YOLOv3目标检测算法

聚类确定锚框尺寸

@tm9161

L3

聚类确定锚框尺寸

1. Kmeans 聚类算法原理

2. 聚类确定锚框尺寸

最近邻分类原理

1.**计算**测试点到各训练数据的**距离**

2.**选取**距离**最近**的邻居（K=1）

3.根据这个**邻居类别**判断**待测点类别**

距离：

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum |I_1^p - I_2^p|$$

L = |2-1|+|1-1| = 1 √

L = |2-3|+|1-5| = 5

L = |2-5|+|1-3| = 5

聚类算法原理

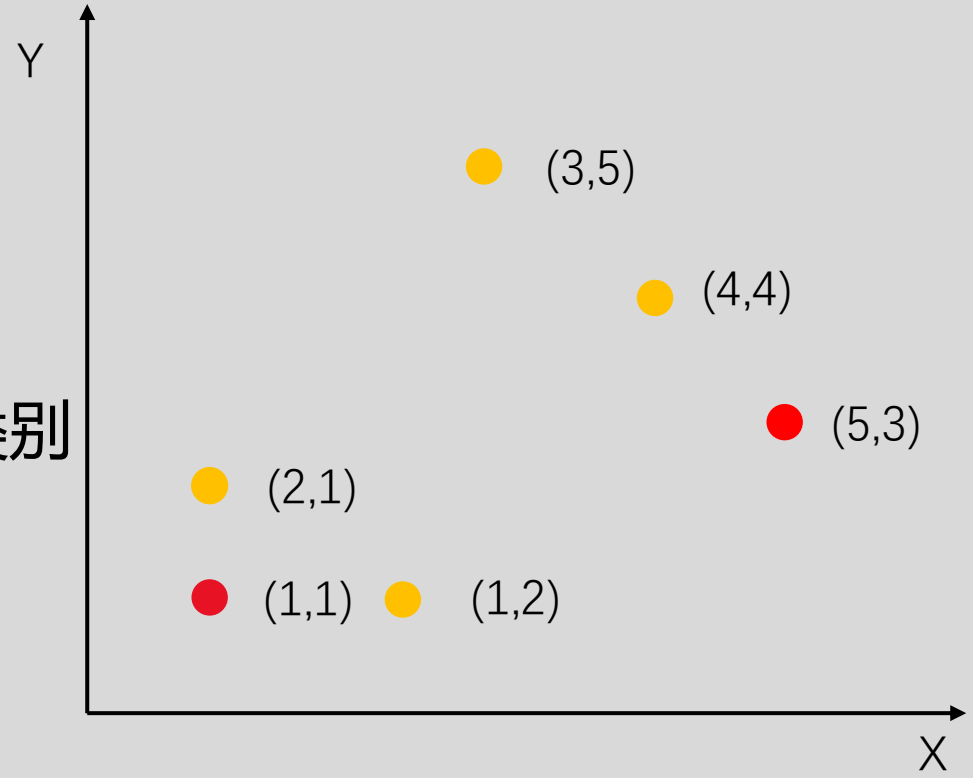
1.根据k值，随机确定k个聚类中心。

2.计算所有样本到k个聚类中心的距离。

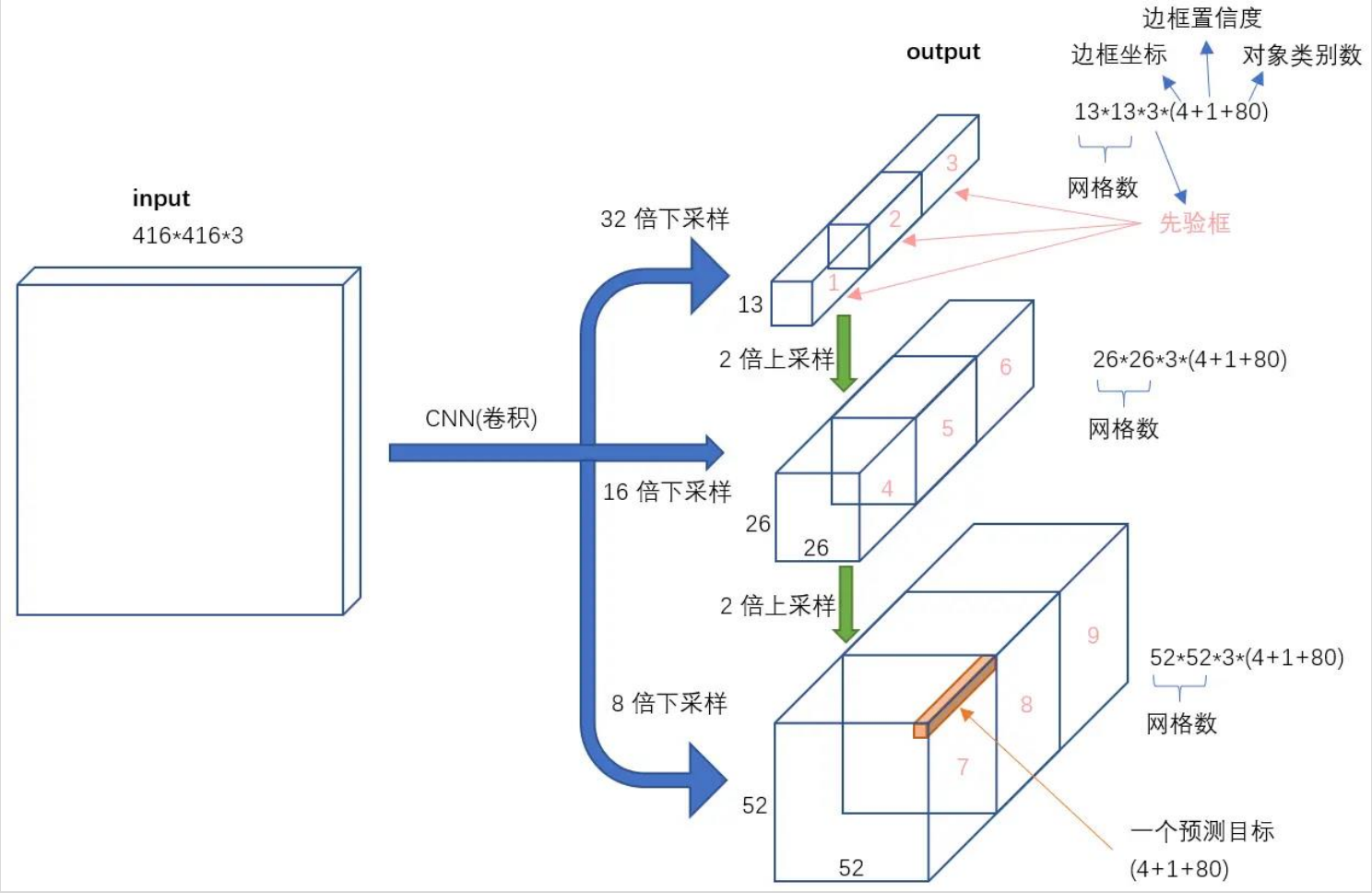
3.每个样本将距离最近的质心作为自己的类别

4.每个类别计算各自新的聚类中心。

5.重复此过程，直到聚类中心不再变化。

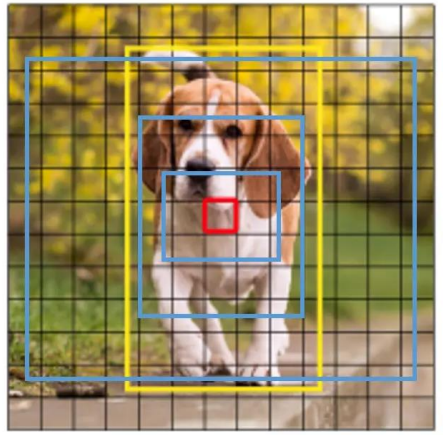


YOLO V3

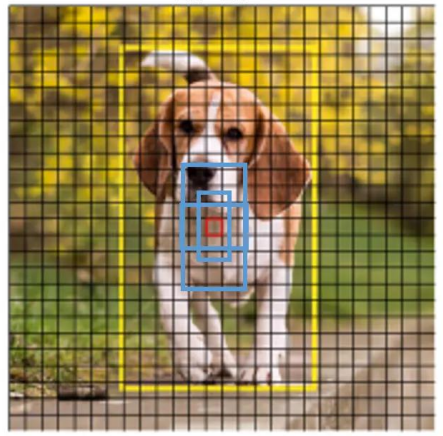


图片来源：https://www.jianshu.com/p/d13ae1055302?tdsourcetag=s_pcqq_aiomsg

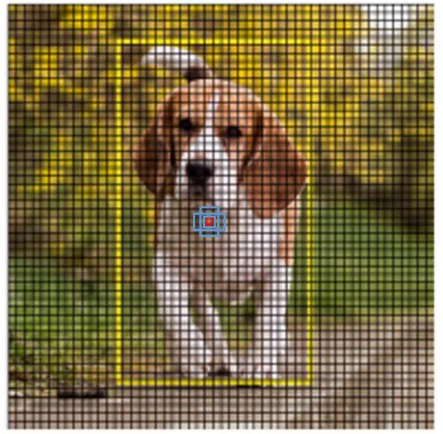
13*13



26*26



52*52



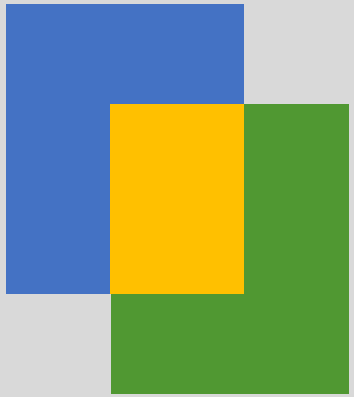
1. 读取边界框信息

```
def txt2boxes():
    # 打开包含边界框信息的文本文件
    f = open(filename, 'r')
    dataSet = []  # 初始化存储边界框尺寸的列表
    for line in f:  # 遍历文件的每一行
        infos = line.split(" ")  # 通过空格分割每行的信息
        length = len(infos)
        for i in range(1, length):  # 从第二个元素开始遍历（跳过图像路径）
            # 计算边界框的宽度：xmax - xmin
            width = int(infos[i].split(",")[2]) - int(infos[i].split(",")[0])
            # 计算边界框的高度：ymax - ymin
            height = int(infos[i].split(",")[3]) - int(infos[i].split(",")[1])
            dataSet.append([width, height])  # 添加到数据集列表
    result = np.array(dataSet)  # 将列表转换为NumPy数组
    f.close()  # 关闭文件
    return result  # 返回包含所有边界框尺寸的NumPy数组
```

```
/home/tm/Desktop/share/object_detection/yolo3-keras-3/VOCdevkit/VOC2007/JPEGImages/001001.jpg 62,64,325,375,7 315,102,427,356,14
/home/tm/Desktop/share/object_detection/yolo3-keras-3/VOCdevkit/VOC2007/JPEGImages/001689.jpg 122,86,227,376,14 1,183,203,500,14 60,238,158,367,8
/home/tm/Desktop/share/object_detection/yolo3-keras-3/VOCdevkit/VOC2007/JPEGImages/000896.jpg 2,189,256,294,0
```

2. 计算边界框的IOU

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$



距离：1—IOU

```python
def iou(boxes, clusters):  # 计算一组边界框和聚类（锚框）的IoU
    n = boxes.shape[0]  # 边界框数量
    k = cluster_number  # 聚类数量

    # 计算边界框的面积
    box_area = boxes[:, 0] * boxes[:, 1]  # 宽*高
    box_area = box_area.repeat(k)  # 每个边界框面积重复k次
    box_area = np.reshape(box_area, (n, k))  # 重塑为n*k的矩阵

    # 计算聚类的面积
    cluster_area = clusters[:, 0] * clusters[:, 1]  # 聚类的宽*高
    cluster_area = np.tile(cluster_area, [1, n])  # 每个聚类面积重复n次
    cluster_area = np.reshape(cluster_area, (n, k))  # 重塑为n*k的矩阵

    # 计算宽度的最小值矩阵
    box_w_matrix = np.reshape(boxes[:, 0].repeat(k), (n, k))  # 边界框宽度重复k次并重塑
    cluster_w_matrix = np.reshape(np.tile(clusters[:, 0], (1, n)), (n, k))  # 聚类宽度重复n次并重塑
    min_w_matrix = np.minimum(cluster_w_matrix, box_w_matrix)  # 宽度的最小值

    # 计算高度的最小值矩阵
    box_h_matrix = np.reshape(boxes[:, 1].repeat(k), (n, k))  # 边界框高度重复k次并重塑
    cluster_h_matrix = np.reshape(np.tile(clusters[:, 1], (1, n)), (n, k))  # 聚类高度重复n次并重塑
    min_h_matrix = np.minimum(cluster_h_matrix, box_h_matrix)  # 高度的最小值

    # 计算交叉区域的面积
    inter_area = np.multiply(min_w_matrix, min_h_matrix)  # 交叉区域面积

    # 计算IoU
    result = inter_area / (box_area + cluster_area - inter_area)  # IoU计算公式
    return result  # 返回IoU值矩阵
```

3. 聚类生成锚框尺寸

```
def kmeans(boxes, k, dist=np.median):
    box_number = boxes.shape[0]  # 边界框的总数
    distances = np.empty((box_number, k))  # 存储每个边界框到聚类中心的距离
    last_nearest = np.zeros((box_number,))  # 上一次每个边界框最近的聚类中心
    np.random.seed()  # 初始化随机种子
    clusters = boxes[np.random.choice(box_number, k, replace=False)]  # 随机初始化k个聚类中心

    while True:
        distances = 1 - iou(boxes, clusters)  # 计算所有边界框与聚类中心的1-IoU作为距离

        current_nearest = np.argmin(distances, axis=1)  # 找到每个边界框最近的聚类中心
        if (last_nearest == current_nearest).all():  # 如果聚类中心不再变化，则结束迭代
            break
        for cluster in range(k):  # 更新每个聚类中心
            clusters[cluster] = dist(boxes[current_nearest == cluster], axis=0)

        last_nearest = current_nearest  # 更新上一次的聚类中心为当前聚类中心

    return clusters  # 返回聚类中心，即最终确定的锚框尺寸
```

参考资料：

1. 3.1 YOLO系列理论合集(YOLOv1~v3)
https://www.bilibili.com/video/BV1yi4y1g7ro

2. qqwweee/keras-yolo3
https://github.com/qqwweee/keras-yolo3

3.大语言模型