全称是Common Objects in COntext，是微软团队提供的一个可以用来进行图像识别的数据集。MS COCO数据集中的图像分为训练、验证和测试集。COCO通过在Flickr上搜索80个对象类别和各种场景类型来收集图像，其使用了亚马逊的Mechanical Turk（AMT）。

COCO通过大量使用Amazon Mechanical Turk来收集数据。COCO数据集现在有3种标注类型：object instances（目标实例）, object keypoints（目标上的关键点）, 和image captions（看图说话），使用JSON文件存储。

基本的JSON结构体类型

object instances（目标实例）、object keypoints（目标上的关键点）、image captions（看图说话）这3种类型共享这些基本类型：info、image、license。

而annotation类型则呈现出了多态：

{

"info": info,

"licenses": [license],

"images": [image],

"annotations": [annotation],

}

info{

"year": int,

"version": str,

"description": str,

"contributor": str,

"url": str,

"date\_created": datetime,

}

license{

"id": int,

"name": str,

"url": str,

}

image{

"id": int,

"width": int,

"height": int,

"file\_name": str,

"license": int,

"flickr\_url": str,

"coco\_url": str,

"date\_captured": datetime,

}

1，info类型，比如一个info类型的实例：

"info":{

"description":"This is stable 1.0 version of the 2014 MS COCO dataset.",

"url":"http:\/\/mscoco.org",

"version":"1.0","year":2014,

"contributor":"Microsoft COCO group",

"date\_created":"2015-01-27 09:11:52.357475"

},

2，Images是包含多个image实例的数组，对于一个image类型的实例：

{

"license":3,

"file\_name":"COCO\_val2014\_000000391895.jpg",

"coco\_url":"http:\/\/mscoco.org\/images\/391895",

"height":360,"width":640,"date\_captured":"2013-11-14 11:18:45",

"flickr\_url":"http:\/\/farm9.staticflickr.com\/8186\/8119368305\_4e622c8349\_z.jpg",

"id":391895

},

3，licenses是包含多个license实例的数组，对于一个license类型的实例：

{

"url":"http:\/\/creativecommons.org\/licenses\/by-nc-sa\/2.0\/",

"id":1,

"name":"Attribution-NonCommercial-ShareAlike License"

},

### Object Instance 类型的标注格式

* 1. **整体JSON文件格式**

比如上图中的instances\_train2017.json、instances\_val2017.json这两个文件就是这种格式。

Object Instance这种格式的文件从头至尾按照顺序分为以下段落：

{

"info": info,

"licenses": [license],

"images": [image],

"annotations": [annotation],

"categories": [category]

}

是的，你打开这两个文件，虽然内容很多，但从文件开始到结尾按照顺序就是这5段。其中，info、licenses、images这三个结构体/类型 在上一节中已经说了，在不同的JSON文件中这三个类型是一样的，定义是共享的。不共享的是annotation和category这两种结构体，他们在不同类型的JSON文件中是不一样的。

images数组元素的数量等同于划入训练集（或者测试集）的图片的数量；

annotations数组元素的数量等同于训练集（或者测试集）中bounding box的数量；

categories数组元素的数量为80（2017年）；

**2，annotations字段**

annotations字段是包含多个annotation实例的一个数组，annotation类型本身又包含了一系列的字段，如这个目标的category id和segmentation mask。segmentation格式取决于这个实例是一个单个的对象（即iscrowd=0，将使用polygons格式）还是一组对象（即iscrowd=1，将使用RLE格式）。如下所示：

annotation{

"id": int,

"image\_id": int,

"category\_id": int,

"segmentation": RLE or [polygon],

"area": float,

"bbox": [x,y,width,height],

"iscrowd": 0 or 1,

}

注意

单个的对象（iscrowd=0)可能需要多个polygon来表示，比如这个对象在图像中被挡住了。而iscrowd=1时（将标注一组对象，比如一群人）的segmentation使用的就是RLE格式。

要是iscrowd=0那么segmentation就是polygon格式；只要iscrowd=1那么segmentation就是RLE格式。另外，每个对象（不管是iscrowd=0还是iscrowd=1）都会有一个矩形框bbox ，矩形框左上角的坐标和矩形框的长宽会以数组的形式提供，数组第一个元素就是左上角的横坐标值。

area是area of encoded masks，是标注区域的面积。如果是矩形框，那就是高乘宽；如果是polygon或者RLE，那就复杂点。

最后，annotation结构中的categories字段存储的是当前对象所属的category的id，以及所属的supercategory的name。

下面是从instances\_val2017.json文件中摘出的一个annotation的实例，这里的segmentation就是polygon格式：

{

"segmentation": [[510.66,423.01,511.72,420.03,510.45......]],

"area": 702.1057499999998,

"iscrowd": 0,

"image\_id": 289343,

"bbox": [473.07,395.93,38.65,28.67],

"category\_id": 18,

"id": 1768

},

polygon格式比较简单，这些数按照相邻的顺序两两组成一个点的xy坐标，如果有n个数（必定是偶数），那么就是n/2个点坐标。

如果iscrowd=1，那么segmentation就是RLE格式(segmentation字段会含有counts和size数组)，在json文件中gemfield挑出一个这样的例子，如下所示：

segmentation :

{

u'counts': [272, 2, 4, 4, 4, 4, 2, 9, 1, 2, 16, 43, 143, 24......],

u'size': [240, 320]

}

COCO数据集的RLE都是uncompressed RLE格式（与之相对的是compact RLE）。 RLE所占字节的大小和边界上的像素数量是正相关的。RLE格式带来的好处就是当基于RLE去计算目标区域的面积以及两个目标之间的unoin和intersection时会非常有效率。 上面的segmentation中的counts数组和size数组共同组成了这幅图片中的分割 mask。其中size是这幅图片的宽高，然后在这幅图像中，每一个像素点要么在被分割（标注）的目标区域中，要么在背景中。很明显这是一个bool量：如果该像素在目标区域中为true那么在背景中就是False；如果该像素在目标区域中为1那么在背景中就是0。对于一个240x320的图片来说，一共有76800个像素点，根据每一个像素点在不在目标区域中，我们就有了76800个bit，比如像这样（随便写的例子，和上文的数组没关系）：00000111100111110...；但是这样写很明显浪费空间，我们直接写上0或者1的个数不就行了嘛（Run-length encoding)，于是就成了54251...，这就是上文中的counts数组。

**3，categories字段**

categories是一个包含多个category实例的数组，而category结构体描述如下：

{

"id": int,

"name": str,

"supercategory": str,

}

从instances\_val2017.json文件中摘出的2个category实例如下所示：

{

"supercategory": "person",

"id": 1,

"name": "person"

},

{

"supercategory": "vehicle",

"id": 2,

"name": "bicycle"

},

至2017年的时候，一共有80个category。

### Object Keypoint 类型的标注格式

**1，整体JSON文件格式**

比如上图中的person\_keypoints\_train2017.json、person\_keypoints\_val2017.json这两个文件就是这种格式。

Object Keypoint这种格式的文件从头至尾按照顺序分为以下段落，看起来和Object Instance一样啊：

{

"info": info,

"licenses": [license],

"images": [image],

"annotations": [annotation],

"categories": [category]

}

是的，你打开这两个文件，虽然内容很多，但从文件开始到结尾按照顺序就是这5段。其中，info、licenses、images这三个结构体/类型 在第一节中已经说了，在不同的JSON文件中这三个类型是一样的，定义是共享的。不共享的是annotation和category这两种结构体，他们在不同类型的JSON文件中是不一样的。

images数组元素数量是划入训练集（测试集）的图片的数量；

annotations是bounding box的数量，在这里只有人这个类别的bounding box；

categories数组元素的数量为1，只有一个：person（2017年）；

**2，annotations字段**

这个类型中的annotation结构体包含了Object Instance中annotation结构体的所有字段，再加上2个额外的字段。

新增的keypoints是一个长度为3\*k的数组，其中k是category中keypoints的总数量。每一个keypoint是一个长度为3的数组，第一和第二个元素分别是x和y坐标值，第三个元素是个标志位v，v为0时表示这个关键点没有标注（这种情况下x=y=v=0），v为1时表示这个关键点标注了但是不可见（被遮挡了），v为2时表示这个关键点标注了同时也可见。

num\_keypoints表示这个目标上被标注的关键点的数量（v>0），比较小的目标上可能就无法标注关键点。

annotation{

"keypoints": [x1,y1,v1,...],

"num\_keypoints": int,

"id": int,

"image\_id": int,

"category\_id": int,

"segmentation": RLE or [polygon],

"area": float,

"bbox": [x,y,width,height],

"iscrowd": 0 or 1,

}

从person\_keypoints\_val2017.json文件中摘出一个annotation的实例如下：

{

"segmentation": [[125.12,539.69,140.94,522.43...]],

"num\_keypoints": 10,

"area": 47803.27955,

"iscrowd": 0,

"keypoints": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,142,309,1,177,320,2,191,398...],

"image\_id": 425226,"bbox": [73.35,206.02,300.58,372.5],"category\_id": 1,

"id": 183126

},

**3，categories字段**

最后，对于每一个category结构体，相比Object Instance中的category新增了2个额外的字段，keypoints是一个长度为k的数组，包含了每个关键点的名字；skeleton定义了各个关键点之间的连接性（比如人的左手腕和左肘就是连接的，但是左手腕和右手腕就不是）。目前，COCO的keypoints只标注了person category （分类为人）。

定义如下：

{

"id": int,

"name": str,

"supercategory": str,

"keypoints": [str],

"skeleton": [edge]

}

从person\_keypoints\_val2017.json文件中摘出一个category的实例如下：

{

"supercategory": "person",

"id": 1,

"name": "person",

"keypoints": ["nose","left\_eye","right\_eye","left\_ear","right\_ear","left\_shoulder","right\_shoulder","left\_elbow","right\_elbow","left\_wrist","right\_wrist","left\_hip","right\_hip","left\_knee","right\_knee","left\_ankle","right\_ankle"],

"skeleton": [[16,14],[14,12],[17,15],[15,13],[12,13],[6,12],[7,13],[6,7],[6,8],[7,9],[8,10],[9,11],[2,3],[1,2],[1,3],[2,4],[3,5],[4,6],[5,7]]

}

### Image Caption的标注格式

**1，整体JSON文件格式**

比如上图中的captions\_train2017.json、captions\_val2017.json这两个文件就是这种格式。

Image Caption这种格式的文件从头至尾按照顺序分为以下段落，看起来和Object Instance一样，不过没有最后的categories字段：

{

"info": info,

"licenses": [license],

"images": [image],

"annotations": [annotation]

}

是的，你打开这两个文件，虽然内容很多，但从文件开始到结尾按照顺序就是这4段。其中，info、licenses、images这三个结构体/类型 在第一节中已经说了，在不同的JSON文件中这三个类型是一样的，定义是共享的。不共享的是annotations这种结构体，它在不同类型的JSON文件中是不一样的。

images数组的元素数量等于划入训练集（或者测试集）的图片的数量；

annotations的数量要多于图片的数量，这是因为一个图片可以有多个场景描述；

**2，annotations字段**

这个类型中的annotation用来存储描述图片的语句。每个语句描述了对应图片的内容，而每个图片至少有5个描述语句（有的图片更多）。annotation定义如下：

annotation{

"id": int,

"image\_id": int,

"caption": str

}

从captions\_val2017.json中摘取的一个annotation实例如下：

{

"image\_id": 179765,

"id": 38,"caption": "A black Honda motorcycle parked in front of a garage."

}

此处未使用测试集

coco/

annotations/

instances\_train2017.json

instances\_val2017.json

images/

train2017/

0d4c5e4f-fc3c-4d5a-906c-105.jpg

...

val2017

0ddfc5aea-fcdac-421-92dad-144.jpg

...

## Labelme格式

labelme/

0d4c5e4f-fc3c-4d5a-906c-105.json

0d4c5e4f-fc3c-4d5a-906c-105.jpg

0ddfc5aea-fcdac-421-92dad-144.json

0ddfc5aea-fcdac-421-92dad-144.jpg

Json file 格式: （imageData那一块太长了，不展示了）

{

"version": "3.6.16",

"flags": {},

"shapes": [

{

"label": "helmet",

"line\_color": null,

"fill\_color": null,

"points": [

[

131,

269

],

[

388,

457

]

],

"shape\_type": "rectangle"

}

],

"lineColor": [

0,

255,

0,

128

],

"fillColor": [

255,

0,

0,

128

],

"imagePath": "004ffe6f-c3e2-3602-84a1-ecd5f437b113.jpg",

"imageData": "" # too long ,so not show here

"imageHeight": 1080,

"imageWidth": 1920

}

<https://github.com/spytensor/prepare_detection_dataset>

## 格式转换

pip install git+https://github.com/philferriere/cocoapi.git#subdirectory=PythonAPI

pip install -i https://pypi.tuna.tsinghua.edu.cn/simple Cython

pip install -i https://pypi.tuna.tsinghua.edu.cn/simple pycocotools

在PythonAPI的文件夹路径下：

python setup.py build\_ext install

pip安装慢的，请配置镜像源，下面是清华的镜像源。

```bash

$ pip install pip -U

$ pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple

```

想配其他镜像源直接把网址替换即可，以下是国内常用的镜像源：

```yaml

豆瓣 https://pypi.doubanio.com/simple/

网易 https://mirrors.163.com/pypi/simple/

阿里云 https://mirrors.aliyun.com/pypi/simple/

腾讯云 https://mirrors.cloud.tencent.com/pypi/simple

清华大学 https://pypi.tuna.tsinghua.edu.cn/simple/

### voc转换为yolo格式计算

标注好的VOC格式的标签xml文件，存储的主要信息为：

图片的名字

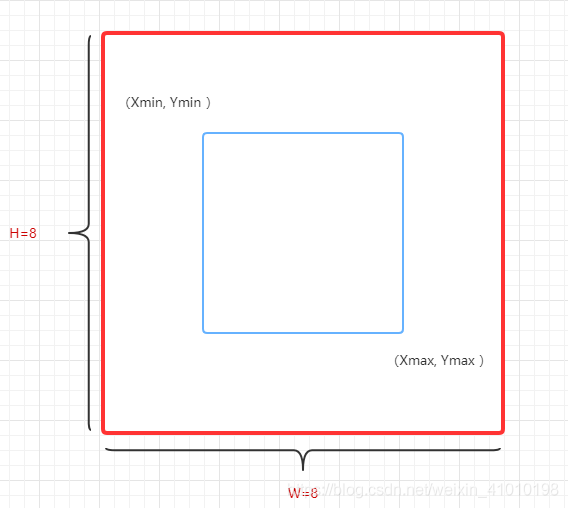
图片的高height、宽width、通道depth

标定框的坐标位置：xmin、ymin、xmax、ymax

例如下图代表的是一样图片：

红框代表的是原图大小：height=8，width=8

蓝框代表的是标注物体的框：左上角坐标为 (xmin, ymin)=(2,2)，右下角的坐标为 (xmax, ymax)=(6,6)



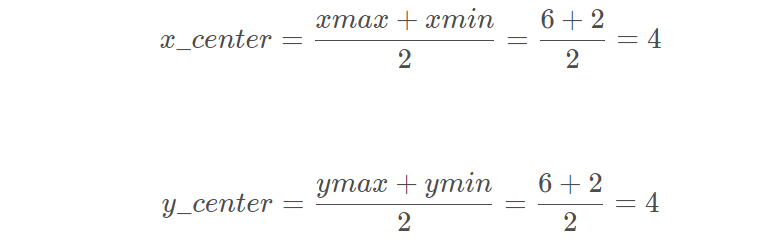
而voc\_label.py目的就是把标注为VOC格式数据转化为标注为yolo格式数据：

VOC格式标签：图片的实际宽和高，标注框的左上角和右下角坐标

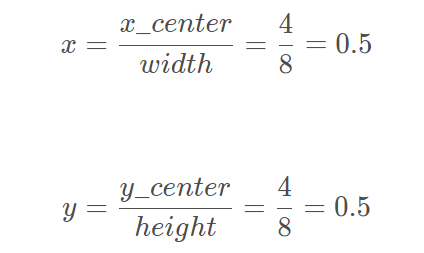
YOLO格式标签：标注框的中心坐标（归一化的），标注框的宽和高（归一化的）

VOC格式标签转换为YOLO格式标签计算公式：

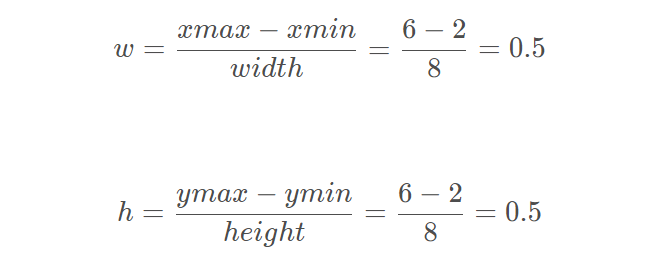
框中心的实际坐标（x, y）：（一般可能还会在后面减去1）



框归一化后的中心坐标（x, y）：



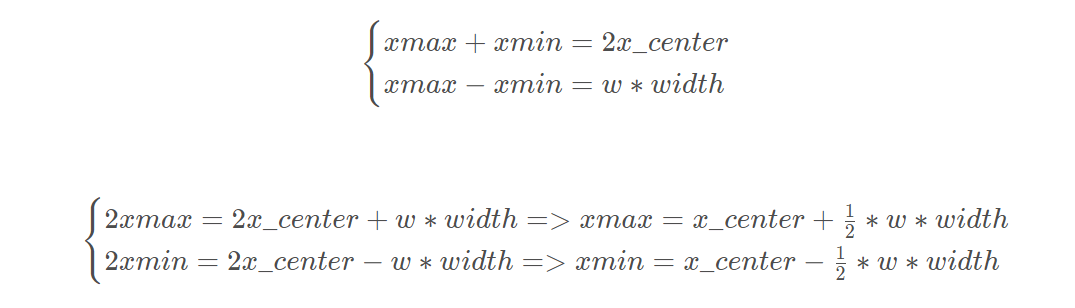
框的高和框（归一化的）：



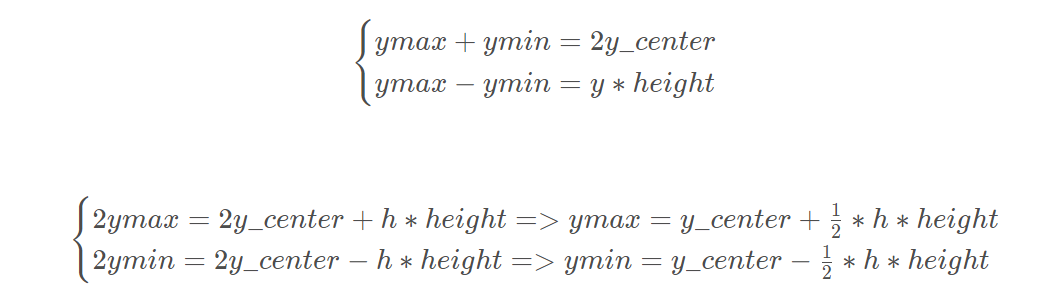
### yolo转换为voc格式计算

voc中保存的坐标信息为：xmin, ymin, xmax, ymax，所以只要根据上面的公式，推导出这四个值即可，推导如下：

推导：xmin, xmax



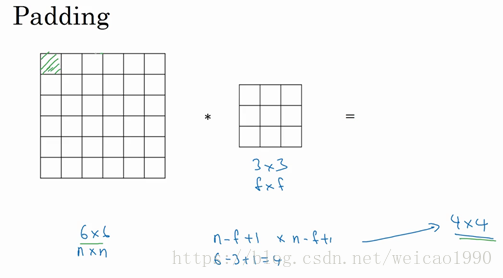
推导：ymin, ymax

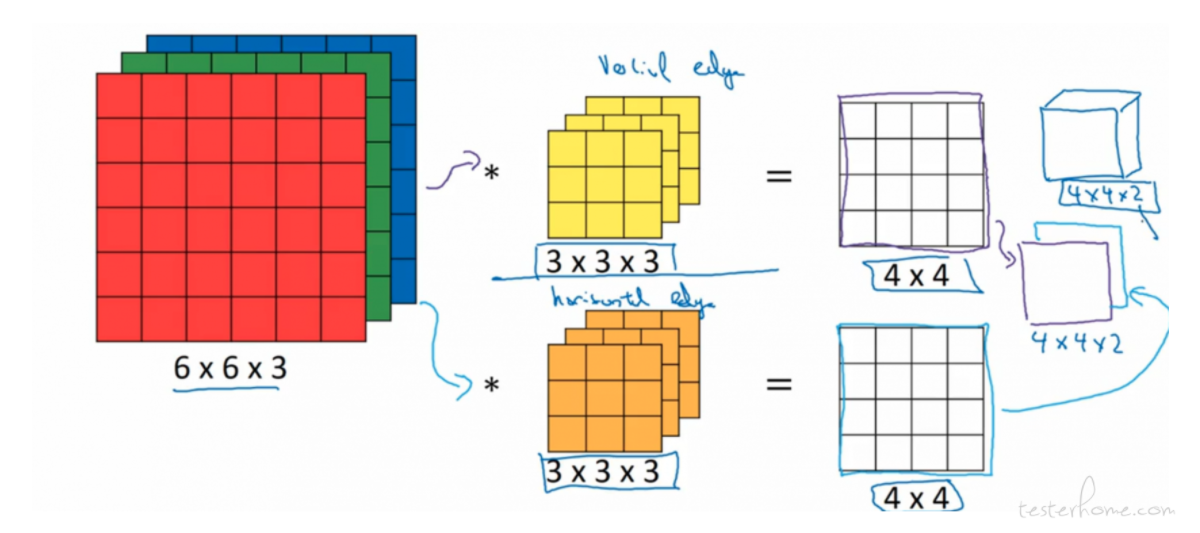


# 深度学基本概念

## padding

为了构建深度神经网络，使用的一个基本的卷积操作就是padding，

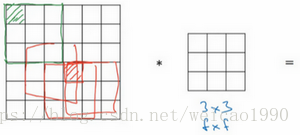




如果用一个3×3的过滤器卷积一个6×6的图像，最后会得到一个4×4的输出，也就是一个4×4矩阵。那是因为3×3过滤器在6×6矩阵中，只可能有4×4种可能的位置。，如果我们有一个nxn的图像，用fxf的过滤器做卷积，那么输出的维度就是(n-f+1)\*(n-f+1)。在这个例子里是6-3+1=4，因此得到了一个4×4的输出。

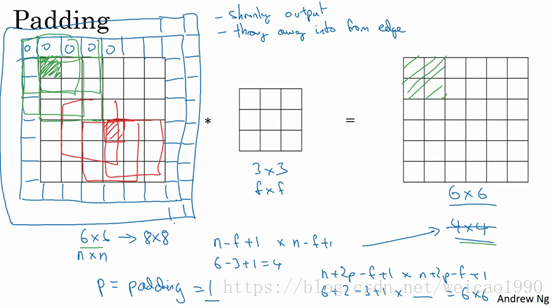
这样的话会有两个缺点：

第一个缺点是，每次做卷积操作，图像就会缩小，从6×6缩小到4×4，可能做了几次之后，图像就会变得很小了，可能会缩小到只有1×1的大小。可不想让图像在每次识别边缘或其他特征时都缩小，这就是第一个缺点。



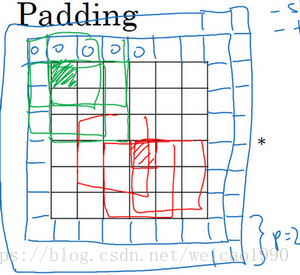
第二个缺点是，注意角落边缘的像素，这个像素点（绿色阴影标记）只被一个过滤器输出所触碰或者使用，因为它位于这个3×3的区域的一角。但如果是在中间的像素点，比如这个（红色方框标记），就会有许多3×3的区域与之重叠。所以那些在角落或者边缘区域的像素点在输出中采用较少，意味着丢掉了图像边缘位置的许多信息。

为了解决这两个问题，一是输出缩小。当我们建立深度神经网络时，就会知道为什么不希望每进行一步操作图像都会缩小。比如有100层深层的网络，如果图像每经过一层都缩小的话，经过100层网络后，就会得到一个很小的图像，所以这是个问题。另一个问题是图像边缘的大部分信息都丢失了。



为了解决这些问题，可以在卷积操作之前填充这幅图像。在这个案例中，可以沿着图像边缘再填充一层像素。如果这样操作了，那么6×6的图像就被你填充成了一个8×8的图像。如果用3×3的图像对这个8×8的图像卷积，得到的输出就不是4×4的，而是6×6的图像，就得到了一个尺寸和原始图像6×6的图像。习惯上，可以用0去填充，如果p是填充的数量，在这个案例中，p = 1，因为在周围都填充了一个像素点，输出也就变成了(n+2p-f+1)\*(n+2p-f+1)，所以就变成了(6+2\*1-3+1)\*(6+2\*1-3+1)=6\*6，和输入的图像一样大。这个涂绿的像素点（左边矩阵）影响了输出中的这些格子（右边矩阵）。这样一来，丢失信息或者更准确来说角落或图像边缘的信息发挥的作用较小的这一缺点就被削弱了。

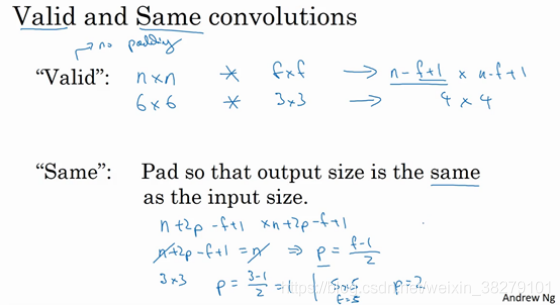
刚才已经展示过用一个像素点来填充边缘，也可以填充两个像素点，也就是说在这里填充一层。实际上还可以填充更多像素。这里画的这种情况，填充后。



至于选择填充多少像素，通常有两个选择，分别叫做Valid卷积和Same卷积。

Valid卷积意味着不填充，这样的话，如果有一个n\*n的图像，用一个f\*f的过滤器卷积，它将会给一个(n-f+1)\*(n-f+1)维的输出。这类似于前面展示的例子，有一个6×6的图像，通过一个3×3的过滤器，得到一个4×4的输出。

另一个经常被用到的填充方法叫做Same卷积，那意味填充后，输出大小和输入大小是一样的。根据这个公式n-f+1，当填充p个像素点，n就变成了n+2p，最后公式变为n+2p-f+1。因此如果有一个n\*n的图像，用p个像素填充边缘，输出的大小就是这样的(n+2p-f+1)\*(n+2p-f+1)。如果想让n+2p-f+1=n的话，使得输出和输入大小相等，如果用这个等式求解p，那么p =(f-1)/2。所以当f是一个奇数的时候，只要选择相应的填充尺寸，就能确保得到和输入相同尺寸的输出。这也是为什么前面的例子，当过滤器是3×3时，和上一张幻灯片的例子一样，使得输出尺寸等于输入尺寸，所需要的填充是(3-1)/2，也就是1个像素。另一个例子，当过滤器是5×5，如果f=5，然后代入那个式子，就会发现需要2层填充使得输出和输入一样大，这是过滤器5×5的情况。



习惯上，计算机视觉中，f通常是奇数，甚至可能都是这样。很少看到一个偶数的过滤器在计算机视觉里使用，认为有两个原因。

其中一个可能是，如果是一个偶数，那么只能使用一些不对称填充。只有f是奇数的情况下，Same卷积才会有自然的填充，可以以同样的数量填充四周，而不是左边填充多一点，右边填充少一点，这样不对称的填充。

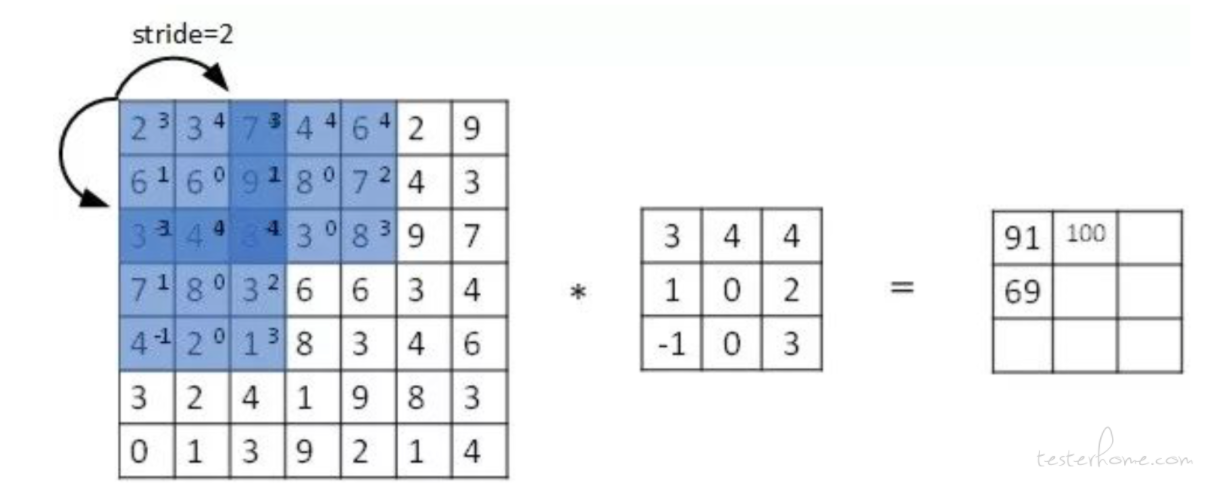
第二个原因是当有一个奇数维过滤器，比如3×3或者5×5的，它就有一个中心点。有时在计算机视觉里，如果有一个中心像素点会更方便，便于指出过滤器的位置。

也许这些都不是为什么f通常是奇数的充分原因，但如果看了卷积的文献，经常会看到3×3的过滤器，也可能会看到一些5×5，7×7的过滤器。后面也会谈到1×1的过滤器，以及什么时候它是有意义的。但是习惯上，推荐你只使用奇数的过滤器。如果你使用偶数f也可能会得到不错的表现，如果遵循计算机视觉的惯例，通常使用奇数值的f。

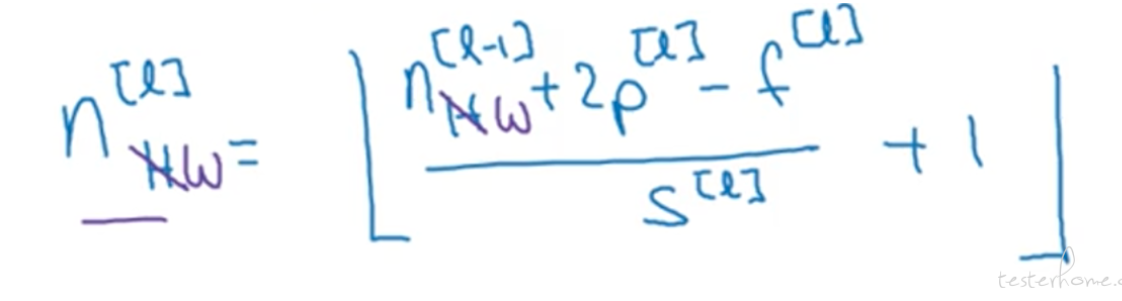
已经看到如何使用padding卷积，为了指定卷积操作中的padding，可以指定的值。也可以使用Valid卷积，也就是。也可使用Same卷积填充像素，使输出和输入大小相同。以上就是padding，在接下来的视频中我们讨论如何在卷积中设置步长。

## 卷积步长 (Stride)

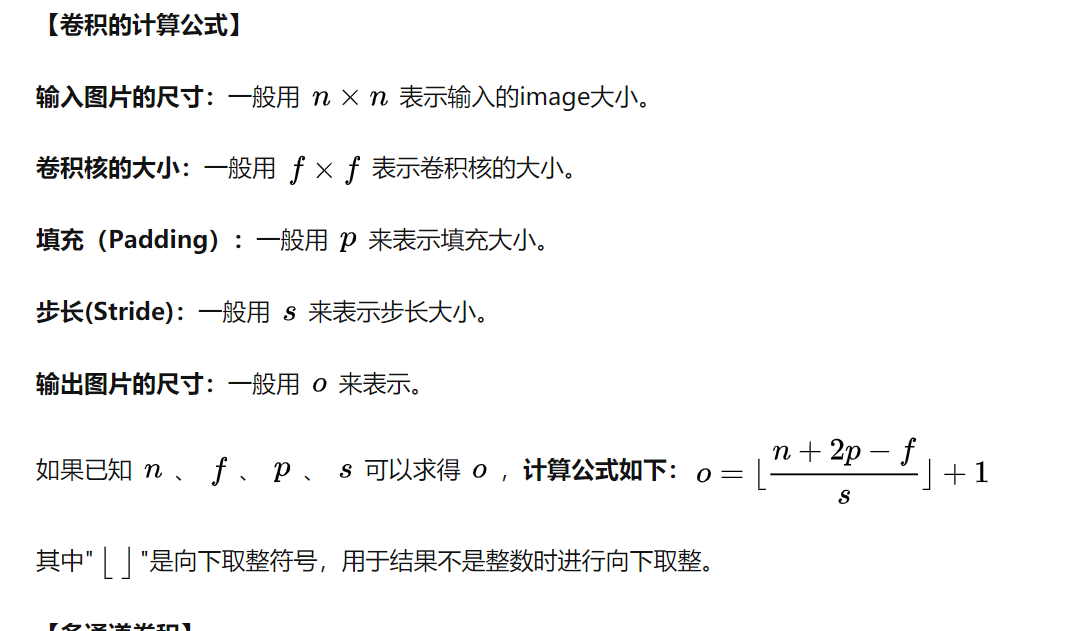
Stride 表示 filter 在原图片中水平方向和垂直方向每次的步进长度。若 stride=2，则表示 filter 每次步进长度为 2，即隔一点移动一次。

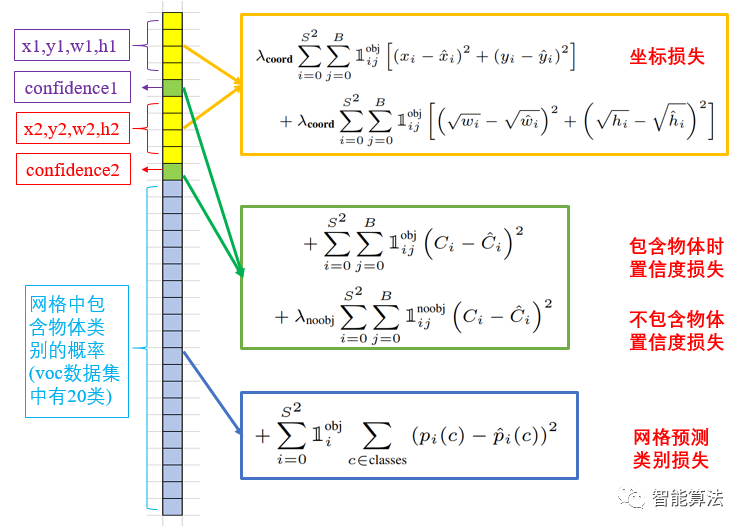


那么如果我们想要计算出任意一个卷积层输出图片的大小 (或者说维度的话)，可以按照下面的公式计算。



其中 n 为上一层图片的维度，p 是 padding， f 是过滤器的大小，s 是卷积步长。





# 参考资料

sudo pip install -i http://pypi.douban.com/simple/ labelme

pip install labelme

pip install -U scikit-learn

pip install -i http://pypi.douban.com/simple/ cython --trusted-host pypi.douban.com

<https://www.cnblogs.com/ZhangRuoXu/p/6370107.html>

数据标注软件labelme详解

<https://blog.csdn.net/u014061630/article/details/88756644>

<https://github.com/wkentaro/labelme>

<https://github.com/wkentaro/labelme/releases>

<https://github.com/spytensor/prepare_detection_dataset>

把labelme形成的文件转为VOC数据集格式时遇到的问题

<https://blog.csdn.net/AWhiteDongDong/article/details/107576592?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_title~default-4.control&spm=1001.2101.3001.4242>

total\_files = glob("./VOCdevkit/VOC2007/Annotations/\*.xml")

<https://blog.csdn.net/weixin_45963617/article/details/105449408?utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-5.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-5.control>

ValueError: With n\_samples=0, test\_size=0.2 and train\_size=None, the resulting train set will be emp

<https://blog.csdn.net/weixin_45963617/article/details/105449408>

labelme标注样本之后转coco数据类型python文件

<https://blog.csdn.net/chanshibing/article/details/86304346?utm_medium=distribute.pc_relevant_download.none-task-blog-2~default~BlogCommendFromBaidu~default-11.test_version_3&depth_1-utm_source=distribute.pc_relevant_download.none-task-blog-2~default~BlogCommendFromBaidu~default-11.test_version_>

COCO数据集的标注格式

<https://zhuanlan.zhihu.com/p/29393415>

把LabelImg标注的YOLO格式标签转化为VOC格式标签 和 把VOC格式标签转化为YOLO格式标签

<https://blog.csdn.net/weixin_41010198/article/details/107175968>

目标检测使用LabelImg标注VOC数据格式和YOLO数据格式——LabelImg使用详细教程

<https://shliang.blog.csdn.net/article/details/106378404>

YOLOv3 Darknet安装编译与训练自己的数据集

<https://blog.csdn.net/weixin_41010198/article/details/106072347>

目标检测-数据准备系列(一)--voc2coco

<https://blog.csdn.net/fengxinzioo/article/details/103102139>

<https://github.com/maozezhong/CV_ToolBox>

深度学习卷积笔记之Full卷积、Same卷积、Valid卷积

<https://blog.csdn.net/weekdayh/article/details/104317359?utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-1.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-1.control>

[Batch Normalization详解](https://www.cnblogs.com/shine-lee/p/11989612.html)

<https://www.cnblogs.com/shine-lee/p/11989612.html>

YOLO-tensorflow代码解析二（yolo\_net.py）

<http://www.manongjc.com/article/68717.html>

[yolo源码解析（一）](https://www.cnblogs.com/sddai/p/10288074.html)

<https://www.cnblogs.com/sddai/p/10288074.html>