一种低表面亮度星系的检测方法

A Way of Detection of Low Surface Brightness Galaxies

作者：张丰毅

院校：山东大学

指导老师：衣振萍

摘要

低表面亮度星系（low-surface-brightness galaxy），或LSB星系，是一种弥漫星系，当从地球观察这些星系时，在夜空的环境中，它们的表面亮度至少比周围的背景天光低一个星等。LSB 星系对于暗物质的研究具有重要意义。为此，本文提出了一种对于低表面亮度星系的识别办法。主要进行了对天文图像进行增强、通道合并；对数据表进行重复合并、异常值剔除；使用 YOLOv3 框架，利用darknet53 卷积残差网络，在谷歌提供的 Colab 云计算平台上训练对 LSB 星系图像的识别和定位等工作。经过反复试验不同参数，最终得到 mAP=93.40%（IoU_thresh=0.5），recall = 94%（conf_thresh = 0.25）的优良训练结果，能够较为准确的找出含有 LSB 星系的天文图像，并给出其中心坐标。为 LSB 星系的识别提供了一种思路。

关键词：LSB，目标检测，YOLO，Darknet-53

ABSTRACT

A low-surface-brightness galaxy, or LSB galaxy, is a diffuse galaxy with a surface brightness that, when viewed from Earth, is at least one magnitude lower than the ambient night sky. LSB galaxies are of great importance to the study of dark matter. Therefore, this paper presents a method to identify low surface brightness galaxies. The enhancement and channel merging of astronomical images, as well as merging and outlier elimination of data table were mainly carried out; With YOLOv3 framework, darknet53 convolution residual network was utilized to train the identification and positioning of LSB galaxy images on the Colab cloud computing platform provided by Google. After repeated experiments with different parameters, the excellent training results of mAP=93.40% (IoU_thresh=0.5) and recall = 94% (conf_thresh = 0.25) were finally obtained, and the astronomical images containing LSB galaxies could be more accurately marked and their central coordinates were given. It provides an idea for the detection of LSB galaxy.

Keywords: LSB, Target detection, YOLO, Darknet-53

目录

一、研究意义：

低表面亮度星系（low-surface-brightness galaxy），或LSB星系，是一种弥漫星系，当从地球观察这些星系时，在夜空的环境中，它们的表面亮度至少比周围的背景天光低一个星等[1]。

对LSB星系的自转曲线测量显示出极高比率的质量-光度比（γ）。这意味着恒星和发光气体对 LSB 星系的整体质量平衡贡献很少。在 LSB 星系中心的恒星中没有大的超高密度恒星，这和普通漩涡星系的核球有很大差异。因此，即使在星系中心似乎也是由暗物质主导的。这使得它们成为研究暗物质的绝佳实验室[1]。

二、现有研究

B. Vollmer，M. Petremand, M. Petremand等人提出了一种基于多尺度马尔可夫模型的低面亮度星系探测算法，称为 MARSIAA (MARkovian Software for Image Analysis in astronauts)。MARSIAA可以同时应用于不同的波段。 它根据图像的面亮度和周围环境，将图像分割成用户定义的多个类——通常，一个或两个类包含 LSB 结构。他们开发了一种名为 DetectLSB的算法，它允许从 MARSIAA 选择的候选源中有效地识别 LSB 星系。这种方法与匹配滤波器的应用和 SExtractor 的优化使用是互补的，具有无标度，可以同时应用于多个波段，并且适合于太空中拥挤的区域的优点。[2]

三、研究过程

3.1 天文图像处理

3.1.1 提取图像

一幅数字化图像在数学上等效为一个矩阵。图像处理的方法很多,对于天文图像而言，常见的处理有图像增强，图像恢复，图像分割等。

对于fits类型的天文图像文件，由文件头和数据组成。在文件头中存储有对该文件的描述，如观测时间、观测对象、拍照温度、曝光时间等信息，而数据部分则保存其对应的的16位图像矩阵。

```
In[3]: rootdir = './target'
   ...: list = os.listdir(rootdir)  # 列出文件夹下所有的目录与文件
   ...: path = rootdir+"/fpC-001035-g3-0133.fit"
In[4]: hdu_list = fits.open(path, cache=True)
   ...: hdu_list.info()
Filename: ./target/fpC-001035-g3-0133.fit
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      73   (2048, 1489)   int16 (rescales to uint16)
```

图 1：fits 信息

可以看到只有一个 PrimaryHDU，而且第一个 HDU 里面就存放了数据。实验数据均为 1489*2048 的矩阵，使用 hdu[0].data 或者 fits.getdata()查看矩阵：

```
In[6]: image_data
Out[6]:
array([[1090, 1101, 1101, ..., 1094, 1098, 1103],
       [1096, 1092, 1094, ..., 1096, 1106, 1101],
       [1099, 1091, 1100, ..., 1088, 1090, 1098],
       ...,
       [1101, 1099, 1103, ..., 1095, 1090, 1094],
       [1097, 1100, 1098, ..., 1104, 1102, 1106],
       [1100, 1099, 1090, ..., 1097, 1100, 1091]], dtype=uint16)
```

图 2：灰度矩阵分布

可以看到矩阵里面为 16 位灰度值。

为了知道图像灰度的总体分布，进行下列四项指标的评估：

```
In[7]: print('Min:', np.min(image_data))
   ...: print('Max:', np.max(image_data))
   ...: print('Mean:', np.mean(image_data))
   ...: print('Stdev:', np.std(image_data))
Min: 1070
Max: 54713
Mean: 1098.8260807116774
Stdev: 178.801973769512
```

图 3：数理统计指标

可以发现大量数据集中在（1098-178，1098+178）附近，而最大值 54713 与平均值相差了 2^15/2^10 = 5 个数量级

用 plt.hist(image_data.flatten(), bins=50) 查看灰度分布的直方图

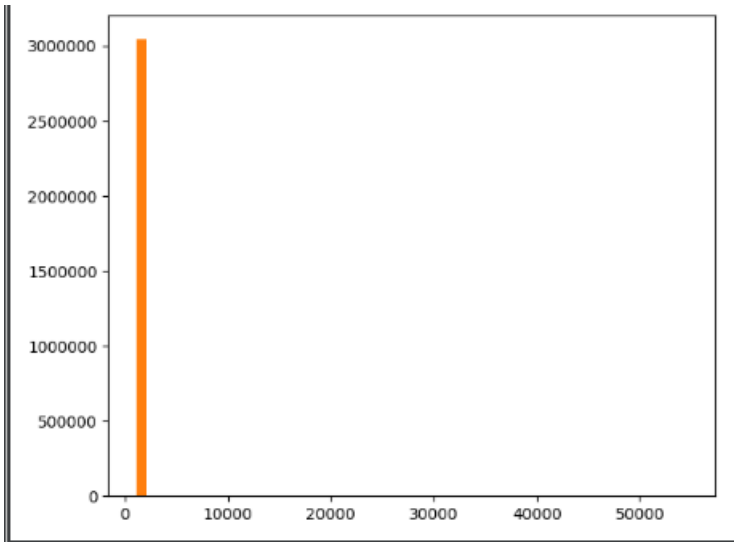


图 4：图像灰度直方图

直观的说明了图像集中分布在均值附近，于是我们一定要对图像进行处理，以防归一化后退化问题。为什么要归一化？为了输出标准 8 位图像，我们不需要那么高的位深度。对 16 位图像进行归一化处理，然后再乘以 255，让像素分布在[0,255]区间内

```
def img_normalization(img):
    '''
    图像归一化
    '''
    img=img.astype(np.float32)
    img[np.isnan(img)] = 0
    if np.amax(img) == 0:
        return img
    else:
        img -= np.amin(img)
        img = img/(np.amax(img)-np.amin(img))
        img *= 255
        return img.astype(np.uint8)
```

图 5：归一化转 8 位图像

所谓退化(degeneration) ，是指图像归一化后，较小部分的值被映射为接近或等于 0（即黑色）的灰度，如果原始图像的大部分像素属于这部分区间，那么这张图像将丢失暗部的细节。如果我们不加处理的输出，那么视觉效果将会这样：

图 6：未处理图像

可以说，这样的图像，连亮度最高的星系也很难用肉眼辨别，更不用说 LSB 星系了

3.1.2 图像增强

方案 1：直方图均衡化（Histogram equalization）

直方图均衡化就是对灰度值 r 进行如下变换 s=T(r)，使得变换后的灰度分布概率均衡的分布在灰度区间中。这样能够发现一些原先肉眼很难发现的细节。实现代码如下：

```
def histeq(img, nbr_bins=65536):
    """ Histogram equalization of a grayscale image. """
    # 获取直方图p(r)
    imhist, bins = histogram(img.flatten(), nbr_bins, normed = True)

    # 获取T(r)
    cdf = imhist.cumsum() # cumulative distribution function
    cdf = 65535 * cdf / cdf[-1]

    # 获取s，并用s替换原始图像对应的灰度值
    result = interp(img.flatten(), bins[:-1], cdf)

    return result.reshape(img.shape), cdf
```

图 7：直方图均衡化

处理后的直方图如下：

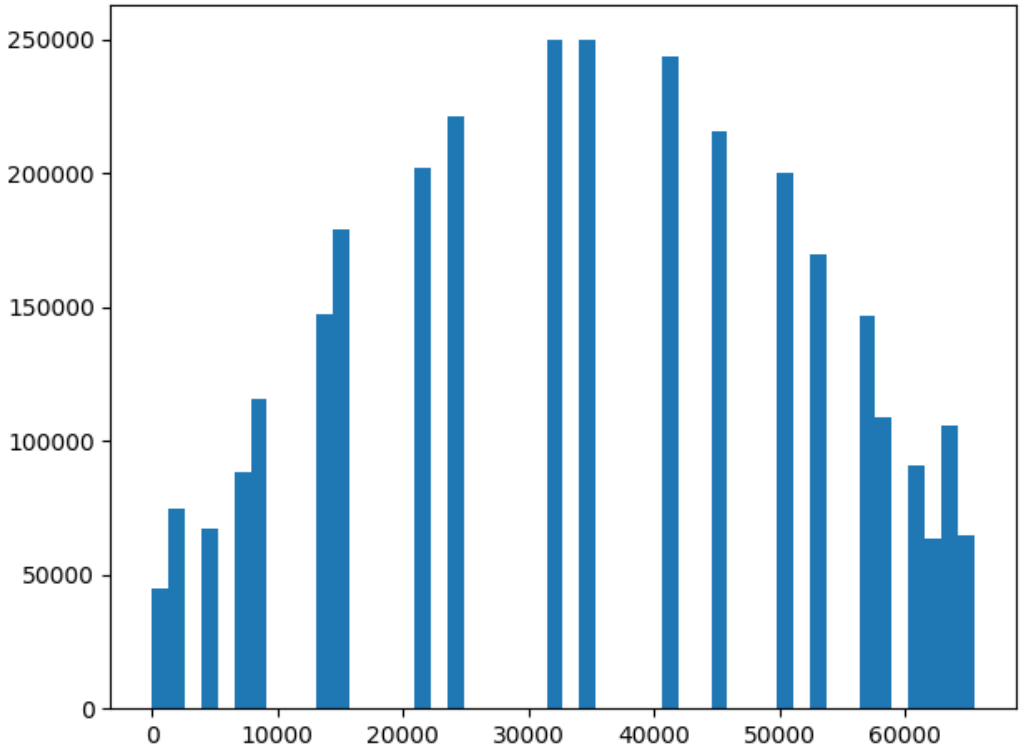


图 8：均衡化后的直方图

保存为 tif 格式，可以看到图像的对比度增加了，原本看不见的星系也清晰可见。

图 9：直方图均衡化后的图片

遗憾的是，这种办法得到的 8 位图像保存为 jpg 格式时会严重失真，且低亮度星表面星系特征不明显。

方案 2：主成分增强法：

这种方法适用于数据集中分布在某值附近，而极端值相差甚远，数量又少。那我们完全可以抛弃这些值。我的方法是，取灰度分布在[mean-std,mean+std]之间的像素，超过或不够的像素改为边缘值。代码也很简单：

```
def limit(image_data):


    mean = np.mean(image_data)
    std = np.std(image_data)
    max = mean+std
    min = mean-std
    image_data[image_data>max] = max
    image_data[image_data<min] = min
    return image_data
```

图 10：主成分增强法

我们将输出的矩阵归一化后转为 8 位图像输出：

图 11：主成分增强后的图片

可以看到，对比度比之前有所降低，保留了更多暗部的细节。

3.1.3 合并通道

对于给定的数据集来说，对于同一幅图像，有 G（green）和 R(red)两个波段的图像。为此，我们需要对相同图像的不同波段进行合并。

有两种方法可以进行通道的合并，一种是使用 Numpy 库中的 np.dstack()进行第三维上的堆叠。第二种是使用 opencv 库种的 cv2.merge()对不同通道进行合并。

这里使用第二种方法，由于 merge 的通道最少为 3，对第三个通道（blue）矩阵置 0

```
b = np.zeros((img1.shape[0], img1.shape[1]), dtype=img1.dtype)
g = np.zeros((img1.shape[0], img1.shape[1]), dtype=img1.dtype)
r = np.zeros((img2.shape[0], img2.shape[1]), dtype=img2.dtype)
#
g[:, :] = img1[:, :]
r[:, :] = img2[:, :]
#
merged = cv2.merge([b, g, r])
```

图 12：合并通道

合并后效果如下：

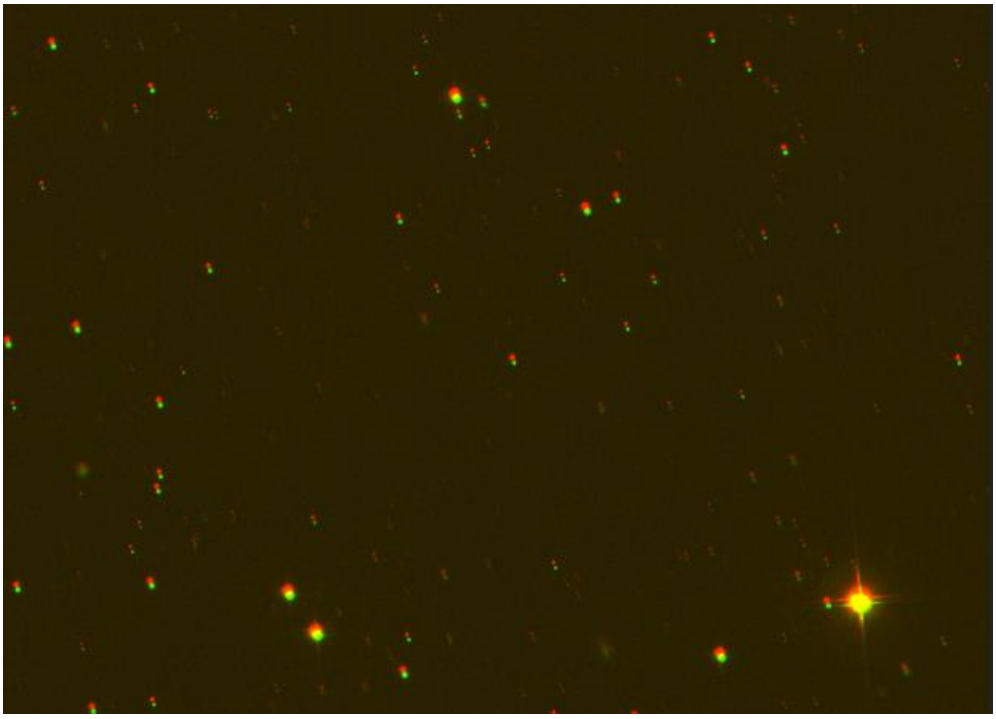


图 13：合并通道后的 8 位 RGB 图像

于是我们只要将 target.csv 中所有出现的文件对于其对应波段的图片合并输出即可。

3.2 数据预处理

3.2.1.数据去重：

给出的 csv 文件中存在 1129 项，但是经过查找重复项，发现有 9 项存在重复，他们在表中的序号为（从 0 起）：

```
281 282 fpC-003630-g1-0200.fit
419 420 fpC-003836-g4-0589.fit
460 461 fpC-003841-g6-0272.fit
583 584 fpC-003971-g2-0056.fit
619 620 fpC-003996-g3-0157.fit
758 759 fpC-004646-g5-0177.fit
794 795 fpC-004674-g6-0046.fit
851 852 fpC-005080-g3-0040.fit
952 953 fpC-005140-g2-0129.fit
```

图 14：重复的表项序号

这意味着，这些图片中存在 2 个 LSB 星系，于是要对其进行合并。

3.2.2 剔除异常值

为了防止脏数据影响训练效果，在制作 label 之前要对数据进行检查。包括 nan 值检查，超限值检查等。经检测，发现序号 266-272 的 LSB 中心坐标记录，远远超过了图像的长宽(1489*2048)，于是将这些项及对应文件剔除。

314744.4116	507442.1913	fpC-00353 fp
199240.6227	413436.1564	fpC-00353 fp
260495.1796	471654.3784	fpC-00353 fp
75920.0404	312534.3111	fpC-00353 fp
62991.83891	299244.3911	fpC-00353 fp
44335.9121	282262.9926	fpC-00353 fp
85055.48857	327149.8104	fpC-00353 fp

图 15：异常数据

3.3 YOLO v3 算法综述

对于目标检测(Object Detection)，从 R-CNN 到 Faster R-CNN 一直采用的思路是 proposal+分类（proposal 提供位置信息，分类提供类别信息）精度已经很高，但由于 two-stage（proposal 耗费时间过多）处理速度不行达不到 real-time 效果。

YOLO 提供了另一种更为直接的思路： 直接在输出层回归 bounding box 的位置和 bounding box 所属的类别，把整张图作为网络的输入，把 Object Detection 的问题转化成一个 Regression 问题

关于 YOLO 算法，由于之前已经有所熟悉，这里不在赘述。这里着重介绍一下 YOLO 的第三个改进版本，YOLOv3 主要调整了网络结构，利用多尺度特征进行对象检测并且对象分类用 Logistic 取代了 softmax。原著论文见参考文献[4]。

3.3.1 YOLO v3 基本结构

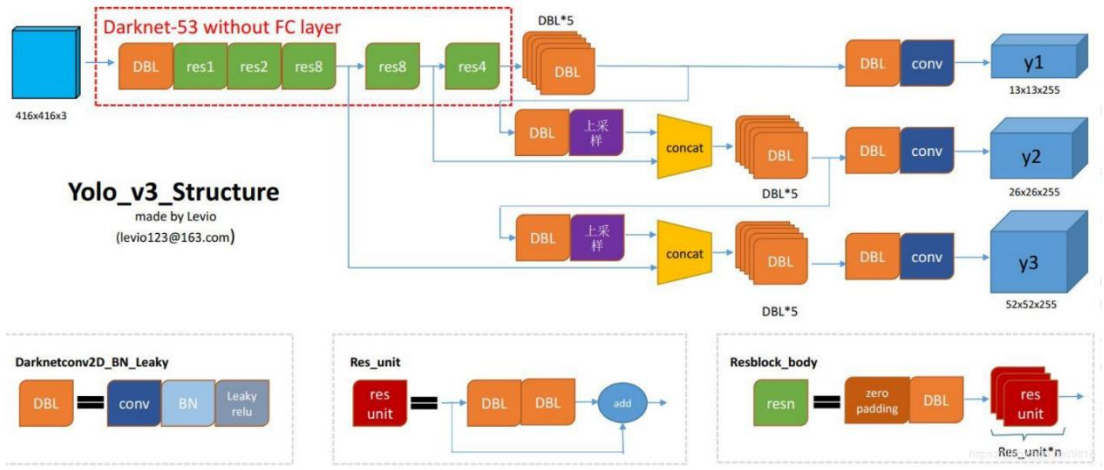


图 16：YOLO_v3 结构图[5]

DBL：代表代码中的 Darknetconv2d_BN_Leaky，是 yolo_v3 的基本组件。就是卷积+BN+Leaky relu。

resn：n 代表数字，有 res1，res2，…，res8 等等，表示这个 res_block 里含有多少个 res_unit

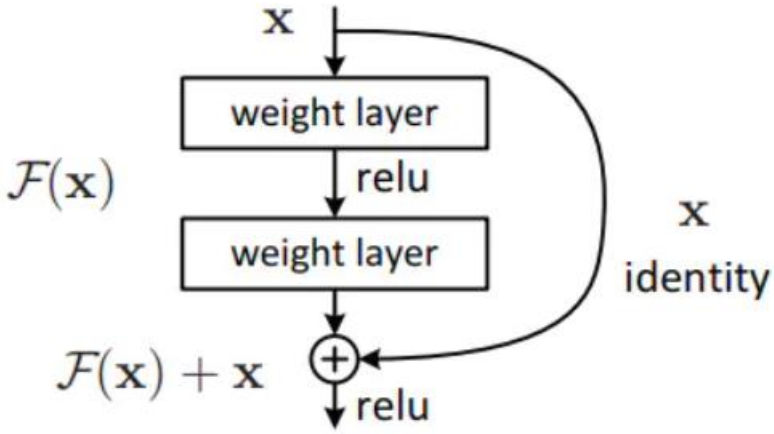


图 17：一个残差组件

concat：张量拼接。将 darknet 中间层和后面的某一层的上采样进行拼接

3.3.2 Darknet-53 卷积残差网络

为了达到更好的分类效果，作者自己设计训练了 darknet-53 卷积网络，大量使用残差的跳层连接，并且为了降低池化带来的梯度负面效果，没有池化层和全连接层，用 conv 的 stride 来实现降采样。在这个网络结构中，使用的是步长为 2 的卷积来进行降采样。每次将边长缩小一半，一共 5 次，即将图像缩小到原图的 1/32

	Type	Filters	Size		Output	
	Convolutional	32	3	3	256	256
	Convolutional	64	3	3 / 2	128	128
	Convolutional	32	1	1		
1	Convolutional	64	3	3		
	Residual				128	128
	Convolutional	128	3	3 / 2	64	64
	Convolutional	64	1	1		
2	Convolutional	128	3	3		
	Residual				64	64
	Convolutional	256	3	3 / 2	32	32
	Convolutional	128	1	1		
8	Convolutional	256	3	3		
	Residual				32	32
	Convolutional	512	3	3 / 2	16	16
	Convolutional	256	1	1		
8	Convolutional	512	3	3		
	Residual				16	16
	Convolutional	1024	3	3 / 2	8	8
	Convolutional	512	1	1		
4	Convolutional	1024	3	3		
	Residual				8	8
	Avgpool		Global			
	Connected		1000			
	Softmax					

图 18：Darknet-53 卷积残差网络[4]

上图的 Darknet-53 网络采用 256*256*3 作为输入，最左侧那一列的 1、2、8 等数字表示多少个重复的残差组件。

3.3.3 多尺度特征对象检测：

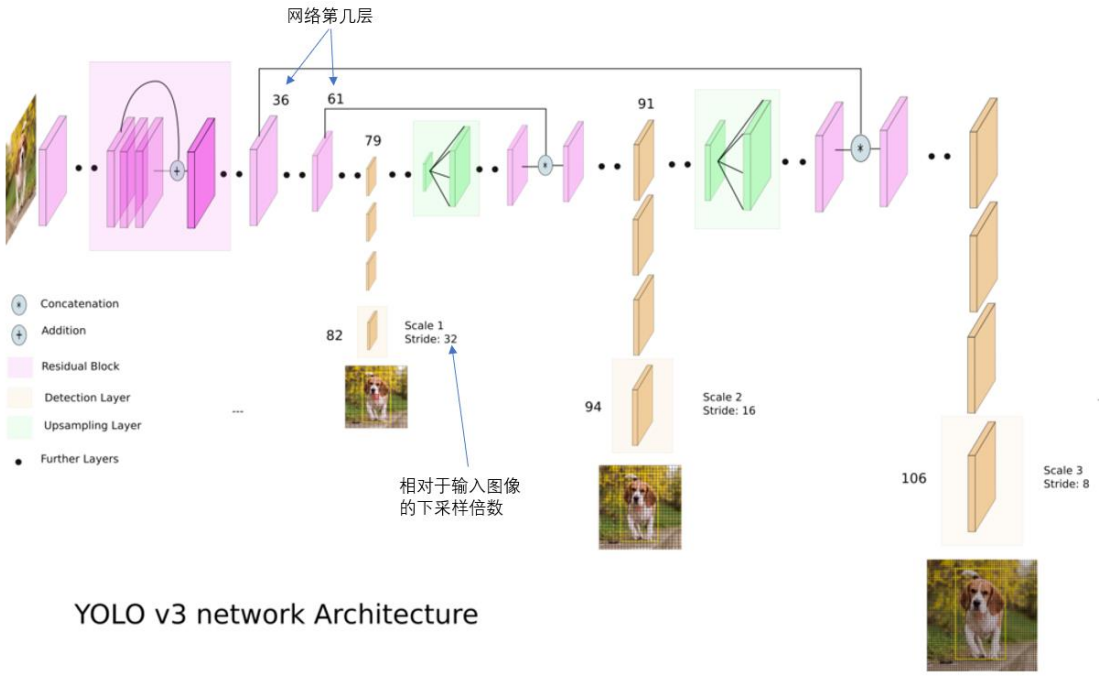


图 19：YOLO v3 网络结构[6]

YOLO v3 采用了 3 个不同尺度的特征图来进行对象检测。卷积网络在 79 层后，经过下方几个黄色的卷积层得到一种尺度的检测结果。相比输入图像，这里用于检测的特征图有 32 倍的下采样。比如输入是 416*416 的话，这里的特征图就是 13*13 了。由于下采样倍数高，这里特征图的感受野比较大，因此适合检测图像中尺寸比较大的对象。

为了实现细粒度的检测，第 79 层的特征图又开始作上采样（从 79 层往右开始上采样卷积），然后与第 61 层特征图融合（Concatenation），这样得到第 91 层较细粒度的特征图，同样经过几个卷积层后得到相对输入图像 16 倍下采样的特征图。它具有中等尺度的感受野，适合检测中等尺度的对象。

最后，第 91 层特征图再次上采样，并与第 36 层特征图融合（Concatenation），最后得到相对输入图像 8 倍下采样的特征图。它的感受野最小，适合检测小尺寸的对象。

3.3.4 9 种尺度的 Anchor Box

随着输出的特征图的数量和尺度的变化，先验框的尺寸也需要相应的调整。YOLO_v3 采用 K-means 聚类得到先验框的尺寸，对应不同尺寸的特征图。

特征图	13*13			26*26			52*52		
感受野	大			中			小		
先验框	(116x90)	(156x198)	(373x326)	(30x61)	(62x45)	(59x119)	(10x13)	(16x30)	(33x23)

图 20：特征图与 Anchor Box[7]

分配上，在最小的 13*13 特征图上（有最大的感受野）应用较大的先验框(116x90)，

(156x198)，(373x326)，适合检测较大的对象。中等的 26*26 特征图上（中等感受野）应用中等的先验框(30x61)，(62x45)，(59x119)，适合检测中等大小的对象。较大的 52*52 特征图上（较小的感受野）应用较小的先验框(10x13)，(16x30)，(33x23)，适合检测较小的对象。

3.3.5 Logistic 对象分类

预测对象类别时不使用 softmax，改成使用 logistic（逻辑回归）对象预测。这样能够支持多标签对象（比如一个人有 Woman 和 Person 两个标签）

3.3.6 网络输出

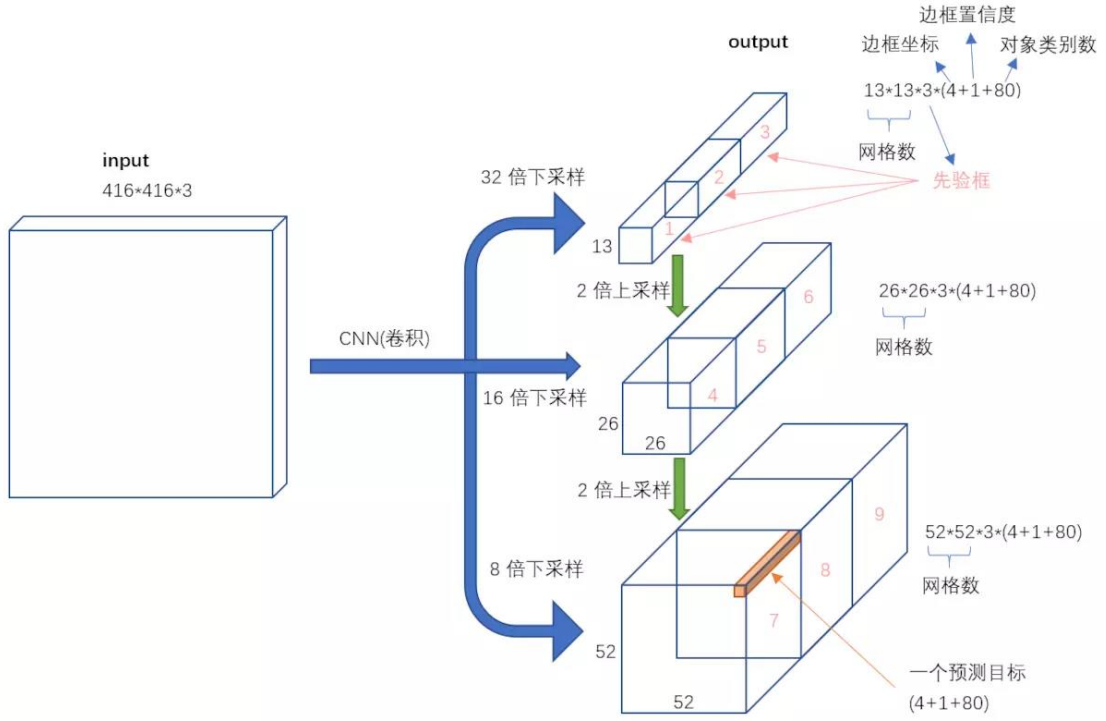


图 21：输入映射到输出[7]

总的来说，对于一个输入图像，YOLO v3 将其映射到 3 个尺度的输出张量，代表图像各个位置存在各种对象的概率。

对于一个 416*416 的输入图像，在每个尺度的特征图的每个网格设置 3 个先验框，总共有 13*13*3 + 26*26*3 + 52*52*3 = 10647 个预测。每一个预测是一个(4+1+80)=85 维向量，这个 85 维向量包含边框坐标（4 个数值），边框置信度（1 个数值），对象类别的概率（对于 COCO 数据集，有 80 种对象）。

对比一下，YOLO2 采用 13*13*5 = 845 个预测，YOLO3 的尝试预测边框数量增加了 10 多倍，而且是在不同分辨率上进行，所以 mAP 以及对小物体的检测效果有一定的提升。

综上所述，YOLOv3 相比于前代的改进，不仅维持了速度快的优势（比 R-CNN 快 1000 倍，比 Fast R-CNN 快 100 倍），并且改善了对于小物体、粗粒度的检测，所以非常适合用于检测 LSB 星系。

3.4 在 Colab 上搭建 YOLO v3 环境

由于电脑性能不足以支撑如此庞大的计算量， 从而选择在谷歌的云平台 Colab 上远程操作 vm 实例训练网络。这部分涉及到的代码量和说明比较多，请参见代码文件中的”YOLOv3_Colab.ipynb” Jupiter 笔记本。

3.5 制作自己的数据集进行训练

感谢 AlexeyAB，他的 GitHub repo 中仔细讲解了 YOLOv3 的配置、运行和改进步骤[8]

3.5.1 制作 Annotation

关于如何制作 YOLO v3 格式的数据集，网上有很多教程[3]。大多数是把用 LabelImg 标注出的 VOC 格式的数据集转为 YOLO v3 格式的数据集。由于已经给出 LSB 的中心坐标，且 LSB 靠肉眼较难识别，所以这里直接以给出的坐标为中心点，以一定大小的矩形框作为 Bounding box，制作标签。下面是 YOLOv3 格式的 label：

```
1 0.716797 0.395833 0.216406 0.147222
0 0.687109 0.379167 0.255469 0.158333
1 0.420312 0.395833 0.140625 0.166667
```

图 22：YOLOv3 格式的 label

其中，第一个参数为类别，后面的参数依次为(x_centre,y_centre,width,height)，而且均为（0-1）之间的小数，为比例值

<x> = <absolute_x> / <image_width>

<height> = <absolute_height> / <image_height>

例如，对于如下 LSB 坐标，设 bounding box 的大小为整幅图像的 10%，有如下 label：

1262.307481	188.9087811	0 0.616361 0.12687 0.05 0.05
1376.975269	1069.247915	0 0.672351 0.718098 0.05 0.05

图 23：label 示例

将制作好的 label 和对应的图片放在一起，保持对应文件名相同，打包成 zip 文件。然后上传到 Google Drive，解压到云服务器的 content/darknet/data/obj 文件夹中备用。

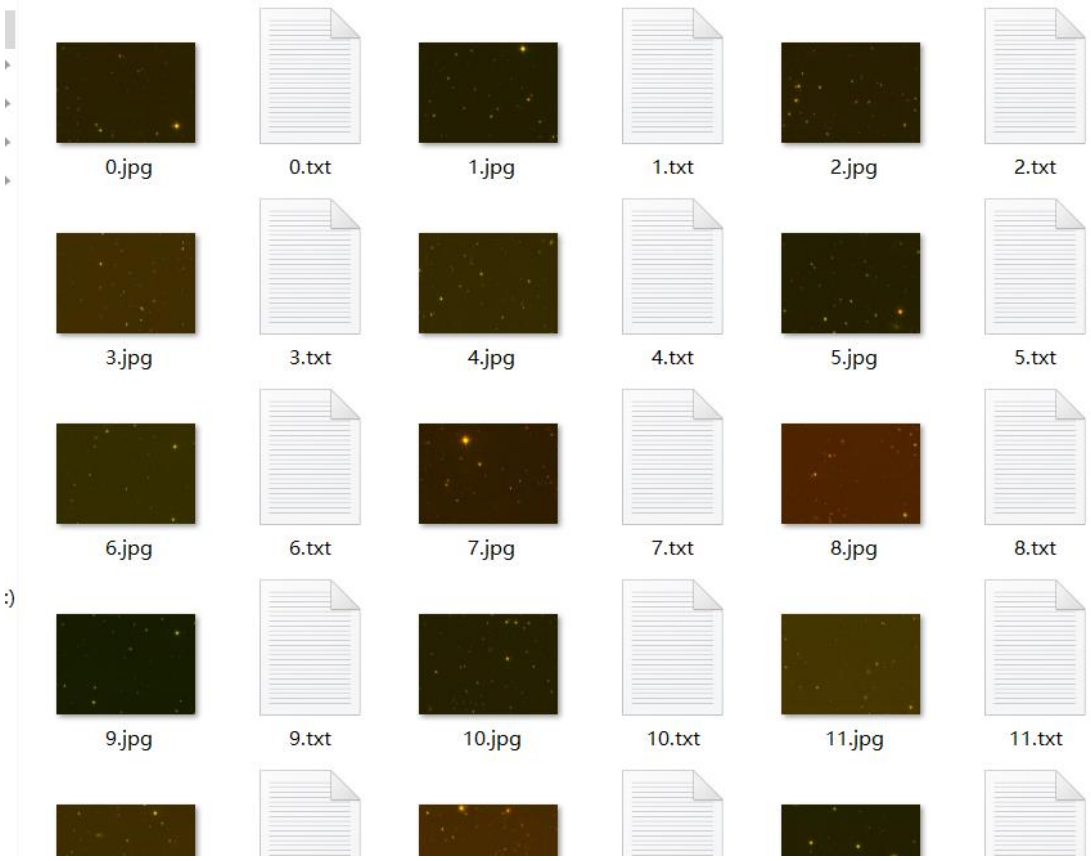


图 24：obj 文件夹

3.5.2 创建 train.txt 和 test.txt

编写 generate_train.py，生成存放训练集和测试集的图片地址的 txt 文件,分配比例为 9:1

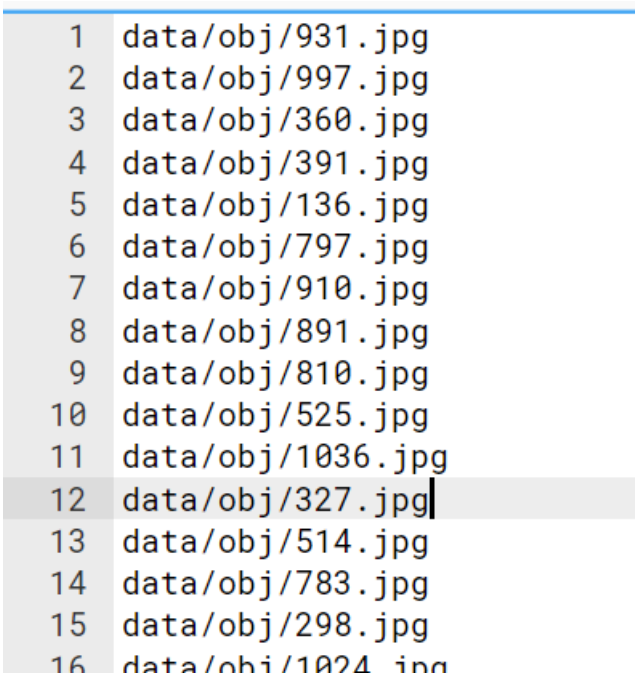


图 25：train.txt

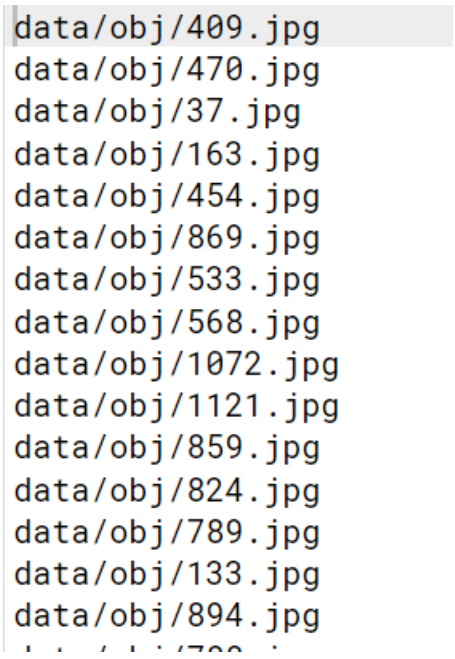


图 26：test.txt

上传到云服务器 darknet/data 备用

3.5.3 创建 obj.names 和 obj.data

obj.names 文件里面列举需要检测目标的名字，obj.data 里面指明类别数，训练集地

址，测试集地址，以及保存权重文件的地址

```
classes = 1
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = /mydrive/yolov3/backup
```

图 27：obj.data

```
LSB
```

图 28：obj.names

3.5.4 修改 yolov3.cfg

原版的 cfg 文件是以 COCO 数据集的 80 个物体编写的，不能直接使用，主要修改下面几个参数：

batch=64
subdivisions=16
…
max_batches = 2000//2000*class_num
steps=1600,1800//max_batches*0.8,max_batches*0.9
[convolutional]
…
filters=18//(5+class_num)*3
…
[yolo]
…
classes=1
…
random=0//random=1 会改变输入图片尺寸训练，增强对不同尺寸图片识别的灵活性，但会增加训练时间
…
[convolutional]
…
filters=18//(5+class_num)*3
…
[yolo]
…
classes=1

…

random=0

…

[convolutional]

…

filters=18//(5+class_num)*3

…

[yolo]

…

classes=1

…

random=0

修改结束后,重命名为 yolov3_custom_4 上传到云服务器 darknet/cfg 备用

3.5.5 开始训练

在 3.4 中配置的 Colab 会话中键入如下代码开始训练：

```
!./darknet detector train data/obj.data cfg/yolov3_custom4.cfg darknet53.conv.74 -dont_show -map
```

其中，detector 后面的参数含义依次为：训练集训练，obj.data,位置 yolov3.cfg 位置，预训练权重，不显示训练中间过程，在最终形成的 loss 趋势图中显示 mAP 变化曲线。

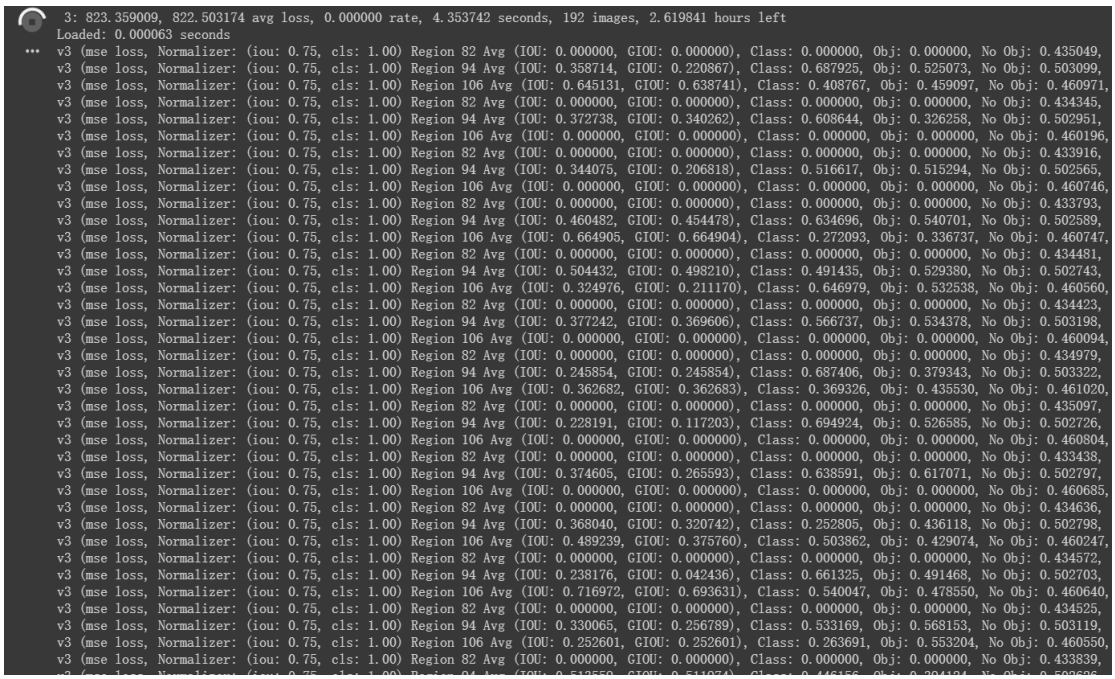


图 29：训练输出

训练时长取决于喂入图片的数量、大小以及 cfg 文件的参数配置，这里我的配置需要 7 小时左右。

3.6 训练结果

3.6.1 绘制 loss-mAP 趋势图

根据训练过程中 log 文件，可以绘制出如下 loss 趋势图：

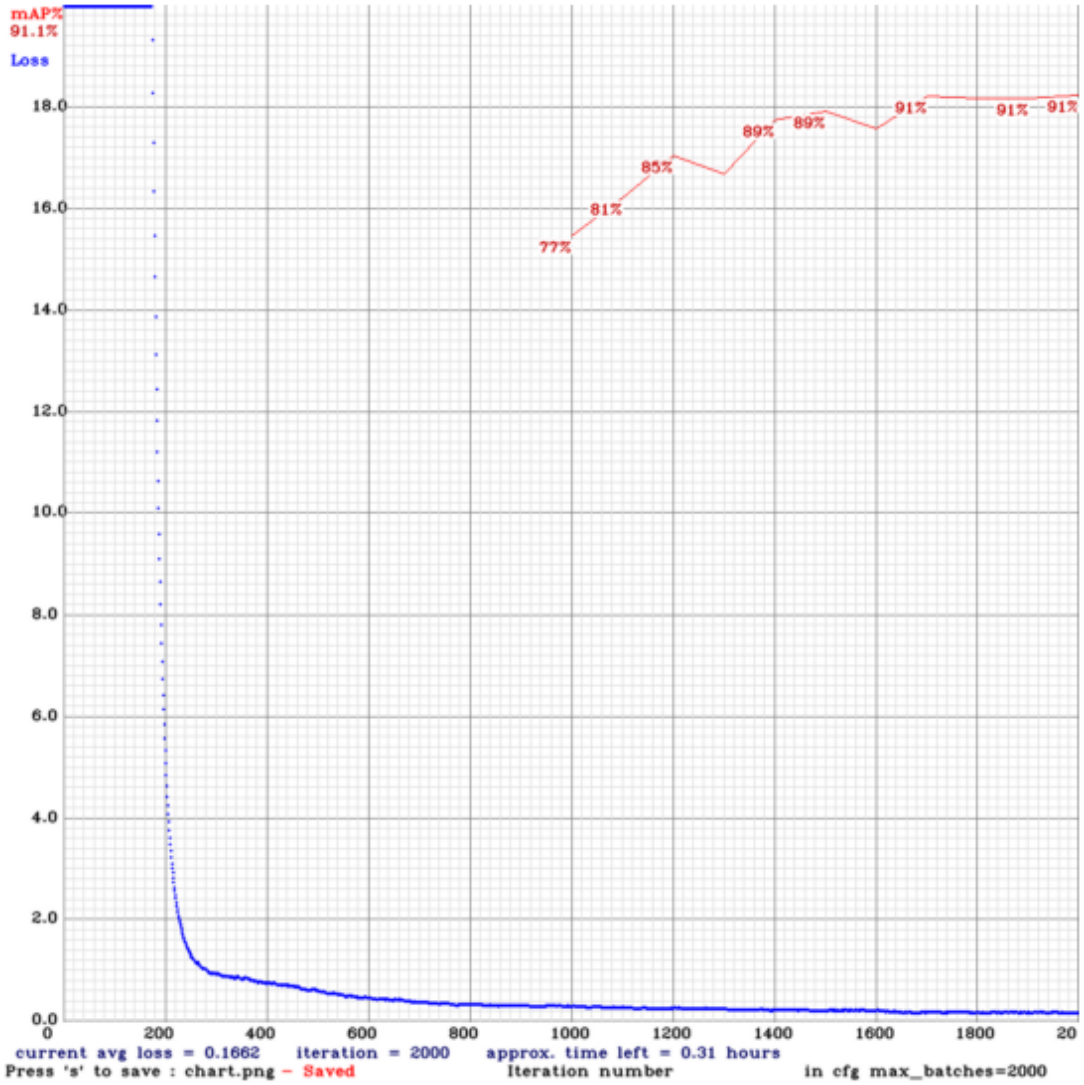


图 27：loss-mAP 趋势图

可以看出，经过 2000 次迭代，最终的 loss 已经接近很接近 0（0.1662），mAP 在 1600 代就开始收敛。说明训练的 max_batches 还可以再往前提一提。

3.6.2 检测单张图片

使用

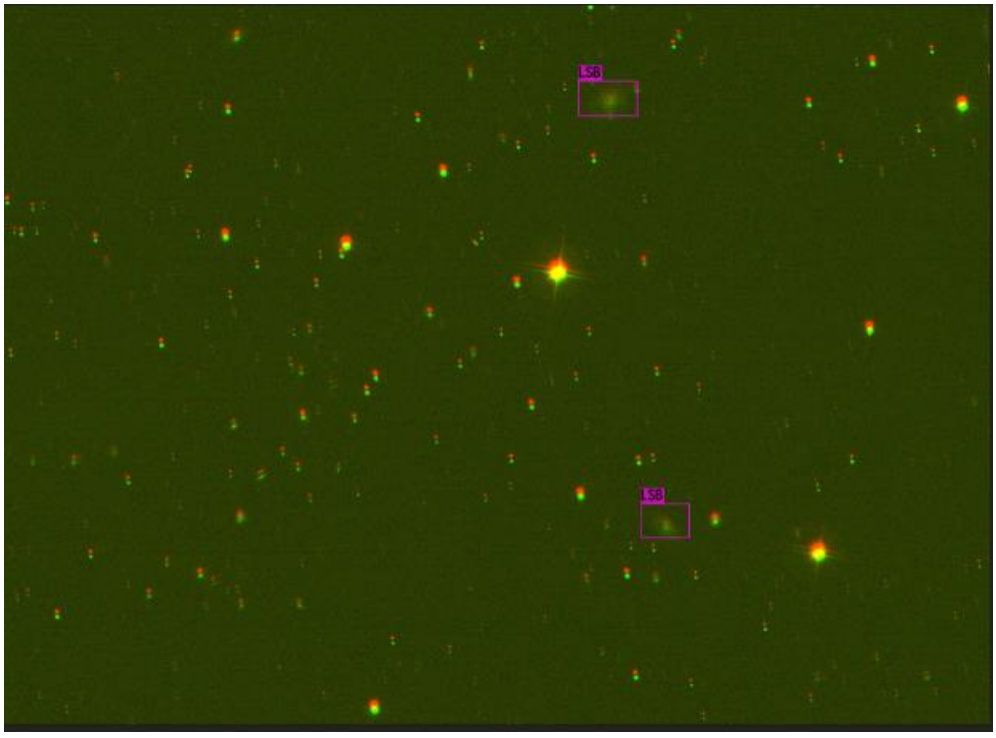
!sed -i 's/batch=64/batch=1/' yolov3_custom4.cfg

!sed -i 's/subdivisions=16/subdivisions=1/' yolov3_custom4.cfg

更改 cfg 文件中的配置，切换到检测模式（batch=1/subdivisons=1 即单张图像一次全部喂入）

键入如下命令对单张图片进行检测（thresh=0.3 即只显示概率在 30%以上的预测框）

```
!./darknet detector test data/obj.data cfg/yolov3_custom4.cfg /mydrive/yolov3/backup/yolov3_custom4_last.weights /mydrive/images/281.jpg -thresh 0.3
imShow('predictions.jpg')
```



1262.307481	188.9087811	fpC-003630-g1-0200.fit	fpC-003630-r1-0200.fit
1376.975269	1069.247915	fpC-003630-g1-0200.fit	fpC-003630-r1-0200.fit

图 28：单张图片识别结果

这与其在表格中的记录相对应

3.6.3 批量检测测试集

键入如下命令，将测试集的识别结果保存到 result.txt 中

```
!./darknet detector test data/obj.data cfg/yolov3_custom4.cfg /mydrive/yolov3/backup/yolov3_custom4_final.weights -dont_show -ext_output < data/test.txt > result.txt
```

```
Enter Image Path: data/obj/409.jpg: Predicted in 33.041000 milli-seconds.
LSB: 87% (left_x:  12  top_y:  917  width:  180  height:  179)
LSB: 90% (left_x: 1031  top_y:  535  width:  218  height:  182)
Enter Image Path: data/obj/470.jpg: Predicted in 32.505000 milli-seconds.
LSB: 28% (left_x: 1414  top_y: 1180  width:  286  height:  174)
LSB: 84% (left_x: 1592  top_y: 1163  width:  220  height:  170)
Enter Image Path: data/obj/37.jpg: Predicted in 32.950000 milli-seconds.
LSB: 57% (left_x:  6  top_y:  705  width:  164  height:  100)
Enter Image Path: data/obj/163.jpg: Predicted in 32.624000 milli-seconds.
Enter Image Path: data/obj/454.jpg: Predicted in 32.704000 milli-seconds.
LSB: 95% (left_x:  123  top_y: 1112  width:  193  height:  151)
Enter Image Path: data/obj/869.jpg: Predicted in 32.624000 milli-seconds.
LSB: 98% (left_x:  722  top_y:  818  width:  224  height:  175)
Enter Image Path: data/obj/533.jpg: Predicted in 32.717000 milli-seconds.
LSB: 96% (left_x:  666  top_y: 1249  width:  254  height:  182)
Enter Image Path: data/obj/568.jpg: Predicted in 32.752000 milli-seconds.
LSB: 96% (left_x: 1630  top_y:  637  width:  190  height:  168)
Enter Image Path: data/obj/1072.jpg: Predicted in 32.854000 milli-seconds
LSB: 60% (left_x:  892  top_y: 1116  width:  204  height:  179)
LSB: 28% (left_x: 1166  top_y: 1287  width:  210  height:  155)
```

图 29：批量识别结果

或使用

```
!./darknet detector valid data/obj.data cfg/yolov3_custom4.cfg /mydrive/yolov3/backup/yolov3_custom4_final.weights -out detect_result.txt
```

输出测试集全部的识别结果，每列含义依次为：图片，概率，xmin, ymin, xmax, ymax

409 0.899268 1032.471924 535.703430 1250.268066 717.227966
409 0.873681 12.528305 917.712341 192.406448 1096.773926
409 0.006644 1240.280029 1100.677612 1472.837646 1213.517944
409 0.002409 1079.510742 471.498016 1290.652832 767.088379
409 0.001587 939.959167 521.404358 1370.072021 761.561218
409 0.001543 5.246750 1453.859619 190.452408 1487.512939
409 0.001363 786.480957 1.226358 980.222168 38.986961
409 0.001288 1401.711670 1415.816772 1643.819580 1489.000000
470 0.842636 1593.022217 1164.495361 1812.832764 1334.426758
470 0.278832 1415.379395 1181.065063 1701.315674 1354.942749
470 0.225003 55.059357 36.607468 237.624481 208.632507
470 0.008697 139.461075 228.855301 305.646851 388.004395
470 0.003018 1259.242188 1134.110596 1795.402832 1336.696533
470 0.001610 1722.809448 162.140045 1951.484497 348.804779
470 0.001594 1.000000 1455.044189 217.535034 1489.000000
37 0.565677 6.563919 705.542419 170.880936 805.648010
37 0.002191 14.014027 719.045044 92.500809 805.592651
37 0.002107 1.000000 697.046753 221.458008 832.294556
37 0.001618 734.183044 342.626038 1026.934937 501.192871
37 0.001314 1460.232422 1193.087769 1661.781738 1312.938843
163 0.124252 1926.927979 818.532959 2041.794922 979.860474
163 0.110653 996.326050 457.461090 1237.148315 616.710693
163 0.063814 133.783630 1275.563843 370.232513 1427.818970

图 30：全部识别结果

3.6.4 模型性能评估

键入

!./darknet detector recall data/obj.data cfg/yolov3_custom4.cfg /mydrive/yolov3/backup/yolov3_custom4_final.weights

进行召回率的评估

0	1	1	RPs/Img: 3.00	IOU: 78.04%	Recall:100.00%
1	2	2	RPs/Img: 7.50	IOU: 77.62%	Recall:100.00%
2	3	3	RPs/Img: 7.67	IOU: 81.89%	Recall:100.00%
3	4	4	RPs/Img: 8.00	IOU: 81.66%	Recall:100.00%
4	4	5	RPs/Img: 8.60	IOU: 65.33%	Recall:80.00%
5	5	6	RPs/Img: 8.17	IOU: 66.01%	Recall:83.33%
6	6	7	RPs/Img: 8.00	IOU: 67.80%	Recall:85.71%
7	7	8	RPs/Img: 8.12	IOU: 69.01%	Recall:87.50%
8	8	9	RPs/Img: 8.33	IOU: 69.69%	Recall:88.89%
9	9	10	RPs/Img: 7.80	IOU: 71.82%	Recall:90.00%
10	10	11	RPs/Img: 7.55	IOU: 71.90%	Recall:90.91%
11	11	12	RPs/Img: 7.33	IOU: 72.67%	Recall:91.67%
12	12	13	RPs/Img: 7.23	IOU: 72.62%	Recall:92.31%
13	13	14	RPs/Img: 7.36	IOU: 72.84%	Recall:92.86%
14	14	15	RPs/Img: 7.13	IOU: 73.60%	Recall:93.33%
15	15	16	RPs/Img: 7.06	IOU: 73.94%	Recall:93.75%
16	16	17	RPs/Img: 7.12	IOU: 74.74%	Recall:94.12%
17	17	18	RPs/Img: 7.00	IOU: 75.21%	Recall:94.44%
18	18	19	RPs/Img: 6.89	IOU: 75.50%	Recall:94.74%
19	19	20	RPs/Img: 6.90	IOU: 75.77%	Recall:95.00%
20	20	21	RPs/Img: 6.86	IOU: 75.78%	Recall:95.24%
21	21	22	RPs/Img: 6.82	IOU: 76.15%	Recall:95.45%
22	22	23	RPs/Img: 6.78	IOU: 76.02%	Recall:95.65%
23	23	24	RPs/Img: 6.71	IOU: 75.99%	Recall:95.83%
24	24	25	RPs/Img: 6.52	IOU: 75.89%	Recall:96.00%
25	25	26	RPs/Img: 6.46	IOU: 75.98%	Recall:96.15%
26	26	27	RPs/Img: 6.44	IOU: 76.31%	Recall:96.30%
27	27	28	RPs/Img: 6.54	IOU: 76.17%	Recall:96.43%
28	28	29	RPs/Img: 6.52	IOU: 76.32%	Recall:96.55%
29	29	30	RPs/Img: 6.50	IOU: 76.38%	Recall:96.67%
30	30	31	RPs/Img: 6.61	IOU: 76.37%	Recall:96.77%
31	31	32	RPs/Img: 6.47	IOU: 76.24%	Recall:96.88%
32	32	33	RPs/Img: 6.39	IOU: 76.14%	Recall:96.97%
33	33	34	RPs/Img: 6.35	IOU: 75.81%	Recall:97.06%
34	34	35	RPs/Img: 6.34	IOU: 76.02%	Recall:97.14%
35	35	36	RPs/Img: 6.33	IOU: 75.97%	Recall:97.22%

图 31：召回率检测

键入

!./darknet detector map data/obj.data cfg/yolov3_custom4.cfg /mydrive/yolov3/backup/yolov3_custom4_final.weights

进行 mAP 等参数的评估

```
calculation mAP (mean average precision)...
124
 detections_count = 326, unique_truth_count = 122
class_id = 0, name = LSB, ap = 91.11%           (TP = 108, FP = 18)

 for conf_thresh = 0.25, precision = 0.86, recall = 0.89, F1-score = 0.87
 for conf_thresh = 0.25, TP = 108, FP = 18, FN = 14, average IoU = 66.80 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.911129, or 91.11 %
Total Detection Time: 9 Seconds
```

图 32：评估指标结果

可以看到置信度阈值 0.25，IoU 阈值为 0.5 的情况下，精度为 86%，召回率为 89%，F1 评分为 87%，真正例 108，假正例 18，假负例 14，平均交并比为 66.80%。对于全体训练集，mAP=91.11%

3.6.7 模型改进

增大输入 Bounding Box 大小为 0.1，输入 resize 变为更高的 608*608（保持 32 的整数倍）提高检测精度，并且设置 random 参数为 1。由于这样的调整导致计算量的猛增，预计训练时间达到了 24 小时以上（Colab 一次会话的最长时限为 12 小时）。于是只应用了其中某些改进，最终模型训练结果如下：

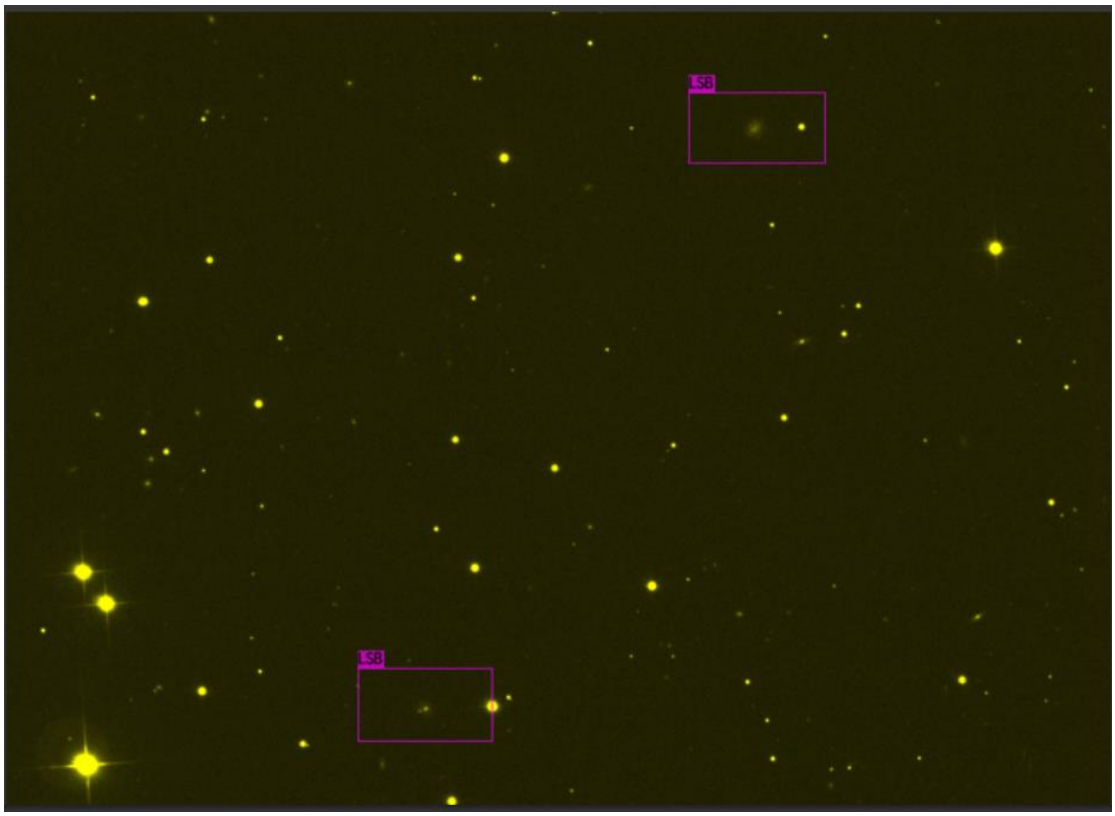
```
calculation mAP (mean average precision)...
128
 detections_count = 286, unique_truth_count = 128
class_id = 0, name = LSB, ap = 93.40%           (TP = 120, FP = 25)

 for conf_thresh = 0.25, precision = 0.83, recall = 0.94, F1-score = 0.88
 for conf_thresh = 0.25, TP = 120, FP = 25, FN = 8, average IoU = 66.49 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.934049, or 93.40 %
```

图 33：最终模型性能

可以观察到，相比第一次训练，精度有所下降，但召回率和 mAP 均有相当的提升。对于 LSB 星系的识别来说，我们更看重的是召回率。故应采用最终模型。

四、研究结果

现在我们已经拥有训练好的模型，只要找出表中没有记载 LSB 坐标的图片进行预测即可。经过筛选，发现只有 fpC-001035-g2-0011/ fpC-001035-r2-0011 这一组没有记录，将其喂入模型：




```
fpC-001035-gr2-0011 0.855715 1266.399536 150.003052 1519.078491 281.723419
fpC-001035-gr2-0011 0.767334 653.282288 1216.318115 902.600769 1352.041748
fpC-001035-gr2-0011 0.024049 172.378387 121.989258 385.647919 251.328186
fpC-001035-gr2-0011 0.002061 934.043213 259.033325 1227.636963 384.670227
fpC-001035-gr2-0011 0.001252 1666.375000 718.696655 1917.474121 861.770752
fpC-001035-gr2-0011 0.001000 266.548920 1.061590 483.358063 34.249916
```

图 34：LSB 检测结果

显然，该幅图像的 LSB 星系有两个，若取检测框中心作为 LSB 星系的坐标，则其坐标分别为（1392.73902，215.863236），（777.94153，1284.17993）

五、总结与展望

对于前文所述的 LSB 星系的识别，出现了召回率比精确率高的情况。初步认为，是由于候选框过大，导致虽然包含 LSB 的概率很高，但是对于 LSB 的精确定位表现却不如人意。换

言之，以候选框中心作为 LSB 位置的朴素算法还比较粗糙。囿于计算量的限制，本文也没有进行更高分辨率的输入和更合适的 Bounding Box 的选取。

在模型的改进方面，或许可以尝试使用最新开源的 YOLOv4 模型[9]，其选择更优的 CSPDarkNet53 作为检测模型的 backbone，具有更高的输入分辨率来检测小目标；更多的层，使之具有更大的感受野，以及更多的参数，更大的模型可以同时检测不同大小的目标。在此基础上，作者添加了 SPP 模块，因其可以提升模型的感受野、分离更重要的上下文信息、不会导致模型推理速度的下降；与此同时，作者还采用 PANet 中的不同 backbone 级的参数汇聚方法替代 FPN。

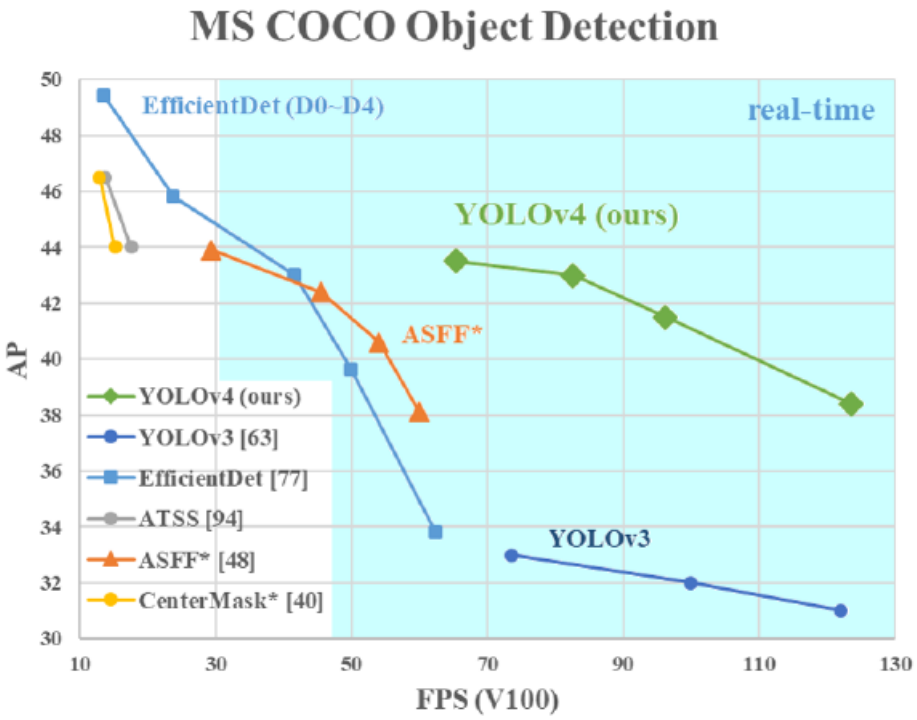


图 35：YOLOv4 与其他目标检测模型对比图[9]

从最终性能上看，对比当前最优目标检测器，YOLOv4 在取得与 EfficientDet 同等性能的情况下，速度是 EfficientDet 的二倍；与 YOLOv3 相比，新版本的 AP 和 FPS 分别提高了 10% 和 12%。以上种种优势，相信能在缩小 BoundingBox 的基础上取得更快、更精准的 LSB 星系识别结果。

六、致谢

感谢研究过程中衣振萍老师的悉心指导，StackOverflow 上相关问题的解答者以及 Github 的开源贡献者们！

附录：参考文献

[1]https://baike.baidu.com/item/%E4%BD%8E%E8%A1%A8%E9%9D%A2%E4%BA%AE%E5%BA%A6%E6%98%9F%E7%B3%BB/10853956?fr=aladdin

[2] Vollmer, B. et al. “SIMULTANEOUS MULTI-BAND DETECTION OF LOW SURFACE BRIGHTNESS GALAXIES WITH MARKOVIAN MODELING.” The Astronomical Journal 145.2 (2013): 36. Crossref. Web.

[3] https://blog.csdn.net/shuiyixin/article/details/82623613

[4] Joseph Redmon and Ali Farhadi.YOLOv3: An Incremental Improvement[J].Computer Vision and Pattern Recognition (cs.CV),2018,1804.02767:.

[5]https://blog.csdn.net/leviopku/article/details/82660381

[6]https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

[7]https://www.jianshu.com/p/d13ae1055302

[8] https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects

[9]Alexey Bochkovskiy and Chien-Yao Wang and Hong-Yuan Mark Liao:YOLOv4: Optimal Speed and Accuracy of Object Detection[J]. Computer Vision and Pattern Recognition (cs.CV),2020