

SHANGHAIUNIVERSITY

毕业设计（论文）

**UNDERGRADUATEPROJECT(THESIS)**

**题目:** **Android手机上的人脸实时检测APP的开发**

|  |  |
| --- | --- |
| **学院** | **计算机工程与科学学院** |
| **专业** | **计算机科学与技术** |
| **学号** | **14122556** |
| **学生姓名** | **雷佳霖** |
| **指导教师** | **支小莉** |
| **起讫日期** | **2018.01.29 – 2018.06.08** |

目录

[摘要 III](#_Toc515552325)

[ABSTRACT IV](#_Toc515552326)

[第1章 绪论 1](#_Toc515552327)

[§1.1 基于YOLO v2的人脸模型构建的背景及意义 1](#_Toc515552328)

[§1.2 目标检测研究现状及存在的问题 2](#_Toc515552329)

[§1.2.1 研究现状 2](#_Toc515552330)

[§1.2.2 深度神经网络目标检测模型存在的问题 3](#_Toc515552331)

[§1.2.3 目标检测模型在移动端构建的难点 3](#_Toc515552332)

[§1.3 本文研究内容及目标 3](#_Toc515552333)

[§1.3.1 研究内容 3](#_Toc515552334)

[§1.3.2 研究目标 4](#_Toc515552335)

[§1.4 本文组织结构 4](#_Toc515552336)

[第2章 基于RPN的目标检测方法 5](#_Toc515552337)

[§2.1 目标检测概念综述 5](#_Toc515552338)

[§2.1.1 目标检测常用术语 7](#_Toc515552339)

[§2.1.2 目标检测常用数据集 8](#_Toc515552340)

[§2.2 常见的深度神经网络目标检测领域模型 9](#_Toc515552341)

[§2.3 目标检测典型深度神经网络模型的分析与比较 10](#_Toc515552342)

[§2.4 本章小结 11](#_Toc515552343)

[第3章 基于YOLO的目标检测方法 12](#_Toc515552344)

[§3.1 YOLO模型概述 12](#_Toc515552345)

[§3.2 YOLO系列模型结构发展与变化 12](#_Toc515552346)

[§3.2.1 YOLO v1模型结构概述 12](#_Toc515552347)

[§3.2.2 YOLO v2模型结构概述 14](#_Toc515552348)

[§3.2.3 YOLO v3模型结构概述 22](#_Toc515552349)

[§3.3 本章小结 24](#_Toc515552350)

[第4章 Android人脸实时检测应用设计与实现 25](#_Toc515552351)

[§4.1 系统整体设计 25](#_Toc515552352)

[§4.1.1 开发环境简介 25](#_Toc515552353)

[§4.1.2 应用整体架构 26](#_Toc515552354)

[§4.2 基于YOLO v2的人脸检测模型设计 26](#_Toc515552355)

[§4.3 功能模块设计 29](#_Toc515552356)

[§4.3.1 Android端模型优化模块 30](#_Toc515552357)

[§4.3.2 Android应用构建与显示 30](#_Toc515552358)

[§4.4 本章小结 30](#_Toc515552359)

[第5章 总结与展望 31](#_Toc515552360)

[§5.1 本文总结 31](#_Toc515552361)

[§5.1.1 本文的主要工作 31](#_Toc515552362)

[§5.1.2 本文的主要创新点 31](#_Toc515552363)

[§5.2 展望 31](#_Toc515552364)

[致谢 33](#_Toc515552365)

[参考文献 34](#_Toc515552366)

[附录：部分源程序清单 36](#_Toc515552367)

Android手机上的人脸实时检测APP的开发

# 摘要

人脸检测一直是目标检测中最重要的几个应用场景之一。近年来，随着深度神经网络的爆发式发展，目标检测领域获得了新的突破性成果。在众多目标检测模型中，YOLO v2以其实时性、易用性、通用性和健壮性脱颖而出，其在VOC2007上测得的mAP为78.6%，实时检测速度为40fps。YOLO v2在拥有实时检测速度的同时保持着出色的定位准确度和检测精确度。综合模型体积、速度、定位和分类正确率、计算消耗及内存占用等方面因素考虑，本文所选的基于YOLO v2的人脸检测模型很适合运行于计算能力受到限制，速度又要求高的移动端。

本文主要构建了基于YOLO v2模型的Android端人脸检测应用。重新训练并优化微调后的模型在手机上进行人脸检测时间控制在1s左右，这为人脸摄像、摄像头监控等提供了一种技术方案。

关键词：YOLO v2，目标检测，区域建议网络，矩阵乘法加速

Development of Android APP for real-time face detection Based on YOLO v2

# ABSTRACT

Face detection has always been one of the most important application scenarios in Object Detection. In recent years, with the explosive development of Deep Neural Networks, new breakthroughs have been achieved in the field of Object Detection. In many Object Detection models, YOLO v2 stands out with its traits including real-time, accessibility, versatility and robustness. The mean Average Precision(mAP) of YOLO v2 measured on VOC2007 is 78.6%, and the speed of it is 40fps. YOLO v2 maintains real-time detection speed as well as excellent detection location accuracy and detection classification accuracy. Considering the aspects of model volume, speed, location accuracy and classification accuracy, calculation consumption and memory occupation, the YOLO v2-based face detection model discussed in this paper is suitable for running on mobile terminals with limited computing capacity and high demand on speed.

This paper mainly constructs Android application for face-detection based on YOLO v2 model. After retraining and optimizing, the fine-tuned face-detection model costs at around one second on the mobile phone, which provides a solution for face snapshot, camera face-monitoring and so on.

**Keywords:** YOLO v2, Object Detection, Region Proposal Networks, Matrix multiplication acceleration

# 绪论

本章主要描述了基于YOLO v2模型的人脸检测Android端应用的背景与研究意义，分析了目标检测领域相关课题国内外的研究现状，进而提出了本文所要研究的内容及目标。

## 基于YOLO v2的人脸模型构建的背景及意义

计算机视觉领域里充满了各种各样有趣的问题，**目标检测**便是其中之一。目标检测在生活中有许多应用：比如对行人，汽车，花朵甚至是细胞进行**计数；寻找相似的商品**；对无人机或者卫星图像进行分析；**探测**某一位置周围的环境设施（如图1‑1）；对危险的地方进行**自动侦察**，对空中军事单位进行检测；还有最著名的**人脸检测**等。大数据时代，使用计算机视觉将原始信息转化为结构化数据的能力越来越强，而数据越多模型表现就越好。



图 1‑1 TensorFlight为保险公司使用目标检测算法自动更新用户不动产的最新数据信息

人脸检测属于目标检测的其中的一个具体的分支。要在内存及CPU能力都受限制的手机上运行人脸检测模型，需要模型在体积、速度、正确率、计算消耗及内存占用方面都达到比较理想的水平。传统的人脸检测方法是基于哈尔特征进行检测的，但其对人脸的标准程度很敏感效果较差。进入深度神经网络时代后，目标检测网络的性能得到了巨大的提升。其中YOLO v2作为目标检测领域中被广泛应用的模型之一，一直享有速度很快完全可以满足实时检测的盛名。YOLO v2实际上可以针对很多种目标进行检测，易于上手且对计算能力消耗相对较少，其中的YOLO v2在达到40fps以上的速度的同时保持很好的准确度和检测精确度，因此基于YOLO v2的人脸检测模型很适合运行于计算能力受到限制，速度又要求高的移动端。

## 目标检测研究现状及存在的问题

人脸检测作为常见的基础应用已经活跃在机器学习领域很多年，并拥有成熟完备的解决方案。传统的人脸检测方法对环境干扰敏感，在深度神经网络流行起来后，基于CNN的目标检测模型也开始逐步成熟起来，下面具体阐述相关研究现状以及存在的问题。

### 目标检测模型研究现状

传统的人脸检测有两种方法：一种是利用人工提取特征来进行训练分类，利用**哈尔特征(Haar-like features)**来提取图像特征对相机实现人脸检测，但这种方法对环境干扰很敏感，当人脸被遮住、侧脸、光照很弱或者曝光度很高的情况下检测效果差。另一种是方向梯度直方图(HOG)来提取图像特征再将特征传给SVM进行分类。在深度神经网络(DNN)席卷整个机器学习领域的时候，目标检测领域的模型也迎来了新一轮的爆发提升。

2013年第一个基于卷积神经网络（CNN）多尺度滑动窗口的目标检测模型OverFeat诞生。此后加利福尼亚大学伯克利分校提出了**R-CNN(Regions with CNN features)**，使目标检测领域取得了里程碑式突破。R-CNN的大体思想是先使用**启发式搜索**方法选取最有可能的区域，然后使用卷积层从每个预处理过的区域里提取特征，最后使用SVM对每个区域里的物体进行分类回归。

但R-CNN分开执行提取特征、分类、回归、微调这样的任务很消耗时间。一年后，改进版的Fast R-CNN被提出，它去掉了SVM，借鉴了SPP Net的思想设计出ROI池化层(ROI pooling)使得不同尺寸的特征图能被统一送入后面固定输入大小的全连接层，这样Fast R-CNN可以同时进行候选区域分类和候选框回归两个任务，且这两个任务在一个CNN网络里共享特征，即把原本SVM该做的任务移到深度神经网络内部。

虽然Fast R-CNN准确率提升的同时速度也大大提升，但启发式搜索找出所有候选区域这个过程仍然很慢。这个问题在Faster R-CNN被提出后得到解决，Faster R-CNN引入了Region Proposal Network(RPN)，将启发式搜索寻找候选区域的任务放在ROI池化层之前，归入到网络内部。据分析，RPN的出现使得Faster R-CNN相比之前的模型速度至少提高了250倍以上。

2015年，将RPN的候选区域提取任务完全移到CNN内部形成一个整体的思想开始发酵，2016年，YOLO v1与SSD初露锋芒出现在人们的视线里。自此以后，DSSD、SSD、MultiBox、R-FCN、MTCNN、FPN FRCN、YOLO v2、YOLO v3等等目标检测模型悉数登场，它们在性能和速度上都拥有着不俗的表现（详细比较见表2‑3）。在2018年的今天，深度神经网络的快速发展仍推动着整个计算机视觉领域跳跃前行。

### 深度神经网络目标检测模型存在的问题

基于区域选择的R-CNN系列方法虽然打开了CNN与候选区域共同合作的大门，但仍然存在很多问题：

1. 计算消耗大。由于专门设立了一个RPN网络进行对候选区域的提取所以有比较高的准确率，但其速度和计算消耗也同时因此变慢变大。比如Faster R-CNN光损失函数就有四个，网络本身与RPN各包含两个：RPN的分类与回归损失函数衡量候选框选取效果；网络本身的分类与回归损失函数衡量物体分类和定位效果。
2. 速度慢，5fps左右的帧率不适合实时检测，但是实时目标检测可以从CNN与候选区域的合作上找到突破口。

所以目标检测模型如何在提高速度与准确率的同时保持两者的一个平衡，成为当下深度神经网络目标检测模型都需要考虑并优化的主要问题。

### 目标检测模型在移动端构建的难点

虽然基于深度神经网络的人脸检测的模型已相当成熟，但针对移动端这样CPU与GPU计算能力都有限的设备，仍然存在以下一些技术难点：

1. Android的arm系CPU架构不同于PC，如何进行针对性优化。
2. Android端的数学计算加速库很少。PC端的数学计算加速库已经非常成熟，并且有众多开源社区及商业公司进行维护。
3. Android端直到Android 8.0才内置支持TensorFlow深度学习API，然而Android通常需要兼容4.4以上的版本。
4. PC端模型如何移植到Android端。

## 本文研究内容及目标

本文针对深度学习目标检测领域模型遇到的问题，选择使用可以实时监测且体积小和计算能力消耗少的YOLO v2-tiny模型作为基础模型，优化并修改模型结构，对拍照及其他方式输入的图片进行人脸检测，并将其配置到Android端运行，同时试图对摄像头进行实时检测，为人脸摄像、摄像头监控等提供了一种技术方案。

### 研究内容

本文研究基于YOLO v2的人脸检测模型构建方法，具体研究内容有一下几个方面。

（1）人脸检测模型构建与优化；

（2）图片通过摄像头的采集；

（4）人脸检测模型移植到Android端；

（5）Android端矩阵乘法计算能力加速。

### 研究目标

针对本文的研究内容，制定了以下几项指标：

1. 重新训练YOLO v2模型并优化其结构，只对图片人脸进行检测；
2. 将训练好的模型移植到Android端部署，实现手机图片中人脸检测；
3. 要求给出最终**检测精度(AP，average precision)**和**检测速度(FPS，frames per second)**；
4. 尝试进行Android端的人脸实时检测。

## 本文组织结构

整篇论文分为五章。

第一章介绍了研究背景及研究意义，阐述了YOLO系列检测方法在目标检测领域的现状以及存在的问题和难点，章节末尾引出了本文的研究内容以及研究目标。

第二章主要介绍了目标检测领域主要使用的几个著名模型，如SSD系列、Faster R-CNN系列等。同时解释了目标检测领域的基本术语，汇总考察了适用于该领域的数据集。

第三章介绍了YOLO系列YOLO v1、v2等网络模型的基本结构，并分析对比了该领域现存的几个著名模型与YOLO v2的差异。

第四章主要描述了基于YOLO v2的人脸检测模型应用在Android端的整体设计方式与构建方式，以人脸检测为例进行，实现了深度学习模型在Android端的落地，并展示了模型训练效果和矩阵乘法优化等各个模块的内容。

第五章对全文进行了总结，归纳了本文的主要工作与创新点，并指出了需要进一步研究的问题。

# 基于RPN的目标检测方法

主要介绍了目标检测的基本概念以及该领域常用的基本方法，简单介绍了YOLO系列、SSD系列和基于RPN的R-CNN系列网络模型，并且解释了目标检测领域的基本术语和主要的数据集。

## 目标检测概念综述

本节介绍了目标检测的基本概念以及发展现状。计**算机视觉(Computer Vision)**主要指识别和理解图像的内容，包括分类、定位、目标检测、语义分割和实例分割，每一步都是在前一步的基础上进行的，如图 2‑1所示。**图像分类**是最常见的计算机视觉问题，对网络输入查看图像并对其中的对象进行分类。图像分类具有广泛的应用范围，从社交网络上的人脸检测到医学中的癌症检测，这些问题通常使用卷积神经网络建模进行分类就可以解决；**定位**进一步判断图像中的目标具体位置，只需要测出固定数量的预测框，即标记对象的类别，给出物体的确切位置围绕目标的位置绘制边框；**目标检测**需要对图像中的多个物体给出其的分类和定位。比如在自动驾驶汽车中，不仅需要检测汽车，还需要检测车架中的行人，摩托车，树木和其他物体。图像分类和定位是目标检测的基础；**语义分割**需要判断哪些像素属于哪个目标，将相同整体的所有像素预测同一类别，注意这是并不区分同种类别的不同物体的，比如第一个人，第二个人是不加区分地被分在一起的；**实例分割**需要区分出相同类别的所有实例，所有实例都需要被区分出来，即使是同一种类别的物体；除此之外，还有基于视频时序的**目标跟踪**等等，都是计算机视觉的问题。

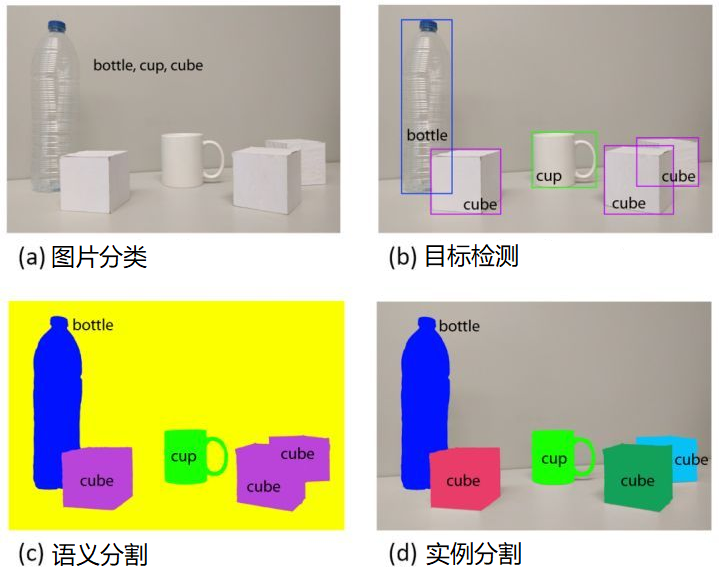


图 2‑2 计算机视觉主要任务

图像分类输入一张的是图片，输出的是物体的类别，用准确率来评判；定位还需要输出检测到的目标位置；目标检测模型的主要性能指标是**检测准确度**和**速度**，其中准确度包含定位准确度和分类准确度。

目前主流的目标检测算法主要是基于深度学习模型，其可以分成两大类，**一步算法(one-stage)**和**两步算法(two-stage)**。

1. 两步算法，顾名思义其将检测问题划分为两个阶段：定位和分类，首先产生候选区域，对候选区域分类（必要情况下位置还需要精修）。这类算法的典型代表是基于RPN(Region Proposal Networks)的R-CNN系算法，如R-CNN，Fast R-CNN，Faster R-CNN等，由于注重的是区域选择，所以物体定位精确度高，但是速度比一步算法慢。
2. 一步算法，其不需要额外的区域选择阶段，同时回归网络产生物体的类别概率和位置坐标值。典型代表模型比如YOLO和SSD。由于没有额外的区域选择模块，一般速度很快，可以达到实时检测，但精确度不如两步算法。

深度神经网络模型中卷积层的基本思想是根据图片的空间位置信息的位移、尺度、形变不变性，即**局部不变性原则**进行的，通过浅层提取出简单特征，然后在高层将这些特征抽象出可能的对象。特征提取的目标是将可变大小的图像缩小为一组的视觉特征，无论是基于传统的计算机视觉特征提取，例如基于滤波器的逼近方法，直方图方法等还是深度学习方法的特征提取，它们都从输入图像中提取特征并使用这些特征来确定图像的类别。

在目标检测框架中，人们通常使用预训练的图像分类模型（如VGG-16，ResNet等）来提取视觉特征，因为这些模型有很强的泛化性，将模型应用在COCO数据集上训练能够提取基本的通用特征。目前目标检测领域有很多用于物体检测的预训练模型，不同的分类网络具有不同的优势和劣势，比如Inceptionv3网络对大物体小物体的检测都很好，ResNet网络的总体精度非常高，而MobileNet是一个精简的网络可以最大限度地减少所需的计算资源。

### 目标检测常用术语

目标检测模型评估条件分两个方面，一个是分类的准确度，一个是定位的准确度。单纯比较绝对坐标值很复杂，因此我们引入了**IOU**(**Intersection over Union**)交并比来表示预测框和真实框的交叉部分比上预测框与真实框的并集部分（如图 2 2），值越大说明预测框与真实框的相交部分越多，也离真实框越近。当IOU超过某个阈值时，判定检测正确，否则认为检测错误。

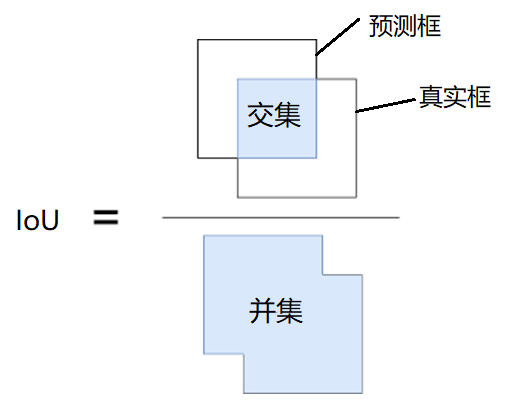


图 2‑3 IOU(Intersection over Union，交并比)计算图例

由于是标注的数据，所以我们可以计算得某种类别的物体的测试精确度，假设有一张图片，对于给定的类，可以计算该图中类别*C*的精确度如下：

其次，多种类目标检测中，有多种物体种类，若要计算对于所有图片来说某类别的精确度，则需要验证集中的每个图像计算，然后对其求平均，得到了单个类别的精确度：

最后为了用单个数字来衡量一个模型的表现，用所有类别的精确度求取平均值有：

这个值就是目标检测中衡量识别精度的指标，称为**mAP(mean average precision)**。多类别物体检测中，**召回率(recall)**指的是样本中真正正确的有多少被预测正确了，**准确率(precision)**预测为正确的样本中有多少是真正正确的。每一个类别都可以根据这两个值绘制一条曲线，*AP*(*Average Precision*)就是该曲线下的面积，*mAP*是多个类别*AP*的平均值。

除了定位精确度*mAP*，目标检测算法的另外一个重要性能指标是速度，只有速度快，才能实现实时检测，评估速度的常用指标是**FPS(Frame Per Second，帧率)**，即每秒内可以处理的图片数量。对比FPS需要在同一硬件上进行，因为CPU或者CPU计算能力对图片处理能力影响极大。另外也可以使用平均处理一张图片所需时间(毫秒，ms)来评估检测速度。

### 目标检测常用数据集

表2‑1列出了常见的大型图片数据集，包含各种各样的物体和类别，众多著名的算法及网络都是在这上面训练并测试的，本文使用的YOLO v2也是基于ImageNet和COCO训练及测试的。表2‑2列出了常见的人脸数据集，本文也是借助了其中的FDDB与CelebA人脸数据集进行训练的。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据集 | 图片数 | 类别数 | 特征 | 最后更新 |
| ImageNet | 450K | 200 | 最大图像识别数据集之一 | 2015 |
| COCO | 120K | 80 | 图像识别和分割数据集 | 2014 |
| Pascal VOC | 12K | 20 | 物体检测和分类数据集 | 2012 |
| Oxford-IIIT Pet | 7K | 37 | 宠物图像数据分割数据集 | 2012 |
| KITTI Vision | 7K | 3 | 自动驾驶场景数据集 | 2014 |

表 2‑1 常见的大型图片数据集，包含各种物体种类

|  |  |  |  |
| --- | --- | --- | --- |
| 人脸数据集 | 图片数 | 特征 | 最后更新 |
| FDDB | 3K | 椭圆形标注人脸 | 2010 |
| AFLW | 25K | 图像识别和分割数据集 | 2017 |
| LFW | 13K | 物体检测和分类数据集 | 2012 |
| WIDER FACE | 3K | 宠物图像数据分割数据集 | 2017 |
| CelebA | 200K | 最大的人脸数据集之一 | 2015 |

表 2‑2 常见的人脸数据集

## 常见的深度神经网络目标检测领域模型

上一节介绍了目标检测领域基本概念和会应用到的术语，对目标检测有了一个粗浅的了解。目标检测模型包括本文应用到的YOLO系列，SSD系列和R-CNN，Fast R-CNN，Mask R-CNN，SSD，Multibox等，这些模型的速度和定位精确度比较如下表所示：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **模型名称** | **基础结构** | **测试分辨率** | **不同数据集上的准确率(mAP)** | | | | **实时速度** |
| **VOC2007** | **VOC2012** | **COCO** | **ILSVRC 2013** |
| R-CNN | AlexNet | - | 58.5% | 53.3% |  | 31.4% |  |
| R-CNN | VGG16 | - | 66.0% |  |  |  |  |
| SPP-net | ZF-5 | - | 54.2% |  |  | 31.84% |  |
| DeepID-Net |  | - | 64.1% |  |  | 50.3% |  |
| NoC |  | - | 73.3% |  |  | 68.8% |  |
| Fast-RCNN | VGG16 | - | 70.0% | 68.4% |  |  |  |
| MR-CNN | 78.2% | - | 73.9% |  |  |  |  |
| Faster-RCNN | VGG16 | - | 78.8% | 75.9% |  |  | 5fps |
| Faster-RCNN | ResNet101 | - | 85.6% | 83.8% |  |  | 5fps |
| YOLO |  | - | 63.4% | 57.9% |  |  | 45 fps |
| Tiny YOLO |  | - | 66.4% |  | 23.7% |  | 244 fps |
| YOLOv2 |  | 448x448 | 78.6% | 73.4% | 48.1% |  | 40 fps |
| YOLOv3 |  | 320x320 |  |  | 51.5% |  | 45 fps |
| YOLOv3 |  | 416x416 |  |  | 55.3% |  | 35 fps |
| YOLOv3 |  | 608x608 |  |  | 57.9% |  | 20 fps |
| Tiny YOLOv3 |  | - |  |  | 33.1% |  | 220 fps |
| SSD | VGG16 | 300x300 | 77.2% | 75.8% | 41.2% |  | 46 fps |
| SSD | VGG16 | 512x512 | 79.8% | 78.5% | 46.5% |  | 19 fps |
| SSD | ResNet101 | 300x300 |  |  | 45.4% |  | 16 fps |
| SSD | ResNet101 | 512x512 |  |  | 50.4% |  | 8 fps |
| DSSD | ResNet101 | 321x321 |  |  | 46.1% |  | 12 fps |
| DSSD | ResNet101 | 513x513 |  |  | 53.3% |  | 6 fps |
| R-FCN |  | - |  |  | 51.9% |  | 12fps |
| FPN FRCN |  | - |  |  | 59.1% |  | 6fps |
| ION |  | - | 79.2% | 76.4% |  |  |  |
| CRAFT |  | - | 75.7% | 71.3% |  | 48.5% |  |
| OHEM |  | - | 78.9% | 76.3% |  |  |  |

表 2‑3 常见目标检测深度神经网络模型的速度及准确度等一系列对比表

## 目标检测典型深度神经网络模型的分析与比较

YOLO系列模型与目标检测领域其他模型的速度与精确度比较详细见表2‑5。

前文有提到在主流目标检测模型中 YOLO和SSD都属于一步算法模型。而Faster R-CNN单独将区域选择设立为网络所以术语两步算法模型，绪论中有详细介绍R-CNN，Fast R-CNN, Faster R-CNN系列深度神经网络模型，这一系列模型需先使用**启发式选择方法(selective search)**或者**RPN(Region Proposal Networks)**产生**候选区域(Region Proposal)**，然后再在选中的候选区域上做分类与回归，提取视觉特征然后进行评估，确定候选区域中是否存哪些目标物体。

事实上不论是传统机器学习方法还是深度神经网络方法，目标检测中候选区域选择的主要中心思想都是围绕**滑动窗口**展开的**，滑动窗口**算法思路非常简单，它将检测问题转化为了图像分类问题。其基本原理就是采用不同大小和比例的窗口在整张图片上以一定的步长进行滑动，然后对这些窗口对应的区域做图像分类，如DPM(Deformable Part Model)就是采用滑动窗口的模型。但是滑动窗口的缺点就是需要设置不同大小和比例和步长的窗口去滑动，但是这样会产生很多的子区域送入分类器去判断种类，计算量很大，并且无法保证每一个子区域都是有效的（刚好包含物体而没有偏移，只包含物体的一部分，或者除开真实物体的长与宽外还剩下巨大的边距），是一种非常蛮力的方法。弥补这一缺点的解决方法之一就是减少要分类的子区域，R-CNN便是这样通过采用了**启发式搜索方法**过滤掉很多子区域，找到最有可能包含目标的子区域，显著提升效率。

**启发式选择方法(selective search)**是一种基于聚类的方法，它尝试对像素进行分组并根据生成的聚类生成提案。其他方法使用从图像中提取的更复杂的视觉特征来生成区域或采用暴力方法来生成区域，类似于应用于图像的滑动窗口，用不同比例的大小和长宽比在图片上滑动，需要注意的是这些区域是按像素和步长滑动生成的，而没有考虑图像特征。

**SSD**(**Single Shot MultiBox Detector，单发多盒检测器**)模型去掉了RPN，结合区域选择和特征提取融合到一个独立的深度神经网络中。为了达成合二为一的目的，YOLO和SSD都借鉴了RPN中的锚箱(Anchor Boxes)的思想。SSD使用不同宽高比和比例的一组默认框，称其为**默认框**(Default Box)，并将默认框应用于多个中间特征卷积层。默认框的大小在每一层卷积层都不同，由于这些特征映射是通过将图像传递到图像分类网络(类CNN网络)来计算的，故可以在每个特征卷积层中进行边界框的特征提取，并为每个默认边界框中的每个目标物体类别生成分数。卷积网络中的不同特征映射与不同的接受域相对应，用于处理不同尺度大小的物体。由于所有计算都封装在一个深度神经网络中，所以SSD可以达到相当不错的速度（SSD300的速度为46FPS）。

**YOLO(You Only Look Once，只看一眼)**系列算法，其仅仅使用一个CNN网络直接预测不同目标的类别与位置。YOLO算法直接将原始图片分割成互不重合的小方块，然后通过卷积最后生产这样大小的特征图，基于上面的分析，可以认为特征图的每个元素也是对应原始图片的一个小方块，然后用每个元素来可以预测那些中心点在该小方格内的目标，这就是YOLO算法的朴素思想。关于YOLO系列模型的具体细节将会在第三章进行详细介绍。

## 本章小结

总结来看，大部分模型都是借鉴其它论文的一些技巧如Faster R-CNN的锚箱，YOLOv2采用锚箱和卷积做预测，SSD也是借鉴了Faster R-CNN的RPN。在两步方法中，区域选择网络起到的作用是给出候选区域，然后再采用R-CNN网络提升检测的准确度同时输出类别。这些模型都选择在卷积层进行区域选择预测，这是因为经过卷积操作后不但图片大小减少，其图片的空间位置对应关系也得到了相当的保留。区域选择方法生成需要考虑区域数量与计算复杂度的平衡，生成的候选区域越多，就越有可能找到该目标物体，但如果详尽地生成所有可能的候选区域会使网络计算变得非常复杂并且速度很慢，因此需要减少候选区域数量。从某种意义上来说，YOLO和SSD使用其他方法代替RPN将其融合到整个网络中去，不过RPN只负责候选区域与候选框的定位而不做类别的预测只是定位物体和背景。对于YOLO和SSD这一类一步算法来说要想要作更精确的预测，还需要在网络设计上做更多的提高精确度的策略。

# 基于YOLO的目标检测方法

本章是全文的重点章节，主要介绍了YOLO v2模型的概念及基本结构，将本文的目标检测模型过程进行了完整的解剖和描述。最后分析对比了该领域现存的几个著名模型与YOLO v2模型的差异。

## YOLO模型概述

YOLO系列模型网络的特点是直接在输出层回归物体的位置和的类别，该模型速度非常快可以实时运行。每次将分析的都是输入图片的整张图片，不会只参考局部，保证了判断时图像参考上下文的完整性，把背景错认为物体的情况少，而且模型很稳定，收敛速度良好，泛化能力强。本节将专注解析关于YOLO所有运行过程和判断方法。

## YOLO系列模型结构发展与变化

YOLO系列的模型是在不断变化并提高的，从2016年初登场的YOLO v1，到2017年的YOLO v2，再到2018年的YOLO v3，其精确度和速度一直在飞速提升。本节将逐步详细介绍YOLO的基本概念和提速策略。

### YOLO v1模型结构概述

YOLO v1作为系列模型的初代，是YOLO系列模型的基础，其后所有系列模型的改进都是从YOLO v1最初的结构开始的。前文提到，YOLO模型和Faster-RNN等基于Region Proposal方法的模型不一样，YOLO结构完全是在一个单独的深度卷积神经网络中实现了目标检测，把检测框选择和类别分类放在了一个网络中。由于输入的是一整张图片，直接经过单个网络回归输出的是检测框，所以是端到端的神经网络模型。

YOLO v1模型首先将输入的图片重新调整为448×448像素大小，将图片分割为7×7网格，对每个网格来说，如果物体中心点落在该格内，则由该格负责检测这个物体属于哪个种类，记为，表示确定网格包含物体的情况下属于的概率。

每个网格内含有两个限位框，**限位框**就是勾勒物体四周的目标结果框（图 3‑1所示），每个限位框用来表示位置与**置信度**，其中是限位框的中心坐标与网格左上角的位移，是限位框的宽与高。为了将这四个参数标准化使其取值落入[0, 1]区间内，最后还需要除以原始图片的长和宽；而指该限位框是否有物体落入，表示限定框与真实框位置的交并比(intersection over union)，即是限位框置信度。

当我们将限位框置信度与网格种类概率相乘时，有：

这样得到了限位框对于包含第类物体的置信度，即同时得到限位框（而不是网格）包含第类物体的概率和限位框有多准确地勾勒真实物体。

如果设定每个网格有2个限位框，这样得到了7×7×2=98个限位框，YOLO v1模型根据设定的阈值去除置信度低的限位框，最后再用非极大值抑制方法去除相近的限位框。由于一个网格负责一个物体，所以一个网格只属于一种分类，即每个限位框在测得坐标和长宽的同时，测得7×7=49个网格属于20种物体分类的概率。每个网格可以得到2个限位框×(4个坐标+1个限位框置信度)+20个分类概率的张量，因此7×7个格子共得到：7×7×(2×5+20)=7×7×30的张量。

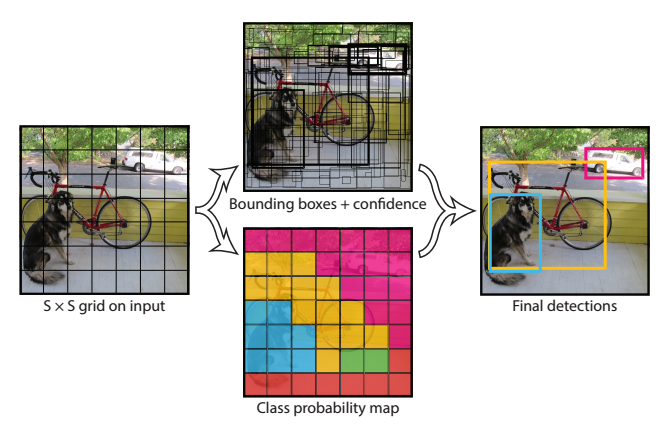


图 3‑2 图片被分割为7×7的网格，每个网格只检测一种类别（图中不同颜色被判定为不同类别），每个网格又只探测两个物体中心落在本格的限位框（图中黑线框）

和所有网络一样，YOLO v1通过卷积层来提取特征，通过全连接层来预测和坐标。YOLO v1模型网络受到了GoogLeNet模型的启发，采用了24个卷积层和2个全连接层，并且用1×1的卷积核代替了GoogLeNet的inception Module。将1×1卷积核跟在3×3卷积层后面。**1×1卷积核**有改变维度的作用：对于单通道输入张量来说，设置不同的padding和stride相当于对输入张量进行缩放操作；当遇到多通道时，由于一个卷积核对应输出结果张量的最后一维的维度数，如果维度数小于原输入通道则属于**降维**，反之升维，相当于跨通道的池化操作。相比传统的降维手段比如白化PCA，使用1×1卷积核降维的特点在于可以帮助减少模型参数（计算量），也可以对不同特征进行尺寸的归一化；也可以用于不同通道上特征的融合，同时保留了空间信息。

YOLO v1网络将目标检测看作一个对限位框位置和置信度进行回归的问题，模型采用均方误差作为损失函数。损失函数如图3-3所示：

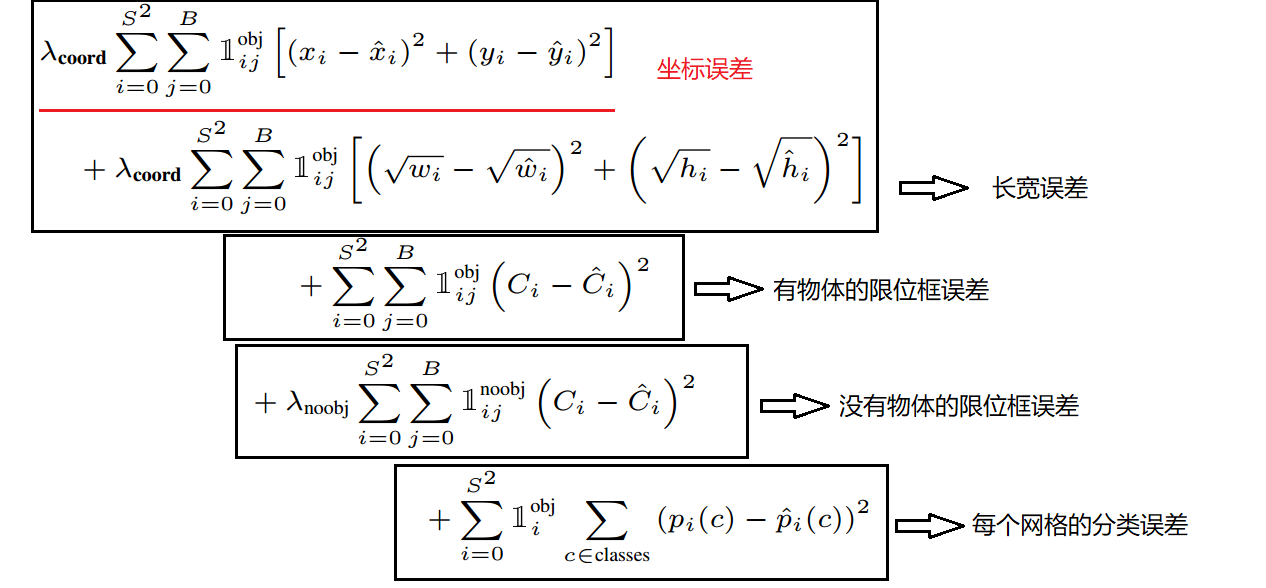


图 3‑3 YOLO v1模型的损失函数设置，采用均方差为基础，引入λx设置不同的惩罚

采用均方误差会导致另一个问题：当网格不包含物体时置信度为0，导致梯度更新时传递0的网格很快覆盖掉其他有物体的网格的参数，使得网络不能收敛，所以网络通过引入参数的方式削弱了不包含物体的网格对损失函数的影响；另一方面，加重限位框坐标位置偏差对损失函数的影响，减少限位框分类偏差对损失函数的影响；在普通测试中发现，较小的限位框的坐标误差往往很敏感，限位框偏离一点就会与真实框在偏差很多（相同的偏差小限位框计算得到的损失函数更大），而大限位框的坐标误差并不敏感，为了均衡限位框大小对损失函数的影响，用长与宽的平方根来进行对比。

### YOLO v2模型结构概述

YOLO v1虽然检测速度很快，但是在检测精度上却不如基于区域选择的R-CNN系检测方法，在限位框定位方面精确度明显低于R-CNN系列。YOLO v1的另一个缺点在于对小物体的探测不足，由于每个网格只有2个限位框，因此当一个网格落入两个以上物体时，YOLO v1模型就无法探测出这些小物体并且可能会出现定位错误。因此在YOLO v2中使用了多种策略，在提升模型的定位的精准度的同时也要继续保持实时的检测速度。

首先，YOLO v2在模型中引入了可以加快模型收敛防止模型网络参数爆炸的著名的**批正则化层**(**BN层**, **Batch Normalization**)。神经网络学习权重参数的根本是为了学习输入数据的数据分布，要求模型需要具备良好的泛化能力，以适应不同的测试数据的分布。在最初的深度神经网络模型中，通常需要对**超参数**进行很多次的尝试分析择取最优，尤其是学习率和参数的初始化。在实际模型的迭代收敛中发现，当输入网络的训练数据每次被随机分为不同小批次(mini-batch)训练时，其分布都不尽相同并不是稳定的，而网络每一层的输入都是前一层的输出，那么在每次迭代梯度更新时模型都需要学习新的数据分布形式，在经过比如*sigmoid*这样的激励函数时会使深层反向传播的梯度变得趋于0近乎失效，*sigmoid*激励函数形式如下：

其导数为：

可以看到*sigmoid*激励函数只有在靠近*x*=0的部分导数才不为0（如图 3‑4），对于落在[-4,4]区间外的数据几乎变化很小导数趋近于0，那么根据链式传播法则，深层反向传播回来的残差回溯的层数越多，信息损失就越多，具体参见当误差结果从*h*层反向传播到第*k*层时，残差*L*对第*k*层输入求偏导过程，有：

其中*L*表示残差，表示第*k*层输入，表示第*k*层输出（），表示第i层权重，表示对第k层输出的加上激励函数处理。

考虑到除1以外的任何数的幂次最后会变得极小或极大（比如0.15=0.00001），由此每一层的权重和激励函数的导数和返回的残差在不断累乘最后也会变得极小或极大：激励函数的导数在0处最大只能取 ，所以在权重小于1的情况下连乘梯度更新结果会迅速趋近于0；而在大于1的情况下也有可能会出现梯度变化巨大的现象。事实上除开*sigmoid*激励函数，换作其他函数（单侧激励抑制的*Relu*，取值全为正的*tanh*，*sigmoid*是最简单的连续可导0-1跃迁函数），只要连续出现权重小于1的情况就有可能梯度更新结果会迅速趋近于0。总的来说，就是在深层累积学习到的残差在反向传播梯度更新时可能会失效或者变得无限大而使网络无法正常更新，即出现臭名昭著的**梯度弥散**或**梯度爆炸**，这将会大大降低网络的训练速度，并使得模型无法收敛。因此为保证模型在进行梯度更新前保证输入数据已经经过归一化，Google提出了BN层，经过BN层后输入数据被划分在均值为0，方差为1范围内，使得反向传播更新权重参数时，不同批次的数据对权重造成的改变不会互相影响，并且可以通过简单变换使数据理论上能够还原最初的形式，避免将已学习到的特征分布破坏。

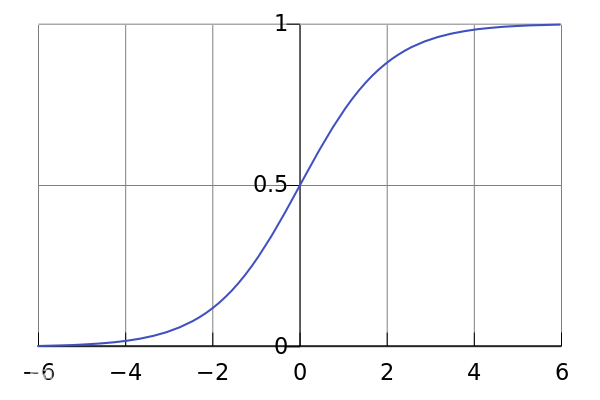


图 3‑5 sigmoid激励函数（也被称为Logistic逻辑回归函数），取值在[0,1]之间，在靠近0的部分增长最快，当绝对值＞5的时候几乎停止增长

YOLO v2使用了新的限位框生成预测策略。在YOLO v1模型中，是先直接使用全连接层来预测限位框的初始坐标，再在其后接上卷积层去提取特征，但是光凭全连接层来进行预测会导致丢失较多的空间信息，造成限位框定位不准，所以YOLO v2 去掉了全连接层，将整个网络变成一个全卷积网络，可以保存更多的空间信息，并且去掉了后面的一个池化层以确保输出的特征图有更高的分辨率。Faster R-CNN模型是人工选取先验预测坐标作为限位框的初始坐标，而没有直接预测限位框的初始坐标，实际探测中比YOLO v1定位精确度更好。Faster R-CNN中的**区域建议网络**(**Region Proposal Network**)使用卷积层来预测**锚箱**(**Anchor Boxes**)的偏移和置信度。**锚箱**是使用9种的候选窗口：三种面积（1282，2562，5122）三种比例（1:1，2:1，1:2）来预测限位框相对先验框的绝对偏移位置。预测偏移值比预测坐标更容易，并使网络更易于学习，所以YOLO v2借鉴了锚箱的思想，在YOLO v1的基础上，改进为预测限位框中心点相对于对应网格左上角的相对偏移值。如图 3‑6所示，其中*pw*，*ph*代表了先验框的长度；*tx*，*ty*，*tw*，*th*表示限位框的位置和长宽。同时为了将限位框中心点约束在当前网格中，使用σ(sigmoid)激励函数处理偏移值，使用这样预测的偏移值可以落在[0,1]范围内。

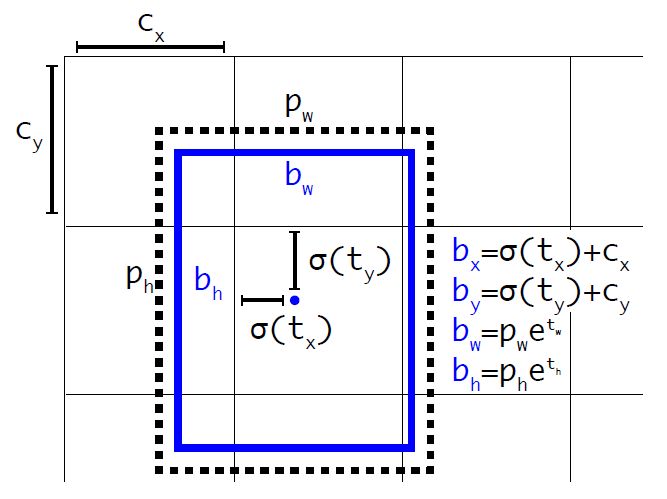


图 3‑7 YOLO v2先验框位置相对偏移预测

进一步改进了YOLO v1对物体尤其是小物体识别不全的问题。由于大物体通常占据输入图像的中间位置，YOLO v2可以只用中心的一个网格来预测这些大物体的位置。YOLO v2缩减模型结构让图片输入分辨率从之前的448×448改为416×416，这是为了让后面产生的卷积特征图的宽与高都为奇数，来产生一个中心网格（偶数会产生四个中心网格）。YOLO v2使用了因子为32的卷积层降采样，使得输入卷积网络的416×416图片最终得到13×13的卷积特征图（416/32=13），即一张图片最终被分为了13×13个网格，而YOLO v1是一开始固定大小的图片就被分割为7×7。对比在YOLO v1中是由每个网格本身来负责预测类别，网格对应的2个限位框负责预测坐标，在YOLO v2中限位框生成策略改为了每个网格对应k个（后文有实验表明最好是5个）限位框预测相对偏移位置和类别概率。根据锚箱策略，如果每个网格可以预测9个限位框，那么总共会预测13×13×9 = 1521个限位框，而之前的YOLO v1网络仅仅预测7×7×2 = 98个限位框，可以探测更多的物体。

在引入锚箱的时候YOLO v2遇到了两个问题，第一个问题在于：Faster R-CNN在初始化锚箱的宽与高的时候，是使用的手工精选的先验框。也就是说如果有更好的初始化锚箱策略，尽可能多地减少网络需要学习的迭代次数，为模型在初始化时选择更好的、更有代表性的先验框宽高，无疑会有更好的效果。因此YOLO v2使用了**K-means聚类方法**来判断需要多少个先验框，和其大概的初始化位置。第二个问题在于：传统的K-means聚类方法使用的是欧氏距离函数，这会使得较大的限位框会比较小的限位框产生更多的误差损失值（大数间的平方差远大于小数间的平方差）。因此YOLO v2将K-means聚类方法的距离函数改为和IOU（交并比）相关，和限位框的大小无关，IOU越大两个框相交得越多，通过与IOU来划分聚类，具体如下：

其中是第*i*个限位框与簇中心限位框的距离，而是第*i*个限位框与簇中心限位框的交并比。

最后，为了保持复杂度不太高的情况下先验框的平均IOU的比较高，要谨慎地选择k的取值，根据实验最后发现k=5时效果最好，即每个网格最好选择5个限位框，根据实验测试改进后的5个先验框策略与完全使用Faster R-CNN锚箱策略得到的平均IOU是相同的，如果再将先验框提升到9个，即每个网格9个先验框平均IOU会更好，但是网络会更大。并且统计发现聚类方法选择扁长的框较少，瘦高的框较多，即选择长宽2：1的较多。

13×13的卷积特征图对大尺寸物体的检测来说可能足够了，但是对小尺度的物体检测可能还需要加强一下。相对于Faster R-CNN和SSD直接在不同尺寸的卷积特征层中使用区域建议策略（Region Proposal）探测不同分辨率图片的不同大小的物体来确保小物体不会被遗漏，YOLO v2参考了Resnet网络的**Residual结构**（如图3‑8所示），选择增加一个**通过层**(**passthrough layer**)来得到更精细的特征层，加强对小物体的探测正确率，具体做法是将最后一层13×13×1024的卷积层前面的 26×26×512大小的卷积层进行隔行隔列地采样，通道增加4倍，变成了13×13×2048，两者进行叠加，使得邻接的特征被叠加到不同的通道中，经过扩张的特征图再进行下一步检测，加强了网络中图像细小的特征的信息，将高一级分辨率的特征连接到低分辨率，有利于检测小目标。

这里介绍一下ResNet中的Residual结构（如图 3‑9所示），越深层的卷积可表达的特征越丰富越抽象，但经过的网络层越深信息的损益也越多，反向传播的梯度在传播的过程中不断衰减使得错误率反而这个问题在层数很深的网络中很突出，即网络深度增加，网络效果反而变差甚至出现欠拟合。实验发现浅层的网络不容易出现该问题，因此如果能使深层的变化为浅层网络就可以解决。通过加上浅层的x来保证残差反向传播时尽可能地保存来自深层学习到的梯度信息，之前的层直接作为输入再次叠加到新层，再次记忆起网络浅层的输入流，对求导有，当时H，保证了梯度可以正常更新，减少了信息损益。

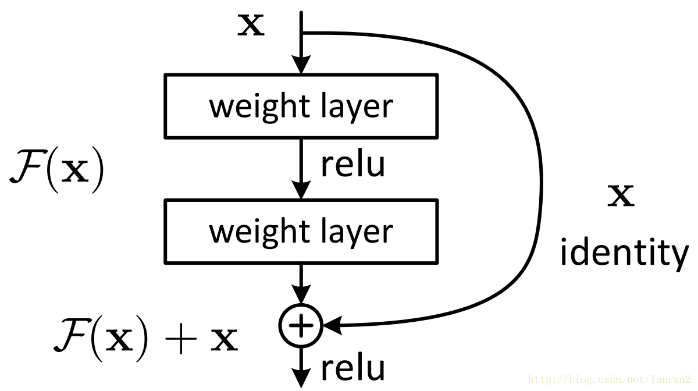


图 3‑10 ResNet网络中的Residual结构，通过加上浅层的x来保证残差反向传播时尽可能地保存来自深层学习到的梯度信息，经过的网络层越深信息的损益也越多。使用该结构可有效减少梯度衰减，配合达百层之多的深层神经网络梯度相关性得到了保持。

前文提到最初的YOLO接受输入的图片分辨率是448×448，加上锚箱后输入图片的分辨率改为416×416，为了使模型能够预测不同分辨率的输入图片，模型在训练的时候每隔10个批次会改变一下模型中输入图片的分辨率，由于还使用了因子为32的卷积层降采样，所以一般会随机选择是32的倍数的分辨率，最小的分辨率是320×320，最大的分辨率是608×608。因为没有固定输入尺寸而使用了不同输入分辨率的图片来训练，YOLO v2可以适应不同大小的图片，并且即使分辨率大也可以在精确度与速度上取得良好的结果（见表 3‑1）。

|  |  |  |
| --- | --- | --- |
| **目标检测网络模型** | **mAP** | **FPS** |
| Fast R-CNN | 70.0 | 0.5 |
| Faster R-CNN VGG-16 | 73.2 | 7 |
| Faster R-CNN ResNet | 76.4 | 5 |
| YOLO | 63.4 | 45 |
| SSD300 | 74.3 | 46 |
| SSD500 | 76.8 | 19 |
| YOLOv2 288 × 288 | 69.0 | 91 |
| YOLOv2 352 × 352 | 73.7 | 81 |
| YOLOv2 416 × 416 | 76.8 | 67 |
| YOLOv2 480 × 480 | 77.8 | 59 |
| YOLOv2 544 × 544 | 78.6 | 40 |

表 3‑2 YOLO v2不同分辨率下的速度与精确度，与其他目标检测网络模型的比较

YOLO v2将可测类别扩展到9000多种，并有效利用了现存的分类数据集作为训练集。目标检测领域的数据集的数据标注要远远比分类数据集的打单属性标签复杂，同理ImageNet分类数据集比VOC等目标检测数据集数量高出很多，并且有众多的维护者和使用者。由于YOLO中限位框的位置预测的是相对偏移，所以可以实现在分类和检测数据集上的联合训练。这种分类和检测的联合训练策略是指，对于检测数据集，可以用来学习预测物体的限位框、置信度以及为物体分类，而对于分类数据集可以仅用来学习分类。YOLOv2在训练网络时把目标检测数据集与分类数据集放在一起，遇到属于目标检测数据集的图片就基于的分类部分和检测部分的损失函数做反向传播；遇到属于分类数据集的图片就仅基于分类部分的损失函数做反向传播。

但是有一个问题需要解决：目标检测数据集的标签太抽象涵盖范围太广，例如“狗”；而分类数据集的标签更为细致，例如ImageNet就有一百多类狗的品种，大多数深度神经网络分类的方法是在最后的全连接层再接一个*softmax*来计算最终的概率分布，*softmax*计算的类别之间是互斥的、完全不相交的，但是融合的数据集之间的类别存在包含关系了，如何融合两类标签成了一个问题。YOLO v2提出了全新的**层级分类方法**(**Hierarchical classification**)，根据各个类别之间的从属关系建立一种树结构，称为**WordTree**（图 3‑11）。大多数分类数据集的分支关系转化为关系树后全部分类都只是互不相关的子节点。每个节点的子节点都属于同一子类，这时它们是互斥的，可以进行*softmax*计算分类。

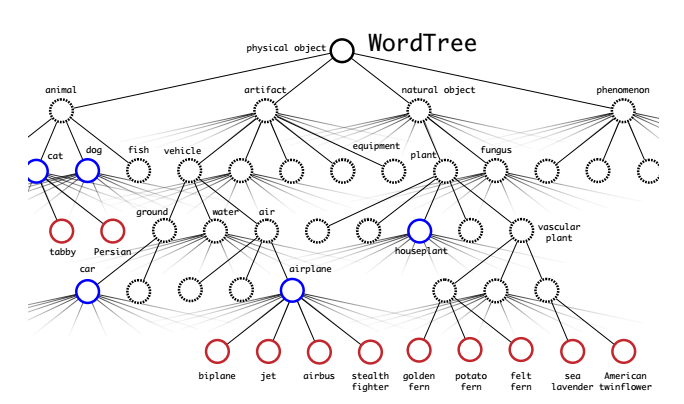


图 3‑12 WordTree的树形结构，同一节点的子节点是互不相关的，因此可以用softmax计算分类概率，分类数据集的分支关系转化为关系树后全部分类都是互不相关的子节点，被标记为节点会继承它所有父节点的属性

WordTree参考了WordNet，WordNet是基于ImageNet分类的**有向图**，对每一个种类，在WordNet上寻找从该种类所在节点到根节点的**最短路径**，将其加入层次树结构，循环这个过程，最终得到了 WordTree。对指定节点即分类种类的绝对概率的计算，有：

我们假定根节点，因此分类种类概率的计算只需要沿着根节点到该节点做连续乘积。YOLO v2用限位框位置和树结构中的概率和类别作为最终输出结果，在层级分类树结构中寻找概率之积最高的路径，当概率之积达阈值时停下，停止处的节点就表示结果类别。注意被标记为节点会继承它所有父节点的属性，当遇到新的或未知物体类别，仍然可以判断大的种类，而不能继续判断明确的是什么分类。使用树状图的另一个好处是可以融合不同数据集的标签，无论是分类数据集还是目标检测数据集。YOLO v2使用ImageNet的前9000类和COCO检测数据集以4：1的比例创造了一个联合数据集并打乱顺序。和之前的策略一样：

1. 当遇到检测数据集图片，对于其他损失部分如常操作，而对于分类损失部分，只在当前路径所有经过的节点分类上进行反向传播，不能传播到子节点，因为没有更精确的信息来了解是否属于子节点的分类情况。比如已知是“狗”，则“狗”之前经过的节点比如“哺乳动物”都会一起传播损失，但是子节点没有经过不能传播信息，比如“猎犬”，“小型犬”就不能传播因为不知道是属于哪一种分类。
2. 当遇到分类数据集图片，仅反向传播分类损失。假设限位框与真实框至少满足IOU >0.31，取该类别所有限位框中置信度最高的来进行分类预测，同样只在当前路径所有经过的节点分类上进行反向传播。

用这种联融合的方法，YOLO v2从检测数据集中学习物体位置，分类数据集中学习更广泛的、可扩展的物体分类。由于分类数据库数量巨大， YOLO v2只从大多数测试数据集中看到过分类数据而非检测数据。最终YOLO v2在ImageNet的评估数据集上测试获得的精度为19.7mAP，在156个从未在任何目标检测数据集见过的物体检测数据类别上测试获得的精度为16.0mAP，高于**DPM**(**Discriminatively trained deformable part models**)，同时还可以以实时的速度检测9000多种其它物体类别。实验发现YOLO v2学习新动物表现很好，但是学习衣服和设备的表现不好。这是因为COCO数据只标注了人的整体类别，如果要检测任何别的细致的物体，需要使用标注相应部位的数据集重新进行训练。

YOLO v2采用了全新的网络结构。大多数深度神经网络目标检测模型是基于VGG-16的，但VGG-16作为特征提取器计算复杂度太高，它的卷积层仅通过一次224×224图片就需要306.9亿次的浮点计算。YOLO模型是基于GoogLeNet的，GoogLeNet比VGG-16快，它的前向传递只需要85.2亿次浮点计算，虽然YOLO的普通模型在ImageNet获得了88.0%的准确度，逊于VGG-16的90.0%的准确度。所以在YOLO v2中采用了全新的**Darknet-19网络**，Darknet-19有19层卷积层，5层最大池化层，令人称赞的是Darknet-19只需要55.8亿次浮点计算，其在ImageNet上的top-1准确率为72.9%，top-5准确率为91.2%。Darknet-19吸收了很多其他网络比如VGG模型的结构的优点，比如使用了较多的3×3卷积核，每进行一次池化都把通道数翻倍，参考**NIN**(**Network in Network**)使用了**全局平局池化**(**Global Average Pooling**)操作来预测，新增了三个 3×3×2048卷积层，添加1×1卷积核插进在3×3卷积层之间来减少参数，在最后一层3×3×2048卷积层前新增一个通过层，同时使用BN批正则化层来加快收敛速度，提升模型鲁棒性和泛化能力。最后使用 1×1 卷积层输出预测结果，输出的通道数等于每个网格的锚箱限位框个数k×(2个相对偏移坐标+2个相对偏移长宽+置信度+分类概率向量) ，即：

YOLO v2模型成型过程首先是在ImageNet分类数据集上预训练Darknet-19，模型初始输入图片分辨率设定为 224×224，在预训练了160轮后，然后将网络的输入分辨率提高到 448×448 ，继续在ImageNet数据集上调优分类模型，训练10轮后在ImageNet上的top-1准确度已经达到76.5%，而top-5准确度是93.3%，并且可以达到实时检测。

### YOLO v3模型结构概述

YOLO v3是在2018年3月底释出的YOLO系列第三代模型，分辨率为320 × 320的YOLO v3可以比SSD速度快三倍并且拥有一样的精确度。在此之前YOLO在定位小物体的精确度上都逊色于Faster R-CNN及SSD，现在YOLO v3不仅远远快于其他目标检测模型，其小物体探测的精确度都有了显著提高，背景误判为物体的误检率低也降低很多。从图3‑13可以看到，YOLO v3 不论在精确度还是在速度上，都领先于其他模型。

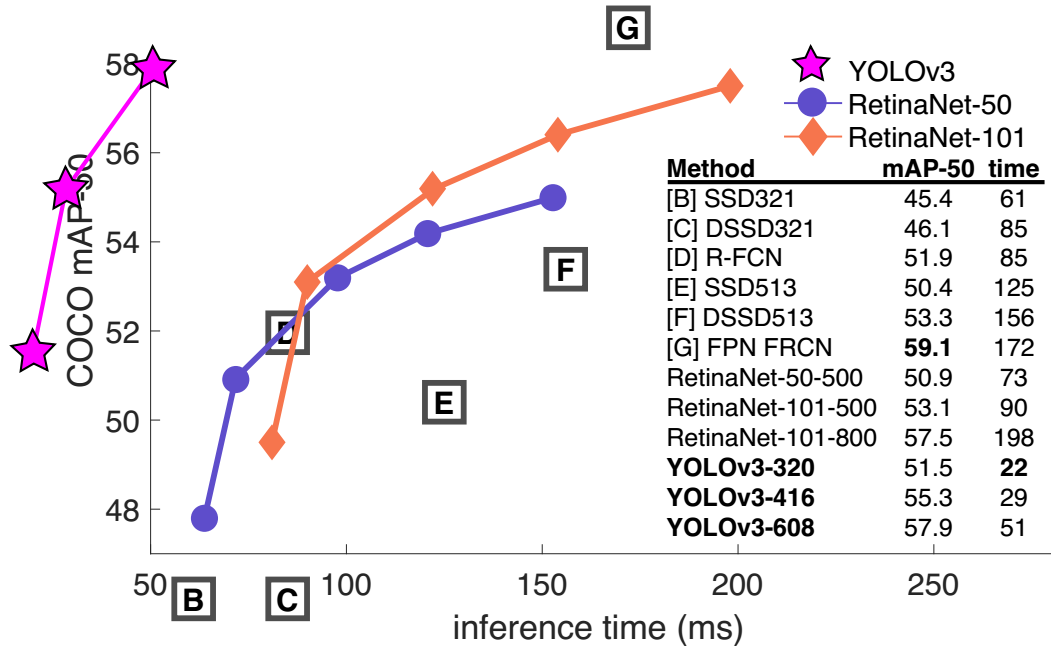


图 3‑14 YOLO v3与其他模型的精确度（单位：mAP）和速度（单位：ms）的对比

从YOLO v1开始，实验就发现模型对小物体探测效果不好，这很大程度上是由于随着网络的加深对输入进行降采样时造成细粒度特征的损失即微小像素被粗糙地池化了，为了弥补这一点，YOLO v2使用了一个类似Residual的回溯连接结构，连接来自上一层的特征映射以补充被模糊掉的浅层网络的特征，但是，YOLO v2的架构仍然缺乏现在大部分目标检测网络最先进的算法中的一些重要元素，YOLO v3整合了所有这些。

YOLO v3网络复杂了不少，其底层体系结构从最初的Darknet-19的19层到Darknet-53的53层，并且在Darknet-53的基础上，总共添加了106层完全卷积层作为最终结构，其中有残余跳过连接结构(residual skip connections)和上采样层，这些结构的中心思想都是与上一层级联保留上一级的特征信息，有助于检测小物体的细微的小像素特征。

YOLO v3最显著的特点是它能够以3种不同的比例作为探测器，即探测器从1个增加到了3个，YOLO v2只有1个13×13的预测卷积核，而YOLO v3现在有3个，在大小分别为13×13、52×52、26×26的卷积核上分别进行预测，这使得YOLO v3的可预测限位框是YOLO v2的可预测限位框的10倍，能探测的物体更多了，但同时速度也相应变慢——不过速度足以应用于实时检测。同时误差函数也进行了变更，YOLO v2中的最后三项是平方误差（如图 3‑15），而在YOLO v3中平方误差已被**交叉熵误差项(cross-entropy error)**，即逻辑回归函数取代，这是因为一张图片标签的分类往往不仅仅是一类，而是多类，这多类别仍然有可能出现有包含的相关关系，同时对置信度和类别的预测也抛弃了原来的二分类函数*softmax*函数，而采用多分类函数*logistic*逻辑回归函数来预测。

YOLO v3更准确了，但是也更复杂了，需要的计算也大大增加，所以本文没有选择YOLO v3，而选择更轻的YOLO v2模型。

## 本章小结

本章主要介绍了YOLO系列模型的基本概念以及在探测过程中模型所使用的策略细节和计算方法。按照时间顺序详细地描述了YOLO系列模型在YOLO v1、YOLO v2、YOLO v3使用不同提高模型精确度和速度的改进策略时的过程及原因。另外，也是本文主要理论内容，为后文的实现打下理论基础。

YOLO v1是最精简的9层网络，但是结构很简朴，速度可能是最慢的，但是它是YOLO系列的开山之作；从YOLO v2开始吸取其他模型的优点，精确度和速度都比YOLO v1提高很多，达到了速度与精确度的平衡；YOLO v3极大地弥补了定位精确度不足的问题，但相对来说最为复杂，计算量最大，网络也最深体积最大。因此，本文通过综合考虑，选择了最适合在移动端运行的YOLO v2模型。

# Android人脸实时检测应用设计与实现

本章主要介绍Android端人脸实时检测应用的设计与实现，包括系统整体设计、模型设计以及系统内各功能模块的设计与界面展示。

## 系统整体设计

### 开发环境简介

本文中设计实现的Android端人脸实时检测应用使用的开发工具是Android开发平台Android Studio，使用的语言是Java与C语言。

**Android Studio**是Google官方开发的Android集成开发环境（IDE），是根据JetBrains公司的IntelliJ IDEA软件的基础上专门为Android开发的，是原生Android应用程序开发的主要平台也是官方唯一推荐的IDE，2018年Google释出了最新的Android Studio 3.1稳定版。Android开发另外还可以使用Eclipse，Eclipse是一个开放源代码的软件开发项目，由于Android从一开始就是用Java语言写的，所以Eclipse不仅可以开发Android应用还用来可以开发Java与JavaWeb应用。本文使用官方推荐的Android Studio进行应用开发。

**Java Native Interference（JNI）**是使得Java可以在代码中调用其他语言如C，C++运行在Java虚拟机（JVM）中，反过来也可以在C，C++代码中调用Java的一种Java与其他语言互相调用的编程框架。JNI诞生的最主要的原因就是在Java诞生之前，就已经有很多程序和库都是由其他语言写的而且运行效果和性能都很好。JNI在Java 1.0就已经存在了，由于Java的跨平台特性，使得Java与本地方法的互相调用能力很差，一旦平台改变或者本地方法及库有改变的话，需要对其进行重新编译。JNI的出现使得Java对于本地方法的调用独立于平台的，并支持从动态共享库加载代码。JNI最重要的好处是它对底层JVM的实现没有任何限制，JVM可以添加对JNI的支持，而不会影响其他部分。但是需要注意的是，JNI在语言调用的时候开销是非常大的。

Android **NDK(Native Development Kit)**是一个工具集，NDK通过JNI调用使得可以在Android上使用C和C ++等语言以本机代码实现部分应用程序。Android生成的.apk文件的java代码很容易被反编译，C/C++语言生成的文件不容易被反编译，并且大部分的开源库都是用C/C++代码编写的，所以有了NDK，Android可以更方便地利用原生代码，并将其移植到不同的平台上。

### 应用整体架构

应用整体将分为五个步骤来构建：

1. 图片输入模块：来自拍照或者相册，并进行预处理（旋转到正确角度）。
2. 人脸检测模型构建：基于人脸数据集的训练人脸检测模型。
3. JNI调用本地语言模块：调用基于本地语言构建的人脸模型，优化JNI调用过程。
4. 移动端模型优化模块：安卓arm系列CPU计算能力弱于PC，需要对模型计算进行一定的优化。
5. 主界面显示模块：安卓端应用构建，主界面显示及检测结果调回。

系统设计流程如图所示。



图 4‑1 应用整体架构流程图

## 基于YOLO v2的人脸检测模型设计

本文使用的人脸检测模型是基于YOLO v2的模型的，一共14层，开头先由6组卷积层与最大池化层组成，然后接2个卷积层3×3的卷积核提取特征减少参数，最后再加上一个1×1的卷积层降维，最终在得到13×13×48的张量上作类锚箱操作预测，参考了YOLO v3中的scale层，把原来只有5个内置限位框预测的锚箱策略改成有8个限位框预测的锚箱策略，大大增加了可探测物体的个数。

其中初始化锚箱尺寸和大小使用(16,16)，(32,32)，(64,64)，(128,128)，(256,256)，(312,312)，(256,256)，(312, 312)等8组高与宽的限位框作为先验框大小。



图 4‑2 基于YOLO v2的模型改进后的结构

由模型图可知，前6组卷积与池化层的组合目的在于学习特征，选择的激活函数均为*leakyRelu*，即单侧激励抑制函数：

*，*

*leakyRelu*函数的好处在于，由于*Relu*类函数是线性函数其收敛速度会比非线性函数如*logistic*快，线性函数计算远远快于非线性，其求导也完全不复杂，所以在反向传播的时候保持了相对的稀疏性，而不像非线性函数一样函数值堆积在0和1附近并且导数值也小于0.25堆叠在[0, 0.25]的区间内。并且为了防止x落在负半轴的时候被完全抑制，*leakyRelu*还添加了一个常数系数使函数值不会完全为0，这样即使学习率大也不会造成因为激活函数值为0导致链式法则连乘后权重更新值为0。

从模型超参数选择上说，模型采用了*darknet-*19的预训练模型作为权重初始化参数，选择的动量梯度更新，每10批次改变图片曝光度等来加强模型泛化能力。学习率一开始为0.0001，30000次的时候学习率变为0.00001。由于批次大小尽量改为能使显存的利用率在50%左右的大小，这样即不会因为显存使用小而请求多造成死机，因为批次大一些时梯度更新的扰动情况少，避免一定程度上的局部收敛。本文所用显卡显存2G十分有限，因此本文每次投入训练的数据量都很小，一开始是32张/批次，后来改为4张/批次，由此设置为最高训练到100,000次停止。

在经过62000次迭代后停止了训练，模型测试人脸检测效果如下所示。

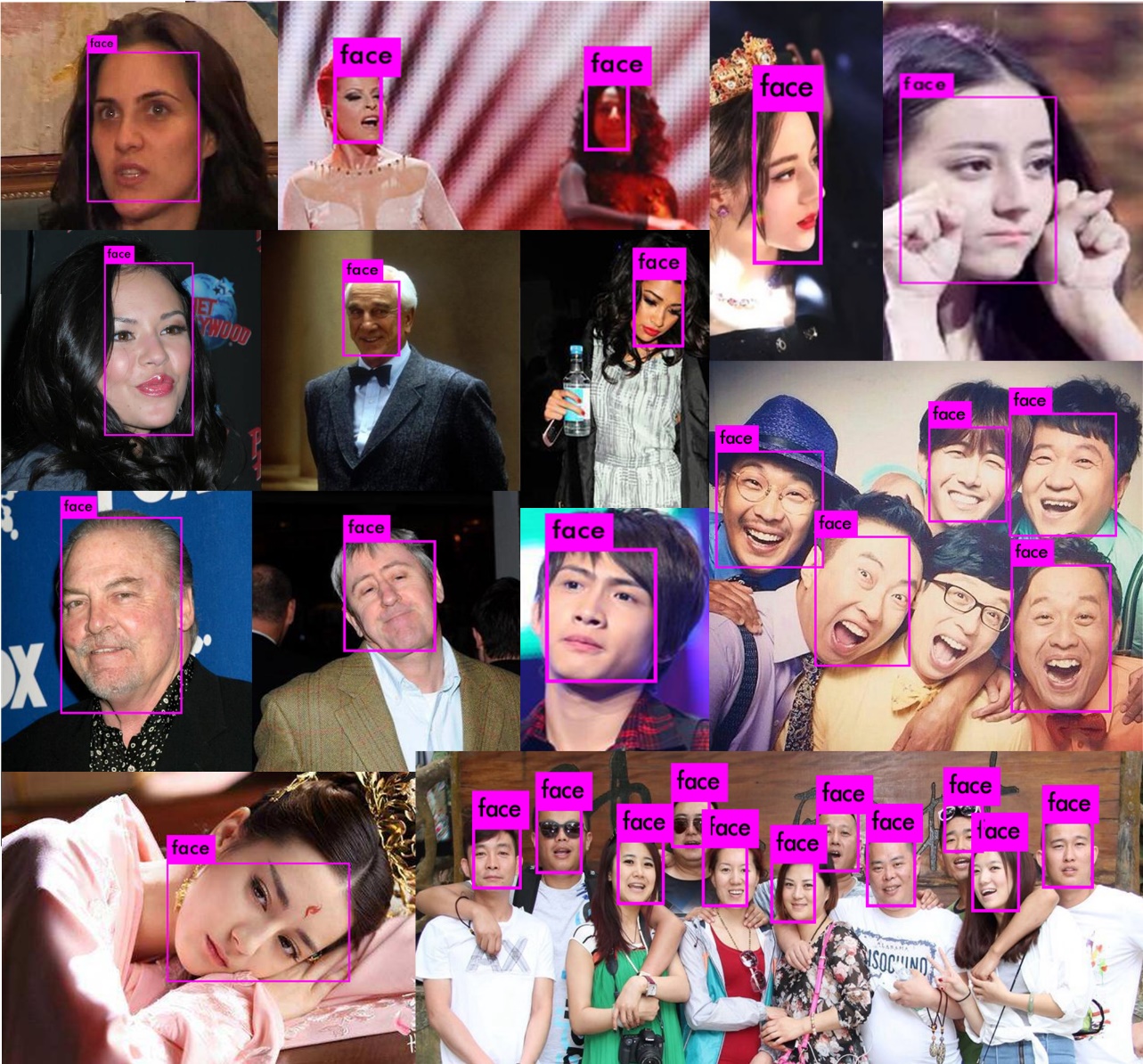


图 4‑3 本文模型的人脸检测测试效果

并将模型放在测试集上进行测试，本文测试环境为Linux 16.04，安装了显卡驱动与Cuda9.1与cudnn7计算库，所使用的显卡为GT740m，位宽64位，计算能力为3在Nvidia显卡中低于平均的3.5。结果如下：

图 4‑4 模型正确率(AP)随批次变化的折线图

图 4‑5 模型定位精确率(IOU)随批次变化的折线图

可以看到在训练44000批次后模型已经接近不错的效果，IOU和AP都在正常范围内波动，例如在训练46000批次后，IOU为61.95%，而AP为88.65%。在本文这样显卡性能较弱的情况下，仍然能保持在200毫秒左右检测出结果。同时在弱光照、模糊、不同对比度、脸部倾斜、中小型脸孔、侧脸等不良检测条件下检测效果依然不错，说明本模型的鲁棒性与泛化性很好，而且准确度和速度也很出色。

## 功能模块设计

本应用的功能模块主要分为图片输入（拍照）模块、JNI本地调用模块、移动端模型优化模块、及安卓应用构建与显示。图片输入拍照模块与JNI本地调用模块由于不在本文讨论范围内，故略去不谈，下面来介绍一下其他功能模块。

### Android端模型优化模块

由于本文选择的模型是基于C语言编写的，故对于Android端C语言模型优化与加速有以下几种优化方式：

1. ARM的SIMD指令优化，即编写底层指令合理安排寄存器及计数器等。
2. ARM的GPU优化，比如OpenCL。
3. 其他加速库。比如Facebook的NNPack，腾讯的ncnn加速库等，还有OpenBLAS等。

使用SIMD指令优化需要较高的汇编能力，而OpenCL更多是作为一种标准；NNPack对于多线程的支持并不好，它在单线程上能力更出众；ncnn是针对底层指令进行优化的，而且支持caffe模型直接转换到移动端实现应用快速落地，但是只针对常用的神经网络层进行了编写，其他层比如YOLO的锚箱层还是需要自己书写指令；本文尝试使用OpenBLAS但是交叉编译没有通过；综上所述，本文最后选择了最简单易试的OpenMP多线程，在小米6上测得时间在1s左右。

本文模型还有另一种实现方式是通过Java编写。由于Android是基于Java书写的，所以可以使用同出于Google之手的TensorFlow来书写，Google在2017年开放了TensorFlow Object Detection API，里面就包含Faster RCNN、R-FCN、SSD等目标检测方法的实现。同时还有转为移动设备定制的TensorFlow Lite，不过里面的方法不是很全面。需要注意的是TensorFlow的学习曲线很陡而且由于是多人维护，代码书写比较复杂。

### Android应用构建与显示

## 本章小结

本章主要介绍了基于YOLO v2的人脸检测模型构建的应用整体设计及各功能模块，以及模型的效果和内容演示，同时提出了模型可以进一步优化的几种策略。

# 总结与展望

## 本文总结

### 本文的主要工作

本文主要研究的是基于YOLO v2的人脸检测模型Android端应用的实现，主要工作内容有以下几个方面：

1. 基于YOLO v2的人脸检测模型的训练及优化改进：手工整合了近3000张人脸数据，将其数据标签并转换为YOLO系列所使用的VOC格式。微调YOLO v2的结构，尤其改变最后三层卷积层的卷积核个数，并且在后一层候选区域提取时锚箱的个数改5个为8个手动设置大小的预置框。模型共训练了62000批次停止。
2. Android端模型移植：本文选择将C语言书写的模型移植到Android端，通过Android-ndk及JNI访问本地语言。
3. Android端模型计算加速与优化：由于是C语言编写的模型，可以使用指令加速和GPU加速，或者现存的加速库等对其进行加速。
4. 手机摄像图片处理：Android端的图片输入输出流处理。对部分手机拍摄时图片会旋转90度、分辨率等的处理。
5. 应用设计与实现：完成人脸检测在Android端界面设计与逻辑实现。

### 本文的主要创新点

本文基于YOLO v2的人脸检测模型Android端应用的创新点主要在于：实现了Android端模型存于本地运行调用。一般商业应用会选择模型保存在服务器，通过网络将结果传输到手机。模型保存于本地可以实现离线调用，并且对于实时检测人脸来说，传输每一帧图片花费的流量太高昂并且需要良好的网络条件。

## 展望

事实上YOLO系列模型虽然解决了速度的问题，但是由于去掉了RPN它们仍然存在以下问题：

1. 定位的准确度及检测精确度下降。这是RPN主要负责的地方，虽然计算消耗很大步骤很繁琐。
2. 细粒度微调程度不如有RPN的模型。Faster R-CNN有很多对细节调优的优化层。
3. 对不同大小尺寸的适应性下降。即对小物体的检测能力弱于有RPN的模型。

虽然本文实现了基于YOLO v2的人脸检测模型Android端应用，但仍有许多不足之处需要改进：

1. 模型的计算能力加速模块有待改进。尤其是针对卷积中矩阵乘法进行指令底层优化，包括SIMD指令优化及GPU优化在内都需要做更多尝试。
2. 需要对人脸实时检测的性能进行优化。由于手机内存及计算能力限制，本文在尝试实现人脸实时检测时固定只允许检测出5张脸，并且设置了较低的分辨率以保持应用能顺畅进行。如果要进行优化可能需要再次提高模型的速度和降低计算耗时，并对视频流进行新的处理。
3. 模型需要对小物体的检测能力还是比较弱，即对小脸检测能力不强。可能是由于YOLO v2只在最后一层卷积层进行了一次浅层细粒度特征和高层抽象特征进行特征重排，YOLO v3中总共进行了3次锚箱操作在不同大小不同深浅的特征图上进行候选区域提取，可以尝试借鉴或引入新结构。

# 致谢

感谢我的老师支小莉与研究生学长张强，他们对本文提出的指导性建议与改进起到了非常大的帮助。感谢我的父母，同学与室友对我平日里的支持和关怀，感谢学校给我们的宝贵的学习机会与学习环境，在此致以我最诚挚的感谢。

# 参考文献

2. 林鸿飞,杨元生.用户兴趣模型的表示和更新机制.计算机研究与发展，200239(7):838-842.
3. 胡学联,潘金贵,李俊,张灵玲. 一个个性化的信息搜集Agent的设计与实现.软件报,2001,12(7):1074-1079.
4. 应晓敏,面向Internet个性化服务的用户建模技术研究. 2003,国防科技大学:长沙
5. 徐振宁,张维明,陈文伟. 基于Ontology的智能信息检索. 计算机科学,20.28(6):21-26.
6. 李勇.智能信息检索中基于本体的个性化用户建模技术及应用. 2002,国防科技学:长沙.
7. FragoudisD.User Modeling in Information Discovery:An overview.InProceedings of Advanced Course on Artificial Intelligence,1999(ACAI99), July,1999, Greece.
8. Balabanovie M, Shoharn Y.Learning Information Retrieval Agents:Experiments with Auto-mated Web BrowsingIn:Proceedings of the AAAISpring Symposium Series on Information Gathering from Heterogeneous, Distributed Environments, March, 1995:13-18.
9. Lieberman H.Letizia. An Agent that Assists Web Browsing.In:Proceedings ofthe International Joint Conference on Artificial Intelligence, Montreal, August, 1995:924-929.
10. Chan P.K.A Non-Invasive Learning Approach to Building Web User Profiles in KDD-99Workshop on Web Usage Analysis and User Profiling, 1999, New York:ACMpress.
11. Schwab L,Kobsa A,and Koychev I. Learning about User from Observation[J].in AAAISpring Symposium on Adaptive User Interface, 2000, Standord,California:AAAI Press.
12. Adomavicius G and Tuzhilin A.Using Data Mining Methods to Build CustomerProfiles.IEEE Computer.Feb 2001:74-82.
13. 余伟. 基于本体的微博客用户行为模型研究[J]. 广东技术师范学院学报, 2010, 31(006): 27-30.
14. 赵岩露, 王晶, and 沈奇威. "基于特征分析的微博用户兴趣发现算法." 电信工程技术与标准化 25.11: 79-83.
15. Uschold M. and Gruninger M., Ontologies: Principle, Methods and Application. Knowledge Engineering Review, 1996, 11(2): p.93-155.
16. Neches R, et al. Enabling technology for knowledge sharing.AI Magazine, 1991, 12(3): p.36-56.
17. Swartout W. Ontologies, 1999. Intelligent Systems and their Applications, IEEE. Issue:1, p.18-19.
18. Gruber T.R.,A Translation Approach to Portable Ontology Specifications .knowledge acquisition, 1993, 5(2): p.199-221.
19. Borst W.N. Construction of Engineering Ontologies for Knowledge Sharing and Reuse. 1997, University of Twente: Enschede.
20. Studer R, Benjamins VR, and Fensel D. Knowledge Engineering: Principles and Methods. Data and Knowledge Engineering, 1998, 25(1-2): p.161-197.
21. 朱晓冰，寇雅楠基于维基技术的本体构建方法探讨图书馆学研究2009.1: p.55-58.
22. 于江生，俞士汝.中文概念词典的结构.中文信息学报.2002年第4期: p.64-68.
23. 由丽凭，杨翠.汉语框架语义知识库概述电脑开发与应用.2007年06期: p.72-74.
24. 张晶，姚建民，赵铁军，李生.基于WordNet和HowNet建设双语语义词典.高技术通讯.2001.12(2-3): p.43-46.
25. 侯汉清，薛春香.用于中文信息自动分类的《中图法》知识库的构建.中国图书馆学报.2005年第5期: p.67-70.
26. 罗志成，马费成，吴晓东，宋倩倩.从维基分类系统构建中文语义词典研究.信息系统学报.2008年02期: p.35-39.
27. 杨玲贤.基于ontology的教学资源知识库构建.计算机与现代化.2009年第11期: p.27-32.
28. Fromkin V, Rodman R. Introduction to Language. London: Holt, Rinehart and Winston, Inc., 1988, 12(1): p.10–16.
29. Brachman R, What IS-A is and isn't: An analysis of taxonomic links in semantic networks [J]. IEEE Computer, 1983, 16(10): p.30–36.
30. Fischer G. User modeling in human–computer interaction[J]. User modeling and user-adapted interaction, 2001, 11(1-2): 65-86.
31. 王金花. 一种利用本体关联度改进的TF-IDF特征词提取方法[D]. 河北大学, 2011.
32. 中国科学院计算技术研究所[EB/0L]. 中文自然语言处理开放平台. http://www.nlp.org.en/project/ project.php?proj\_id=6, 2005-2-2.
33. Scott Deerwester, Susan T Dumais, Furnas W George, et al. Indexing by latent semantic analysis[J]. Journal of the American Society for Information Science, Vol 41, 1990.
34. 王娟琴. 三种检索模型的比较分析研究—布尔模型、向量模型、概率模型[J]. 情报科学, 1998, 16(3): 225-231.
35. 景玉峰等. 概率检索模型.现代图书情报技术, 1987(1): 29-31.
36. 冀胜利, 李波. 基于SVM的中文文本分类算法[J]. 重庆工学院学报(自然科学), 2008, Vol.22 No.7: 84-87.
37. 崔争艳. 中文短文本分类的相关技术研究[D].
38. 埃克尔著, 陈昊鹏译. Java编程思想[M]. 第4版. 北京: 机械工业出版社, 2007.
39. 李刚. 疯狂Java讲义[M]. 第2版. 北京: 电子工业出版社, 2008.
40. 李刚. 轻量级Java EE企业应用实战:Struts2＋Spring3＋Hibernate整合开发[M]. 第3版. 北京: 电子工业出版社, 2011.
41. Dou Shen, Jian-Tao Sun, Qiang Yang, et al. Latent Friend Mining from Blog Data[C]. Sixth International Conference on Data Mining, 2006: 552-561.
42. Katarzyna Musiał, Przemysław Kazienko, and Tomasz Kajdanowicz. Social Recommendations within the Multimedia Sharing Systems[C]. First World Summit on the Knowledge Society, 2008: 364-372.

# 附录：部分源程序清单

//1. 分词、特征词提取

**publicvoid** Feaword()**throws** Exception

{

List<S\_user> userlist =findusers();

Map<String,Integer> globalmap= **new**HashMap<String,Integer>();//存放全部微博特征词

**int** weibo\_count=0;

**for**(**int** u=0;u<userlist.size();u++)

{

String uid=userlist.get(u).getUid();

List<S\_weibo\_content> weibolist=findByUid(uid);

**if**(weibolist==**null**||weibolist.size()==0)

{

}

**else**

{

weibo\_count=weibo\_count+weibolist.size();

**for**(**int** w=0;w<weibolist.size();w++)

{

S\_weibo\_content sweibo=weibolist.get(w);

//单条微博开始

Map<String,Integer> currentmap = **new** HashMap<String,Integer>();//存放单条微博特征词

SplitWord splitWord = SplitWord.*getInstance*();

splitWord.init();

**int** count=0;

String str=sweibo.getContent();

str=str.replaceAll(" ", "");

String result=**null**;

result=splitWord.split(str);

//System.out.println(result);

String[] rr=result.split(" ");

**for**(**int** i=0;i<rr.length;i++)

{

String[] rr1=rr[i].split("/");

**if**(rr1[1].equals("n"))

{

String noun=rr1[0];

count++;

**if**(currentmap.get(noun)==**null**)

{

currentmap.put(noun, **new** Integer(1));

}

**else**

{

Integer number = currentmap.get(noun);

number++;

currentmap.put(noun, **new**Integer(number));

}

}

}

Iterator iter = currentmap.entrySet().iterator();

**while** (iter.hasNext())

{

Map.Entry<String,Integer> entry = (Map.Entry<String,Integer>) iter.next();

String key=entry.getKey();

**if**(globalmap.get(key)==**null**)

{

globalmap.put(key, **new** Integer(1));

}

**else**

{

Integer number=globalmap.get(key);

number++;

globalmap.put(key, **new** Integer(number));

}

}

}//单条结束

}

System.*out*.println("用户"+uid+"的微博分词结束");

}

System.*out*.println("总微博数weibo\_count="+weibo\_count);

//2. 特征词权重计算

**for**(**int** u=0;u<userlist.size();u++)

{

String uid=userlist.get(u).getUid();

List<S\_weibo\_content> weibolist=findByUid(uid);

**if**(weibolist==**null**||weibolist.size()==0)

{

}

**else**

{

**for**(**int** w=0;w<weibolist.size();w++)

{

S\_weibo\_content sweibo=weibolist.get(w);

Map<String,Integer> currentmap = **new** HashMap<String,Integer>();

SplitWord splitWord = SplitWord.*getInstance*();

splitWord.init();

**int** count=0;

String str=sweibo.getContent();

str=str.replaceAll(" ", "");

String result=**null**;

result=splitWord.split(str);

System.*out*.println(result);

String[] rr=result.split(" ");

**for**(**int** i=0;i<rr.length;i++)

{

String[] rr1=rr[i].split("/");

**if**(rr1[1].equals("n"))

{

String noun=rr1[0];

count++;

**if**(currentmap.get(noun)==**null**)

{

currentmap.put(noun, **new** Integer(1));

}

**else**

{

Integer number = currentmap.get(noun);

number++;

currentmap.put(noun, **new** Integer(number));

}

}

}

Iterator iter2 = currentmap.entrySet().iterator();

**while** (iter2.hasNext())

{

Map.Entry<String,Integer> entry2 = (Map.Entry<String,Integer>) iter2.next();

String key=entry2.getKey();

**double** tf=(**double**)currentmap.get(key)/(**double**)count;

System.*out*.println("count="+count);

System.*out*.println("tf="+tf);

**double** idf=(**double**)weibo\_count/(**double**)globalmap.get(key);

idf=Math.*log10*(idf);

System.*out*.println("globalmap.get(key)="+globalmap.get(key));

System.*out*.println("idf="+idf);

**double** weight=tf\*idf;

S\_feaword ff=**new** S\_feaword();

ff.setSweibo(sweibo);

ff.setFeaword(key);

ff.setWeight(weight);

saveFeaword(ff);

}

}

}

}

//3.用户模型兴趣树的显示

<ul class="easyui-tree">

<s:iterator value="#request['menus']" id="c2">

<s:if test="#c2.parent\_id == 0">

<li data-options="state:'closed'">

<span><s:property value="#c2.name" /></span>

<ul>

<s:iterator value="#request['menus']" id="c3">

<s:if test="#c3.parent\_id == #c2.ids">

<li>

<span><s:property value="#c3.name" /></span>

<ul>

<s:iterator value="#request['menus']" id="c4">

<s:if test="#c4.parent\_id == #c3.ids">

<li>

<span><s:property value="#c4.name" /></span>

<ul>

<s:iterator value="#request['menus']" id="c5">

<s:if test="#c5.parent\_id == #c4.ids">

<li>

<s:property value="#c5.name" />

</li>

</s:if>

</s:iterator>

</ul>

</li>

</s:if>

</s:iterator>

</ul>

</li>

</s:if>

</s:iterator>

</ul>

</li>

</s:if>

</s:iterator>

</ul>

//4. 用户兴趣度计算

public void contentinterest() throws Exception

{

List<S\_user> userlist = findByuid("283358587");

//读取所有用户

if(userlist==null||userlist.size()==0)

{

}

else

{

//读取所有本体类别

List<S\_category> categorylist=ms\_categoryDao.getS\_categoryname();

if(categorylist==null||categorylist.size()==0)

{

}

else

{

for(int u=0;u<userlist.size();u++)

{

//读取某一个用户的所有微博

String uid=userlist.get(u).getUid();

List<S\_weibo\_content> weibolist=findByUid(uid);

if(weibolist==null||weibolist.size()==0)

{

}

else

{

for(int c=0;c<categorylist.size();c++)

{

//判断任意一条微博在每一个分类上的权重，如有则类别兴趣累计

Double wc=(double) 0;

Double ws=(double) 0;

System.out.println(weibolist.get(w).getWeibo\_id()); feaword=findBywid(s\_weibo\_content.getWeibo\_id());//找到微博w的所有特征词记录

List<S\_feaword> feaword=findBywfid(categorylist.get(c).getName(),uid);

if(feaword==null||feaword.size()==0)

{

}

else

{

for(int f=0;f<feaword.size();f++)

{

wc=wc+feaword.get(f).getWeight();

ws=ws+categorylist.get(c).getScd();

}

}

if(wc>0)

{

S\_s\_user\_profile user\_profile= new S\_s\_user\_profile();

user\_profile.setUser(userlist.get(u));

user\_profile.setCategory(categorylist.get(c));

user\_profile.setContentweight(wc);

user\_profile.setSemanticweight(ws);

try

{

s\_UserProfileDao.save(user\_profile);

}catch(Exception e)

{

e.printStackTrace();

}

}

}

}

}

}

}

System.out.println("用户完");

}

//5. 用户兴趣度显示

<table id=*"tt"*class=*"easyui-datagrid"*style="width:*auto*;">

<thead>

<tr>

<thdata-options=*"field:'uid',width:150"*align=*"center"*>uid</th>

<thdata-options=*"field:'s\_category.ids',width:100"*align=*"center"*>category\_id</th>

<thdata-options=*"field:'s\_category.name',width:150"*align=*"center"*>category\_name</th>

<thdata-options=*"field:'contentweight',width:200"*align=*"center"*>content weight</th>

</tr>

</thead>

<s:iteratorvalue=*"#request['plansearch']"*id=*"c2"*>

<tr>

<td><s:propertyvalue=*"#c2.uid"*/></td>

<td><s:propertyvalue=*"#c2.category\_id"*/></td>

<td><s:propertyvalue=*"#c2.category\_name"*/></td>

<td><s:propertyvalue=*"#c2.contentweight"*/></td>

</tr>

</s:iterator>

</table>