

计算机视觉大作业

院 系 计算机科学与技术学院

专 业 人工智能创新班

学生姓名 赵家琛

学 号 162050121

**2023** 年 **5 月 15 日**

本次计算机视觉课设题目给的很多，这也给了我们选择的空间，可以选择自己感兴趣的，在任务1当中我 选择了暗通道先验去雾霾算法，在任务2当中我选择了目标检测、语义分割和人脸识别算法，目标检测算 法选用了Yolov4、faster rcnn两个模型进行对比试验，语义分割算法选用pspnet、deeplabv3+两个模 型进行对比试验，人脸识别算法中直接使用了Retinaface进行人脸目标检测并用Facenet进行人脸识别直 接看识别的效果，并没有作对比实验。

**1.** 低层视觉及传统方法

**B.** 暗通道先验及图像去雾霾

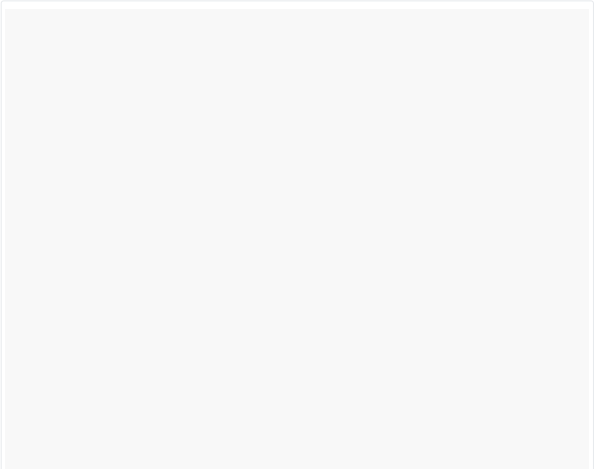
①模型简介

暗通道先验及图像去雾霾算法是一种简单而有效的通过暗通道来去除单一输入图像的雾霾的方法，给出 该算法的流程图如下，具体实现细节参考关键代码中的注释和详解。



②关键代码

图1：暗通道先验去雾霾算法流程图



import cv2 as cv

import numpy as np

def Dark\_Channel(image\_min, r = 7):

return cv.erode(image\_min, np.ones((2 \* r + 1, 2 \* r + 1))) #最小值滤波，滤波的 半径由窗口大小决定

def Guided\_Filter(img, p, r, eps):

#blur(src, dst, ksize, anchor, borderType)等价于boxFilter(src, dst, src.type(), anchor, true, borderType)

mean\_I = cv.blur(img, (r, r)) #I的均值平滑

mean\_p = cv.blur(p, (r, r)) #p的均值平滑

mean\_II = cv.blur(img \* img, (r, r)) #I\*I的均值平滑

mean\_Ip = cv.blur(img \* p, (r, r)) #I\*p的均值平滑

var\_I = mean\_II - mean\_I \* mean\_I #方差

cov\_Ip = mean\_Ip - mean\_I \* mean\_p #协方差

a = cov\_Ip / (var\_I +eps) #相关因子a

b = mean\_p - a \*mean\_I #相关因子b

mean\_a = cv.blur(a, (r, r)) #对a进行均值平滑

mean\_b = cv.blur(b, (r, r)) #对b进行均值平滑

q = mean\_a \* img + mean\_b

return q

def get\_A\_t(Image, img\_origin, V):

#输入：Image最小值图像，img\_origion原图，w是t之前的修正参数，t0阈值，V导向滤波结果 #对于灰度图像：bright = R = G = B

#对于RGB图像，bright = (0.114 \* B) + (0.587 \* G) + (0.299 \* R)

rows, cols, channels = img\_origin.shape

size = rows \* cols

list = [0 for i in range(size)]

m = 0

for t in range(0, rows):

for j in range(0, cols):

list[m] = Image[t][j]

m = m+1

list.sort(reverse=True) #降序排序

index =int(size \* 0.001) #从暗通道中选取亮度最大的前0.1%

Threshold = list[index+1] #阈值，后面要比这个大才行

A = 0

for i in range(0, rows):

for j in range(0, cols):

B = img\_origin[i][j][0]

G = img\_origin[i][j][1]

R = img\_origin[i][j][2]

bright = 0.114 \* B + 0.587 \* G + 0.299 \* R

if Image[i][j] > Threshold and bright > A:

A = bright

t = 1 - 0.95 \* (V/A)

t = np.maximum(t, 0.1) #把数组中小于某个值的数都设为0.1

return A, t

def repair(Image, t, A):

rows, cols, channels = Image.shape

J = np.zeros(Image.shape)

for i in range(0, rows):

for j in range(0, cols):

for c in range(0, channels):

t[i][j] = t[i][j] -0.18 #调参感觉舒服多了

J[i][j][c] = (Image[i][j][c]-A/255.0)/t[i][j]+A/255.0

return J

img = cv.imread('./1/1.png')

img\_arr = np.array(img/255.0) #归一化

img\_min = np.min(img\_arr, 2) #求出每个像素RGB分量中的最小值，存入一副和原始图像大小相同的 灰度图中

img\_dark = Dark\_Channel(img\_min)

A, t = get\_A\_t(img\_dark, img, Guided\_Filter(img\_dark, img\_min, r=75, eps=0.001)) target = repair(img\_arr, t, A)

cv.imshow('Origin', img)

cv.imshow('Target', target)

cv.waitKey()

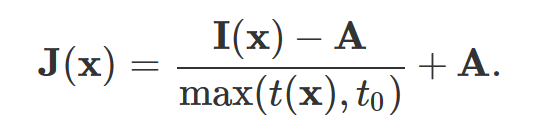
整体代码实现就按论文说到的一样，流程如下：

先用np.min(img\_arr, 2)函数求出每个像素RGB分量中的最小值，存入一副和原始图像大小相同的灰度图中；

调用Dark\_Channel函数，以15\*15的窗口进行最小值滤波，得到暗通道图像；

然后从暗通道图中按照亮度的大小提取最亮的前%0.1像素，然后，在原始有雾图像中寻找对应位置上的 具有最高亮度的点的值，并以此作为A的值；

在按照论文中给出的公式求出t的值(引入导向滤波，得到更好的透射率图)；

最后以 **[1]** 来恢复得到去雾处理后的图像。

③实验设置

按照论文的描述封装好函数，输入一个图片即可获得结果。

④可视化及定量实验结果展示及说明

可以发现去雾霾的结果还是比较好的（图2），主要体现在原本在雾霾之后的物体变得清晰了很多，当然效果和理想的还是有一定差距，并且整张图片的颜色格调差距很大，但我不太能理解，我按照论文的步骤一步步来的，参数也基本都和论文中一致。

然后我在网上找到了近些年的图像去雾代码（效果如图3），发现和我自己写的代码有一些区别：

1. 在最小值滤波上增加了腐蚀（cv2.erode）
2. 大气光照方面原文中计算大气光照时是计算图像亮度的最大值，而后面的代码则是直接取图像RGB的均值与V1相同时的RGB值的最大值作为大气光照。
3. 论文中采用的是对每个RGB通道分别进行修复，而后面的代码则将所有RGB通道的修复结果进行颜色校正。
4. 我自己写的代码引导滤波是最简单的中规中矩的公式代码化，后面的代码直接调用opencv的库函数，可能在某些方面的细节处理要更优一些

还发现了一个比较好玩的基于pytorch的代码，放在了我的仓库【链接】，但是效果一般般

⑤对实验结果现象的原理性分析

在大多数不覆盖天空的局部区域中，一些像素(称为暗像素)在至少一个颜色(RGB)通道中通常具有 非常低的强度，由此可以看出暗通道先验去雾霾算法是基于一种统计信息的方法。

在朦胧的图像中，这些暗像素的强度主要由空气光影响，结合雾图形成模型，这些暗像素可以直接提供 雾霾传输的精确估计，结合无雾成像模型可以恢复出高质量的无雾图像。

一开始做的时候，常常会有最后生成的无雾图像是完全黑的图片，观察公式中有一步是矫正t的值，即取 实际t的值和0.1之间较大的一个，我输出了一些t值发现都比0.1大，就对所有的t同减去一个值，发现效 果也是一点点在变好的，后来上课时，听同学分享说可以调整w的值，回来试了几个值之后，效果不是 很理想，就没有花过多时间再去调参了。

1. 高层视觉及传统**/**深度学习方法

环境

详情见requirements.txt,仅列出几个容易冲突的包，建议创建虚拟环境

|  |
| --- |
| python==3.6  scipy==1.2.1  numpy==1.17.0  matplotlib==3.1.2  opencv\_python==4.1.2.30  torch==1.2.0  torchvision==0.4.0  tqdm==4.60.0  Pillow==8.2.0  h5py==2.10.0 |

软件

 pycharm+anaconda

**E.**语义分割

图像是由许多像素组成，语义分割是对图片的每个像素都做分类。

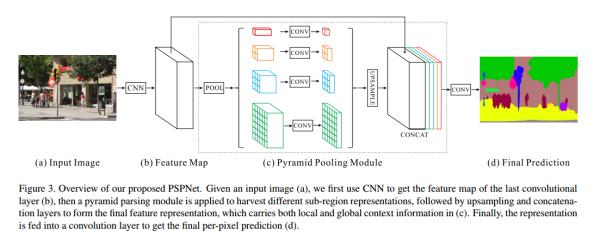
**PSPnet**

①模型简介

PSPnet模型最主要的特点是采用了PSP模块。

该模型提出的金字塔池化模块(Pyramid Pooling Module)能够聚合不同区域的上下文信息，从而提高 获取全局信息的能力。

PSP结构的功能是将获取到的特征层划分成不同大小的网格，每个网格内部各自进行平均池化。实现聚 合不同区域的上下文信息，从而提高获取全局信息的能力。 PSP结构典型情况下，会将输入进来的特征 层划分成6x6 ，3x3 ，2x2 ， 1x1的网格，对应了下图中的绿色、蓝色、橙色、红色的的输出：



我们从左往右看这幅结构图，首先要对输入图像进行一个不失真的resize (通过给周围加灰条的操 作)，在代码实现中为letterbox\_image函数；将resize后的网络输入到主干提取网络中进行特征提取， 获取一个特征层；将feature map分为两部分，一部分传入PSP模块中，一部分直接传入到后一个模块， PSP模块的作用上面已经说到过，将PSP模块输出的特征层进行一个上采样后与传入到后一个模块的特征 进行堆叠；利用获取到的特征上采样后进行对每一个像素的分类。

**a.**主干网络介绍

PSPnet在论文中采用的是Resnet系列作为主干特征提取网络，由于算力限制我自己训练使用的是 MobileNetv2模型(后面Deeplabv3+中也用的这个模型，刚好进行一个对比)。

MobileNet模型是Google针对手机等嵌入式设备提出的一种轻量级的深层神经网络， MobileNetv2是 MobileNet的升级版，它具有两个特征点：

1、 Inverted residuals，在MobileNetv2网络部分，采用Inverted residuals结构，先利用1x1卷积升 维，然后再利用3x3网络卷积，最后再利用1x1卷积降维，即先进行扩张，再进行压缩。

2、 Linear bottlenecks，为了避免Relu对特征的破坏，在利用1x1卷积降维后，不再进行Relu6层，直接 进行残差网络的加法。

**b.**加强特征提取结构

PSPNet所使用的加强特征提取结构是PSP模块，将获取到的特征层划分成不同大小的区域，每个区域内 部各自进行平均池化，实现聚合不同区域的上下文信息，从而提高获取全局信息的能力。具体细节前面 已经提到过，此处不予赘述。

**c.**利用特征获得预测结果

利用特征获得预测结果的过程先利用一个3x3卷积对特征进行整合，然后再利用一个1x1卷积进行通道调 整，调整成Num\_Classes (即类别数量)，最后再利用resize进行上采样使得最终输出层，宽高和输入 图片一样。

②关键代码

github源码地址： <https://github.com/bubbliiiing/pspnet-pytorch>

a.train.py文件用于训练，默认参数已经对应voc数据集所需要的参数，只要在其中进行修改backbone和 model\_path即可运行进行训练。

b.predict.py文件用于预测，修改mode的参数可以选择测试fps和视频检测，不修改就直接进行语义分 割。

c.get\_miou.py文件用于评估，设置num\_classes为预测的类的数量加1， name\_classes为需要去区分的 类别，直接运行即可获得miou大小。

③实验设置

数据及使用细节：

本文使用VOC格式进行训练，下载处理过的VOC12+07的数据集。

语义分割模型训练的文件分为两部分：原图和标签，原图就是普通的RGB图像，标签就是灰度图或者8位 彩色图。原图的shape为[height, width, 3]，标签的shape就是[height, width]，对于标签而言，每个像 素点的内容是一个数字，比如0、 1、2、3、4、5…… ，代表这个像素点所属的类别。

语义分割的工作就是对原始的图片的每一个像素点进行分类，所以通过预测结果中每个像素点属于每个 类别的概率与标签对比，可以对网络进行训练。

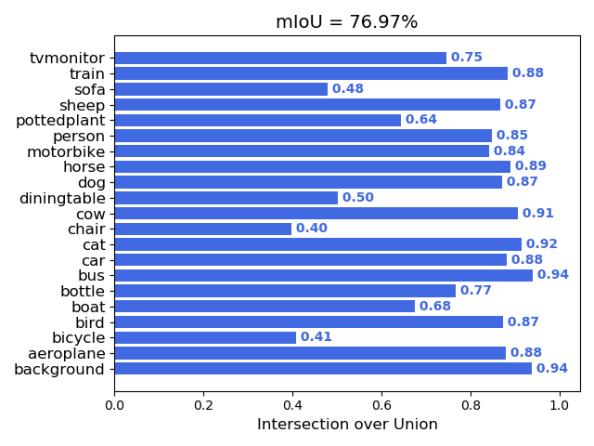
训练前将图片文件放在VOCdevkit文件夹下的VOC2007文件夹下的JPEGImages中。

训练前将标签文件放在VOCdevkit文件夹下的VOC2007文件夹下的SegmentationClass中。

参数设置：

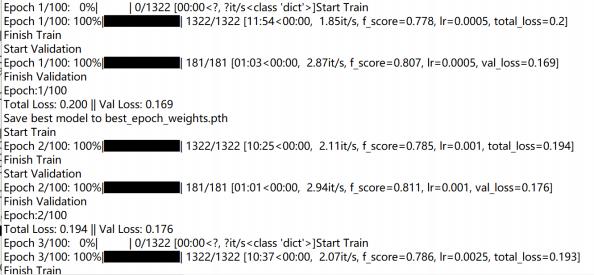
|  |  |  |
| --- | --- | --- |
| ---------------------------------------------------------------------- | | |
| | | keys | | values| |
| ---------------------------------------------------------------------- | | |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | | | num\_classes | | | 21 | | | | | | backbone | | | mobilenet | | | | | | model\_path | | | model\_data/pspnet\_mobilenetv2.pth | | | | | | input\_shape | | | [473, 473] | | | | | | Init\_Epoch | | | 0 | | | | | | Freeze\_Epoch | | | 50 | | | | | | UnFreeze\_Epoch | | | 100 | | | | | | Freeze\_batch\_size | | | 8 | | | | | | Unfreeze\_batch\_size | | | 4 | | | | | | Freeze\_Train | | | True | | | | | | Init\_lr | | | 0.01 | | | | | | Min\_lr | | | 0.0001 | | | | | | optimizer\_type | | | sgd | | | | | | momentum | | | 0.9 | | | | | | lr\_decay\_type | | | cos | | | | | | save\_period | | | 5 | | | | | | save\_dir | | | logs | | | | | | num\_workers | | | 4 | | | | | | num\_train | | | 10582 | | | | | | num\_val | | | 1449 | | | | | |
| ---------------------------------------------------------------------- | | |

由于电脑算力的问题，并没有选用论文中提到的resnet50，而是采用了mobilenetv2为主干提取网路， 所以效果并没有论文中的那么好(比较mIoU)，采用resnet50的权重得到mIoU值如下，具体mIoU的 含义后续会介绍。



实验过程：

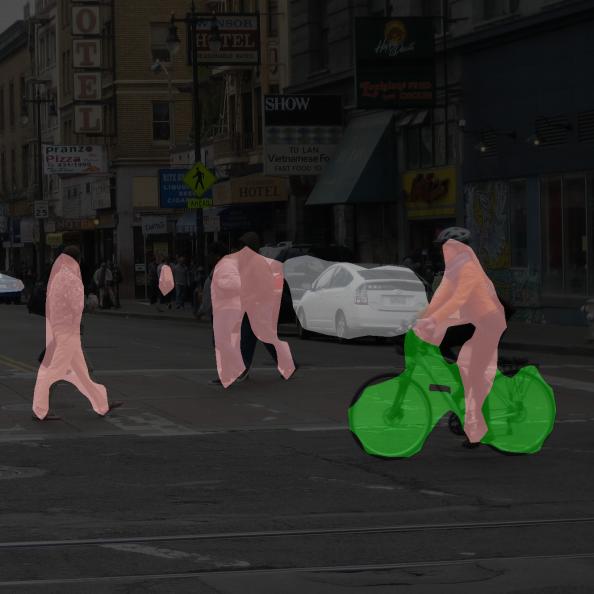
该模型自己训练了一下，训练时长大概3天不到，训练结果如下(截取一小部分)：



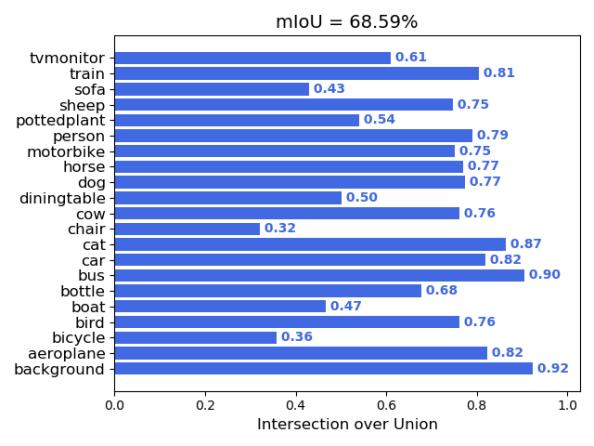
训练完的权重的log文件夹内，后续直接调用就行。

④可视化及定量实验结果展示及说明

启动predict.py程序得到语义分割结果如下：



启动get\_miou.py文件得到miou的值如下：



mIoU意为平均交并比，也就是交和并的比值。在语义分割的问题中，单类的交并比就是该类的真实标签 和预测值的交和并的比值。假设在图中， A为真实标签，占据了图像的一定区域； B为预测结果，占据了 图像的一定区域。那么中间的TP部分(即A和B共同占据的部分)就是真实标签和预测值的交，图像A和 B所占据的部分就是真实标签和预测值的并。

而mIoU就是该数据集中的每一个类的交并比的平均。

计算公式如下：

i表示真实值， j表示预测值 ，表示将i预测为j







显然mIoU的值越大越好，实验结果69.59%还不错，但是和DeeplabV3+比还是差了点。

⑤对实验结果现象的原理性分析

该模型主要功臣就是PSP模块，该模块在不同层次上融合feature，达到语义和细节的融合，与之前的一 些语义分割模型相比，优越性体现在能很好地获得图像的局部和全局特征(pooling层)，原论文在这块 也是做了很多对比实验，首先是对比实验看平均池化和最大池化哪个好，之后也结合了几个trick (数据 增强、辅助loss等)。

**DeeplabV3+**

①模型简介

DeeplabV3+被认为是语义分割的新高峰，因为这个模型的效果非常好。 DeeplabV3+主要在模型的架构 上进行了一些改进，引入了可任意控制编码器提取特征的分辨率，通过空洞卷积平衡精度和耗时。

我们来看一下整体模型图：

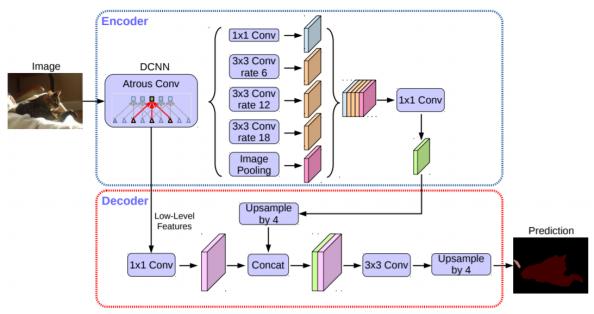


图3： DeeplabV3+模型整体架构图 [3]

我们可以发现整体模型可以分成两部分，分别是Enocoder和Decoder，是一个编码+解码的过程，利用 Encoder可以得到输入图片的特征情况，利用这个Decoder可以将获取到的特征进行解码，得到预测结 果(输入图片每个像素点对应的种类)。

输入的图片传入到Encoder里，先利用DCNN提取特征，获取两个特征层，第一个特征层(浅特征层) 直接传入到Decoder里，另一个特征层(比较深的特征层)使用不同膨胀率的膨胀卷积进行特征提取 (提高网络的感受野)，将特征提取的结果进行堆叠，然后用1×1的卷积进行通道数的调整之后进行上 采样，再与浅特征层进行堆叠(特征融合)，再利用3×3的卷积特征提取，经过一次上采样后再进行图 片大小的调整。

**a.**主干网络介绍

与上一个模型一样均使用MobileNetv2，不予赘述。

**b.**加强特征提取结构

在DeeplabV3+中，加强特征提取网络可以分为两部分：

在Encoder中，会对压缩四次的初步有效特征层利用并行的膨胀卷积，分别用不同膨胀率的膨胀卷积进 行特征提取，再进行合并，再进行1x1卷积压缩特征。

在Decoder中，会对压缩两次的初步有效特征层利用1x1卷积调整通道数，再和膨胀卷积后的有效特征 层上采样的结果进行堆叠，在完成堆叠后，进行两次深度可分离卷积块。

这个时候，就获得了一个最终的有效特征层，它是整张图片的特征浓缩。

**c.**利用特征获得预测结果

利用特征获得预测结果的过程先利用一个3x3卷积对特征进行整合，然后再利用一个1x1卷积进行通道调 整，调整成Num\_Classes (即类别数量)，最后再利用resize进行上采样使得最终输出层，宽高和输入 图片一样。

②关键代码

github源码地址： <https://github.com/bubbliiiing/deeplabv3-plus-pytorch>

a.train.py文件用于训练，默认参数已经对应voc数据集所需要的参数，只要在其中进行修改backbone和 model\_path即可运行进行训练。

b.predict.py文件用于预测，修改mode的参数可以选择测试fps和视频检测，不修改就直接进行语义分

割。

c.get\_miou.py文件用于评估，设置num\_classes为预测的类的数量加1， name\_classes为需要去区分的 类别，直接运行即可获得miou大小。

③实验设置

数据及使用细节：

本文使用VOC格式进行训练，下载处理过的VOC12+07的数据集。

语义分割模型训练的文件分为两部分：原图和标签，原图就是普通的RGB图像，标签就是灰度图或者8位 彩色图。原图的shape为[height, width, 3]，标签的shape就是[height, width]，对于标签而言，每个像 素点的内容是一个数字，比如0、 1、2、3、4、5…… ，代表这个像素点所属的类别。

语义分割的工作就是对原始的图片的每一个像素点进行分类，所以通过预测结果中每个像素点属于每个

类别的概率与标签对比，可以对网络进行训练。

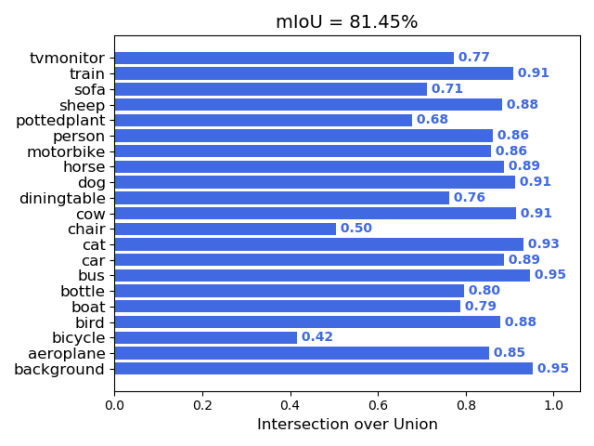
训练前将图片文件放在VOCdevkit文件夹下的VOC2007文件夹下的JPEGImages中。

训练前将标签文件放在VOCdevkit文件夹下的VOC2007文件夹下的SegmentationClass中。

参数设置：

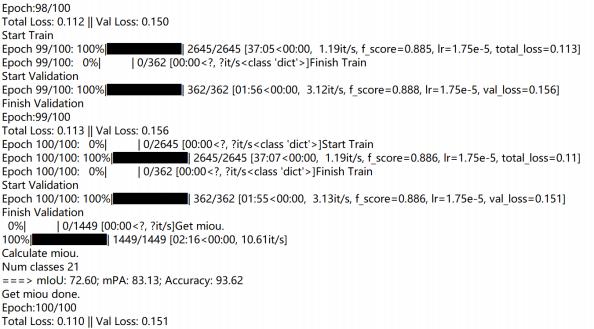
|  |  |  |
| --- | --- | --- |
| ---------------------------------------------------------------------- | | |
| | | keys | | values| |
| ---------------------------------------------------------------------- | | |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | | | num\_classes | | | 21 | | | | | | backbone | | | mobilenet | | | | | | model\_path | | | model\_data/deeplab\_mobilenetv2.pth | | | | | | input\_shape | | | [512, 512] | | | | | | Init\_Epoch | | | 0 | | | | | | Freeze\_Epoch | | | 50 | | | | | | UnFreeze\_Epoch | | | 100 | | | | | | Freeze\_batch\_size | | | 8 | | | | | | Unfreeze\_batch\_size | | | 4 | | | | | | Freeze\_Train | | | True | | | | | | Init\_lr | | | 0.007 | | | | | | Min\_lr | | | 7.000000000000001e-05 | | | | | | optimizer\_type | | | sgd | | | | | | momentum | | | 0.9 | | | | | | lr\_decay\_type | | | cos | | | | | | save\_period | | | 5 | | | | | | save\_dir | | | logs | | | | | | num\_workers | | | 4 | | | | | | num\_train | | | 10582 | | | | | | num\_val | | | 1449 | | | | | |
| ---------------------------------------------------------------------- | | |

同样使用的是MobilenetV2作为主干特征提取网络(刚好与上一个模型进行对比实验)，先来看一下采 用xception的权重得到mIoU值如下。



实验过程：

该模型自己训练了一下，训练时长大概3天不到，训练结果如下(截取一小部分)：



训练完的权重的log文件夹内，后续直接调用就行。

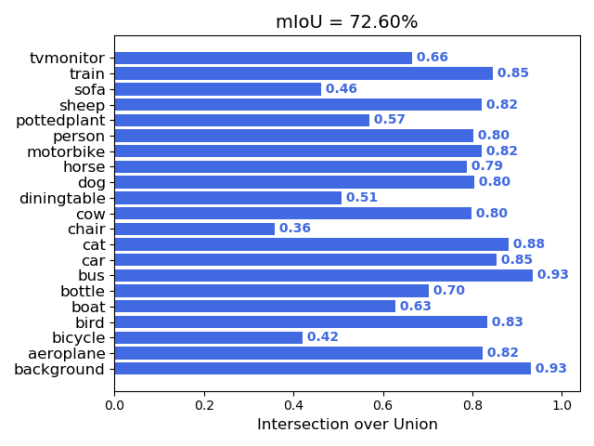
④可视化及定量实验结果展示及说明

启动predict.py程序得到语义分割结果如下：



可以看出来这个效果确实比PSPnet结果好。

我们还是直接启动get\_mIoU程序，和上一个模型进行对比：



mIoU值为72.60%，比PSPnet训练结果好一点。

⑤对实验结果现象的原理性分析

DeepLabv3+使用DeepLabv3作为Encoder模块，并添加一个简单且有效的Decoder模块来获得更清晰 的分割。 Encoder- Decoder网络已经成功应用于许多计算机视觉任务，通常， Encoder- Decoder网络包 含：

·逐步减少特征图并提取更高语义信息的Encoder模块

·逐步恢复空间信息的Decoder模块

这种网络之前我做nlp大量见到Encoder- Decoder网络，原理也很好理解，由Encoder将输入的向量转化 为固定长度的向量表达，最后再经过Decoder相当于一个还原的过程。

**D.**目标检测器

**Yolov4**

①模型简介

Yolo系列算法为one-stage检测算法，其不需要region proposal阶段，直接产生物体的类别概率和位置 坐标值，经过单次检测即可直接得到最终的检测结果，因此有着更快的检测速度。

Yolov4是Yolov3的改进版，在Yolov3的基础上结合了非常多的小Tricks。尽管没有目标检测上革命性的

改变，但是Yolov4依然很好的结合了速度与精度。

列出一些有效的改进如下：

1、主干特征提取网络： DarkNet53 => CSPDarkNet53

2、特征金字塔： SPP ， PAN

3、分类回归层： YOLOv3 (未改变)

4、训练用到的小技巧： Mosaic数据增强、 Label Smoothing平滑、 CIOU、学习率余弦退火衰减

5、激活函数：使用Mish激活函数

关于Yolov4模型最简单清晰的表示： Yolov4 = CSPDarknet53 (主干) + SPP附加模块(颈) + PANet 路径聚合(颈) + Yolov3 (头部)，即由主干特征提取网络Backbone提取特征，然后经过特征金字塔 结构，最后由YoloHead利用获得到的特征进行预测。

**a.**主干网络介绍

使用CSPDarknet53，输入是一个416×416×3的图片，对输入进行不断地特征提取，先用一个Darknet卷 积，再进行几个Resblock\_body (残差网络结构，卷积激活函数+标准化)，经过第一次卷积(Darknet 卷积)之后变成了416×416×32，之后进行Resblock\_body会对高和宽进行压缩(1/2的比例)，通道数 会进行膨胀。不断进行下采样是为了获得更高层次的语义信息，所以最后使用到的也是最后3个特征层。

**b.**特征金字塔结构介绍

在特征金字塔部分， Yolov4结合了两种改进： SPP结构和PANet结构。

SPP结构是对CSPdarknet53的最后一个特征层的卷积里，在对CSPdarknet53的最后一个特征层进行三 次Darknet卷积后，分别利用四个不同尺度的最大池化进行处理，最大池化的池化核大小分别为13x13、 9x9、5x5、 1x1。

PANet结构是对特征的反复提取，从上文可以得知，需要处理的特征层有3个。

②关键代码

github源码地址： <https://github.com/bubbliiiing/yolov4-pytorch>

a.训练的参数较多，均在train.py中，大家可以在下载库后仔细看注释，其中最重要的部分依然是 train.py里的classes\_path。classes\_path用于指向检测类别所对应的txt，这个txt和voc\_annotation.py 里面的txt一样！训练自己的数据集必须要修改！修改完classes\_path后就可以运行train.py开始训练 了，在训练多个epoch后，权值会生成在logs文件夹中。

b.训练结果预测需要用到两个文件，分别是yolo.py和predict.py。在yolo.py里面修改model\_path以及 classes\_path。 model\_path指向训练好的权值文件，在logs文件夹里。 classes\_path指向检测类别所对 应的txt。完成修改后就可以运行predict.py进行检测了。运行后输入图片路径即可检测。

c.运行get\_map.py即可获得评估结果，评估结果会保存在map\_out文件夹中。

③实验设置

数据及使用细节：

数据集的准备

本文使用VOC格式进行训练，训练前需要下载好VOC07+12的数据集，解压后放在根目录

数据集的处理

修改voc\_annotation.py里面的annotation\_mode=2，运行voc\_annotation.py生成根目录下的 2007\_train.txt和2007\_val.txt。

trainval\_percent用于指定(训练集+验证集)与测试集的比例，默认情况下 (训练集+验证集):测试集 = 9:1。train\_percent用于指定(训练集+验证集)中训练集与验证集的比例，默认情况下 训练集:验证集 = 9:1。

参数设置：

[](af://n173/)

|  |  |  |
| --- | --- | --- |
| ---------------------------------------------------------------------- | | |
| | | keys | | values| |
| ---------------------------------------------------------------------- | | |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | | | model\_path | | | model\_data/yolo4\_voc\_weights.pth | | | | | | classes\_path | | | model\_dataoc\_classes.txt| |  | | | | anchors\_path | | | model\_data/yolo\_anchors.txt | | | | | | anchors\_mask | | | [[6, 7, 8], [3, 4, 5], [0, 1, 2]] | | | | | | input\_shape | | | [416, 416] | | | | | | confidence | | | 0.5 | | | | | | nms\_iou | | | 0.3 | | | | | | letterbox\_image | | | False | | | | | | cuda | | | True | | | | | |
| ---------------------------------------------------------------------- | | |

实验过程：

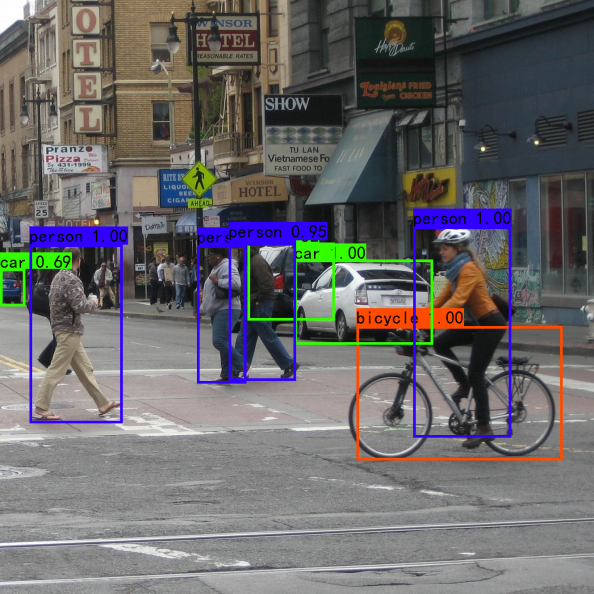
train.py的默认参数用于训练VOC数据集，直接运行train.py即可开始训练；

去yolo.py里面修改model\_path以及classes\_path，然后运行predict.py进行预测；

评估的话直接运行get\_map即可。

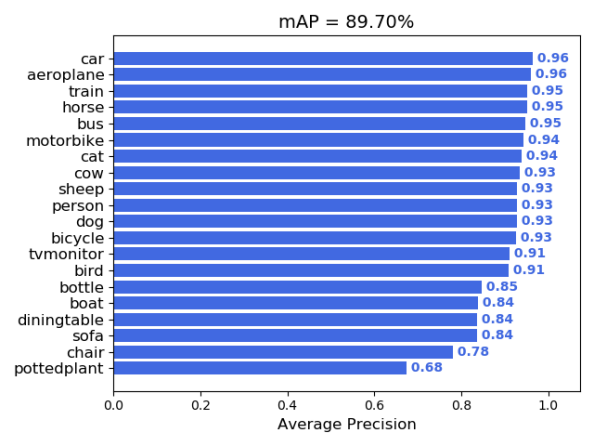
④可视化及定量实验结果展示及说明

查看一下预测的结果图：



对于检测到的物体，准确率还是很高的，但是对于一些物体，我们也可以直观地在图上发现是没有检测 到的，但是Faster RCNN是可以检测到的。

查看一下评估结果图：



mAP是目标检测模型中常用的评价指标，意为平均精确率。我们希望一个模型的Precision和Recall都很 高，所以需要综合考虑这两个因素，我们可以联想到用调和平均数F1-beta值来衡量，另一种方法正是 PR曲线下的面积AUC，这也就是AP。AUC面积越接近1性能越好。曲线下的面积理解为不同召回值的情 况下所有精度的平均值。 AP都是针对一个类别而言的，当我们把所有类别的AP都计算出来后，再对它们 求平均值，即可得到mAP。

⑤对实验结果现象的原理性分析

我们可以发现这个模型的精度是非常高的，到达了89.7%，与后面两个模型对比也可以发现，我也拿这 个模型和Yolov3进行了对比，发现精度提升了7%左右，除了网络架构外，这也源于Yolov4使用了很多的 调优技巧，如下所示：

Bag of freebies ( BoF)

仅仅改变训练策略，并且只增加训练的开销，不增加推理测试的开销的改进，包括：

● backbone ：CutMix和mosaic数据增强、 DropBlock正则化、类标签平滑(label smoothing)。 ● detector：CIoU-loss、CmBN、 DropBlock正则化、 Mosaic数据增强、自对抗训练(Self-Adversarial Training)、消除网格敏感性(Eliminate grid sensitivity)、为单个ground truth使用多个anchors、 余弦退火调度器(Cosine annealing scheduler)、最优超参数(Optimal

hyperparameter)、随机训练形状(Random training shapes)。

Bag of specials ( BoS)

只通过增加很小的计算量就可以极大的提高模型精度的方法，包括：

● backbone： Mish激活函数、跨级部分连接(Cross-stage partial connection ，CSP)、多输入加权剩余 连接(Multi-input weighted residual connection ， MiWRC)。

● detector： Mish激活函数、 SPP-block、SAM-block、 PAN path-aggregation block、 DIoU- NMS。

**Faster-RCNN**

①模型简介

Faster- RCNN作为一种two-stage的算法，与one-stage的算法相比， two-stage的算法更加复杂且速度 较慢，但是检测精度会更高，原理是先根据某种算法得到相应的候选框，然后对于每一个候选框单独预 测其所属类别。

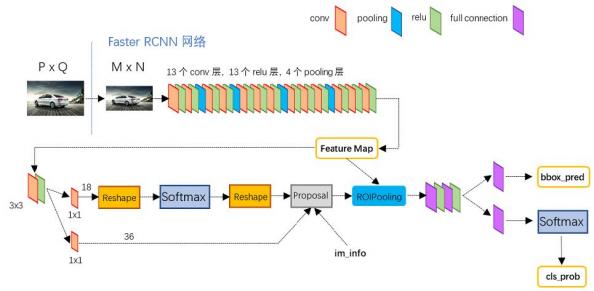


图4： Faster- RCNN模型整体架构图 [4]

图像输入到Faster- RCNN模型之前，需要先经过resize (等比例保证不失真)，首先利用Backbone进行 特征提取，获得一个共享特征层(38×38×1024)，由图上可以看出，分成两条路，往左侧的一条路先 经过一个3×3的卷积，卷积后经过两个1×1的卷积， 1个通道为18 ( 9×2)，这个2表示背景和是否真是包 含物体， 1个通道为36 ( 9×4)，这个4指的是先验框的调整参数(4个参数确定一个先验框)，将先验框 确定为建议框，利用建议框对共享特征层进行截取时大小可能不一样，需要经过一个pooling层使大小一 样，之后进行回归预测和分类预测。

**a.**主干网络介绍

Faster- RCNN可以采用多种的主干特征提取网络，常用的有VGG ， Resnet ，Xception等，此处选用 Resnet50。

ResNet50有两个基本的块，分别名为Conv Block和Identity Block，其中Conv Block输入和输出的维度 是不一样的，所以不能连续串联，它的作用是改变网络的维度； Identity Block输入维度和输出维度相 同，可以串联，用于加深网络的。这两个块都是经过3次卷积，都是残差网络结构。

**b.**获得**Proposal**建议框

在Faster- RCNN中， num\_priors也就是先验框的数量就是9，所以两个1x1卷积的结果实际上也就是9x4 的卷积用于预测公用特征层上每一个网格点上每一个先验框的变化情况。 9x2的卷积用于预测公用特征 层上每一个网格点上每一个预测框内部是否包含了物体，这在上文也提到过，到此位置还只是粗略的一 个框的获取，也就是一个建议框，之后会在建议框里面继续找东西。

**c.Proposal**建议框的解码

通过第二步获得了38x38x9个先验框的预测结果。相当于就是将整个图像分成38x38个网格；然后从每 个网格中心建立9个先验框，一共38x38x9个， 12996个先验框。当输入图像shape不同时，先验框的数 量也会发生改变。先验框虽然可以代表一定的框的位置信息与框的大小信息，但是其是有限的，无法表 示任意情况，因此还需要调整。之后经过ROIPooling层将建议框对应到相应的特征图的区域，得到建议

框特征图，这是一个动态调整步长的池化操作，不同于普通的池化，构造一个ROIPooling不需要指定步 长，只需要指定目标特征图的大小。

**d.**对**Proposal**建议框加以利用

在ROI Pooling结束后，得到了每个proposals对应的49维向量，每个向量都会经过若干个全连接层得到 一个特征向量，该特征向量既用来做回归(算出bbox的偏移量来第二次bbox修正)也用来做分类(通 过softmax来预测该proposal对应的区域是哪个类)。

②关键代码

github源码地址： <https://github.com/bubbliiiing/faster-rcnn-pytorch>

整体结构与Yolov4类似，不予赘述，说一下训练ROIpooling细节：

在ROI网络部分，会将建议框根据进行一定的截取，并获得对应的预测结果，事实上就是将上一步建议 框当作了ROI网络的先验框。因此，需要计算所有建议框和真实框的重合程度，并进行筛选，如果某个 真实框和建议框的重合程度大于0.5则认为该建议框为正样本，如果重合程度小于0.5则认为该建议框为 负样本。每次训练都放入128个建议框进行训练，同时要注意正负样本的平衡。

③实验设置

数据及使用细节：

数据集的准备

本文使用VOC格式进行训练，训练前需要下载好VOC07+12的数据集，解压后放在根目录

数据集的处理

修改voc\_annotation.py里面的annotation\_mode=2，运行voc\_annotation.py生成根目录下的 2007\_train.txt和2007\_val.txt。

trainval\_percent用于指定(训练集+验证集)与测试集的比例，默认情况下 (训练集+验证集):测试集 = 9:1。train\_percent用于指定(训练集+验证集)中训练集与验证集的比例，默认情况下 训练集:验证集 = 9:1。

参数设置：

|  |  |  |
| --- | --- | --- |
| ---------------------------------------------------------------------- | | |
| | | keys | | values| |
| ---------------------------------------------------------------------- | | |
| |  |  |  |  | | --- | --- | --- | --- | | | | model\_path | | | model\_dataoc\_weights\_resnet.pth| | | | | classes\_path | | | model\_dataoc\_classes.txt| | | | | backbone | | | vgg| | | | | confidence | | | 0.5 | | | | | nms\_iou | | | 0.3| | | | | anchors\_size | | | [8, 16, 32] | | | | | cuda | | | True| | | | |
| ---------------------------------------------------------------------- | | |

实验过程：

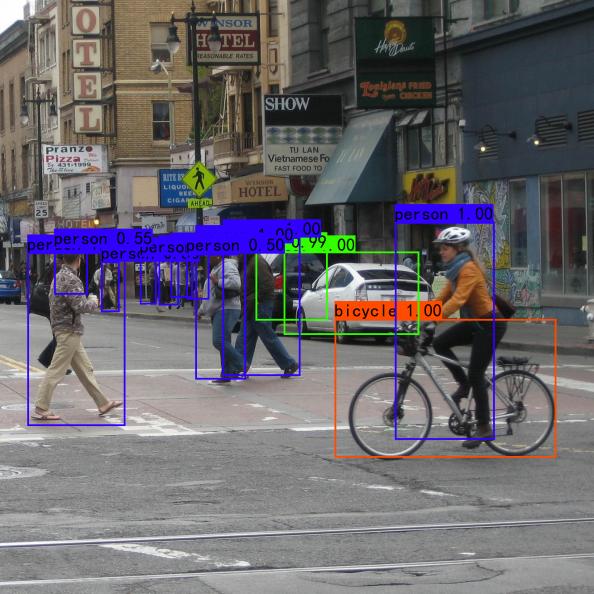
train.py的默认参数用于训练VOC数据集，直接运行train.py即可开始训练；

去frcnn.py里面修改model\_path以及classes\_path，然后运行predict.py进行预测；

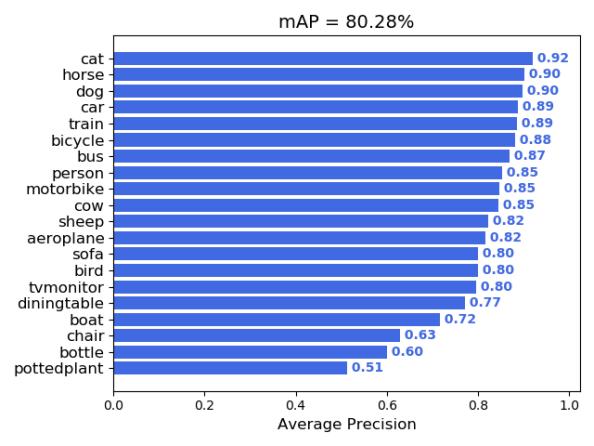
评估的话直接运行get\_map即可。

④可视化及定量实验结果展示及说明

查看一下预测的结果图：



查看一下评估结果图：



⑤对实验结果现象的原理性分析

Faster -rcnn引入了Region Proposal Network (RPN) ，它和检测网络共享整图的卷积特征，这样使得候 选框的计算几乎不额外占用时间。 RPN是一个全卷积网络，可同时预测物体外接框和每个位置是否为物 体的得分。 RPN采用端到端的方式进行训练，产生高质量的候选框，进而被Fast R-CNN用来做检测。 Faster -rcnn通过共享卷积特征，进一步融合RPN和Fast R-CNN为一个网络。

整体的map虽然不如Yolov4的精度高，但是从预测的图中我们就可以发现目标检测的效果是最好的，另 外两个模型达不到检测阈值的目标该模型可以检测出来，不愧是two-stage的模型。

人脸识别

**Retinaface+Facenet**

①课设简介

本项目基于Retinaface (检测) +Facenet (识别)搭建人脸识别平台。

1、 Retinaface

Retinaface是来自insightFace的又一力作，基于one-stage的人脸检测网络。同时开源了代码与数据 集，在widerface上有非常好的表现。该模型的构建非常简单，主要是由三部分组成，第一部分是主干特 征提取网络用于初步特征提取，第二部分是加强特征提取网络对上一步得到的特征进行特征融合再进行 加强特征提取，第三部分是利用获取到的特征得到最后结果。

主干特征提取网络采用Mobilenet，上文有讲，此处不赘述。

加强特征提取是SSH结构，使用三个并行结构，利用3x3卷积的堆叠代替5x5与7x7卷积的效果，得到3个 有效特征层。

[](af://n288/)[](af://n301/)Retinahead的预测对先验框是否包含人脸进行预测，之后对中心进行宽和高的调整，最后再对中心进行 调整获得人脸关键点(每一个人脸关键点需要两个调整参数，一共有五个人脸关键点)，具体实现都是 使用1×1的卷积进行通道数的调整。

2、 Facenet

谷歌人脸识别算法，发表于CVPR 2015，利用相同人脸在不同角度等姿态的照片下有高内聚性，不同人 脸有低耦合性，提出使用cnn + triplet mining方法，在LFW数据集上准确度达到99.63%。

通过CNN将人脸映射到欧式空间的特征向量上，实质上：不同图片人脸特征的距离较大；通过相同个体 的人脸的距离，总是小于不同个体的人脸这一先验知识训练网络。

测试时只需要计算人脸特征EMBEDDING，然后计算距离使用阈值即可判定两张人脸照片是否属于相同 的个体。

简单来讲，在使用阶段， facenet即是：

1、输入一张人脸图片

2、通过深度卷积网络提取特征

3、 L2标准化

4、得到一个长度为128特征向量(这个长度为128的特征向量就是输入图片的特征浓缩)

②关键代码

github源码地址： <https://github.com/bubbliiiing/facenet-retinaface-pytorch> 代码的使用如下： 1、在GITHUB上下载好库后将库解压。 2、下载对应的权重， README中会有下载连接，百度网盘下载或者GITHUB下载均可。 3、在retinaface.py里面，根据自身需求修改retinaface\_model\_path、 retinaface\_backbone、 facenet\_model\_path、facenet\_backbone四个参数。 4、运行encoding.py进行人脸数据集编码。 5、运行predict.py进行人脸图片的预测。

Ps：这里直接使用训练出来的权重，不重复进行训练。

③实验设置

数据及使用细节：

retinaface使用widerface数据集进行训练，可通过上述百度网盘下载widerface数据集，然后覆盖根目 录下的data文件夹。

facenet需要下载好数据集，将训练用的CASIA-WebFaces数据集以及评估用的LFW数据集，解压后放在 根目录。然后在训练前利用txt\_annotation.py文件生成对应的cls\_train.txt。再利用train.py训练facenet 模型，训练前，根据自己的需要选择backbone ， model\_path和backbone一定要对应。

参数设置：

都使用训练好的权重。

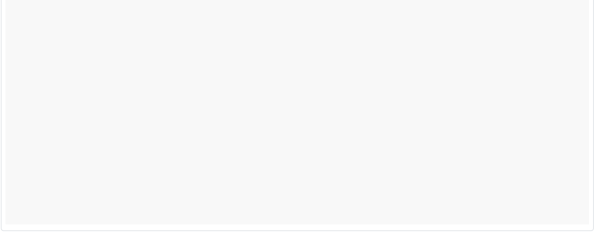
retinaface:



: 'model\_data/Retinaface\_mobilenet0.25.pth',

= { "model\_path"

[](af://n307/)



"backbone"

"confidence"

"nms\_iou"

"cuda"

#----------------------------------------------------------------------#

# 是否需要进行图像大小限制。

# 开启后，会将输入图像的大小限制为input\_shape。否则使用原图进行预测。

# 可根据输入图像的大小自行调整input\_shape，注意为32的倍数，如[640, 640, [3](#_bookmark1)]

# #

: [1280, 1280, 3],

: True

"input\_shape"

"letterbox\_image" }

: 'mobilenet', : 0.5,

: 0.45,

: True,

facenet:

|  |
| --- |
| \_defaults = {  "model\_path" : "model\_data/facenet\_mobilenet.pth",  "input\_shape" : (160, 160, 3),  "backbone" : "mobilenet",  "cuda" : True,  } |

实验过程：

首先单独的完成Retinaface和Facenet两个模型的训练。

然后将数据库初始化，初始化步骤如下：

1、遍历数据库中所有的图片。

2、利用Retinaface检测每个图片中的人脸位置。

3、将人脸截取下来。

4、将获取到的人脸进行对齐。

5、利用Facenet将人脸进行编码。

6、将所有人脸编码的结果放在一个列表中。

7、保存成npy的形式。

再进行检测图片的处理：

1、人脸的截取与对齐

利用Retinaface我们可以获得一张图片中人脸的位置，但是我们截取下来的人脸很明显的看出来是歪着 的，我们如果人脸可以正过来，那么将对人脸的特征提取非常有好处，我们可以利用双眼坐标进行旋 正。

2、利用Facenet对矫正后的人脸进行编码

Facenet是一个人脸特征获取的模型，将第1步获得的对齐人脸传入Facenet模型就可以得到每个人脸的 特征向量。将所有特征向量保存在一个列表中，在第3步进行比对。

3、将实时图片中的人脸特征与数据库中的进行比对

这个比对过程需要循环实现，具体对实时图片中的每一个人脸进行循环：

a.获取实时图片中的每一个人脸特征。

b.将每一个人脸特征和数据库中所有的人脸进行比较，计算距离。如果距离小于门限值，则认为其具有 一定的相似度。

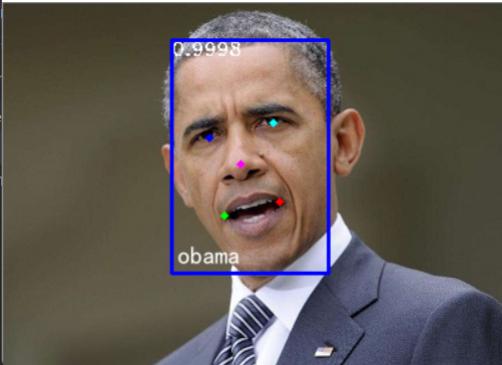
[](af://n319/)c.获得每一张人脸在数据库中最相似的人脸的序号。

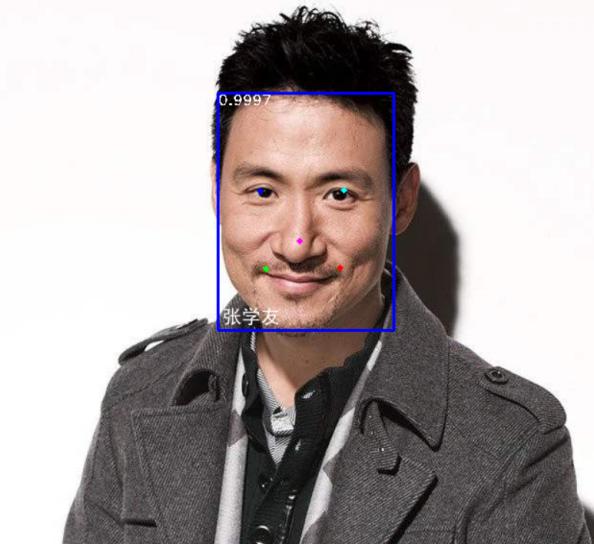
d.判断这个序号对应的人脸距离是否小于门限，是则认为人脸识别成功，他就是这个人。

4、图片绘制

④可视化及定量实验结果展示及说明

本项目没有什么数值评价指标，给大家看一下预测效果：





可以发现，这个预测结果堪称完美，只要数据库里出现过的人脸，都可以识别出来。

⑤对实验结果现象的原理性分析

实验结果和预想的和模型描述的一样，由Retinaface进行人脸检测，标注框，传给Facenet进行编码再传 输到数据库中进行比较，找到对应的人脸名字。当然，如果想训练自己的数据集，你可以多准备几张自 己的照片，然后按照github上的要求进行数据集处理(这一步并不麻烦，讲解很详细)，然后进行训练 得到权重，就可以有全新的权重用来人脸识别啦。

心得体会

本次计算机视觉课设收获很多，先说一下做课设的过程。

任务1的暗通道先验去雾霾算法我是一步一步跟着论文的描述做的，但一开始做完效果不是很好，得到的 最后处理完的图片是黑色的，最后经过调参发现是在得到透射率图矩阵的那一步中，要给这个矩阵都减 去一个值，最后得到的效果还不错，当我将自己拍摄的图片用于去雾霾时，由于分辨率太高，我一度以 为程序跑不出来，结果过了几分钟后才得到结果，但是得到的结果和论文给出的图像来比，效果还是差 了一些。

任务2中目标检测算法是比较熟悉之前科创接触过的，所以做起来不是很困难，不仅对不同的模型之间进 行了对比，每个模型我也换用了不同的主干网络的模型权重进行对比，选用了最高精度的模型。语义分 割这四个字乍一看我是不知道他实现的任务是个图像的像素标类别，知道后感觉还挺有意思的，就了解 了一下实现的原理和网络结构，也是进行了两次对比试验，我先控制了主干网络都为mobilenetv2，发 现deeplabv3+的效果好一些，然后都换用论文提到的主干网络，发现指标都提升了好多。对于人脸识别

[](af://n498/)模型并没有给出什么评价指标，从预测图中就可以很好的看出效果。任务2的开源代码都在报告中有说 明，也有教你如何训练自己的数据集的部分，很方便。

这次课设让我对以前熟悉的和新了解的模型有了一个模块化的认识，并且通过课设将知识体系化，在论 文复现和模型调用中，发现自己的不足之处，也领略到了cv的魅力。

**References**

[1] K. He, J. Sun and X. Tang, "Single Image Haze Removal Using Dark Channel Prior," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 12, pp. 2341-2353, Dec.

2011.

[2] "Pyramid Scene Parsing Network" by Hengshuang Zhao etc, in CVPR 2017 (IEEE Conference on Computer Vision and Pattern Recognition).

[3] "Encoder- Decoder with Atrous Separable Convolution for Semantic Image Segmentation" by Liang-Chieh Chen etc, in CVPR 2018 (IEEE Conference on Computer Vision and Pattern Recognition).

[4] "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks" by Shaoqing Ren etc, in NIPS 2015.