深蓝学院
shenlanxueyuan.com

人脸检测

主讲人 张士峰

中国科学院自动化研究所
模式识别国家重点实验室

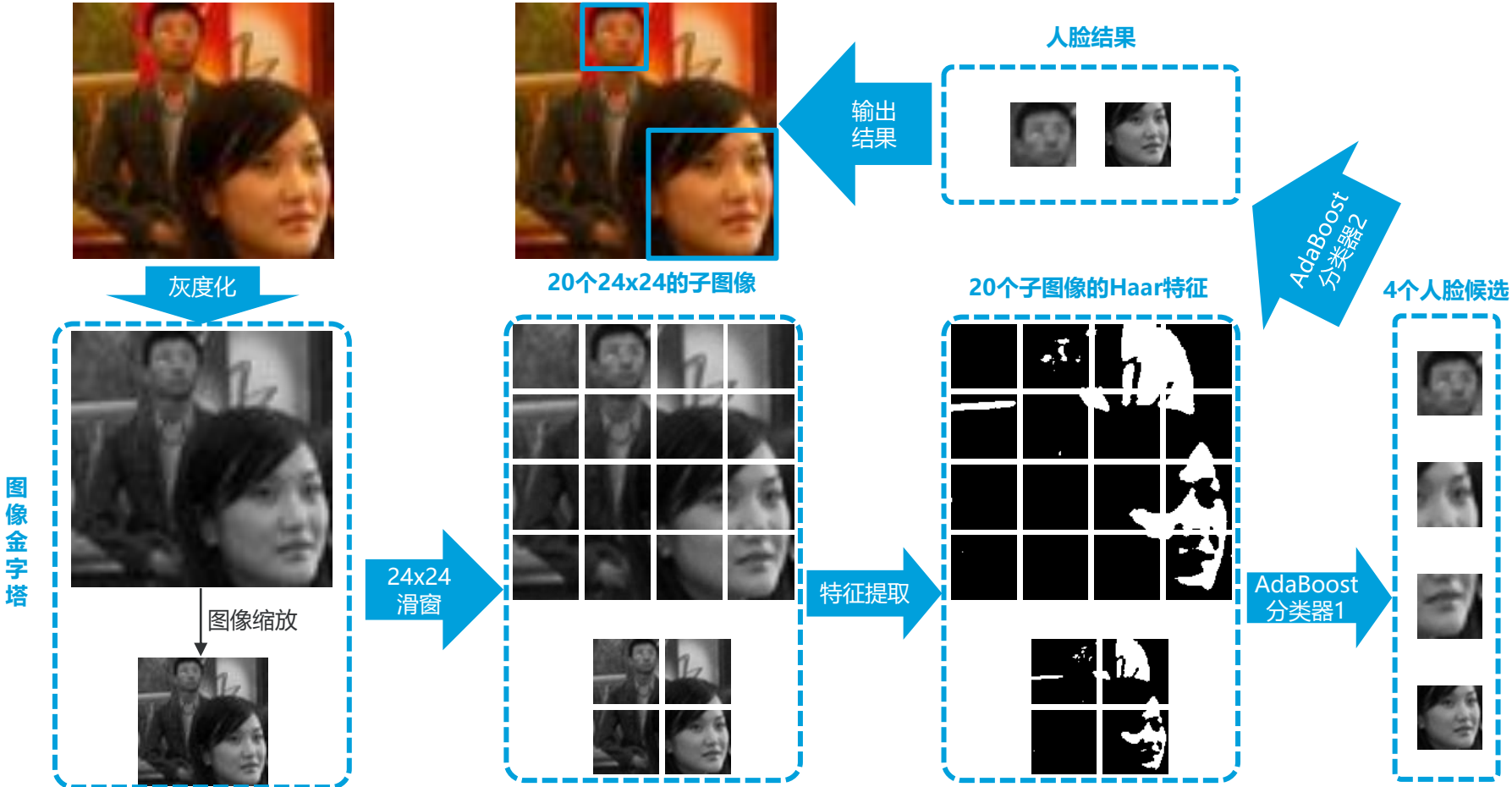
内容回顾：传统人脸检测算法

- 利用手工特征+分类器，以滑窗方式在图像金字塔上遍历所有位置和大小，进行人脸检测

遍历所有位置
滑窗方式

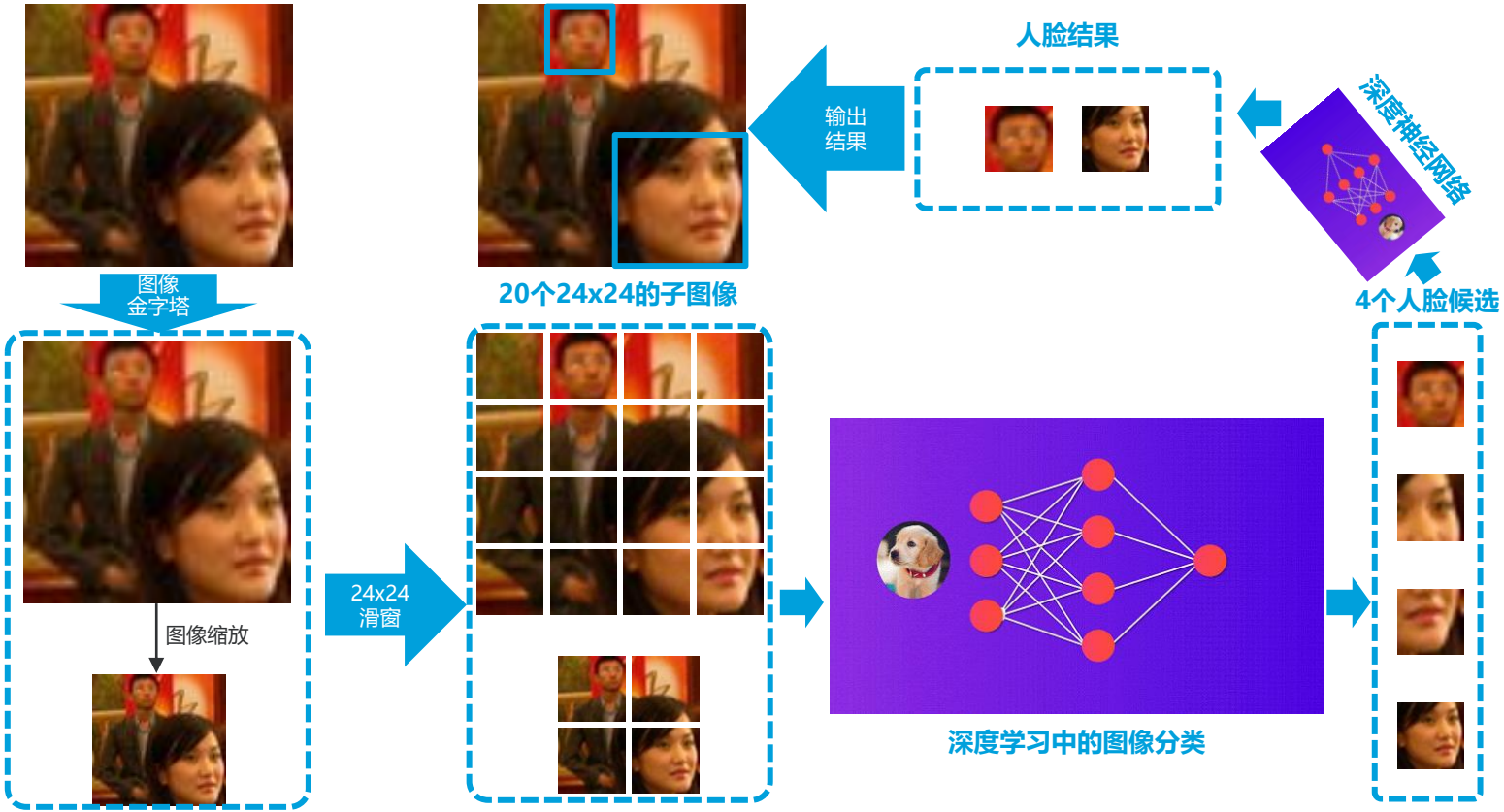
遍历不同大小
图像金字塔

内容回顾：传统人脸检测算法Viola-Jones

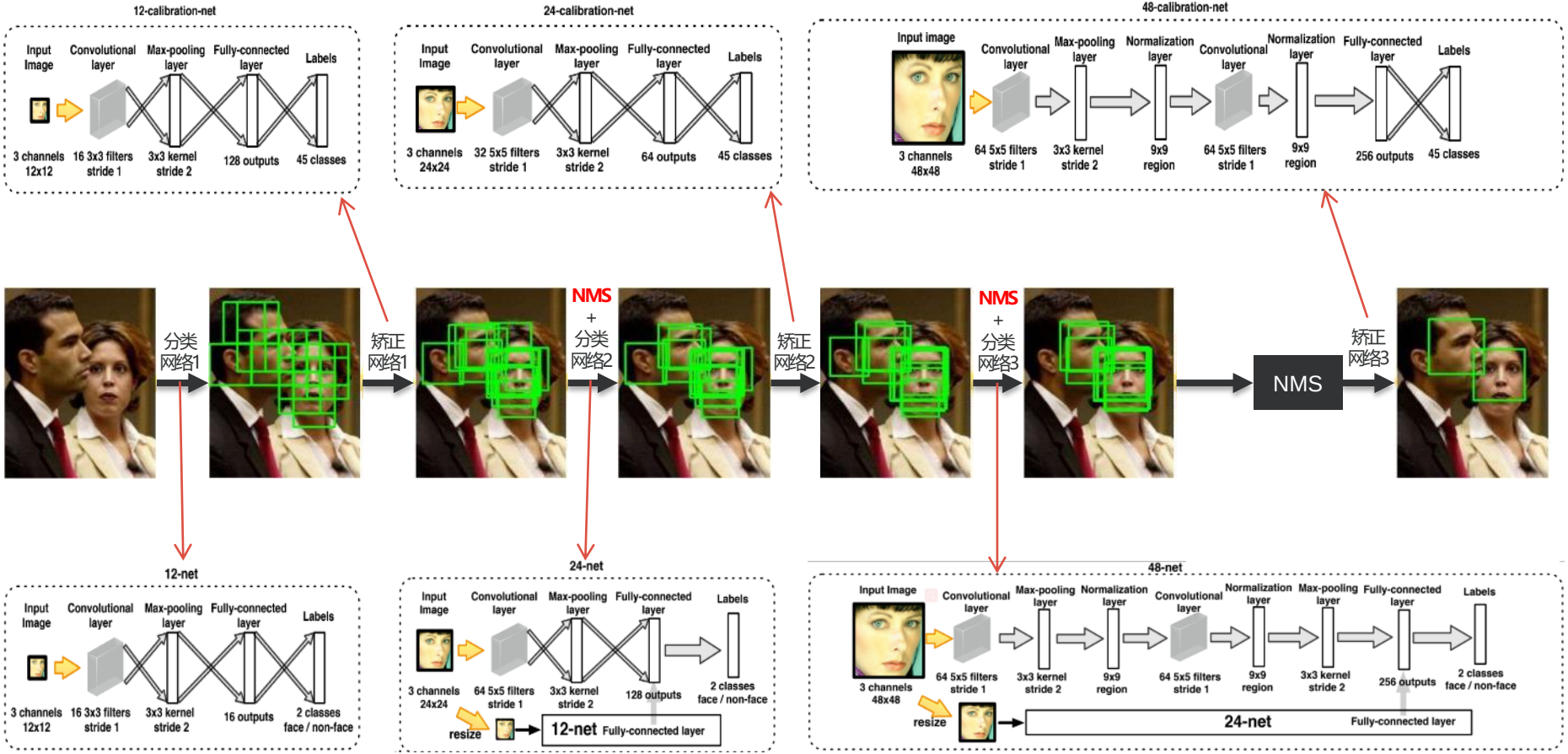


内容回顾：深度学习早期人脸检测算法

- 在传统人脸检测算法的流程中，把**手工设计的特征和分类器**变成**深度学习中的特征和分类器**

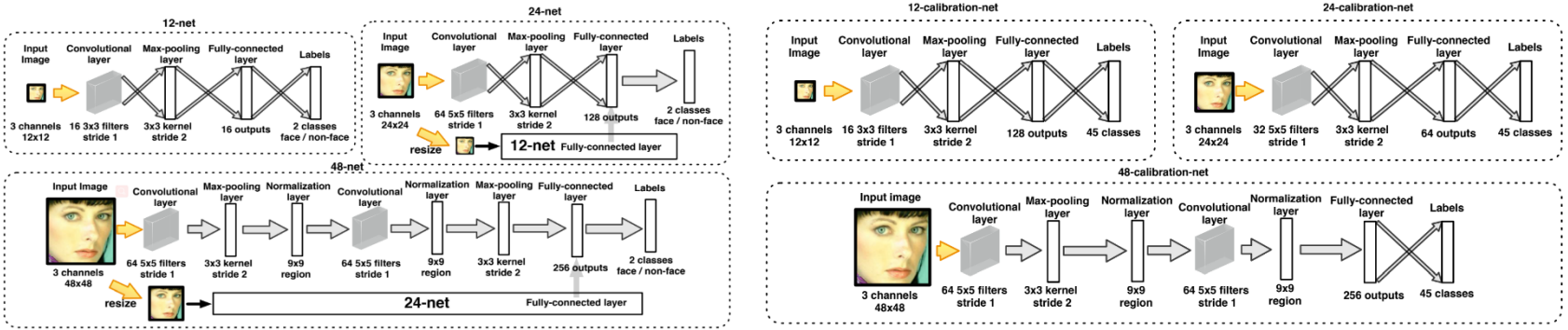


内容回顾：深度学习早期人脸检测算法CascadeCNN

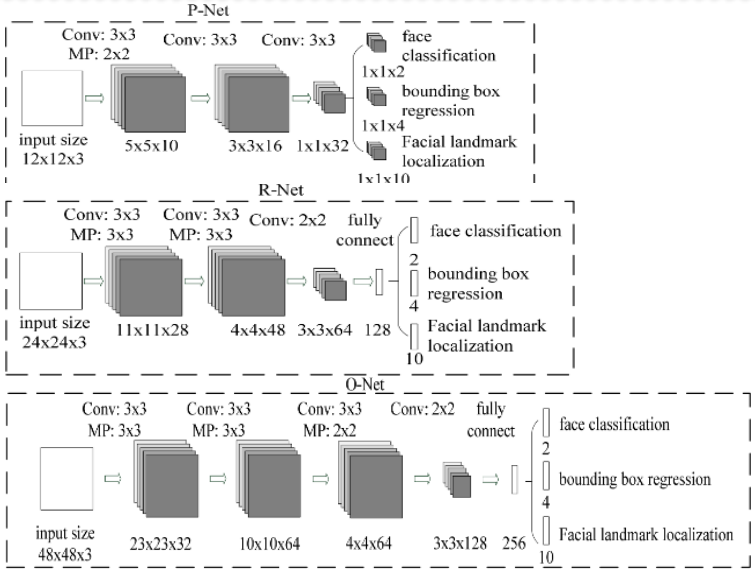


内容回顾：深度学习早期人脸检测算法的发展

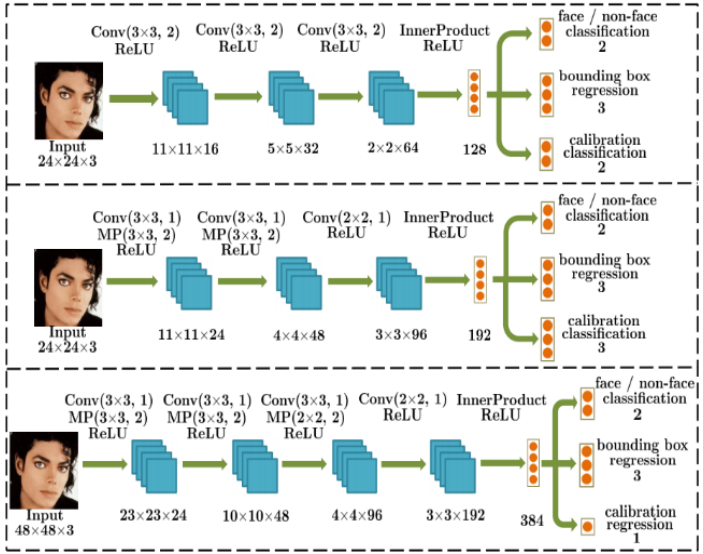
CascadeCNN



MTCNN



PCN



内容回顾：深度学习早期人脸检测算法的总结

CascadeCNN → +关键点分支 → MTCNN → +角度预测分支 → PCN

滑窗	滑窗操作遍历所有的位置
金字塔	图像金字塔遍历所有的大小
级联	3个级联的阶段，人脸数量从多到少，人脸难度从易到难，网络结构从简单到复杂
深度学习	利用深度学习中的神经网络进行特征提取+分类器+矫正器+其他
优点	有着满足实际需求的精度，具备CPU实时的速度
缺点	检测过程仍然分为多个独立的阶段，训练过程比较繁琐
总结	深度学习早期人脸检测算法的代表，开创了深度学习时代下，人脸检测的一个派系，很多实际场景中都在使用该类型的算法

目录

深度学习早期人脸检测算法：痛点问题

CascadeCNN

MTCNN

PCN

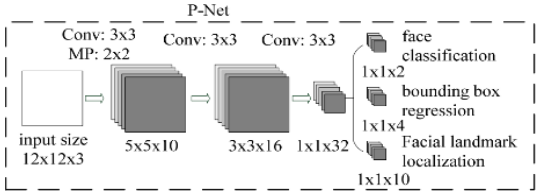
局部最优：三个独立的阶段，容易取得局部最优而非全局最优

训练繁琐：训练不是端到端的，每个阶段单独处理，非常繁琐

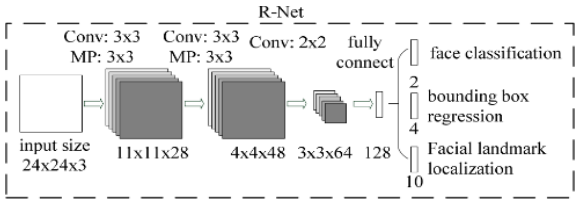
检测速度：检测速度与图像上人脸的数量高度相关

检测精度：在复杂的场景下，检测精度不能够满足性能的需求

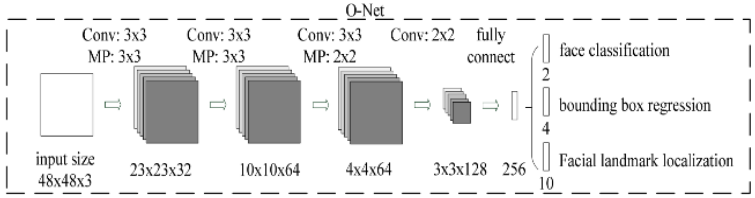
深度学习早期人脸检测算法：局部最优问题



P-Net的最优检测器为P*（局部最优）



R-Net的最优检测器为R* （局部最优）



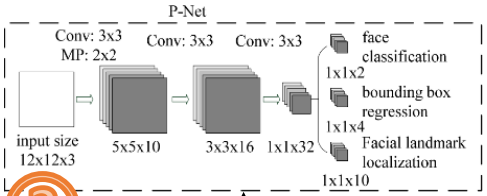
O-Net的最优检测器为O* （局部最优）

- D-Net = P-Net + R-Net + O-Net
- D-Net的最优检测器D*不一定是P* + R* + O*
- 各个局部达到最优，并不代表达到了全局最优

深度学习早期人脸检测算法：训练繁琐问题

1 切图准备海量人脸/非人脸数据

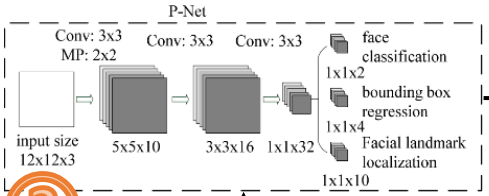
深度学习早期人脸检测算法：训练繁琐问题



② 训练P-Net

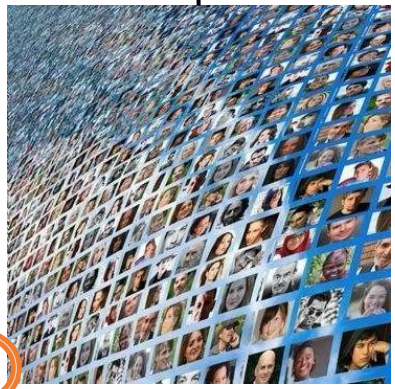
① 切图准备海量人脸/非人脸数据

深度学习早期人脸检测算法：训练繁琐问题



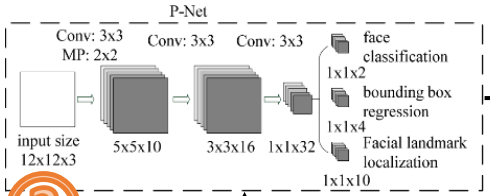
P-Net
筛选数据

训练P-Net



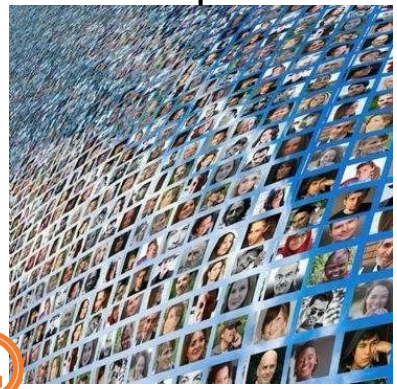
切图准备海量人脸/非人脸数据

深度学习早期人脸检测算法：训练繁琐问题

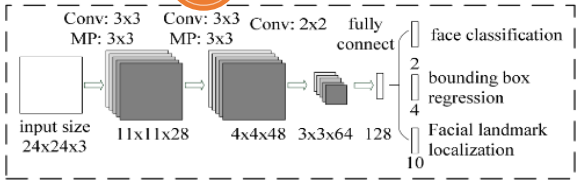


P-Net
筛选数据

训练P-Net

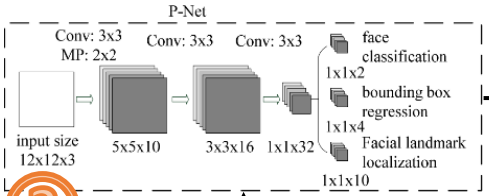


训练R-Net



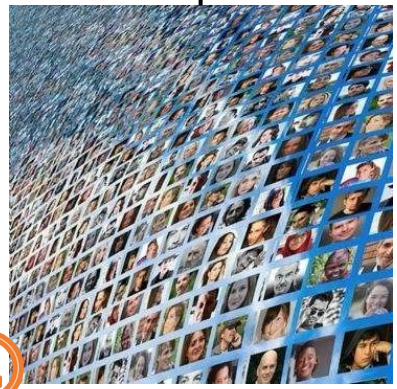
切图准备海量人脸/非人脸数据

深度学习早期人脸检测算法：训练繁琐问题



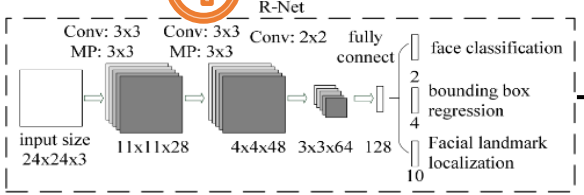
P-Net
筛选数据

训练P-Net



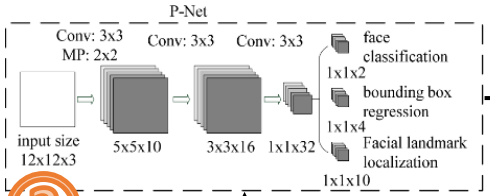
切图准备海量人脸/非人脸数据

训练R-Net

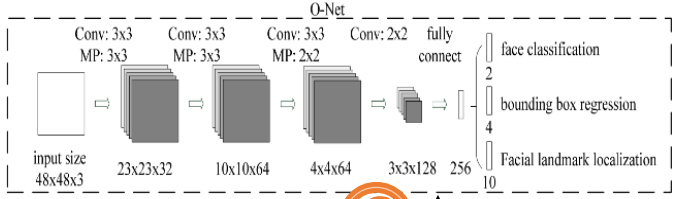


R-Net
筛选数据

深度学习早期人脸检测算法：训练繁琐问题

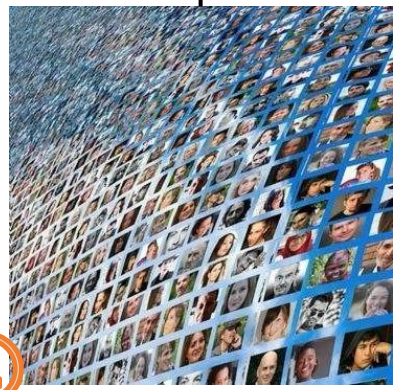


P-Net
筛选数据

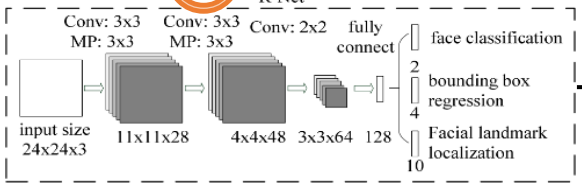


训练P-Net

训练O-Net



训练R-Net



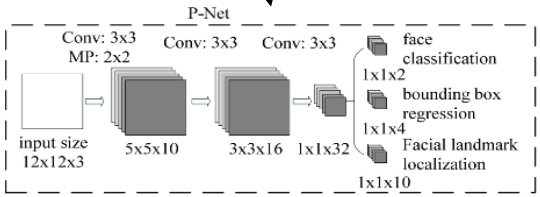
R-Net
筛选数据

切图准备海量人脸/非人脸数据

深度学习早期人脸检测算法：检测速度问题

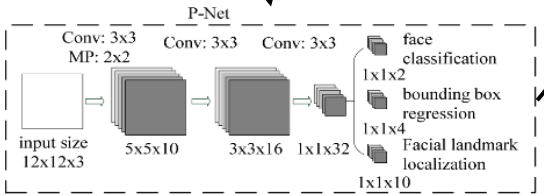
深度学习早期人脸检测算法：检测速度问题

10000个滑动窗口

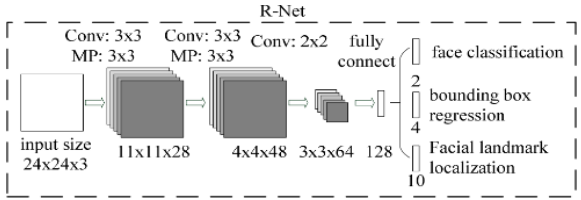


深度学习早期人脸检测算法：检测速度问题

10000个滑动窗口

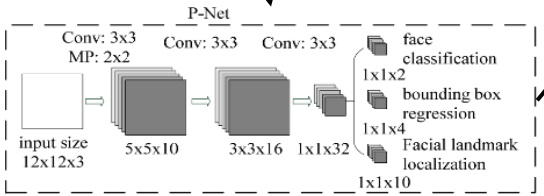


10个窗口

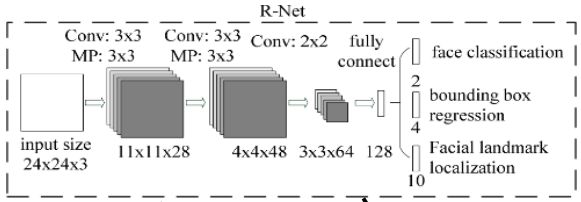


深度学习早期人脸检测算法：检测速度问题

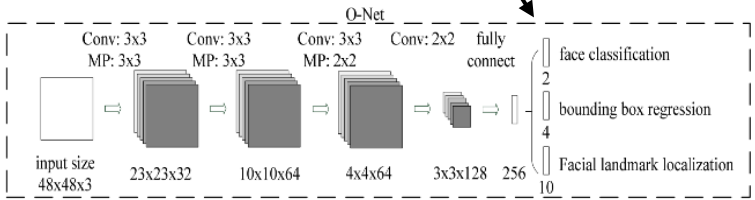
10000个滑动窗口



10个窗口



2个窗口

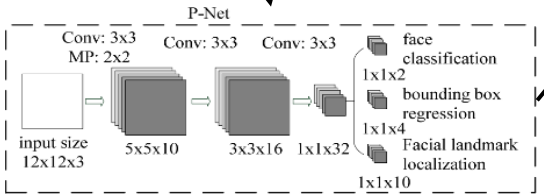


深度学习早期人脸检测算法：检测速度问题

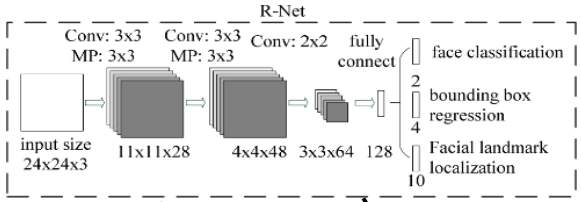
当图像上人脸较少时

检测速度非常快

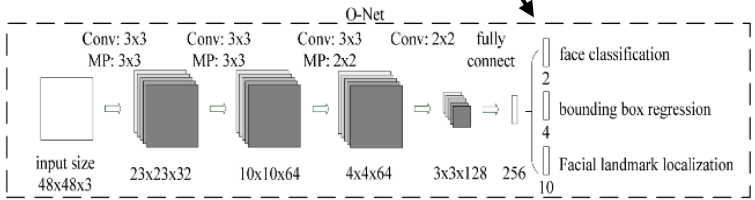
10000个滑动窗口



10个窗口



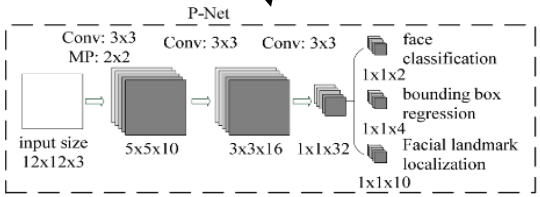
2个窗口



深度学习早期人脸检测算法：检测速度问题

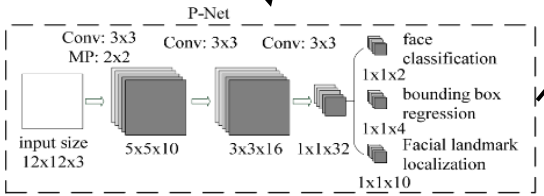
深度学习早期人脸检测算法：检测速度问题

10000个滑动窗口

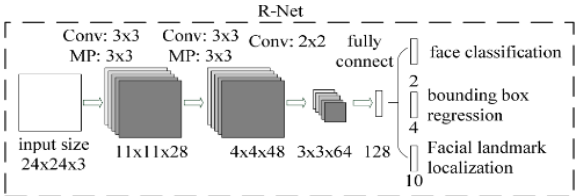


深度学习早期人脸检测算法：检测速度问题

10000个滑动窗口

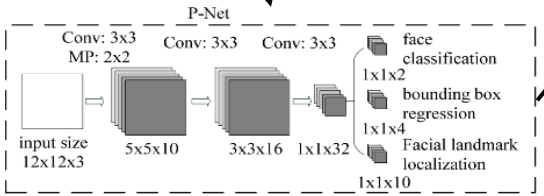


7000个窗口

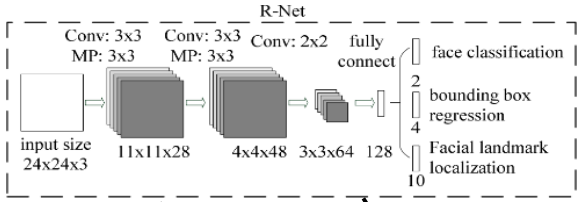


深度学习早期人脸检测算法：检测速度问题

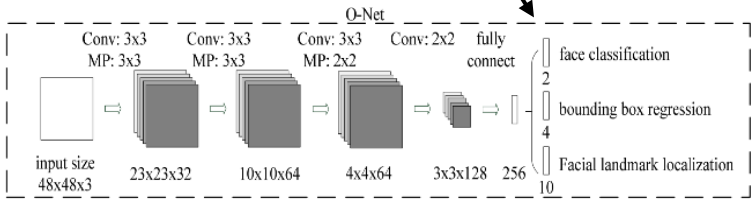
10000个滑动窗口



7000个窗口



5000个窗口



深度学习早期人脸检测算法：检测速度问题

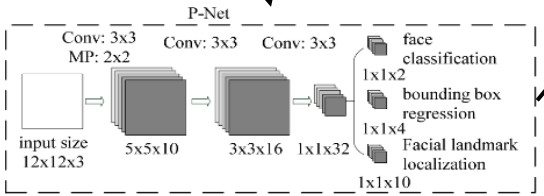
当图像上人脸非常多时

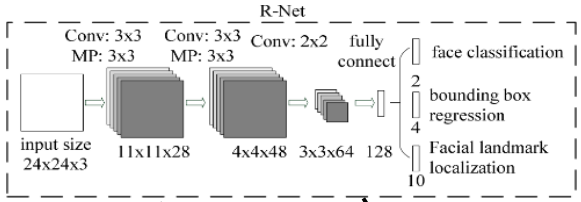
检测速度非常慢！！！

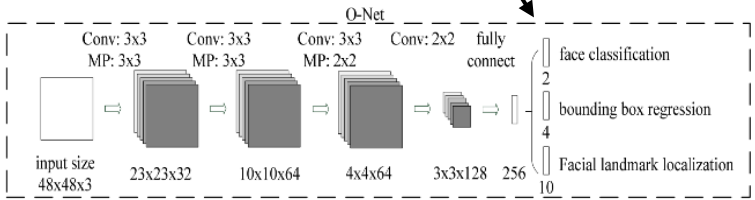
10000个滑动窗口

7000个窗口

5000个窗口



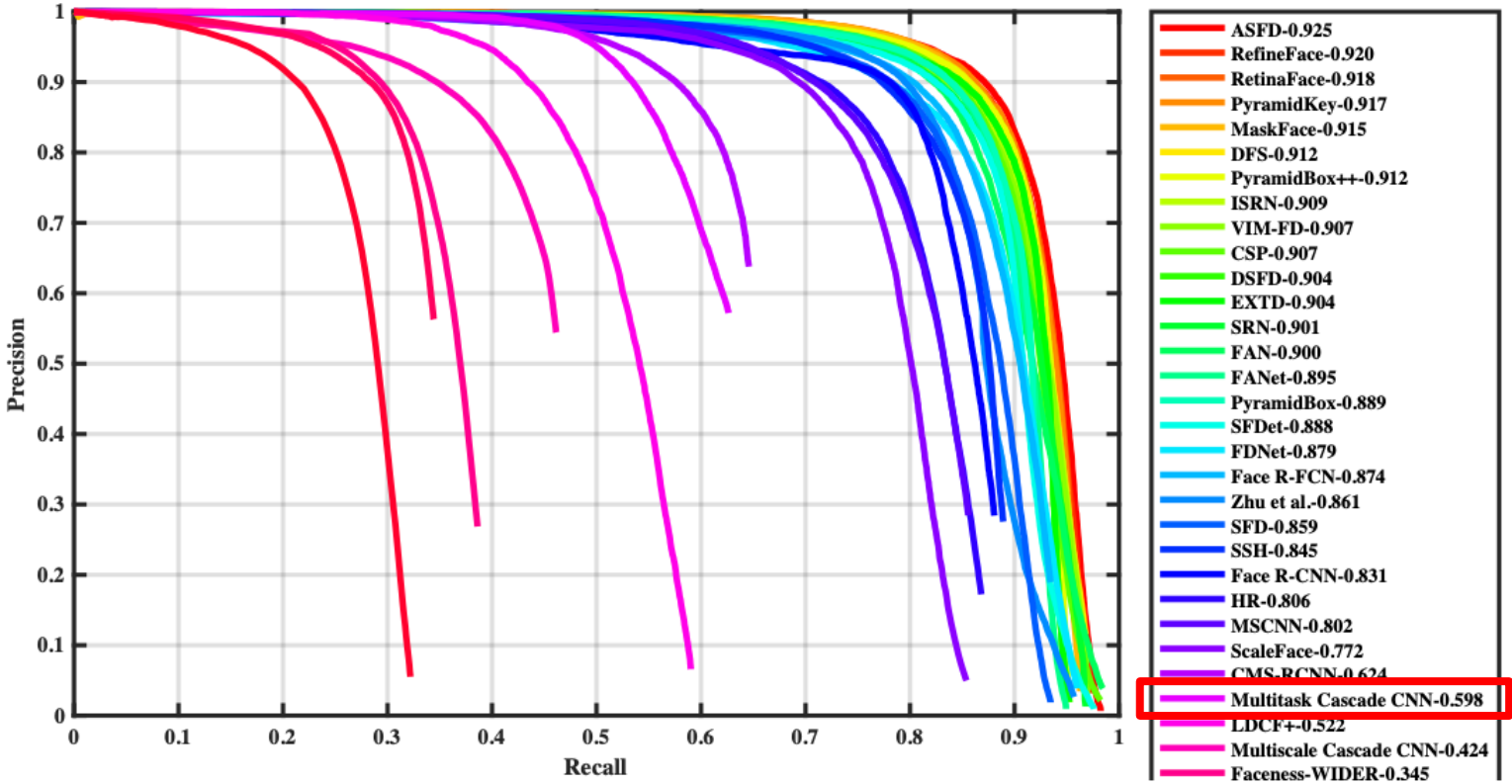




深度学习早期人脸检测算法：检测精度问题

- 在复杂的场景下，例如包含复杂的尺度、姿势、遮挡、表情、装扮、光照等方面的变化
- 该类算法的检测精度就不能够满足性能的需求

深度学习早期人脸检测算法：检测精度问题



在复杂场景下精度不到60%

深度学习早期人脸检测算法：痛点问题

由于这些痛点问题
在深度学习的后期
出现了全新的算法
早期算法不再流行

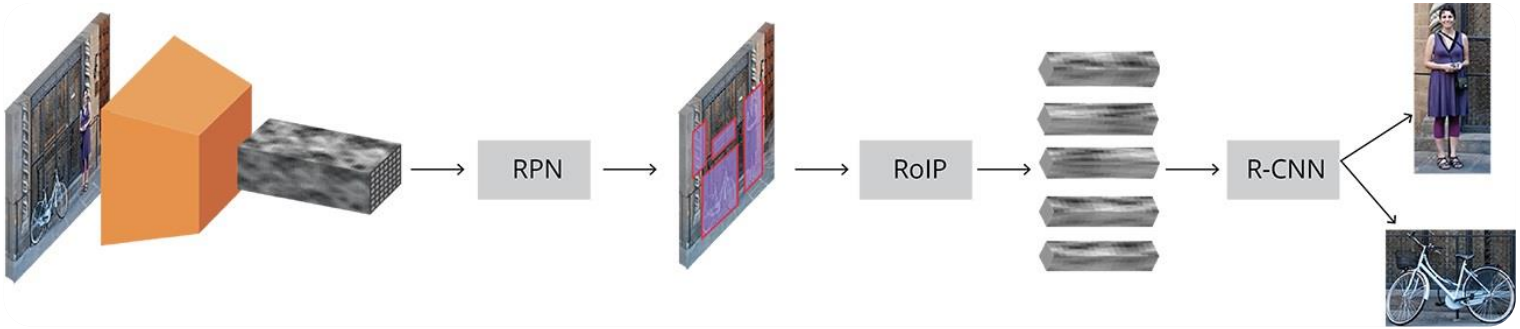
- **局部最优：**三个独立的阶段，容易取得局部最优而非全局最优
- **训练繁琐：**训练不是端到端的，每个阶段单独处理，非常繁琐
- **检测速度：**检测速度与图像上人脸的数量高度相关
- **检测精度：**在复杂的场景下，检测精度不能够满足性能的需求

深度学习后期人脸检测算法

- 深度学习后期人脸检测算法：对**通用物体检测算法**进行**相应改进**，应用于**人脸检测领域**

深度学习后期人脸检测算法@Faster R-CNN

- 深度学习后期人脸检测算法：对**通用物体检测算法**进行**相应改进**，应用于**人脸检测领域**



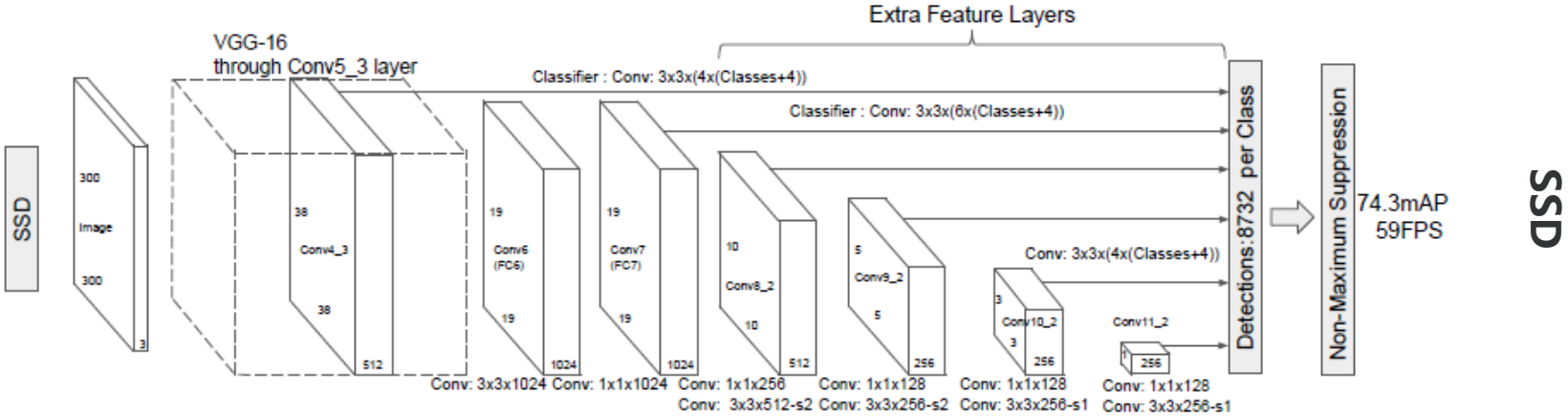
Faster R-CNN

相应改进

人脸检测算法	CMS-RCNN	HR	Face R-CNN	FDNet

深度学习后期人脸检测算法@SSD

- 深度学习后期人脸检测算法：对**通用物体检测算法**进行**相应改进**，应用于**人脸检测领域**

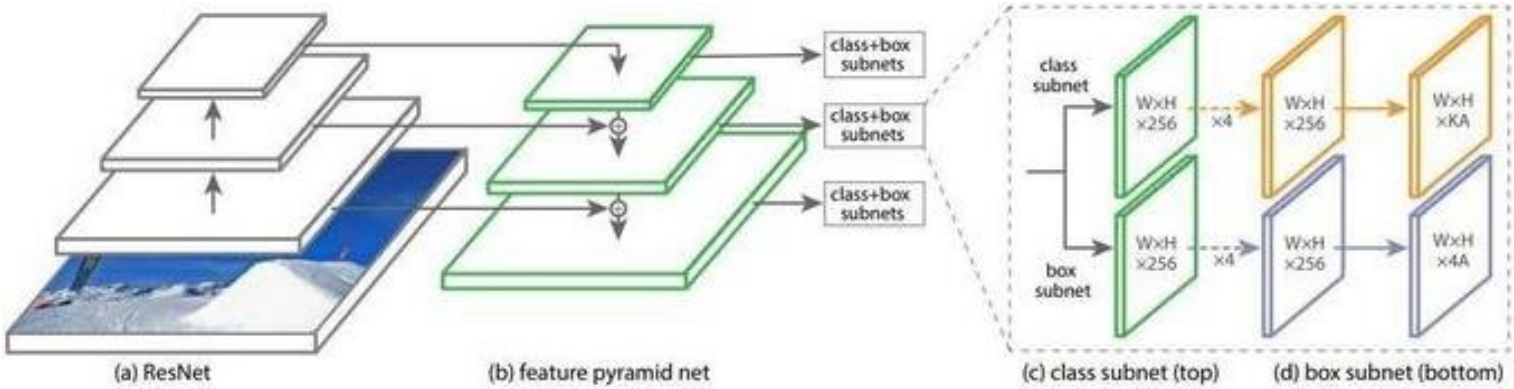


相应改进

人脸检测算法	SFD	SSH	PyramidBox	DSFD	SFDet	FaceBoxes	EXTD

深度学习后期人脸检测算法@RetinaNet

- 深度学习后期人脸检测算法：对**通用物体检测算法**进行**相应改进**，应用于**人脸检测领域**



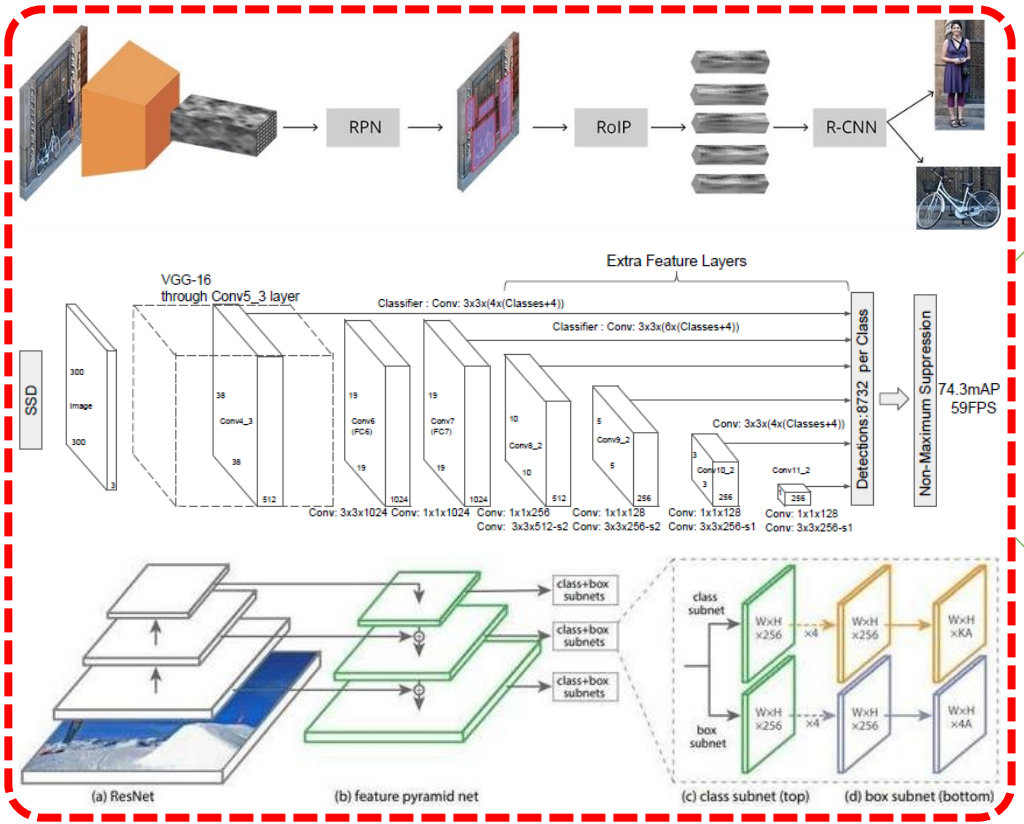
RetinaNet

相应改进

人脸检测算法	FAN	SRN	DFS	ISRN	RetinaFace	AInnoFace	RefineFace

深度学习后期人脸检测算法

- 通用物体检测算法 -> 人脸检测算法：基础改进

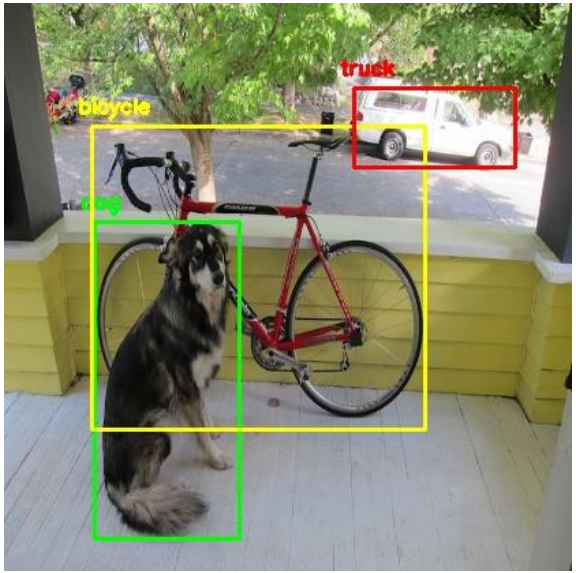


锚框调整

数据增广

深度学习后期人脸检测算法

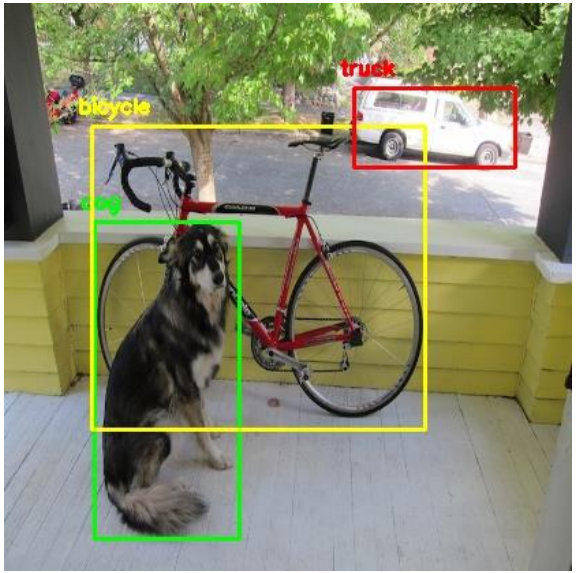
- 通用物体检测算法 -> 人脸检测算法：基础改进之锚框调整



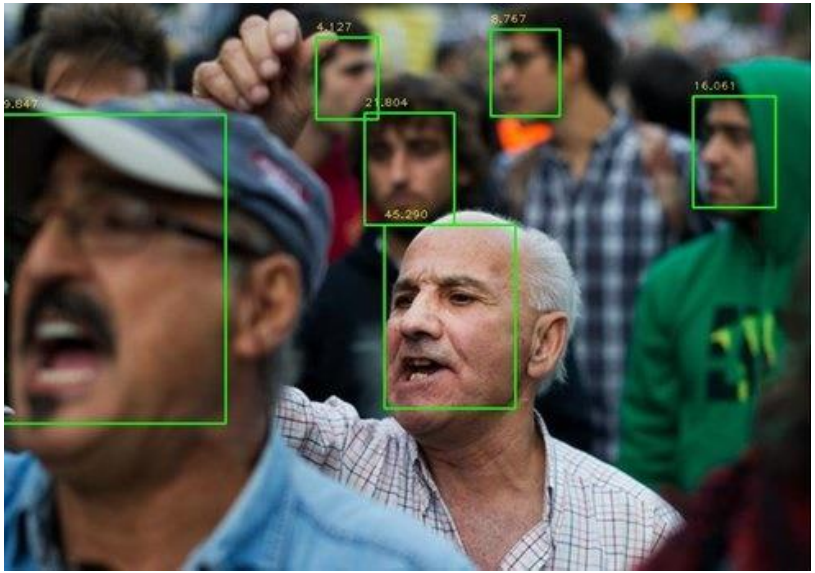
- 通用物体检测需要检测不同的类别
- 不同类别的物体有着不同的长宽比
- 因此锚框比例的设置为3个或5个
- 1:3、1:2、1:1、2:1、3:1

深度学习后期人脸检测算法

- 通用物体检测算法 -> 人脸检测算法：基础改进之锚框调整



锚框比例

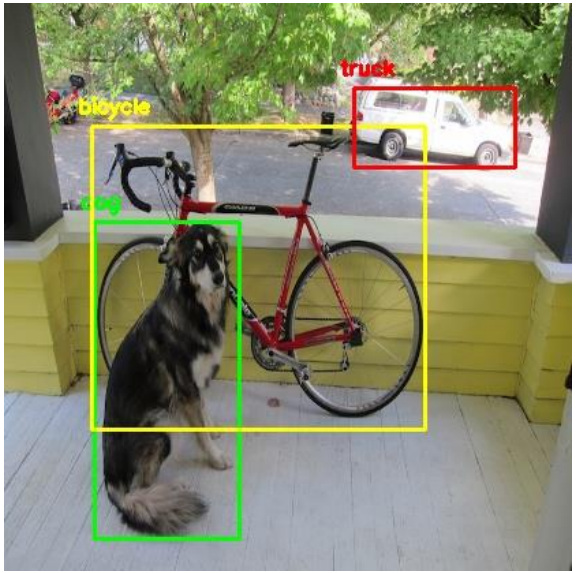


- 通用物体检测需要检测不同的类别
- 不同类别的物体有着不同的长宽比
- 因此锚框比例的设置为3个或5个
- 1:3、1:2、1:1、2:1、3:1

- 人脸检测只需要检测人脸这一特定的类别
- 人脸这一特定的类别有着相对固定的长宽比
- 因此锚框比例的设置一般为1个
- 1:1或1.25:1

深度学习后期人脸检测算法

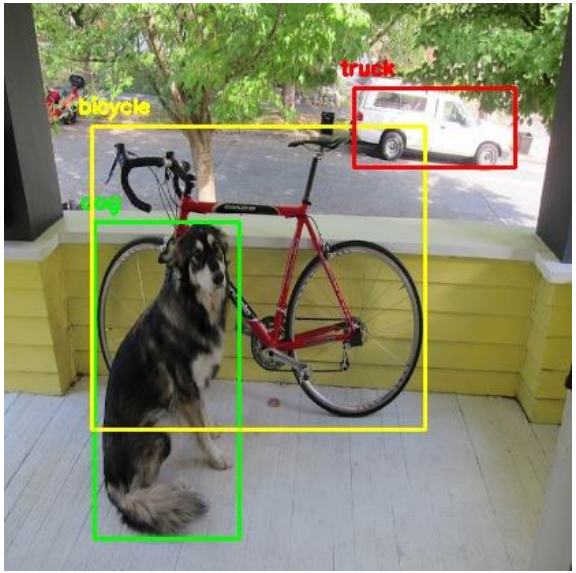
- 通用物体检测算法 -> 人脸检测算法：基础改进之锚框调整



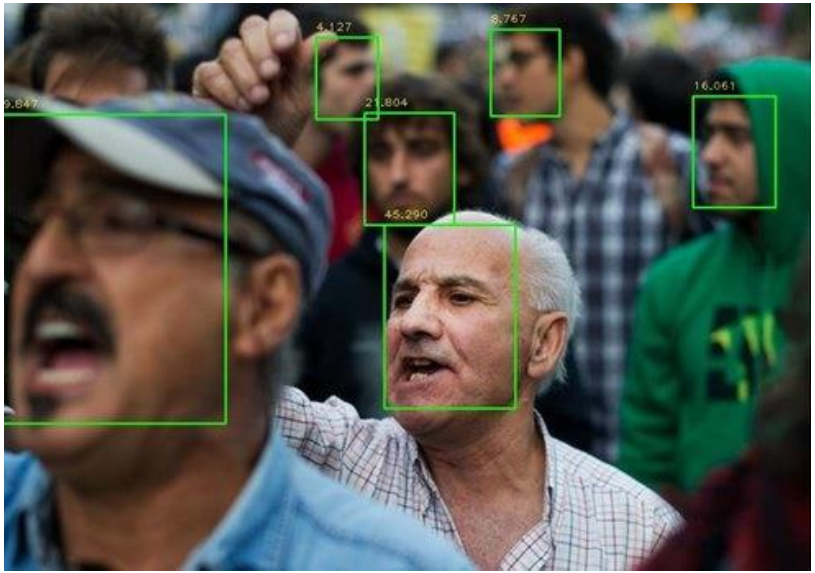
- 通用物体检测中物体尺度相对集中
- 绝大部分物体在64~512像素之间
- 因此锚框尺度设置在32~800之间

深度学习后期人脸检测算法

- 通用物体检测算法 -> 人脸检测算法：基础改进之锚框调整



锚框
尺度

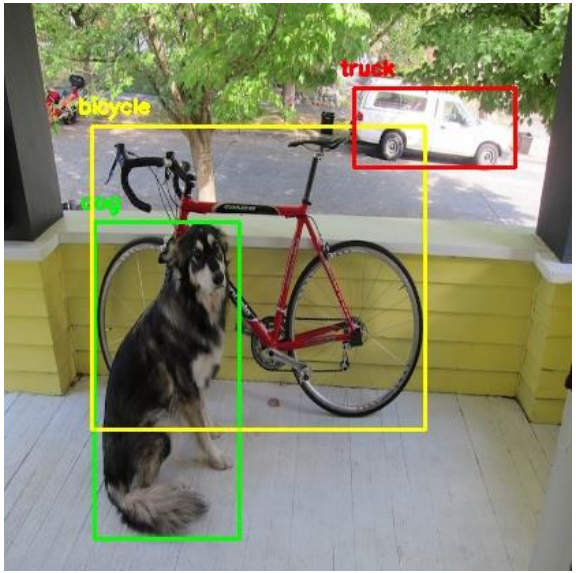


- 通用物体检测中物体尺度相对集中
- 绝大部分物体在64~512像素之间
- 非常小或非常大的物体占比不大
- 因此锚框尺度设置在32~800之间

- 人脸检测中人脸的尺度范围比较大
- 有很大一部分人脸的尺度处于10~50个像素之间
- 也有少部分人脸的尺度在800~1000像素之间
- 因此锚框尺度的设置一般为8~1024

深度学习后期人脸检测算法

- 通用物体检测算法 -> 人脸检测算法：基础改进之数据增广



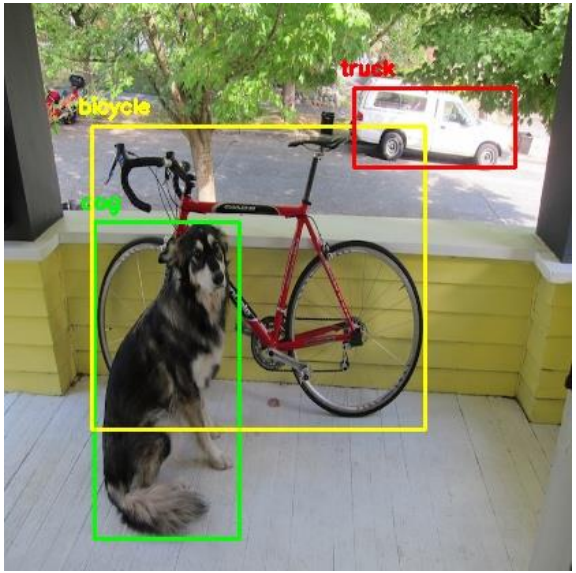
- 通用物体检测中数据增广比较简单

① 随机的水平翻转

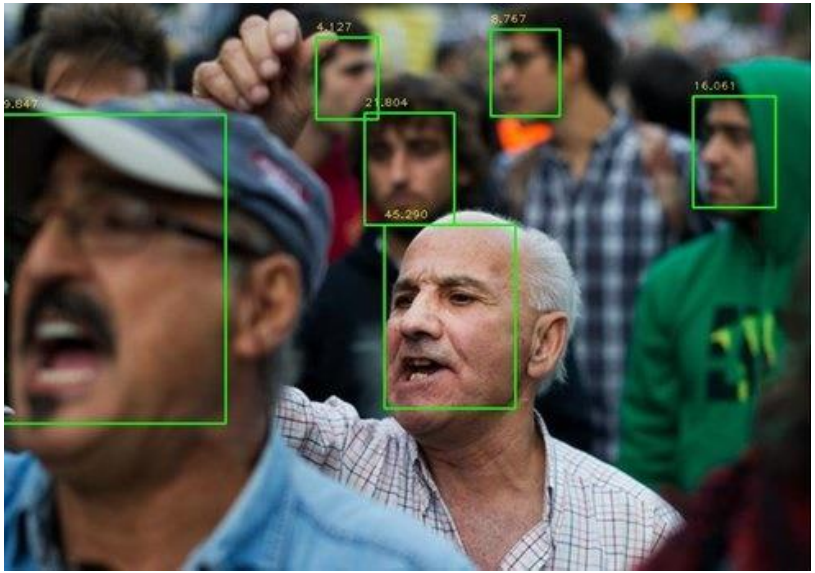
② 离散的多尺度训练

深度学习后期人脸检测算法

- 通用物体检测算法 -> 人脸检测算法：基础改进之数据增广



数据增广



- 通用物体检测中数据增广比较简单

① 随机的水平翻转

② 离散的多尺度训练

- 人脸检测沿用SSD的数据增广，比较丰富

① 随机的水平翻转

② 随机的颜色抖动

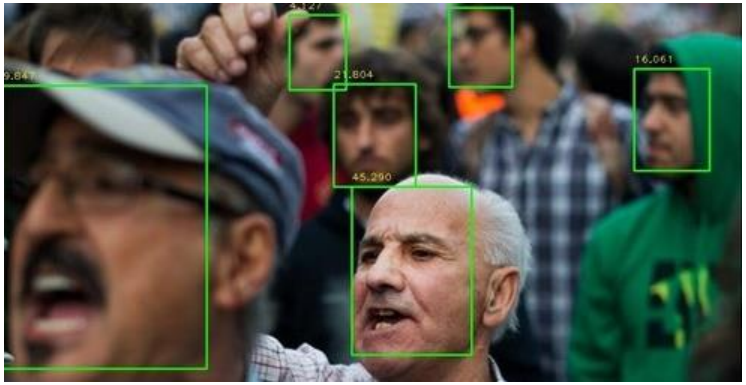
③ 动态的多尺度训练

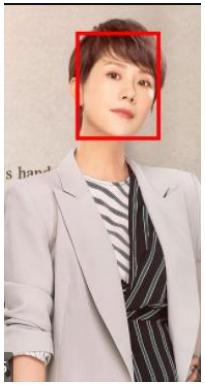
深度学习后期人脸检测算法

- SSD的数据增广有一个问题：**物体的比例会被改变**，从而增大了难度

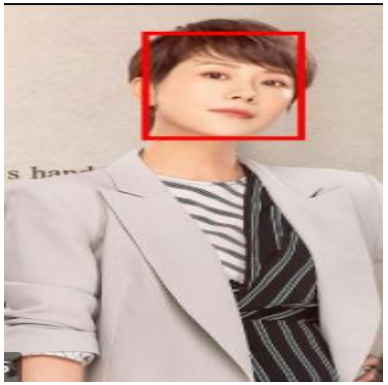
深度学习后期人脸检测算法

- SSD的数据增广有一个问题：**物体的比例会被改变**，从而增大了难度



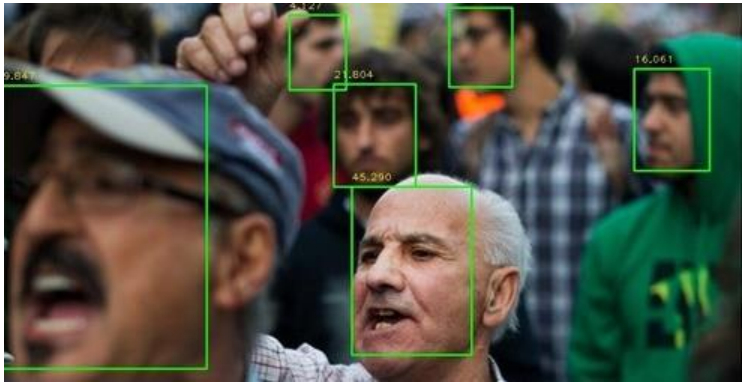


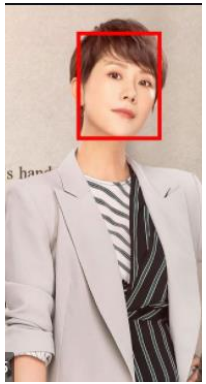
SSD的
数据增广

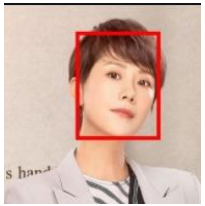


深度学习后期人脸检测算法

- 改进SSD的数据增广：动态多尺度中进行随机裁剪时加一个**1:1**的限制

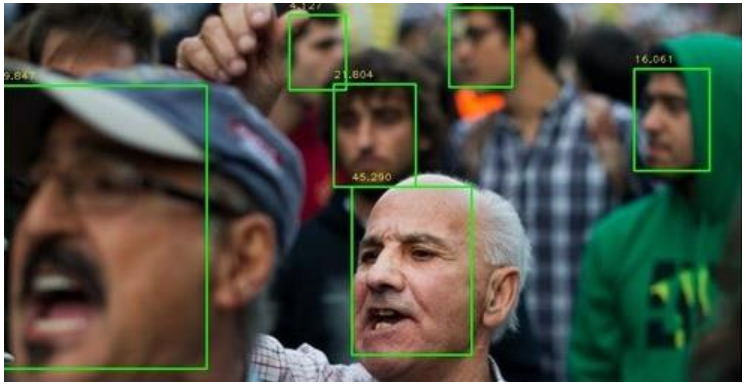


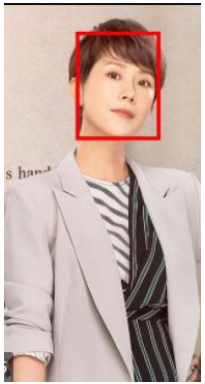


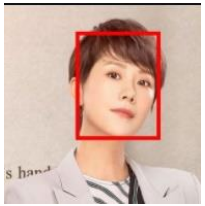


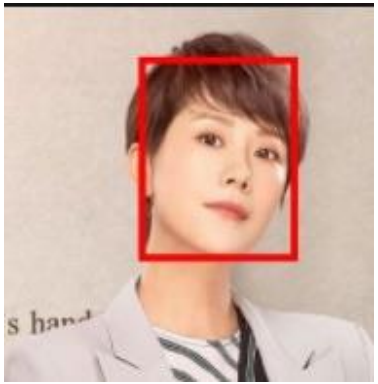
深度学习后期人脸检测算法

- 改进SSD的数据增广：动态多尺度中进行随机裁剪时加一个**1:1**的限制









深度学习后期人脸检测算法

高效率的人脸检测算法

- 基础网络为专门设计的轻量级的网络结构
- 在实际场景中，检测大于30个像素的人脸，有着满足需求的检测精度
- 能够在资源受限的前端设备（CPU、ARM、FPGA等）上实时的运行
- 追求检测速度和检测精度的平衡，满足实用性

深度学习后期人脸检测算法

高效率的人脸检测算法

- 基础网络为专门设计的轻量级的网络结构
- 在实际场景中，检测大于30个像素的人脸，有着满足需求的检测精度
- 能够在资源受限的前端设备（CPU、ARM、FPGA等）上实时的运行
- 追求检测速度和检测精度的平衡，满足实用性

高精度的人脸检测算法

- 基础网络为重量级的VGG16或ResNet-50/101/152等
- 在复杂场景下，有着非常高的检测精度，非常小的人脸也能检测
- 可以在高性能的GPU设备上实时的运行
- 追求极致的检测精度，检测速度可以不考虑

深度学习后期人脸检测算法

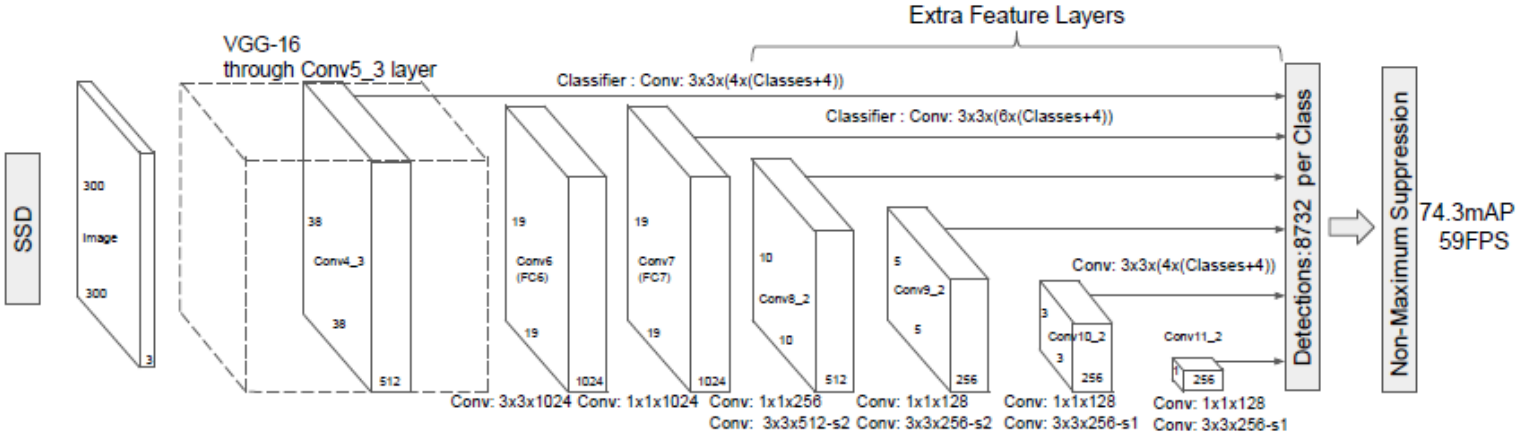
高效率的人脸检测算法

- 基础网络为专门设计的轻量级的网络结构
- 在实际场景中，检测大于30个像素的人脸，有着满足需求的检测精度
- 能够在资源受限的前端设备（CPU、ARM、FPGA等）上实时的运行
- 追求检测速度和检测精度的平衡，满足实用性

高精度的人脸检测算法

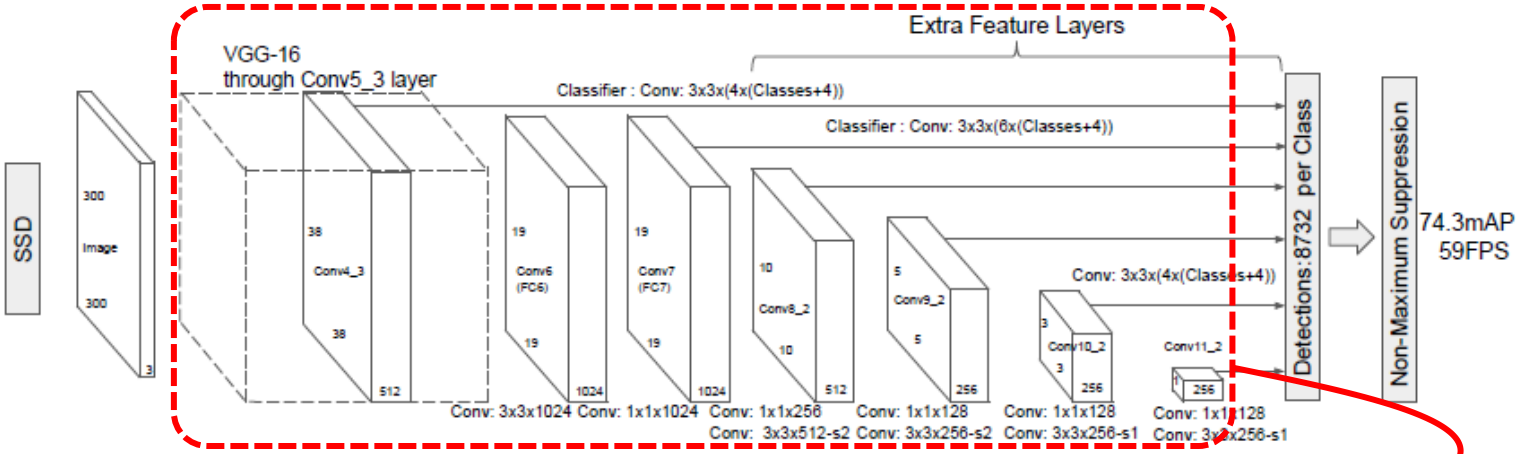
- 基础网络为重量级的VGG16或ResNet-50/101/152等
- 在复杂场景下，有着非常高的检测精度，非常小的人脸也能检测
- 可以在高性能的GPU设备上实时的运行
- 追求极致的检测精度，检测速度可以不考虑

深度学习后期人脸检测算法：高效率



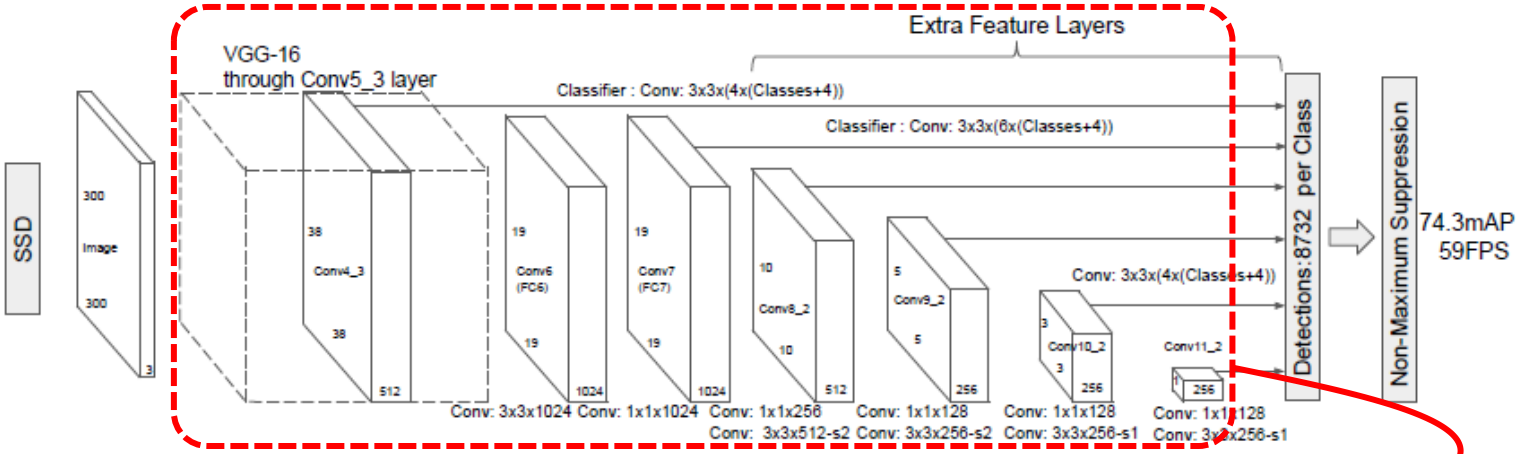
- 高效率的深度学习后期人脸检测算法几乎都是基于通用物体检测算法SSD进行的改进

深度学习后期人脸检测算法：高效率



- 高效率的深度学习后期人脸检测算法几乎都是基于通用物体检测算法SSD进行的改进
- SSD算法不能够在资源受限的设备上实时运行的主要原因是基础网络VGG16太耗时

深度学习后期人脸检测算法：高效率



- 高效率的深度学习后期人脸检测算法几乎都是基于通用物体检测算法SSD进行的改进
- SSD算法不能够在资源受限的设备上实时运行的主要原因是基础网络VGG16太耗时
- 为了资源受限的设备上实时运行，对SSD的主要改进点是设计一个**轻量级的网络结构**
- 在实际场景中，能够在**后端设备**（CPU/ARM等）上实时的运行，且有着满足需求的检测精度
- 代表性论文：FaceBoxes, BlazeFace, LFFD, EagleEye, CenterFace

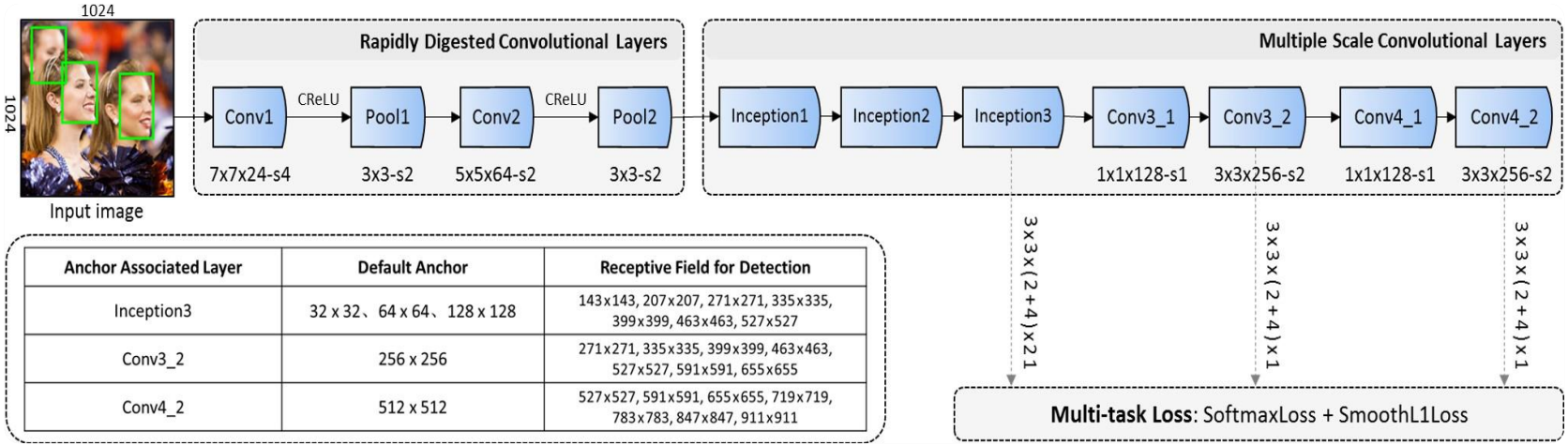
高效率的深度学习后期人脸检测算法：FaceBoxes

- 基于通用物体检测算法SSD所设计的一个轻量级人脸检测算法
- 主要改进点是：①设计了一个轻量级网络结构；②提出了一个锚框密集化操作
- 对于VGA图像（640x480），能够在CPU上以20 FPS的速度运行，且有着满足需求的检测精度
- 所有训练代码、测试代码、模型等已经在github上开源，有Caffe和Pytorch两个版本

Approach	CPU-model	mAP(%)	FPS
ACF [40]	i7-3770@3.40	85.2	20
CasCNN [16]	E5-2620@2.00	85.7	14
FaceCraft [26]	N/A	90.8	10
STN [5]	i7-4770K@3.50	91.5	10
MTCNN [45]	N/A@2.60	94.4	16
Ours	E5-2660v3@2.60	**96.0**	**20**

FaceBoxes: A CPU Real-time Face Detector with High Accuracy, IJCB 2017

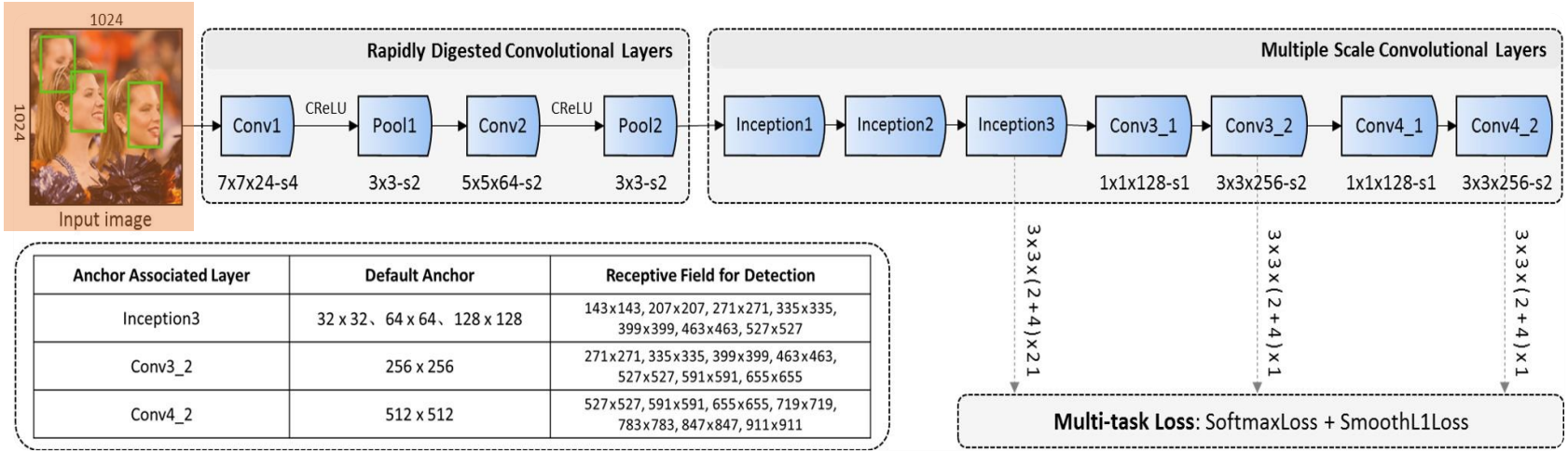
高效率的FaceBoxes人脸检测算法：整体框架



Anchor Associated Layer	Default Anchor	Receptive Field for Detection
Inception3	32 x 32、64 x 64、128 x 128	143x143, 207x207, 271x271, 335x335, 399x399, 463x463, 527x527
Conv3_2	256 x 256	271x271, 335x335, 399x399, 463x463, 527x527, 591x591, 655x655
Conv4_2	512 x 512	527x527, 591x591, 655x655, 719x719, 783x783, 847x847, 911x911

- 整体框架 = 输入图像 + 快速消化网络模块 + 多尺度网络模块 + 锚框设计

高效率的FaceBoxes人脸检测算法：输入图像



Anchor Associated Layer	Default Anchor	Receptive Field for Detection
Inception3	32 x 32、64 x 64、128 x 128	143x143, 207x207, 271x271, 335x335, 399x399, 463x463, 527x527
Conv3_2	256 x 256	271x271, 335x335, 399x399, 463x463, 527x527, 591x591, 655x655
Conv4_2	512 x 512	527x527, 591x591, 655x655, 719x719, 783x783, 847x847, 911x911

- 整体框架 = 输入图像 + 快速消化网络模块 + 多尺度网络模块 + 锚框设计
- 输入图像：经过改进的SSD数据增广，得到一张1024x1024的图像作为训练输入

高效率的FaceBoxes人脸检测算法：输入图像

训练输入图像
1024x1024

高效率的FaceBoxes人脸检测算法：输入图像

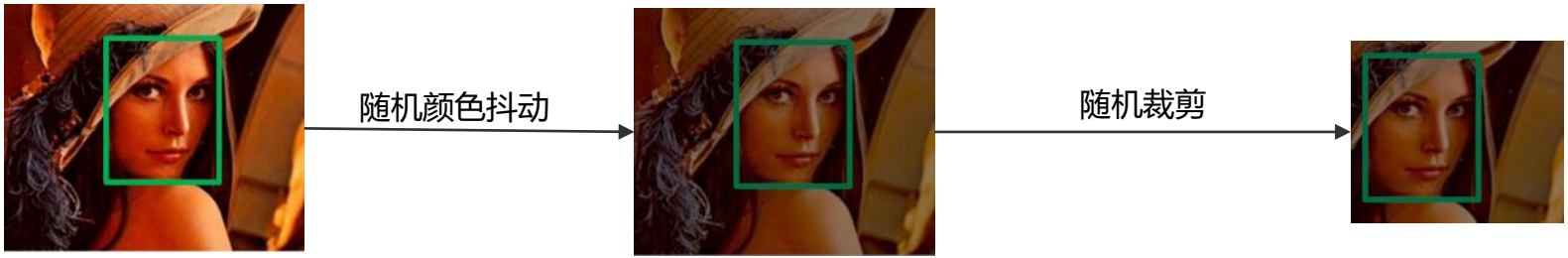
训练输入图像
1024x1024

高效率的FaceBoxes人脸检测算法：输入图像

训练输入图像
1024x1024

高效率的FaceBoxes人脸检测算法：输入图像

训练输入图像
1024x1024

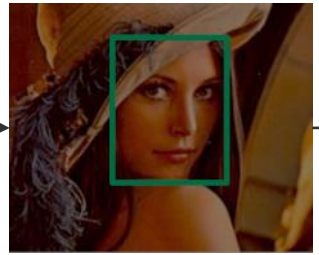


高效率的FaceBoxes人脸检测算法：输入图像

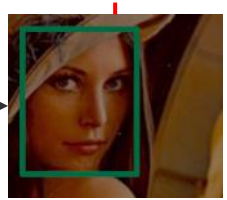
训练输入图像
1024x1024

非正方形图像缩放到正方形图像，图像比例被改变，人脸比例随之改变

随机颜色抖动

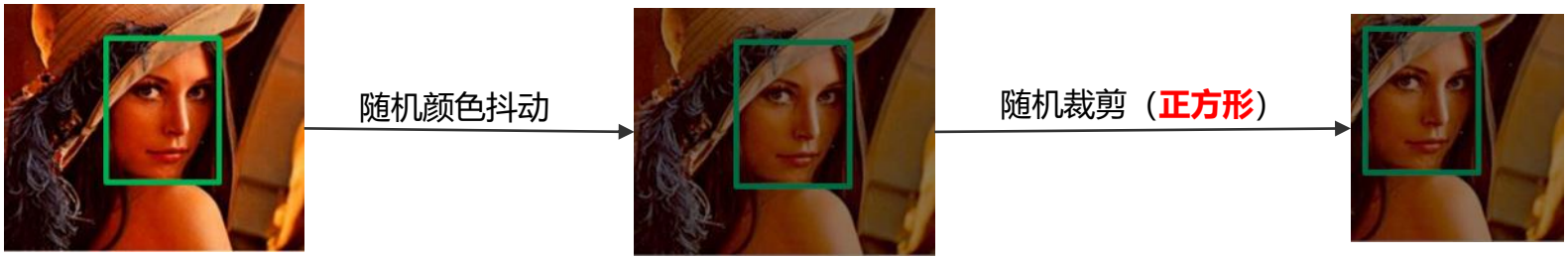


随机裁剪



高效率的FaceBoxes人脸检测算法：输入图像

训练输入图像
1024x1024

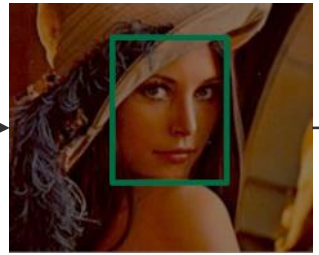


高效率的FaceBoxes人脸检测算法：输入图像

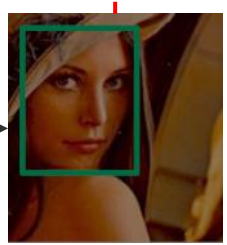
训练输入图像
1024x1024

正方形图像缩放到正方形图像，图像比例没有改变，人脸比例不会被改变

随机颜色抖动



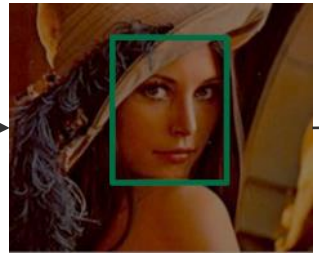
随机裁剪（**正方形**）



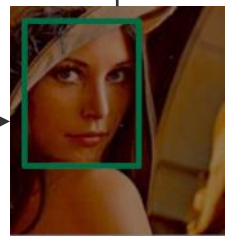
高效率的FaceBoxes人脸检测算法：输入图像

训练输入图像
1024x1024

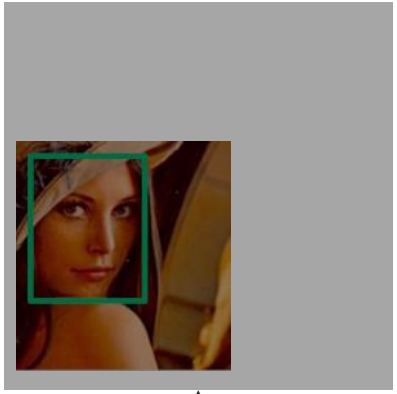
随机颜色抖动



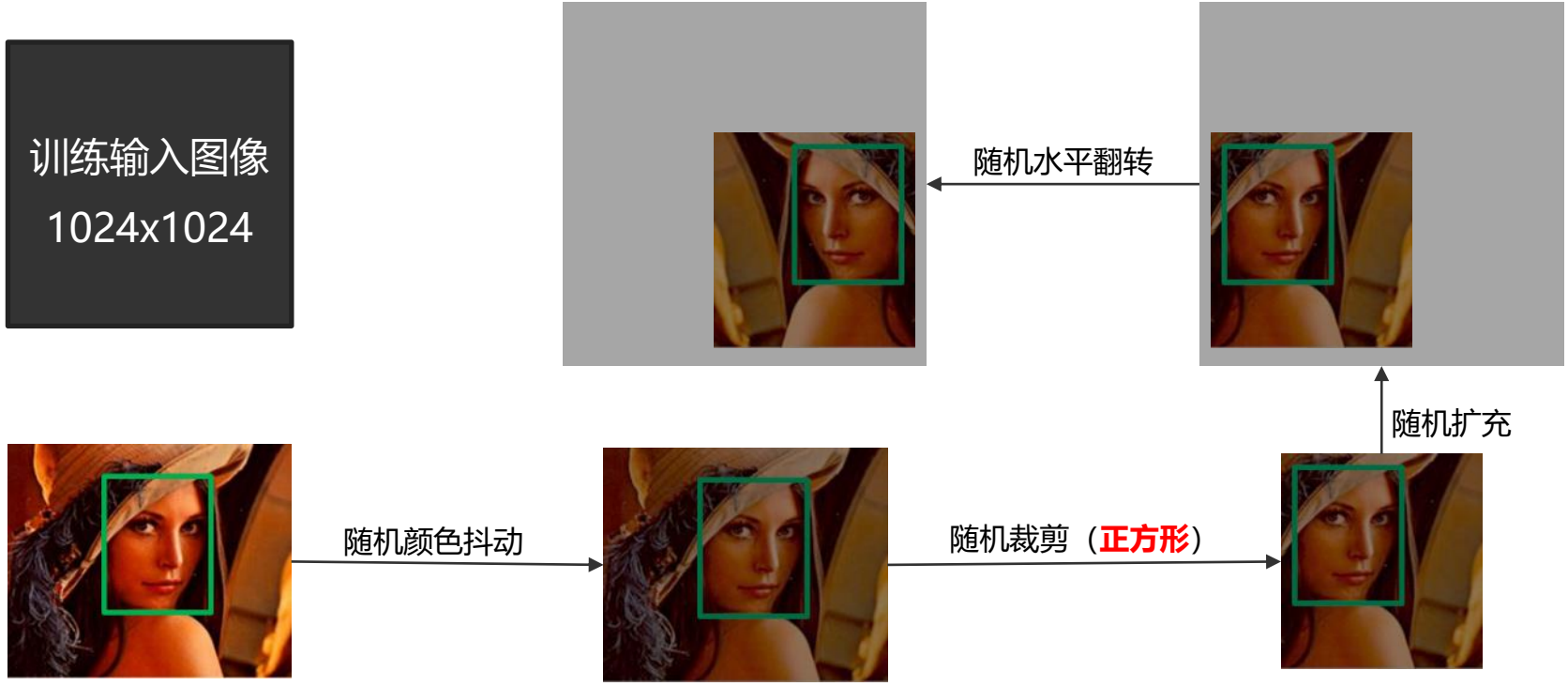
随机裁剪（**正方形**）



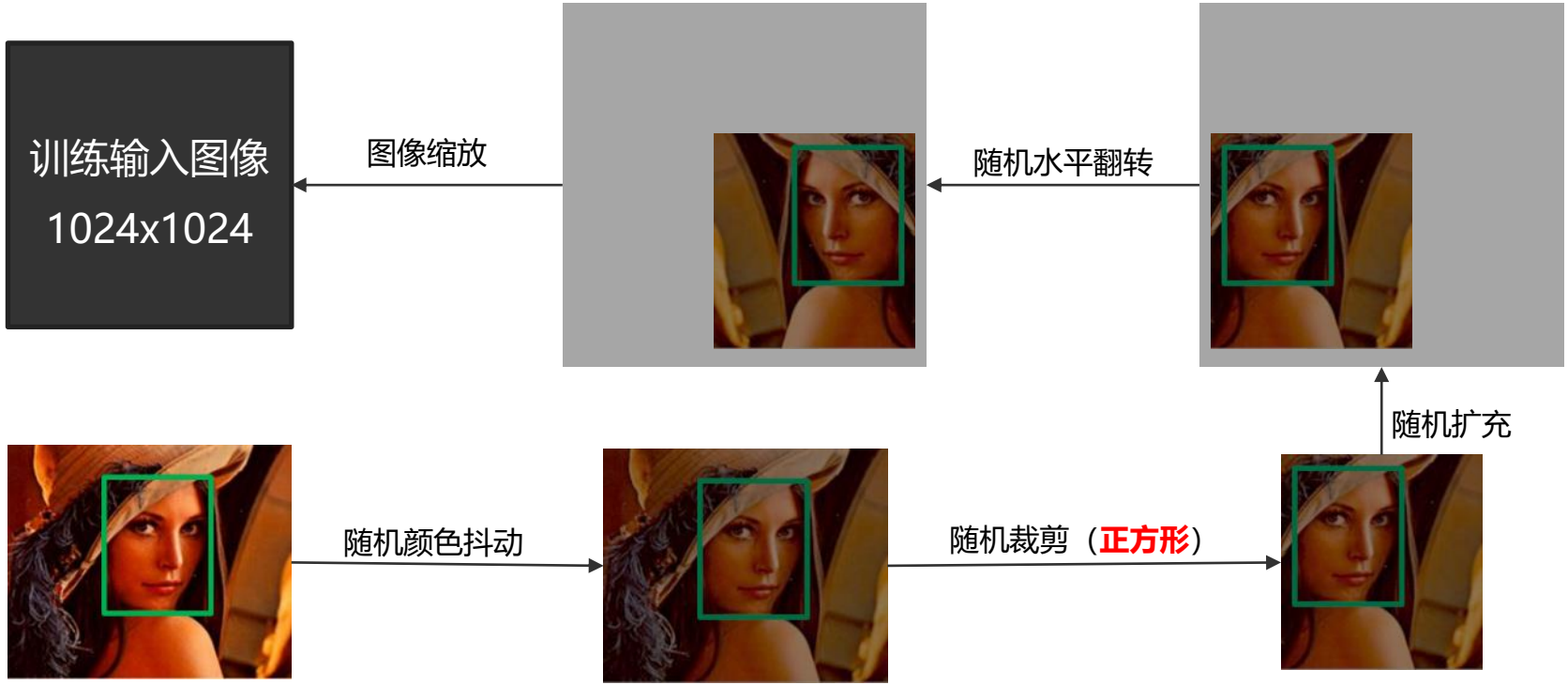
随机扩充



高效率的FaceBoxes人脸检测算法：输入图像

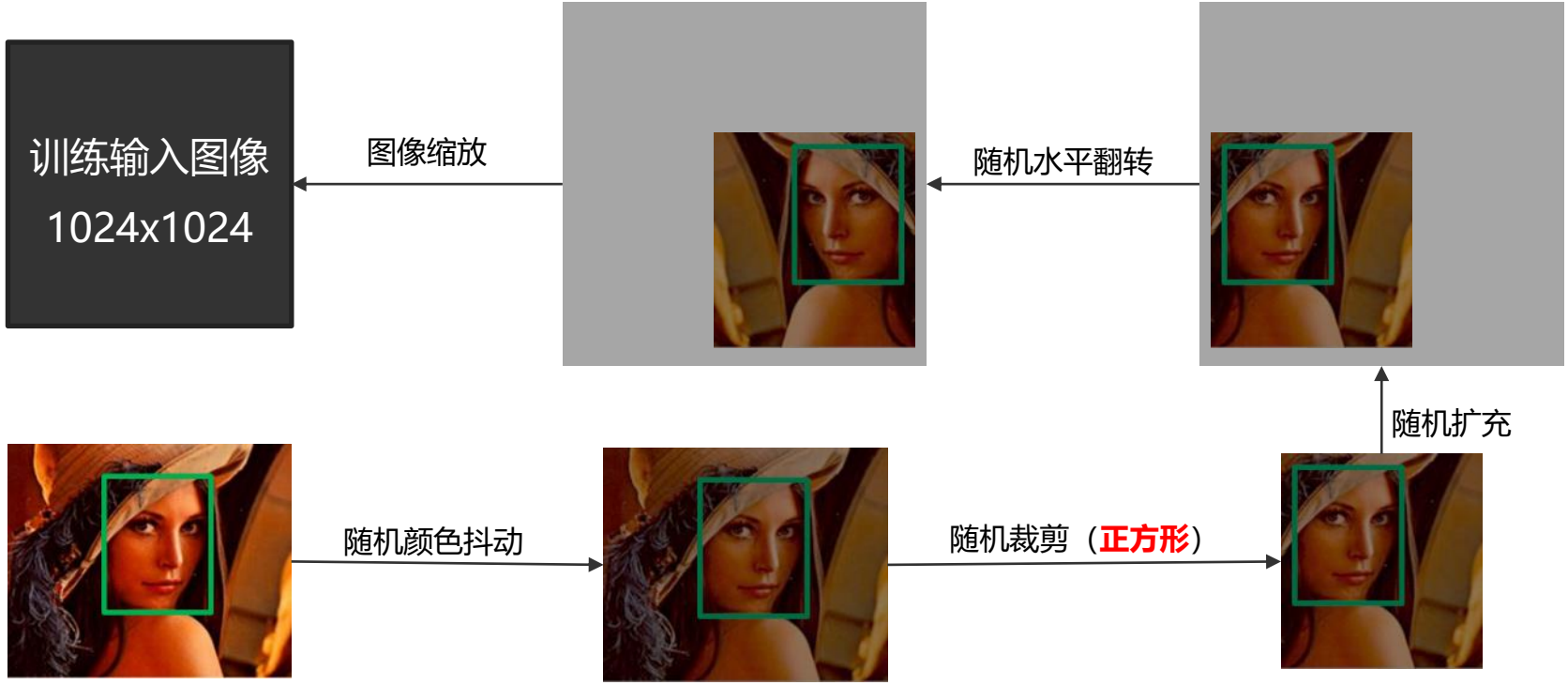


高效率的FaceBoxes人脸检测算法：输入图像

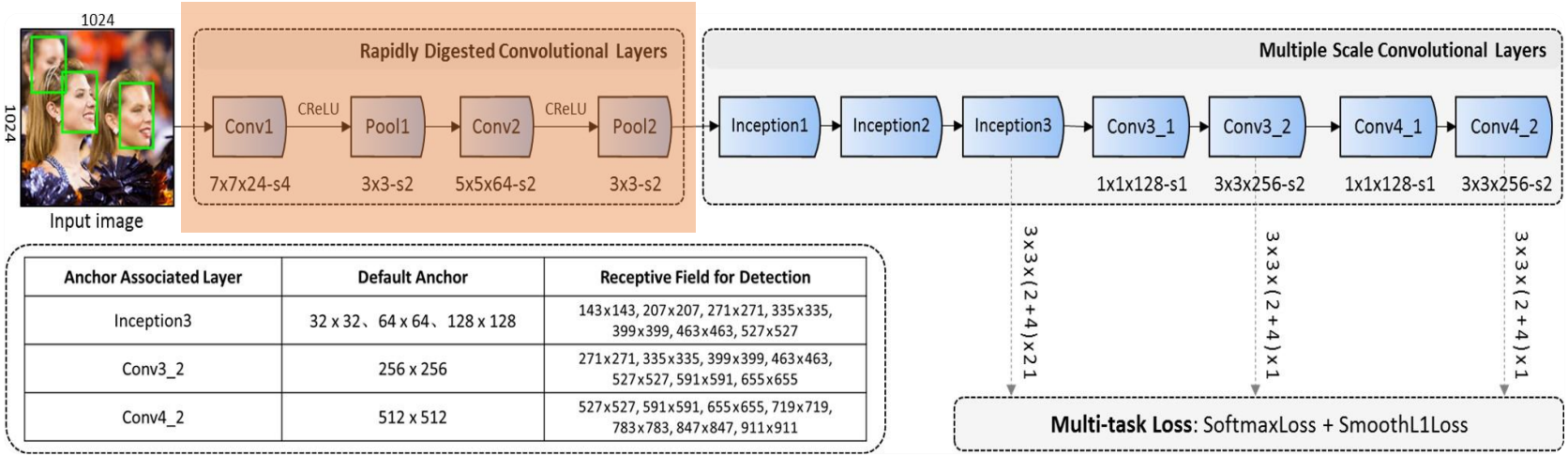


高效率的FaceBoxes人脸检测算法：输入图像

此过程是训练阶段的处理，测试阶段直接输入原图或等比例缩放图像



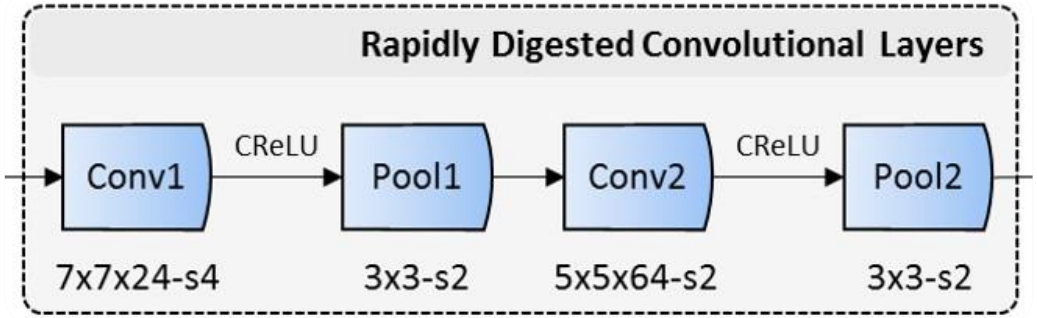
高效率的FaceBoxes人脸检测算法：快速消化网络模块



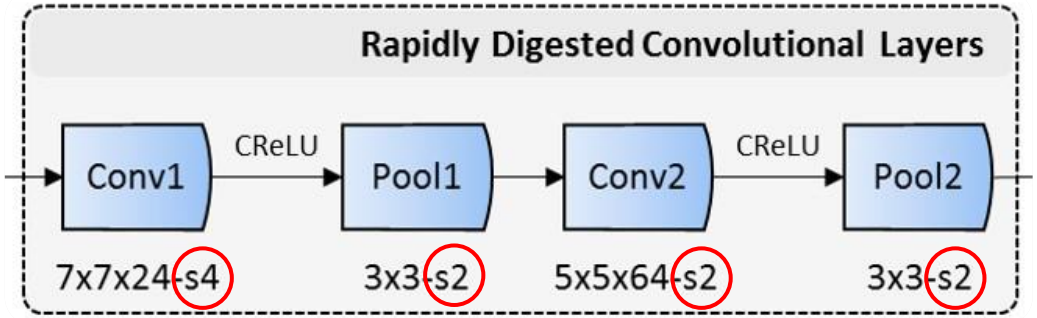
Anchor Associated Layer	Default Anchor	Receptive Field for Detection
Inception3	32 x 32、64 x 64、128 x 128	143x143, 207x207, 271x271, 335x335, 399x399, 463x463, 527x527
Conv3_2	256 x 256	271x271, 335x335, 399x399, 463x463, 527x527, 591x591, 655x655
Conv4_2	512 x 512	527x527, 591x591, 655x655, 719x719, 783x783, 847x847, 911x911

- 整体框架 = 输入图像 + 快速消化网络模块 + 多尺度网络模块 + 锚框设计
- 输入图像：经过改进的SSD数据增广，得到一张1024x1024的图像作为训练输入
- 快速消化网络模块：快速地对图像进行32倍的下采样，以达到CPU实时的检测速度

高效率的FaceBoxes人脸检测算法：快速消化网络模块

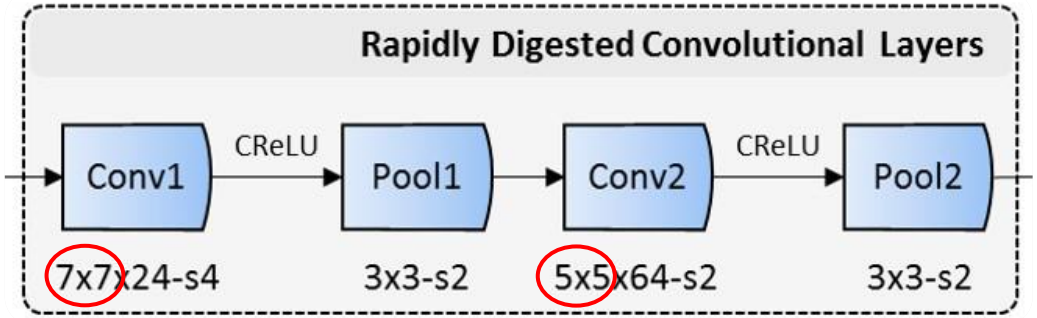


高效率的FaceBoxes人脸检测算法：快速消化网络模块



1. 快速降地低输入图像的空间尺寸：空间尺寸越大，卷积速度越慢，会导致整个网络的耗时较大

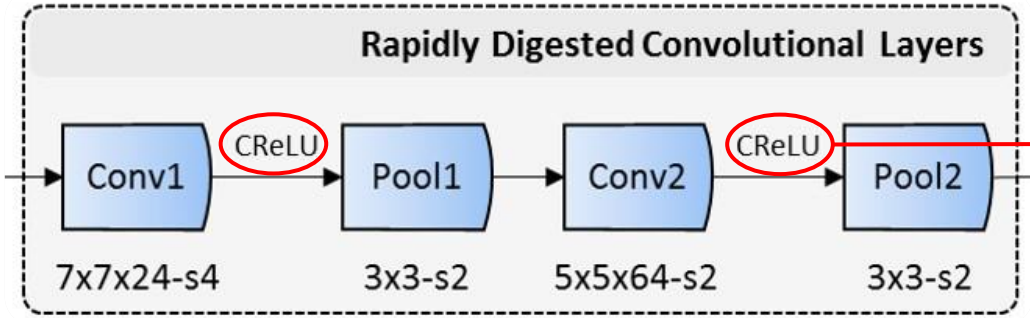
高效率的FaceBoxes人脸检测算法：快速消化网络模块

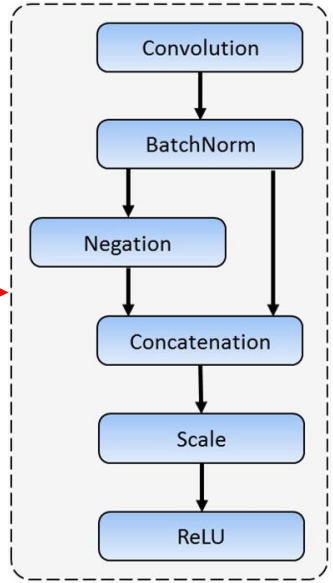


1. 快速降地低输入图像的空间尺寸：空间尺寸越大，卷积速度越慢，会导致整个网络的耗时较大

2. 选择大小合适的卷积核：卷积核太大会导致速度很慢，卷积核太小不能弥补下采样带来的信息损失

高效率的FaceBoxes人脸检测算法：快速消化网络模块



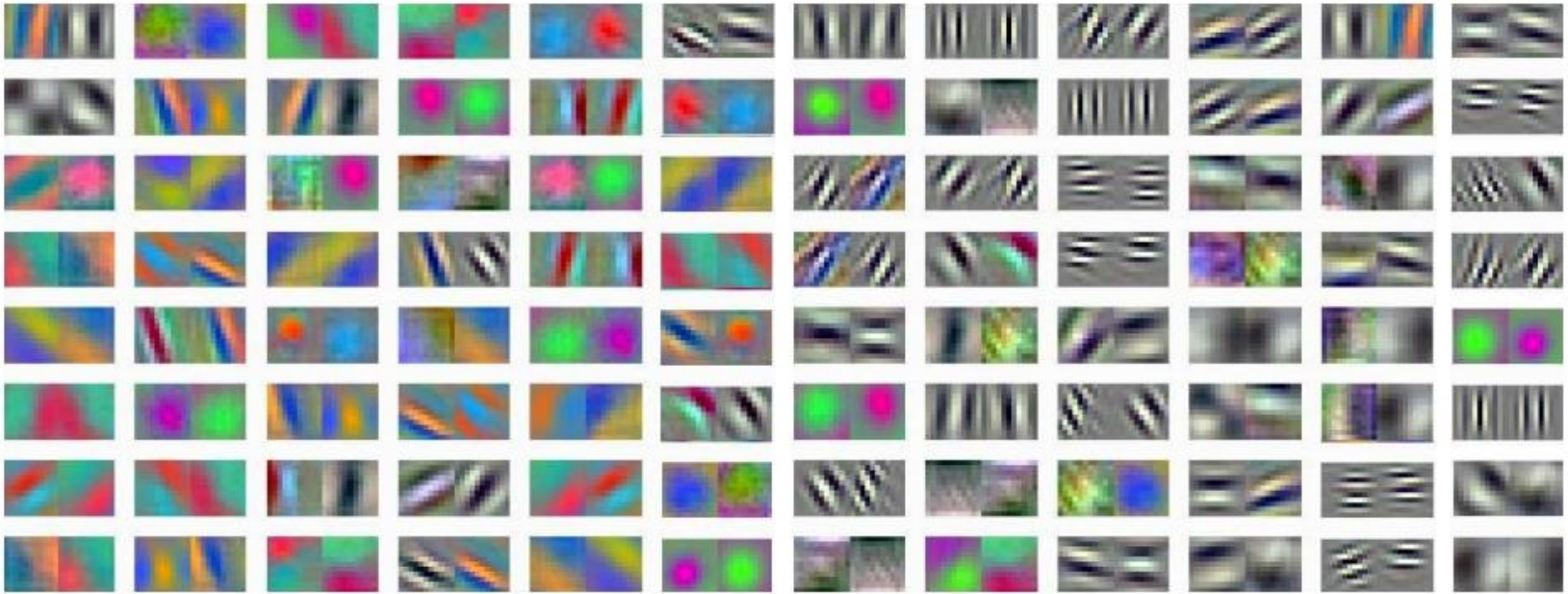


1. 快速降地低输入图像的空间尺寸：空间尺寸越大，卷积速度越慢，会导致整个网络的耗时较大

2. 选择大小合适的卷积核：卷积核太大会导致速度很慢，卷积核太小不能弥补下采样带来的信息损失

3. 有效地减少输出的channel数：利用CReLU激活函数，能够减少卷积输出的通道，从而加快速度

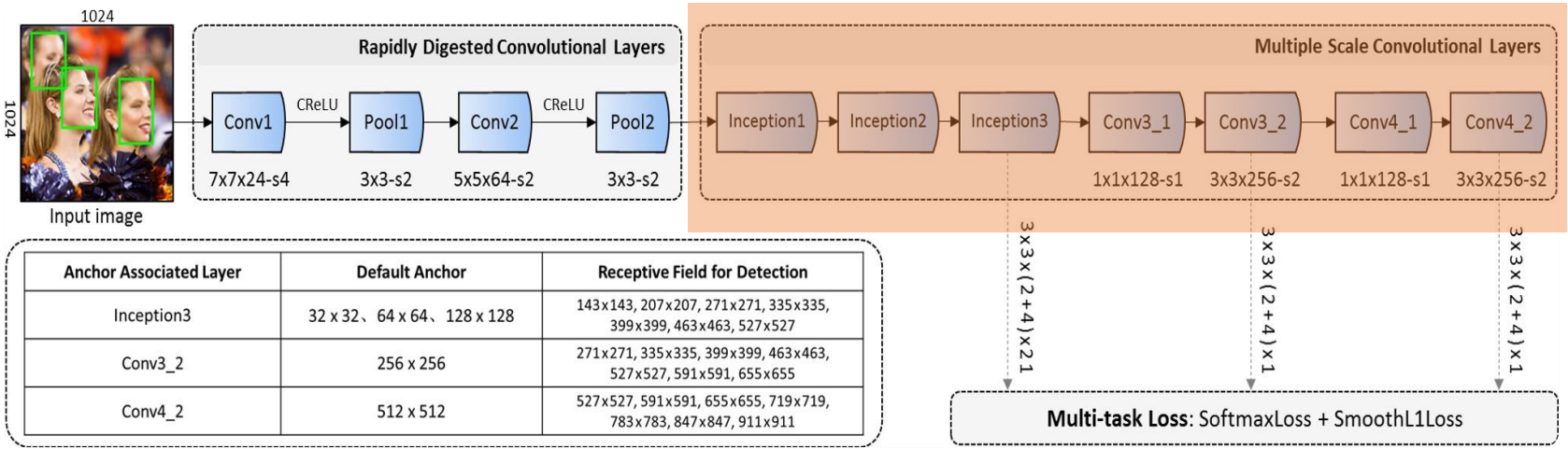
高效率的FaceBoxes人脸检测算法：快速消化网络模块



- 卷积网络前部的卷积核的参数有较强的负相关性，如第一层卷积网络的两个卷积核，对应的参数是互为相反数
- 网络的前部倾向于同时捕获正负相位的信息，但ReLU会抹掉负响应，造成了一部分卷积核没有起到作用
- CReLU则两边的信息都考虑：CReLU(x)=[ReLU(x), ReLU(−x)]

Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units, ICML 2016

高效率的FaceBoxes人脸检测算法：多尺度网络模块

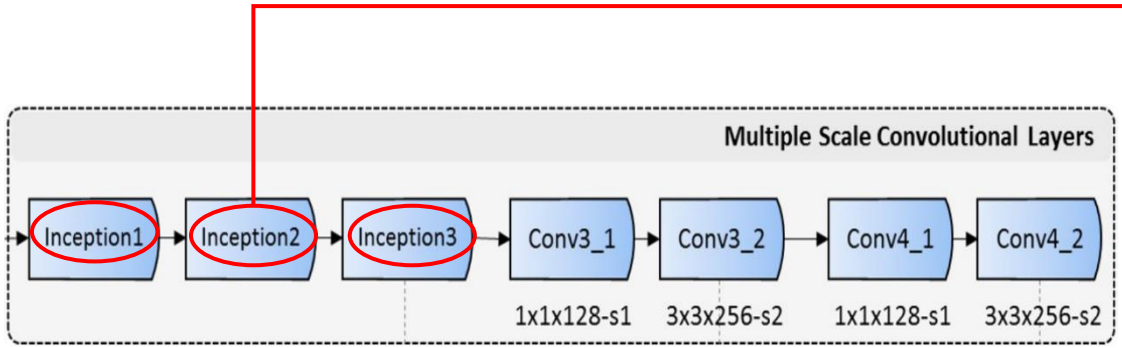


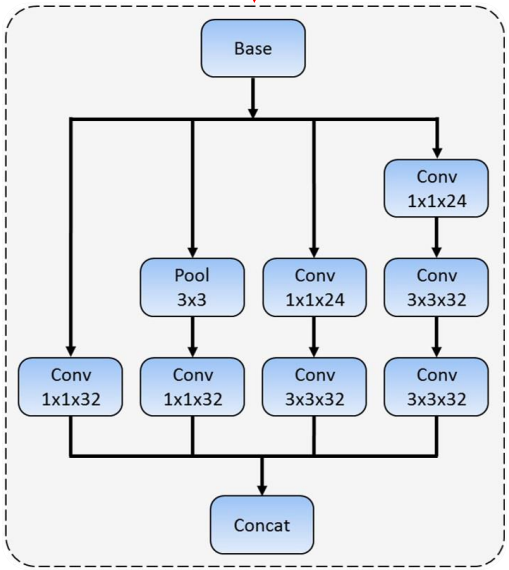
Anchor Associated Layer	Default Anchor	Receptive Field for Detection
Inception3	32 x 32、64 x 64、128 x 128	143x143, 207x207, 271x271, 335x335, 399x399, 463x463, 527x527
Conv3_2	256 x 256	271x271, 335x335, 399x399, 463x463, 527x527, 591x591, 655x655
Conv4_2	512 x 512	527x527, 591x591, 655x655, 719x719, 783x783, 847x847, 911x911

- 整体框架 = 输入图像 + 快速消化网络模块 + 多尺度网络模块 + 锚框设计
- 输入图像：经过改进的SSD数据增广，得到一张1024x1024的图像作为训练输入
- 快速消化网络模块：快速地对图像进行32倍的下采样，以达到CPU实时的检测速度
- 多尺度网络模块：在网络宽度和深度这两个维度上，对特征进行强化以提高检测精度

高效率的FaceBoxes人脸检测算法：多尺度网络模块

- “网络宽度”维度上的多尺度

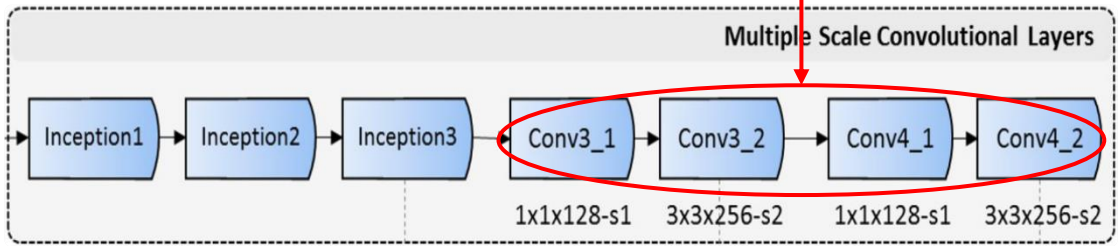


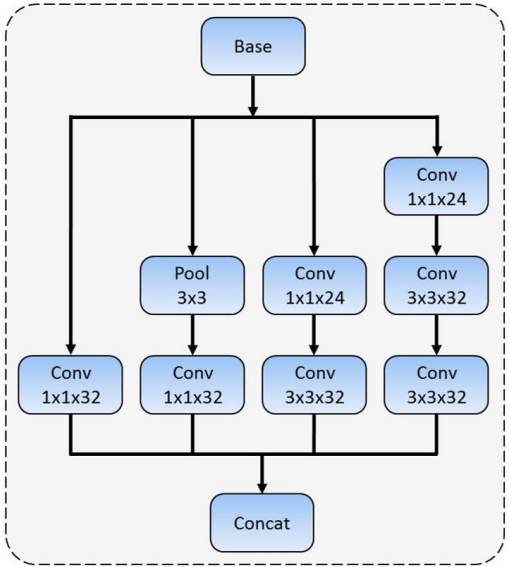


Going Deeper with Convolutions, CVPR 2015

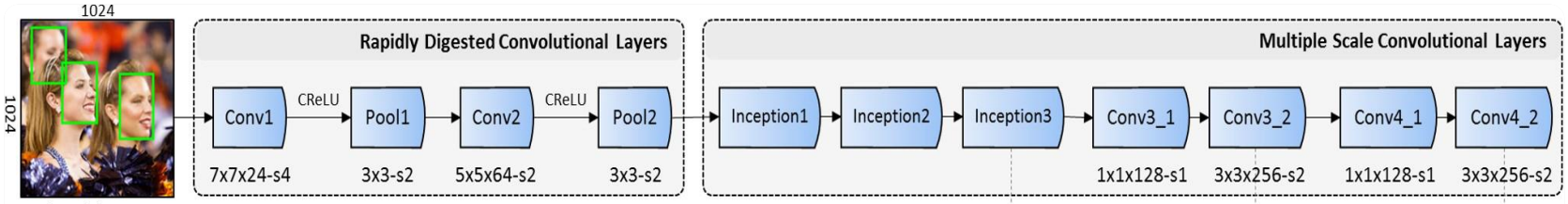
高效率的FaceBoxes人脸检测算法：多尺度网络模块

- “网络宽度”维度上的多尺度
- “网络深度”维度上的多尺度





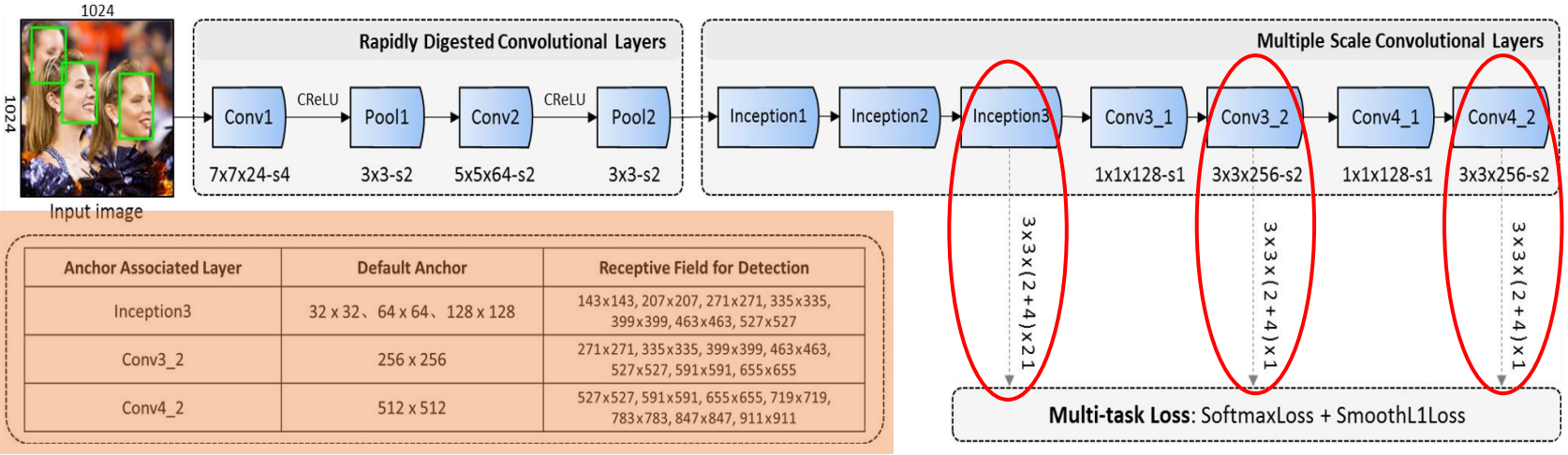
高效率的FaceBoxes人脸检测算法：整体框架



Anchor Associated Layer	Default Anchor	Receptive Field for Detection
Inception3	32 x 32、64 x 64、128 x 128	143x143, 207x207, 271x271, 335x335, 399x399, 463x463, 527x527
Conv3_2	256 x 256	271x271, 335x335, 399x399, 463x463, 527x527, 591x591, 655x655
Conv4_2	512 x 512	527x527, 591x591, 655x655, 719x719, 783x783, 847x847, 911x911

- 整体框架 = 输入图像 + 快速消化网络模块 + 多尺度网络模块 + 锚框设计
- 输入图像：经过改进的SSD数据增广，得到一张1024x1024的图像作为训练输入
- 快速消化网络模块：快速地对图像进行32倍的下采样，以达到CPU实时的检测速度
- 多尺度网络模块：在网络宽度和深度这两个维度上，对特征进行强化以提高检测精度
- 锚框设计：选取3个检测层关联锚框进行人脸检测，并对小尺度锚框进行密集化操作

高效率的FaceBoxes人脸检测算法：整体框架



Anchor Associated Layer	Default Anchor	Receptive Field for Detection
Inception3	32 x 32、64 x 64、128 x 128	143x143, 207x207, 271x271, 335x335, 399x399, 463x463, 527x527
Conv3_2	256 x 256	271x271, 335x335, 399x399, 463x463, 527x527, 591x591, 655x655
Conv4_2	512 x 512	527x527, 591x591, 655x655, 719x719, 783x783, 847x847, 911x911

- 整体框架 = 输入图像 + 快速消化网络模块 + 多尺度网络模块 + 锚框设计
- 输入图像：经过改进的SSD数据增广，得到一张1024x1024的图像作为训练输入
- 快速消化网络模块：快速地对图像进行32倍的下采样，以达到CPU实时的检测速度
- 多尺度网络模块：在网络宽度和深度这两个维度上，对特征进行强化以提高检测精度
- 锚框设计：选取3个检测层关联锚框进行人脸检测，并对小尺度锚框进行密集化操作

高效率的FaceBoxes人脸检测算法：锚框密集化

锚框的关联层	关联层对应的下采样倍数	对应的锚框尺度	锚框的铺设间隔
Inception3	32	32 x 32、64 x 64、128 x 128	32
Conv3_2	64	256 x 256	64
Conv4_2	128	512 x 512	128

高效率的FaceBoxes人脸检测算法：锚框密集化

锚框的关联层	关联层对应的下采样倍数	对应的锚框尺度	锚框的铺设间隔
Inception3	32	32 x 32、64 x 64、128 x 128	32
Conv3_2	64	256 x 256	64
Conv4_2	128	512 x 512	128

$$\text{锚框的铺设密度} = \frac{\text{锚框的尺度}}{\text{锚框的铺设间隔}}$$

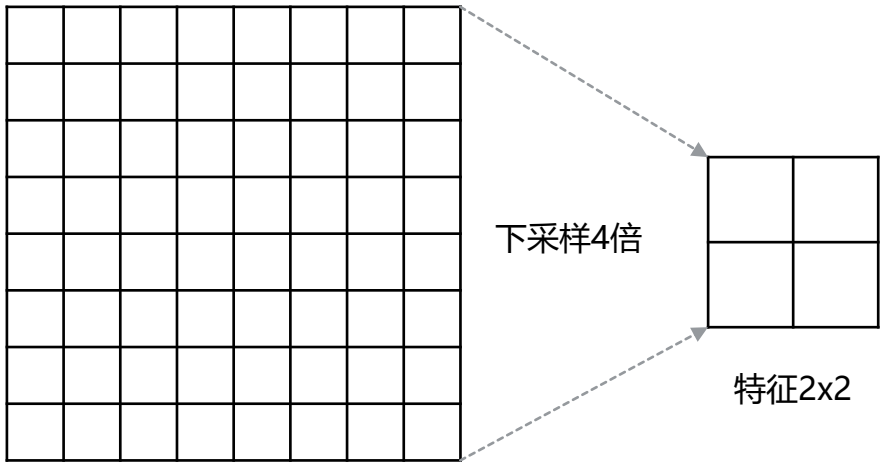
高效率的FaceBoxes人脸检测算法：锚框密集化

锚框的关联层	关联层对应的下采样倍数	对应的锚框尺度	锚框的铺设间隔
Inception3	32	32 x 32、64 x 64、128 x 128	32
Conv3_2	64	256 x 256	64
Conv4_2	128	512 x 512	128

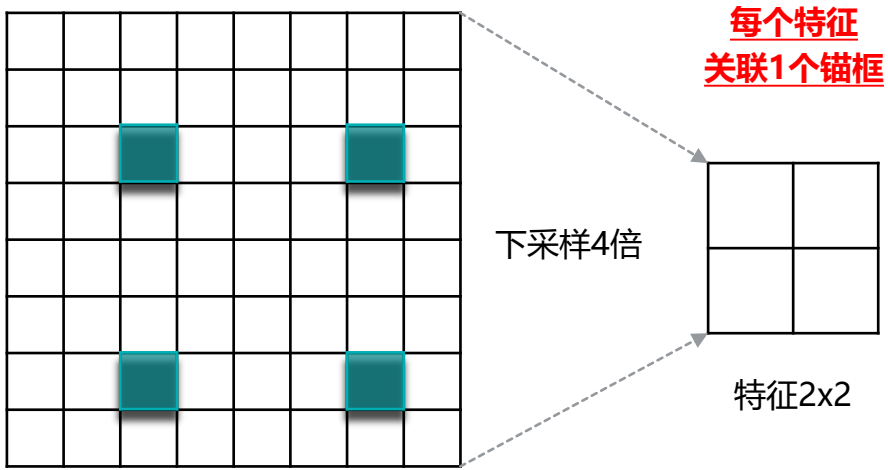
$$\text{锚框的铺设密度} = \frac{\text{锚框的尺度}}{\text{锚框的铺设间隔}}$$

锚框尺度	32	64	128	256	512
锚框铺设间隔	32	32	32	64	128
锚框铺设密度	1	2	4	4	4

高效率的FaceBoxes人脸检测算法：锚框密集化

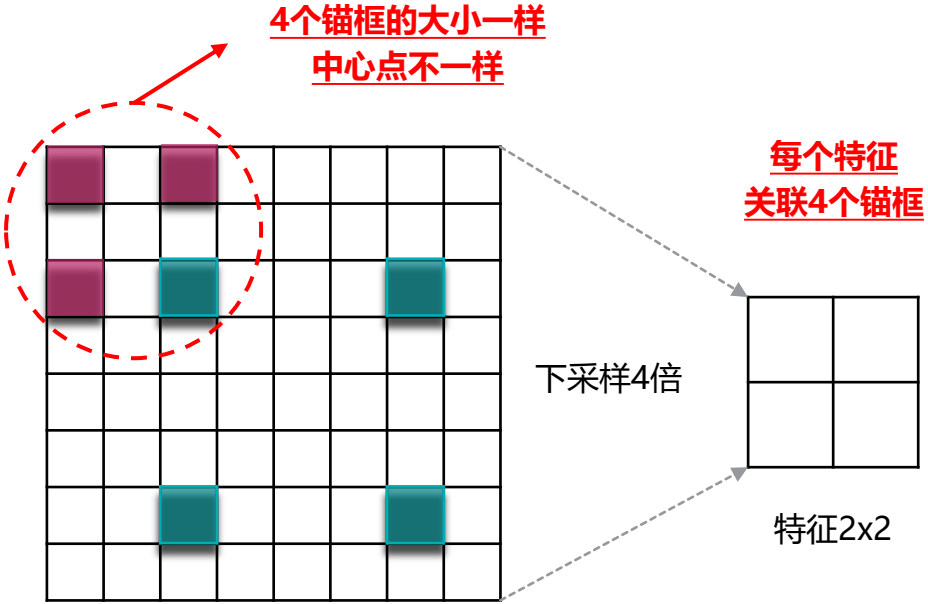


高效率的FaceBoxes人脸检测算法：锚框密集化



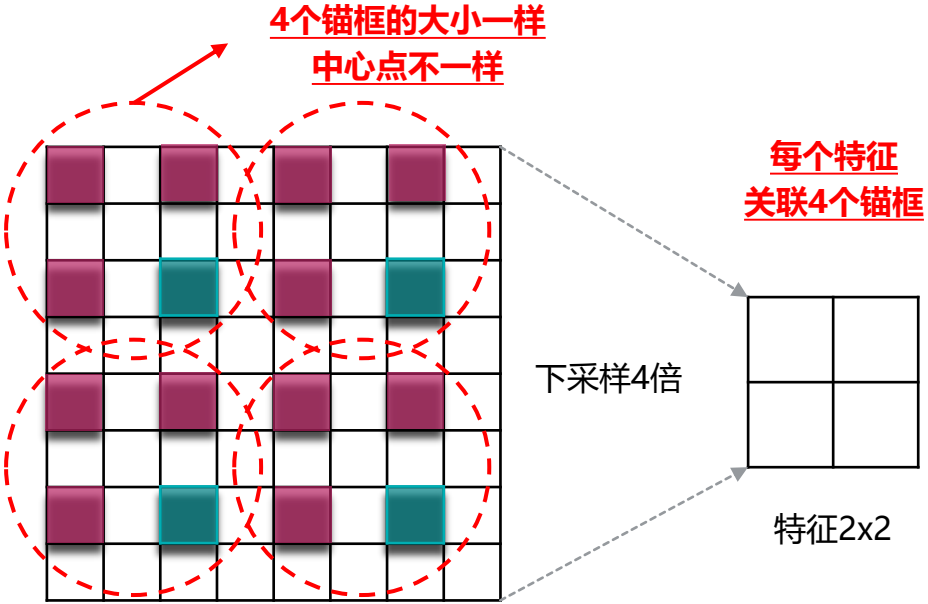
原始的铺设密度1/4

高效率的FaceBoxes人脸检测算法：锚框密集化



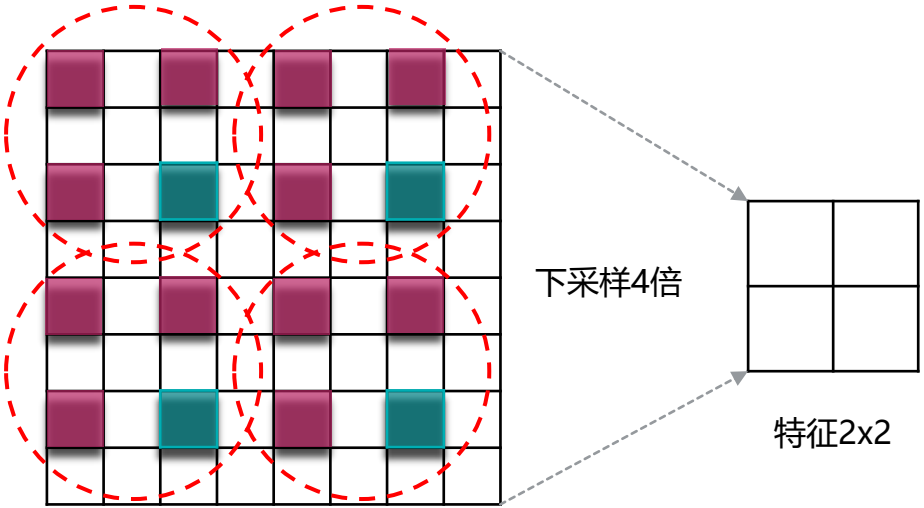
原始的铺设密度1/4

高效率的FaceBoxes人脸检测算法：锚框密集化



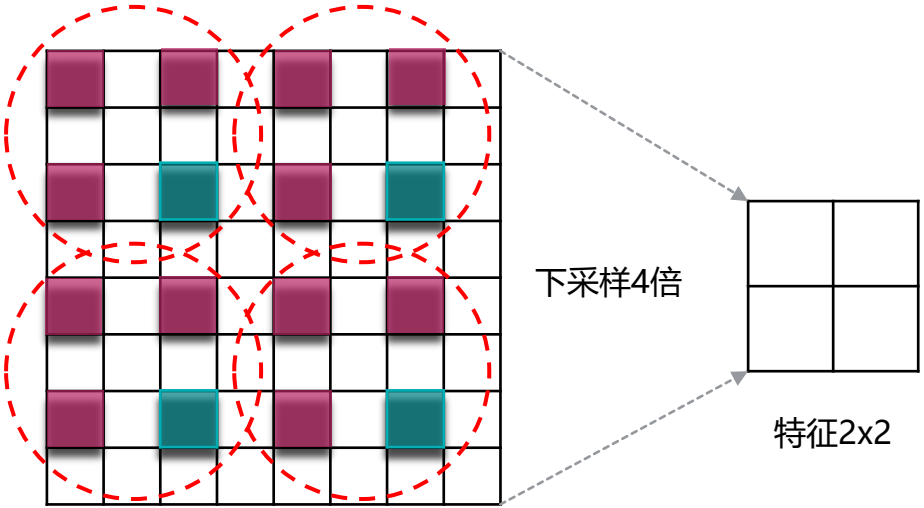
原始的铺设密度1/4

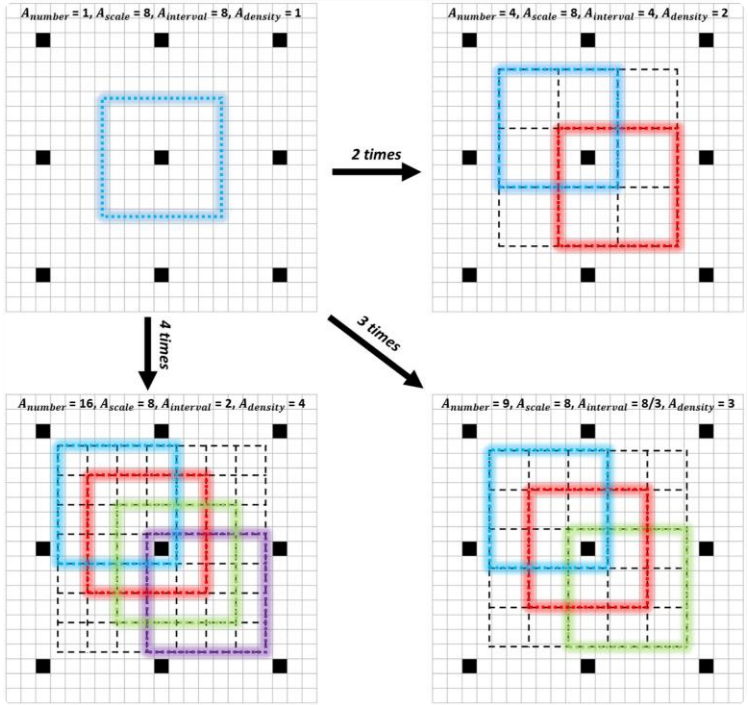
高效率的FaceBoxes人脸检测算法：锚框密集化



原始的铺设密度1/4 密集化 密集化后的铺设密度1/2

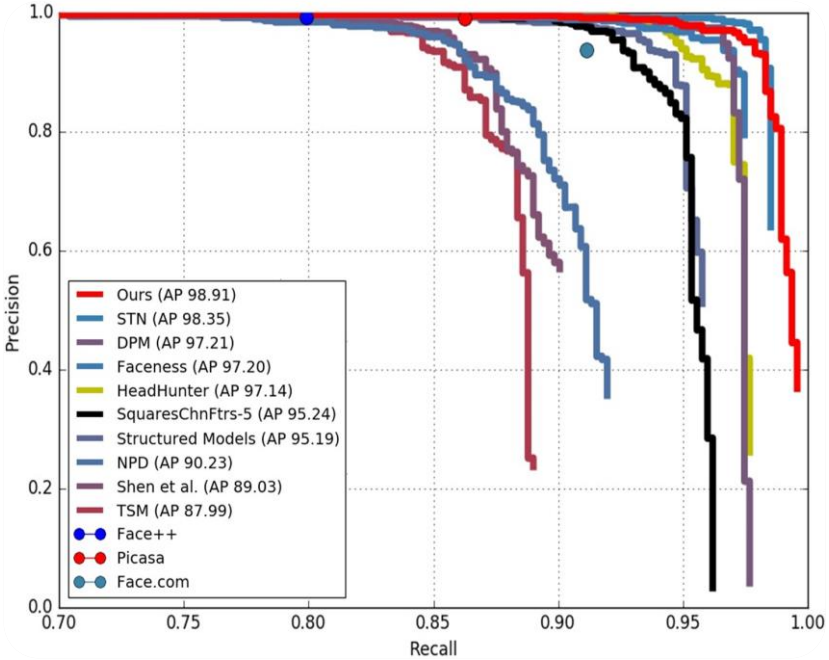
高效率的FaceBoxes人脸检测算法：锚框密集化



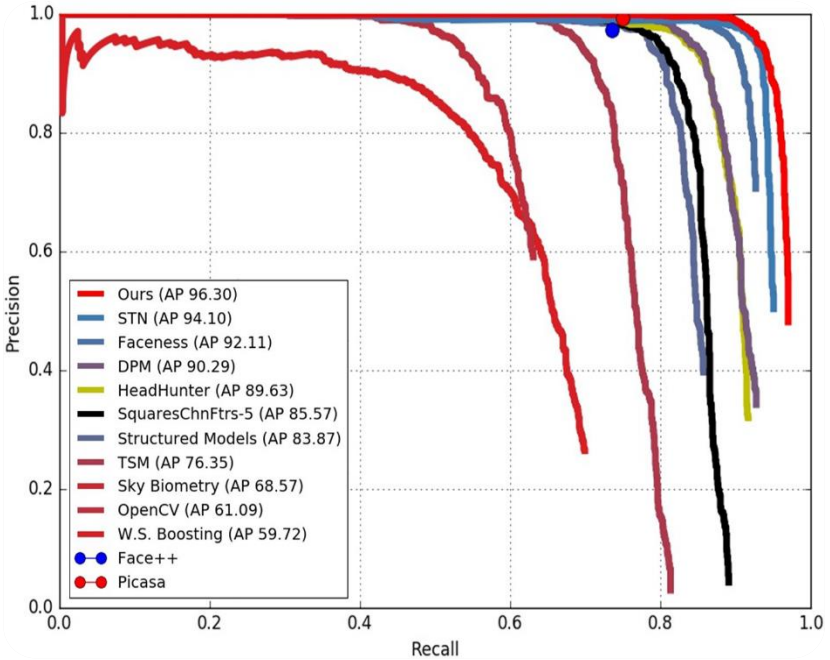


原始的铺设密度1/4 → 密集化 → 密集化后的铺设密度1/2

高效率的FaceBoxes人脸检测算法：检测精度

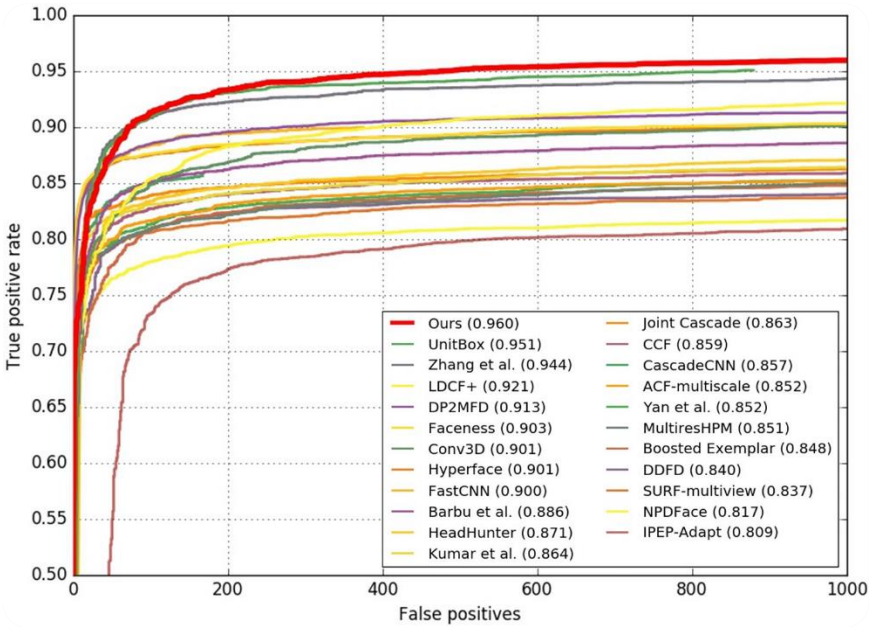


AFW

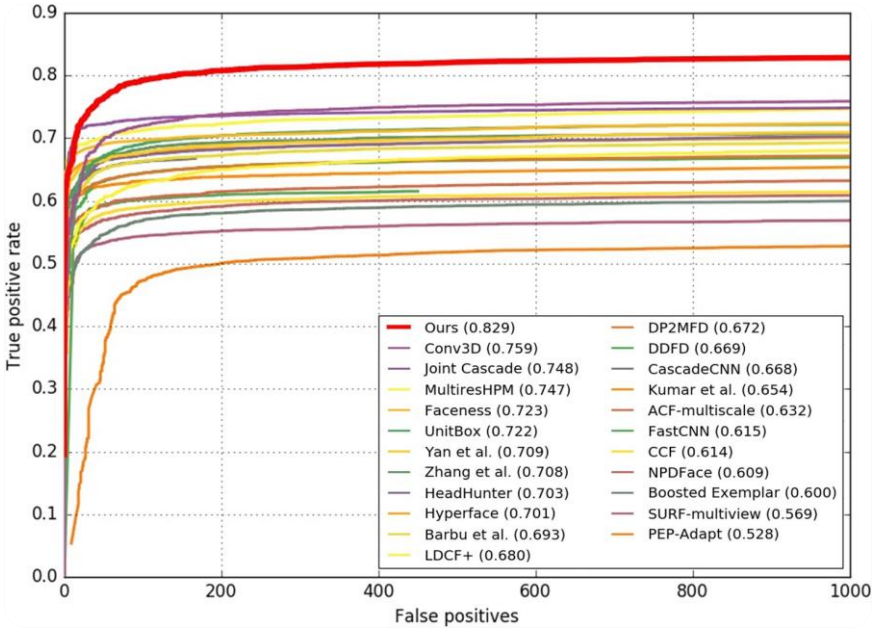


PASCAL Face

高效率的FaceBoxes人脸检测算法：检测精度



FDDB：离散



FDDB：连续

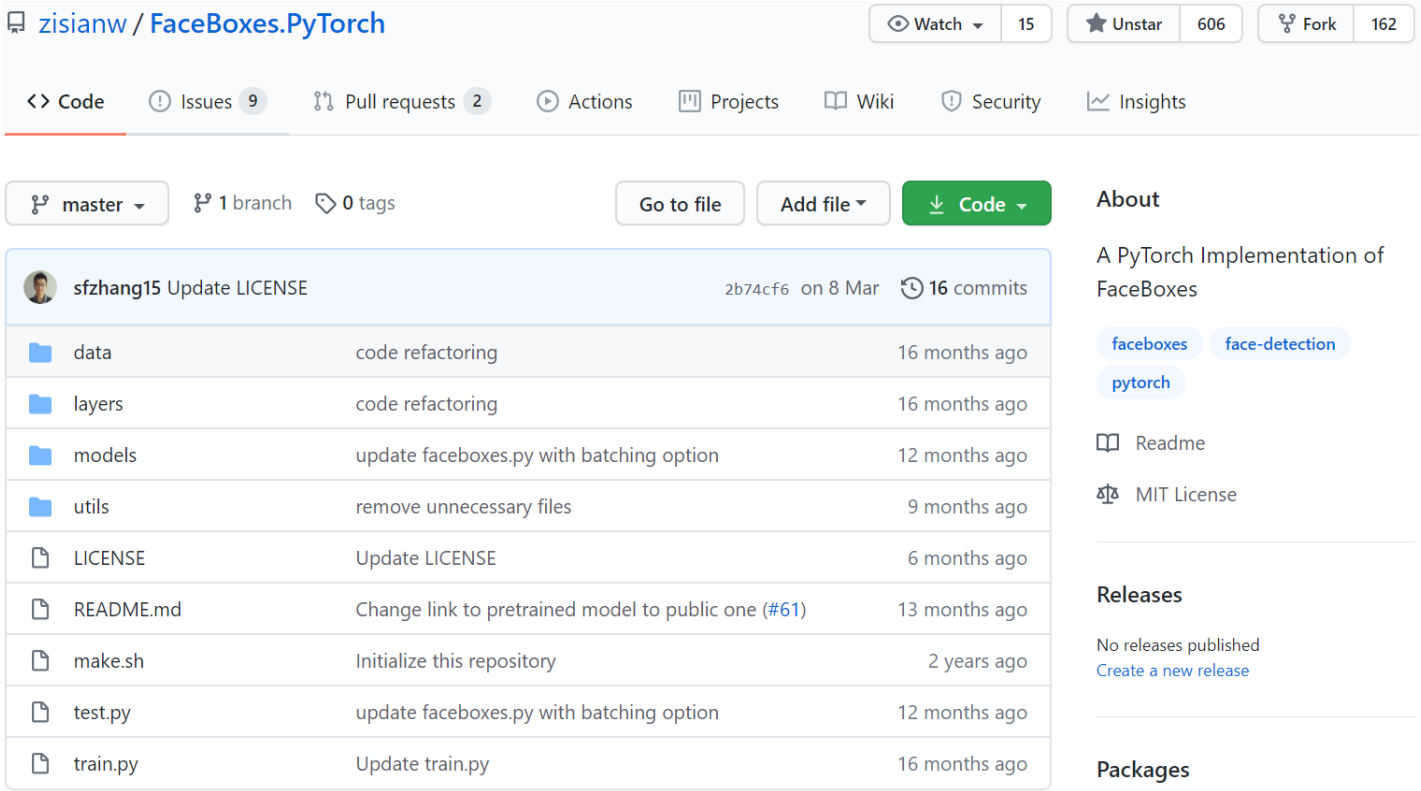
高效率的FaceBoxes人脸检测算法：检测速度

Approach	CPU-model	mAP(%)	FPS
ACF [40]	i7-3770@3.40	85.2	20
CasCNN [16]	E5-2620@2.00	85.7	14
FaceCraft [26]	N/A	90.8	10
STN [5]	i7-4770K@3.50	91.5	10
MTCNN [45]	N/A@2.60	94.4	16
Ours	E5-2660v3@2.60	**96.0**	**20**

VGA图像、CPU设备、FDDB精度

高效率的FaceBoxes人脸检测算法：算法代码

- 代码链接：https://github.com/zisianw/FaceBoxes.PyTorch



高效率的FaceBoxes人脸检测算法：安装编译

- 安装编译步骤：①安装PyTorch；②克隆代码；③编译

Installation

1. Install PyTorch >= v1.0.0 following official instruction.

2. Clone this repository. We will call the cloned directory as `$FaceBoxes_ROOT`.

```
git clone https://github.com/zisianw/FaceBoxes.PyTorch.git
```

3. Compile the nms:

```
./make.sh
```

Note: Codes are based on Python 3+.

高效率的FaceBoxes人脸检测算法：训练模型

■ 训练模型步骤：①下载WIDER FACE数据集；②下载WIDER FACE转换后的标注；③开始训练模型

Training

1. Download WIDER FACE dataset, place the images under this directory:

```
$FaceBoxes_ROOT/data/WIDER_FACE/images
```

2. Convert WIDER FACE annotations to VOC format or download our converted annotations, place them under this directory:

```
$FaceBoxes_ROOT/data/WIDER_FACE/annotations
```

3. Train the model using WIDER FACE:

```
cd $FaceBoxes_ROOT/
python3 train.py
```

If you do not wish to train the model, you can download our pre-trained model and save it in `$FaceBoxes_ROOT/weights`.

高效率的FaceBoxes人脸检测算法：训练模型

■ 训练模型步骤：①下载WIDER FACE数据集；②下载WIDER FACE转换后的标注；③开始训练模型

Training

1. Download WIDER FACE dataset, place the images under this directory:

```
$FaceBoxes_ROOT/data/WIDER_FACE/images
```

2. Convert WIDER FACE annotations to VOC format or download our converted annotations, place them under this directory:

```
$FaceBoxes_ROOT/data/WIDER_FACE/annotations
```

3. Train the model using WIDER FACE:

```
cd $FaceBoxes_ROOT/
python3 train.py
```

If you do not wish to train the model, you can download our pre-trained model and save it in `$FaceBoxes_ROOT/weights`.

高效率的FaceBoxes人脸检测算法：评测模型

■ 评测模型步骤：①下载AFW/PASCAL Face/FDDB数据集；②获取检测结果；③对检测结果进行评测

Evaluation

1. Download the images of AFW, PASCAL Face and FDDB to:

```
$FaceBoxes_ROOT/data/AFW/images/
$FaceBoxes_ROOT/data/PASCAL/images/
$FaceBoxes_ROOT/data/FDDB/images/
```

2. Evaluate the trained model using:

```
# dataset choices = ['AFW', 'PASCAL', 'FDDB']
python3 test.py --dataset FDDB
# evaluate using cpu
python3 test.py --cpu
# visualize detection results
python3 test.py -s --vis_thres 0.3
```

3. Download eval_tool to evaluate the performance.

高效率的FaceBoxes人脸检测算法：训练代码解读

■ data/config.py：相关参数的设定

```
cfg = {
    'name': 'FaceBoxes',
    #'min_dim': 1024,
    #'feature_maps': [[32, 32], [16, 16], [8, 8]],
    # 'aspect_ratios': [[1], [1], [1]],
    'min_sizes': [[32, 64, 128], [256], [512]],
    'steps': [32, 64, 128],
    'variance': [0.1, 0.2],
    'clip': False,
    'loc_weight': 2.0,
    'gpu_train': True
```

锚框尺度

检测层下采样倍数

是否使用GPU

高效率的FaceBoxes人脸检测算法：训练代码解读

■ train.py中16~19行：训练参数的设定

```
16  parser = argparse.ArgumentParser(description='FaceBoxes Training')
17  parser.add_argument('--training_dataset', default='./data/WIDER_FACE', help='Training dataset directory')
18  parser.add_argument('-b', '--batch_size', default=32, type=int, help='Batch size for training')
19  parser.add_argument('--num_workers', default=8, type=int, help='Number of workers used in dataloading')
20  parser.add_argument('--ngpu', default=2, type=int, help='gpus')
21  parser.add_argument('--lr', '--learning-rate', default=1e-3, type=float, help='initial learning rate')
22  parser.add_argument('--momentum', default=0.9, type=float, help='momentum')
23  parser.add_argument('--resume_net', default=None, help='resume net for retraining')
24  parser.add_argument('--resume_epoch', default=0, type=int, help='resume iter for retraining')
25  parser.add_argument('-max', '--max_epoch', default=300, type=int, help='max epoch for retraining')
26  parser.add_argument('--weight_decay', default=5e-4, type=float, help='Weight decay for SGD')
27  parser.add_argument('--gamma', default=0.1, type=float, help='Gamma update for SGD')
28  parser.add_argument('--save_folder', default='./weights/', help='Location to save checkpoint models')
29  args = parser.parse_args()
30
31  if not os.path.exists(args.save_folder):
32      os.mkdir(args.save_folder)
33
34  img_dim = 1024 # only 1024 is supported
35  rgb_mean = (104, 117, 123) # bgr order
36  num_classes = 2
```

训练数据路径

训练批次大小

训练使用GPU数

初始学习率

训练迭代回合

模型保存路径

训练输入大小

归一化用的图像均值

类别数 (人脸 vs 非人脸)

高效率的FaceBoxes人脸检测算法：训练代码解读

■ train.py中16~19行：训练参数的设定

```
16  parser = argparse.ArgumentParser(description='FaceBoxes Training')
17  parser.add_argument('--training_dataset', default='./data/WIDER_FACE', help='Training dataset directory')
18  parser.add_argument('-b', '--batch_size', default=32, type=int, help='Batch size for training')
19  parser.add_argument('--num_workers', default=8, type=int, help='Number of workers used in dataloading')
20  parser.add_argument('--ngpu', default=2, type=int, help='gpus')
21  parser.add_argument('--lr', '--learning-rate', default=1e-3, type=float, help='initial learning rate')
22  parser.add_argument('--momentum', default=0.9, type=float, help='momentum')
23  parser.add_argument('--resume_net', default=None, help='resume net for retraining')
24  parser.add_argument('--resume_epoch', default=0, type=int, help='resume iter for retraining')
25  parser.add_argument('-max', '--max_epoch', default=300, type=int, help='max epoch for retraining')
26  parser.add_argument('--weight_decay', default=5e-4, type=float, help='Weight decay for SGD')
27  parser.add_argument('--gamma', default=0.1, type=float, help='Gamma update for SGD')
28  parser.add_argument('--save_folder', default='./weights/', help='Location to save checkpoint models')
29  args = parser.parse_args()
30
31  if not os.path.exists(args.save_folder):
32      os.mkdir(args.save_folder)
33
34  img_dim = 1024 # only 1024 is supported
35  rgb_mean = (104, 117, 123) # bgr order
36  num_classes = 2
```

训练数据路径

训练批次大小

训练使用GPU数

初始学习率

训练迭代回合

模型保存路径

上述默认设置为：使用2张GPU，每张GPU16张图，数据迭代300个回合

可使用其他设置：使用1张GPU，每张GPU32张图，数据迭代300个回合

训练输入大小

归一化用的图像均值

类别数 (人脸 vs 非人脸)

高效率的FaceBoxes人脸检测算法：训练代码解读

- train.py的49行对模型进行初始化：net = FaceBoxes('train', img_dim, num_classes)

- 调用models.faceboxes
- 初始化FaceBoxes这个类

高效率的FaceBoxes人脸检测算法：训练代码解读

■ train.py的49行对模型进行初始化：net = FaceBoxes('train', img_dim, num_classes)

- 调用models.faceboxes
- 初始化FaceBoxes这个类

```
63  class FaceBoxes(nn.Module):
64
65      def __init__(self, phase, size, num_classes):
66          super(FaceBoxes, self).__init__()
67          self.phase = phase
68          self.num_classes = num_classes
69          self.size = size
70
71          self.conv1 = CRelu(3, 24, kernel_size=7, stride=4, padding=3)
72          self.conv2 = CRelu(48, 64, kernel_size=5, stride=2, padding=2)
73
74          self.inception1 = Inception()
75          self.inception2 = Inception()
76          self.inception3 = Inception()
77
78          self.conv3_1 = BasicConv2d(128, 128, kernel_size=1, stride=1, padding=0)
79          self.conv3_2 = BasicConv2d(128, 256, kernel_size=3, stride=2, padding=1)
80
81          self.conv4_1 = BasicConv2d(256, 128, kernel_size=1, stride=1, padding=0)
82          self.conv4_2 = BasicConv2d(128, 256, kernel_size=3, stride=2, padding=1)
83
84          self.loc, self.conf = self.multibox(self.num_classes)
```

- 轻量级基础网络构建
- 快速消化卷积网络模块
- CReLU模块
- 多尺度卷积网络模块
- Inception模块

高效率的FaceBoxes人脸检测算法：训练代码解读

■ train.py的49行对模型进行初始化：net = FaceBoxes('train', img_dim, num_classes)

- 调用models.faceboxes
- 初始化FaceBoxes这个类

```
63  class FaceBoxes(nn.Module):
64
65    def __init__(self, phase, size, num_classes):
66      super(FaceBoxes, self).__init__()
67      self.phase = phase
68      self.num_classes = num_classes
69      self.size = size
70
71      self.conv1 = CRelu(3, 24, kernel_size=7, stride=4, padding=3)
72      self.conv2 = CRelu(48, 64, kernel_size=5, stride=2, padding=2)
73
74      self.inception1 = Inception()
75      self.inception2 = Inception()
76      self.inception3 = Inception()
77
78      self.conv3_1 = BasicConv2d(128, 128, kernel_size=1, stride=1, padding=0)
79      self.conv3_2 = BasicConv2d(128, 256, kernel_size=3, stride=2, padding=1)
80
81      self.conv4_1 = BasicConv2d(256, 128, kernel_size=1, stride=1, padding=0)
82      self.conv4_2 = BasicConv2d(128, 256, kernel_size=3, stride=2, padding=1)
83
84      self.loc, self.conf = self.multibox(self.num_classes)
```

- 轻量级基础网络构建
- 快速消化卷积网络模块
- CReLU模块
- 多尺度卷积网络模块
- Inception模块

- 总共三个检测层
- 每个检测层一个分类分支
- 每个检测层一个回归分支

高效率的FaceBoxes人脸检测算法：训练代码解读

- train.py的76行对损失函数层进行初始化：criterion = MultiBoxLoss(……)

```
class MultiBoxLoss(nn.Module):
    """SSD Weighted Loss Function
    Compute Targets:
        1) Produce Confidence Target Indices by matching  ground truth boxes
           with (default) 'priorboxes' that have jaccard index > threshold parameter
           (default threshold: 0.5).
        2) Produce localization target by 'encoding' variance into offsets of ground
           truth boxes and their matched  'priorboxes'.
        3) Hard negative mining to filter the excessive number of negative examples
           that comes with using a large number of default bounding boxes.
           (default negative:positive ratio 3:1)
    Objective Loss:
        L(x,c,l,g) = (Lconf(x, c) + αLloc(x,l,g)) / N
        Where, Lconf is the CrossEntropy Loss and Lloc is the SmoothL1 Loss
        weighted by α which is set to 1 by cross val.
        Args:
            c: class confidences,
            l: predicted boxes,
            g: ground truth boxes
            N: number of matched default boxes
        See: https://arxiv.org/pdf/1512.02325.pdf for more details.
    """

    def __init__(self, num_classes, overlap_thresh, prior_for_matching, bkg_label, neg_mining, neg_pos, neg_overlap, encode_target):
        super(MultiBoxLoss, self).__init__()
```

高效率的FaceBoxes人脸检测算法：训练代码解读

- train.py的78~81行生成锚框

```
78  priorbox = PriorBox(cfg, image_size=(img_dim, img_dim))
79  with torch.no_grad():
80      priors = priorbox.forward()
81      priors = priors.to(device)
```

高效率的FaceBoxes人脸检测算法：训练代码解读

- train.py的78~81行生成锚框

```
78    priorbox = PriorBox(cfg, image_size=(img_dim, img_dim))
79    with torch.no_grad():
80        priors = priorbox.forward()
81        priors = priors.to(device)
```

```
7   class PriorBox(object):
8       def __init__(self, cfg, image_size=None, phase='train'):
9           super(PriorBox, self).__init__()
10          #self.aspect_ratios = cfg['aspect_ratios']
11          self.min_sizes = cfg['min_sizes']
12          self.steps = cfg['steps']
13          self.clip = cfg['clip']
14          self.image_size = image_size
15          self.feature_maps = [[ceil(self.image_size[0]/step),
```

高效率的FaceBoxes人脸检测算法：训练代码解读

■ train.py的78~81行生成锚框

```
78    priorbox = PriorBox(cfg, image_size=(img_dim, img_dim))
79    with torch.no_grad():
80        priors = priorbox.forward()
81        priors = priors.to(device)
```

```
7   class PriorBox(object):
8       def __init__(self, cfg, image_size=None, phase='train'):
9           super(PriorBox, self).__init__()
10          #self.aspect_ratios = cfg['aspect_ratios']
11          self.min_sizes = cfg['min_sizes']
12          self.steps = cfg['steps']
13          self.clip = cfg['clip']
14          self.image_size = image_size
15          self.feature_maps = [[ceil(self.image_size[0]/step),
```

```
17      def forward(self):
18          anchors = []
19          for k, f in enumerate(self.feature_maps):
20              min_sizes = self.min_sizes[k]
21              for i, j in product(range(f[0]), range(f[1])):
22                  for min_size in min_sizes:
23                      s_kx = min_size / self.image_size[1]
24                      s_ky = min_size / self.image_size[0]
25                      if min_size == 32:
26                          dense_cx = [x*self.steps[k]/self.image_size[1] for x in [j+0, j+0.25, j+0.5, j+0.75]]
27                          dense_cy = [y*self.steps[k]/self.image_size[0] for y in [i+0, i+0.25, i+0.5, i+0.75]]
28                          for cy, cx in product(dense_cy, dense_cx):
29                              anchors += [cx, cy, s_kx, s_ky]
30                      elif min_size == 64:
31                          dense_cx = [x*self.steps[k]/self.image_size[1] for x in [j+0, j+0.5]]
32                          dense_cy = [y*self.steps[k]/self.image_size[0] for y in [i+0, i+0.5]]
33                          for cy, cx in product(dense_cy, dense_cx):
34                              anchors += [cx, cy, s_kx, s_ky]
35                      else:
36                          cx = (j + 0.5) * self.steps[k] / self.image_size[1]
37                          cy = (i + 0.5) * self.steps[k] / self.image_size[0]
38                          anchors += [cx, cy, s_kx, s_ky]
39          # back to torch land
40          output = torch.Tensor(anchors).view(-1, 4)
41          if self.clip:
42              output.clamp_(max=1, min=0)
43          return output
```

高效率的FaceBoxes人脸检测算法：训练代码解读

■ train.py的78~81行生成锚框

```
78    priorbox = PriorBox(cfg, image_size=(img_dim, img_dim))
79    with torch.no_grad():
80        priors = priorbox.forward()
81        priors = priors.to(device)
```

```
7  class PriorBox(object):
8      def __init__(self, cfg, image_size=None, phase='train'):
9          super(PriorBox, self).__init__()
10         #self.aspect_ratios = cfg['aspect_ratios']
11         self.min_sizes = cfg['min_sizes']
12         self.steps = cfg['steps']
13         self.clip = cfg['clip']
14         self.image_size = image_size
15         self.feature_maps = [[ceil(self.image_size[0]/step),
```

```
17     def forward(self):
18         anchors = []
19         for k, f in enumerate(self.feature_maps):
20             min_sizes = self.min_sizes[k]
21             for i, j in product(range(f[0]), range(f[1])):
22                 for min_size in min_sizes:
23                     s_kx = min_size / self.image_size[1]
24                     s_ky = min_size / self.image_size[0]
25                     if min_size == 32:
26                         dense_cx = [x*self.steps[k]/self.image_size[1] for x in [j+0, j+0.25, j+0.5, j+0.75]]
27                         dense_cy = [y*self.steps[k]/self.image_size[0] for y in [i+0, i+0.25, i+0.5, i+0.75]]
28                         for cy, cx in product(dense_cy, dense_cx):
29                             anchors += [cx, cy, s_kx, s_ky]
30                     elif min_size == 64:
31                         dense_cx = [x*self.steps[k]/self.image_size[1] for x in [j+0, j+0.5]]
32                         dense_cy = [y*self.steps[k]/self.image_size[0] for y in [i+0, i+0.5]]
33                         for cy, cx in product(dense_cy, dense_cx):
34                             anchors += [cx, cy, s_kx, s_ky]
35                     else:
36                         cx = (j + 0.5) * self.steps[k] / self.image_size[1]
37                         cy = (i + 0.5) * self.steps[k] / self.image_size[0]
38                         anchors += [cx, cy, s_kx, s_ky]
39         # back to torch land
40         output = torch.Tensor(anchors).view(-1, 4)
41         if self.clip:
42             output.clamp_(max=1, min=0)
43         return output
```

32x32锚框密集化4倍

64x64锚框密集化2倍

高效率的FaceBoxes人脸检测算法：训练代码解读

- train.py的89行对数据集进行初始化：dataset = VOCDetection(……)
- 调用data/wider_voc.py文件里的VOCDetection类

高效率的FaceBoxes人脸检测算法：训练代码解读

- train.py的89行对数据集进行初始化：dataset = VOCDetection(……)
- 调用data/wider_voc.py文件里的VOCDetection类
- 数据增广代码在data/data_augment.py中，关键代码197~204行如下所示

```
197     image_t, boxes_t, labels_t, pad_image_flag = _crop(image, boxes, labels, self.img_dim)
198     image_t = _distort(image_t)
199     image_t = _pad_to_square(image_t,self.rgb_means, pad_image_flag)
200     image_t, boxes_t = _mirror(image_t, boxes_t)
201     height, width, _ = image_t.shape
202     image_t = _resize_subtract_mean(image_t, self.img_dim, self.rgb_means)
```

随机裁剪

随机颜色抖动

随机扩充

随机水平翻转

图像归一化操作

高效率的FaceBoxes人脸检测算法：训练代码解读

■ train.py的121行，输入一次图像得到分类和回归的预测：out = net(images)

```
112    def forward(self, x):
113
114      detection_sources = list()
115      loc = list()
116      conf = list()
117
118      x = self.conv1(x)
119      x = F.max_pool2d(x, kernel_size=3, stride=2, padding=1)
120      x = self.conv2(x)
121      x = F.max_pool2d(x, kernel_size=3, stride=2, padding=1)
122      x = self.inception1(x)
123      x = self.inception2(x)
124      x = self.inception3(x)
125      detection_sources.append(x)
126
127      x = self.conv3_1(x)
128      x = self.conv3_2(x)
129      detection_sources.append(x)
130
131      x = self.conv4_1(x)
132      x = self.conv4_2(x)
133      detection_sources.append(x)
134
135      for (x, l, c) in zip(detection_sources, self.loc, self.conf):
136          loc.append(l(x).permute(0, 2, 3, 1).contiguous())
137          conf.append(c(x).permute(0, 2, 3, 1).contiguous())
```

快速消化卷积网络模块

多尺度卷积网络模块

检测头子网络模块

高效率的FaceBoxes人脸检测算法：训练代码解读

- train.py的125行根据预测值和真实值计算分类和回归的损失值：loss_l, loss_c = criterion(out, priors, targets)

```
62      # match priors (default boxes) and ground truth boxes
63      loc_t = torch.Tensor(num, num_priors, 4)
64      conf_t = torch.LongTensor(num, num_priors)
65      for idx in range(num):
66          truths = targets[idx][:, :-1].data
67          labels = targets[idx][:, -1].data
68          defaults = priors.data
69          match(self.threshold, truths, defaults, self.variance, labels, loc_t, conf_t, idx)
```

锚框匹配获得真实值

高效率的FaceBoxes人脸检测算法：训练代码解读

- train.py的125行根据预测值和真实值计算分类和回归的损失值：loss_l, loss_c = criterion(out, priors, targets)

```
62      # match priors (default boxes) and ground truth boxes
63      loc_t = torch.Tensor(num, num_priors, 4)
64      conf_t = torch.LongTensor(num, num_priors)
65      for idx in range(num):
66          truths = targets[idx][:, :-1].data
67          labels = targets[idx][:, -1].data
68          defaults = priors.data
69          match(self.threshold, truths, defaults, self.variance, labels, loc_t, conf_t, idx)
```

锚框匹配获得真实值

```
76      # Localization Loss (Smooth L1)
77      # Shape: [batch,num_priors,4]
78      pos_idx = pos.unsqueeze(pos.dim()).expand_as(loc_data)
79      loc_p = loc_data[pos_idx].view(-1, 4)
80      loc_t = loc_t[pos_idx].view(-1, 4)
81      loss_l = F.smooth_l1_loss(loc_p, loc_t, reduction='sum')
```

计算回归损失值

高效率的FaceBoxes人脸检测算法：训练代码解读

■ train.py的125行根据预测值和真实值计算分类和回归的损失值：loss_l, loss_c = criterion(out, priors, targets)

```
62      # match priors (default boxes) and ground truth boxes
63      loc_t = torch.Tensor(num, num_priors, 4)
64      conf_t = torch.LongTensor(num, num_priors)
65      for idx in range(num):
66          truths = targets[idx][:, :-1].data
67          labels = targets[idx][:, -1].data
68          defaults = priors.data
69          match(self.threshold, truths, defaults, self.variance, labels, loc_t, conf_t, idx)
```

锚框匹配获得真实值

```
76      # Localization Loss (Smooth L1)
77      # Shape: [batch,num_priors,4]
78      pos_idx = pos.unsqueeze(pos.dim()).expand_as(loc_data)
79      loc_p = loc_data[pos_idx].view(-1, 4)
80      loc_t = loc_t[pos_idx].view(-1, 4)
81      loss_l = F.smooth_l1_loss(loc_p, loc_t, reduction='sum')
```

计算回归损失值

```
83      # Compute max conf across batch for hard negative mining
84      batch_conf = conf_data.view(-1, self.num_classes)
85      loss_c = log_sum_exp(batch_conf) - batch_conf.gather(1, conf_t.view(-1, 1))
86
87      # Hard Negative Mining
88      loss_c[pos.view(-1, 1)] = 0 # filter out pos boxes for now
89      loss_c = loss_c.view(num, -1)
90      _, loss_idx = loss_c.sort(1, descending=True)
91      _, idx_rank = loss_idx.sort(1)
92      num_pos = pos.long().sum(1, keepdim=True)
93      num_neg = torch.clamp(self.negpos_ratio*num_pos, max=pos.size(1)-1)
94      neg = idx_rank < num_neg.expand_as(idx_rank)
95
96      # Confidence Loss Including Positive and Negative Examples
97      pos_idx = pos.unsqueeze(2).expand_as(conf_data)
98      neg_idx = neg.unsqueeze(2).expand_as(conf_data)
99      conf_p = conf_data[(pos_idx+neg_idx).gt(0)].view(-1,self.num_classes)
00      targets_weighted = conf_t[(pos+neg).gt(0)]
01      loss_c = F.cross_entropy(conf_p, targets_weighted, reduction='sum')
```

难样本挖掘

计算分类损失值

课程作业

- 调试和训练FaceBoxes算法（https://github.com/zisianw/FaceBoxes.PyTorch）

1. 按照指示配好FaceBoxes所需要的的环境
2. 按照指示下载所需要的数据集和标注
3. 利用PyCharm单步调试一遍FaceBoxes的代码
4. 使用1张或2张GPU训练FaceBoxes人脸检测算法（如果确实没有GPU或GPU不够，可以只调试不训练）
5. 利用训练好的模型或官方提供的模型，在AFW、PASCAL Face、FDDB上进行测试，看结果是否正确

感谢聆听

Thanks for Listening