大数据与人工智能技术丛书

# 计算机视觉

## （Python+TensorFlow+Keras）深度学习实战

微课视频版

袁　雪　著

清华大学出版社

北　京

## 内 容 简 介

人工智能正在成为全世界产业变革的方向，处于第四次科技革命的核心地位。计算机视觉（Computer Vision）就是利用摄像机、算法和计算资源为人工智能系统按上“眼睛”，让其可以拥有人类的双眼所具有的前景与背景分割、物体识别、目标跟踪、判断决策等功能。计算机视觉系统可以让计算机看见并理解这个世界的“信息”，从而替代人类完成重复性工作。目前计算机视觉领域热门的研究方向有物体检测与识别、语义分割、目标跟踪等。本书围绕着计算机视觉的关键技术，介绍基于深度学习计算机视觉的基础理论及主要算法。本书结合常见的应用场景和项目实例，循序渐进地带领读者进入美妙的计算机视觉世界。本书共分为 11 章，第 1 章为人工智能概述；第 2~5 章介绍计算机视觉的几种关键技术，即图像分类、目标检测、图像分割和目标跟踪，并将这四项关键技术组合完成人工智能的实际应用；第 6、7 章介绍人工智能的两个典型应用：文字检测与识别系统及多任务深度学习系统；第 8 章介绍一种非常有意思的深度学习网络——对抗生成神经网络；第 9 章介绍制作训练和测试样本的方法；第 10 章介绍如何安装 TensorFlow、Keras API 及相关介绍；第 11 章介绍综合实验。本书提供了大量项目实例及代码解析，均是基于 Python 语言及 TensorFlow、Keras API 的。本书的每章均配有微课视频，扫描书中的二维码，可观看作者的视频讲解。

本书不仅可以作为大学计算机及相关专业的教材，也适合自学者及人工智能开发人员参考使用。



**图书在版编目（CIP）数据**

计算机视觉 : Python+TensorFlow+Keras 深度学习实战 : 微课视频版/袁雪著. —北京：清华大学出版社，2021.9

（大数据与人工智能技术丛书）

ISBN 978-7-302-57925-0

Ⅰ. ①计… Ⅱ. ①袁… Ⅲ. ①计算机视觉–软件工具–程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字(2021)第 061985 号

**策划编辑**：魏江江

**责任编辑**：王冰飞

**封面设计**：刘 键

**责任校对**：郝美丽

**责任印制**：朱雨萌

**出版发行**：清华大学出版社

网 址：http://www.tup.com.cn，http://www.wqbook.com

地 址：北京清华大学学研大厦 A 座 邮 编：100084

社 总 机：010-62770175 邮 购：010-83470235

投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn

质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn

课 件 下 载：http://www.tup.com.cn，010-83470236

**印 装 者**：三河市天利华印刷装订有限公司

**经 销**：全国新华书店

**开 本**：185mm×260mm 印 张：10.5 字 数：242 千字

**版 次**：2021 年 9 月第 1 版 印 次：2021 年 9 月第 1 次印刷

**印 数**：1~2000

**定 价**：39.8 元

产品编号：090080-01

# 前 言

计算机视觉是人工智能领域的一个重要组成部分，它的主要任务是对采集的图片或视频进行处理以获得相应信息。传统的计算机视觉算法的主要步骤是提取包括边缘、角点、颜色等图像特征，然后利用这些图像特征完成图像处理与机器学习的任务。传统算法的主要问题在于需要告诉系统在图像中寻找哪些图像特性。由于提取图像特征部分是人为设计的，在实现的过程中，对于算法、功能及阈值的更改都需要手工完成，这对高质量的项目实现造成了很大的障碍，而深度学习的出现解决了这一问题。当前，深度学习在处理计算机视觉子任务方面取得了重大进展。深度学习的最大不同之处在于它不再通过精心设计的算法来搜索特定的图像特征，而是通过训练大量的神经网络参数来实现。本书将从计算机视觉的四大关键技术出发，详细介绍基于深度学习的计算机视觉技术的基础理论、主要算法项目实战及代码实现。本书结合常见的人工智能应用场景，循序渐进地带领读者进入美妙的计算机视觉世界。

第 1 章介绍人工智能概述，对人工智能的发展历程及常见的应用案例进行详细介绍；第 2 章讲解卷积神经网络的基本原理，几种常见的深度卷积神经网络框架，并介绍图像分类的项目实战；第 3 章主要讲解目标检测的基本原理，几种典型的目标检测算法，并介绍目标检测的项目实战；第 4 章讲解图像分割的基本原理，几种典型的图像分割算法，结合项目实战使读者进一步理解图像分割算法；第 5 章介绍目标跟踪的基本原理，几种典型的目标跟踪算法，并通过项目实战介绍目标跟踪算法的实现过程；第 6 章讲解文字检测与识别系统的基本构成及原理，几种典型的文字检测及识别算法，并通过项目实战进一步介绍文字检测与识别的实现过程；第 7 章讲解多任务深度学习网络的原理、构建方法和实用技巧，并通过项目实例给出了易于理解的项目实战方法；第 8 章讲解生成对抗神经网络的基本原理，介绍几种典型的生成对抗神经网络算法，并通过项目实例介绍生成对抗神经网络的构建过程；第 9 章主要讲解怎样制作训练样本，包括数据的标注及数据增强两部分；第 10 章介绍 Keras 和 API 的安装方法；第 11 章介绍综合实验。按照以上章节介绍的理论及案例，就可以逐步开启计算机视觉的项目实战了。

时光荏苒，岁月如梭，转眼研究计算机视觉与神经网络已经近二十个年头了，感谢引领我进入这个领域的恩师谷获隆嗣教授，在教授那里学习到的研究方法和学术态度让我受益终身。在漫长岁月里，由于计算资源的限制和一些结构上的缺陷，神经网络一度备受冷落，由衷地敬佩和感慨 Geoffrey Hinton 教授在这一领域锲而不舍地坚持和奉献，

让深度学习真正地进入了产业界，解决了我在漫长二十年的学术生涯中遇到的多个百思不得其解的难题。同样感谢我的学生们的支持和奉献。本书的部分章节参考了我指导的研究生黄伟杰、裴柳、李博、李雪倩、支勇、王贝贝的硕士论文及毕业设计成果。在撰写书稿的过程中，重新翻开同学们的毕业论文，在一起奋战的日日夜夜一幕幕地浮现在眼前。传承、融入、影响、身教、合作、困惑与顿悟汇成了我对计算机视觉的全部理解。

编　者

2021 年 7 月

# 目 录

资源下载

# 第 1 章

# 人工智能概述

微课视频

人工智能正在成为全世界产业变革的方向，处于第四次科技革命的核心地位。计算机视觉就是利用摄像机、算法和计算资源（计算机、芯片、云等）为人工智能系统安上“眼睛”，让其可以拥有人类的双眼所具有的前景与背景分割、物体识别、目标跟踪、判断决策等功能。计算机视觉系统可以让计算机看见并理解这个世界的“信息”，从而替代人类完成重复性工作。目前计算机视觉领域热门的研究方向有物体识别和检测、语义分割、目标跟踪等。

**本章学习目标**

- 人工智能的发展趋势
- 人工智能技术的发展历史
- 计算机视觉技术的应用案例

## 1.1 人工智能的发展浪潮

人工智能（Artificial Intelligence，AI）已经成为新一轮产业革命的核心驱动力，正在对世界经济发展、社会进步和人类的生活产生极其深刻的影响，人工智能将带来第四次产业革命。为了抓住人工智能的机遇，各大科技巨头都将人工智能作为企业最核心的战略。截至 2017 年年底，超过 22 个国家把人工智能上升到国家战略。表 1-1 列出我国发布的相关政策。可以说，无论从国家政策的角度，还是从 AI 技术发展的角度，我们已经正式地跨入人工智能时代。

人工智能开启了时代的变革，它可以说是一种认识和思考的方式，重新认识和思考传统行业、传统生活，重新思考和认识这个世界，我把这种重新认识和思考世界的方式定义为 AI 思维。同时，它也是一种技术，可替代人脑完成感知和决策，可以替代肢体执

**表 1-1　我国发布的相关政策**

| 时　间 | 单　位 | 发 布 政 策 |
|---|---|---|
| 2015.7 | 国务院 | 《国务院关于积极推进“互联网+”行动的指导意见》 |
| 2016.3 | 国务院 | 《国民经济与社会发展第十三个五年规划纲要》 |
| 2016.4 | 工信部、国家发改委、财政部 | 《机器人产业发展规划[2016—2020 年]》 |
| 2016.5 | 中共中央、国务院 | 《国家创新驱动发展战略纲要》 |
| 2016.5 | 国家发改委、科技部、工信部、中央网信办 | 《“互联网+”人工智能三年行动实施方案》 |
| 2016.7 | 国务院 | 《“十三五”国家科技创新规划》 |
| 2017.3 | 国务院 | 《政府工作报告》 |
| 2017.7 | 国务院 | 《新一代人工智能发展规划》 |
| 2017.12 | 工信部 | 《促进新一代人工智能产业发展三年行动计划[2018—2020]年》 |
| 2018.4 | 教育部 | 《高等学校人工智能创新行动计划》 |

行大脑发出指令。综上所述，归纳人工智能时代需要的能力包含 AI 思维和 AI 技术，如图 1-1 所示。人工智能将重新定义生产方式，数据成为生产资料，算法+计算资源成为生产力。互联网是连接生产资料及生产力的方式，从而建立新的生产关系，如图 1-2 所示。其中，AI 技术是一种通用技术，如同计算机或互联网技术一样。而 AI 思维，是利用 AI 技术解决现在和未来各行各业存在的问题，实现替代人类并且超越人类的目的。


图 1-1　人工智能时代必备的能力

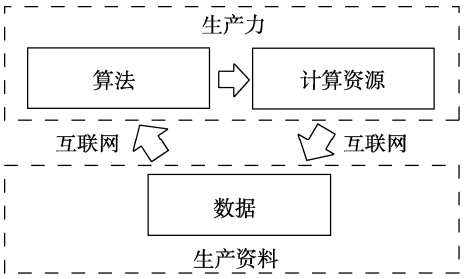


图 1-2　人工智能三要素

我们已经看到，现阶段的 AI 技术在某些领域已经可以替代人类，甚至比人类的生产效率更高。未来的组织人员架构可能是“菱形”的，其中大量处于底部的基础性、重复性日常岗位会被 AI 所取代。人工智能触发的产业变革，将涉及所有行业。行业是否会被 AI 技术改变，甚至被彻底颠覆，以及如何以一种全新的模式重构各行各业，是我们在未来都要思考和实践的。不容置疑，人工智能将带来巨大的社会变革，这次技术革命带来的变化远远超出我们的想象，未来三十年 AI 将深入到社会的方方面面，改变传统制造业、服务业、物流、能源行业，改变教育和医疗，我们所有的生活将随之改变。

深度学习是目前人工智能领域最流行的算法之一。深度学习的概念源于人工神经网络的研究，含多个隐藏层的多层感知器就是一种深度学习结构。深度学习通过组合低层

的特征形成更加抽象的高层属性，从而发现数据的分布式特征表示。研究深度学习的动机在于建立模拟人脑进行分析学习的神经网络，它模仿人脑的机制来解释数据，例如图像、声音和文本等。自从 2012 年起，深度学习就以势如破竹之势破解了一个又一个经典的人工智能问题。如图 1-3 所示，目前深度学习主要的应用领域包括视频分析、语言识别、自然语言处理等。

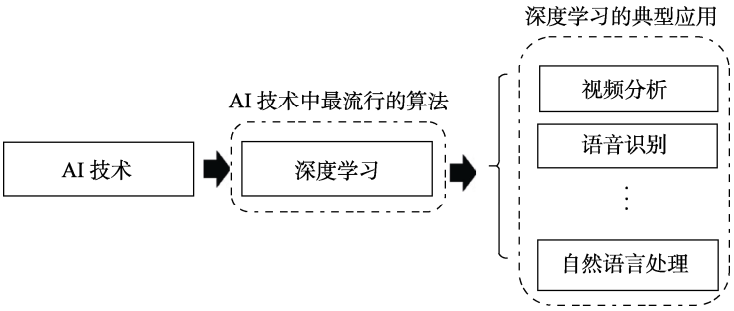


图 1-3　深度学习的典型应用

基于深度学习算法的人工智能系统结构图如图 1-4 所示。人工智能系统中，传感器就好像人的感官，而深度卷积神经网络就好像是人的大脑，对感官采集到的信号进行分析并做出决策。眼睛是人类最重要的感官，人们从外界接收的各种信息中 80%以上是通过视觉获得的，所以，视觉分析对于人工智能系统而言十分重要，也是深度学习技术的重要应用之一。本教程将介绍用于计算机视觉的深度学习技术。

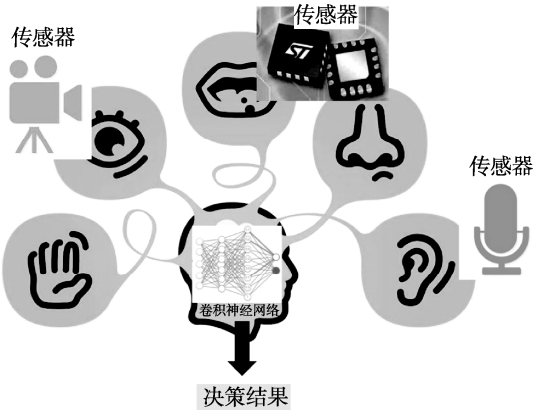


图 1-4　基于深度学习算法的人工智能系统结构图

深度学习利用深层神经网络模拟人脑进行模式识别，从而实现认知外界的目的，从图像中进行特征提取与模式识别，其端到端的学习范式在计算机视觉领域得到广泛应用。相比于基于传统的数字图像处理、几何光学、传统机器学习方法，深度学习往往具有更强大的特征学习和表示能力。

2009 年，美国斯坦福大学李飞飞团队通过 3 年努力，创建了囊括 2.2 万不同种类的物体或场景的 1400 多万张图片的数据集 ImageNet，并组织了相关比赛（ImageNet Large Scale Visual Recognition Challenge，ILSVRC）。2012 年，由 Geoffrey Hinton 及其学生 Alex Krizhevsky 团队所设计的 AlexNet 网络，将卷积神经网络（CNN）首次应用于 ImageNet 数据集上，并以压倒性的优势取得了 LSVRC12 图像分类的冠军，从此开启了深度学习在计算机视觉领域研究的热潮。

## 1.2 AI 技术发展历史

AI 技术的三要素为数据、算法及计算资源。本章将分别介绍 AI 三要素的发展历程。

### 1.2.1 AI 技术三要素之算法

当前，深度学习是 AI 技术中最流行的算法，其基础算法已经较为成熟，各大厂商纷纷发力建设算法模型工具库，并将其封装为软件框架，供开发者使用。目前业内巨头开发了基于自身技术体系的训练及推理的软件框架，将打造开源的深度学习软件框架作为其生态的核心。常用的深度学习软件框架如表 1-2 所示。本书主要结合 TensorFlow 和 Keras 软件模型介绍用于计算机视觉的实用案例。

表 1-2 常用的深度学习软件框架

| 框　架 | 单　位 | 支 持 语 言 | 简　介 |
|---|---|---|---|
| TensorFlow | 谷歌 | Python/C++/Go/… | 神经网络开源库 |
| Caffe | 加州大学伯克利分校 | C++/Python | 卷积神经网络开源框架 |
| PaddlePaddle | 百度 | Python/C++ | 深度学习开源平台 |
| CNTK | 微软 | C++ | 深度学习计算网络工具包 |
| Torch | Facebook | Lua | 机器学习算法开源框架 |
| Keras | 谷歌 | Python | 模块化神经网络库 API |
| Theano | 蒙特利尔大学 | Python | 深度学习开源库 |
| DL4J | Skymind | Java/Scala | 分布式深度学习开源库 |
| MXNet | DMLC 社区 | C++/Python/R/… | 深度学习开源库 |

Keras 是一个非常容易上手的软件框架，它的开发者是 Francois Chollet。Keras 是可以运行于 TensorFlow 或 Theano 上的高端神经网络软件框架，主要有模块化、最小化和易扩展的特点。

深度学习算法的发展历程如表 1-3 所示。1986 年，David Rumelhart 提出了一种适用于多层感知器的反向传播算法——BP（Back Propagation）算法。BP 算法在传统神经网络正向传播的基础上，增加了误差的反向传播过程。反向传播过程不断地调整神经元之间的权值和阈值，直到输出误差减小到允许的范围，或达到预先设定的训练次数为止。

BP 算法完美地解决了非线性分类问题，让人工神经网络再次地引起了人们广泛的关注。但是由于 20 世纪 80 年代计算机的硬件水平有限，运算能力跟不上算法的要求，这就导致当神经网络的规模增大时，使用 BP 算法会出现“梯度消失”的问题，这使得 BP 算法的发展受到了很大的限制。2006 年，Geoffrey Hinton 和他的学生鲁斯兰·萨拉赫丁诺夫正式提出了“深度学习”的概念。他们在世界顶级学术期刊 *Science* 发表的一篇文章中详细地给出了“梯度消失”问题的解决方案，即通过无监督的学习方法逐层训练算法，再使用有监督的反向传播算法进行调优。该深度学习方法的提出，立即在学术圈引起了巨大的反响，以斯坦福大学、多伦多大学为代表的众多世界知名高校纷纷投入巨大的人力、财力进行深度学习领域的相关研究。而后又迅速蔓延到工业界中。2012 年，在著名的 ImageNet 图像识别大赛中，Geoffrey Hinton 领导的小组采用深度学习模型 AlexNet 一举夺冠。AlexNet 采用 ReLU 激活函数，从根本上解决了梯度消失问题，并采用 GPU 极大地提高了模型的运算速度。同年，由斯坦福大学著名的吴恩达教授和世界顶尖计算机专家 Jeff Dean 共同主导的图像识别实验，用 16000 个 CPU Core 的并行计算平台训练一种称为“深度神经网络”（Deep Neural Networks，DNN）的机器学习模型，当把海量的数据投入到模型中，而它自己产生了意识——能够自己领悟到“猫”的概念。深度学习算法在世界大赛脱颖而出，再一次吸引了学术界和工业界对于深度学习领域的关注。随着深度学习技术的不断进步以及数据处理能力的不断提升，2014 年，Facebook 基于深度学习技术的 DeepFace 项目，在人脸识别方面的准确率已经能达到 97%以上，跟人类识

**表 1-3　深度学习算法的发展历程**

| 神 经 网 络 | 深 度 学 习 | |
|---|---|---|
| 神经网络系统开始具有自主学习能力 | 互联网、云计算、大数据技术发展<br>深度学习算法实现应用<br>资本大批进入 | |
| David Rumelhart 于 1986 年提出 BP 算法<br>  <br>1996 年 Deep Blue（深蓝）<br>  | Hinton 于 2006 年正式提出“深度学习”并进行实证<br>  | 2016 年 AlphaGo 首次击败人类围棋世界冠军<br>  |
| 20 世纪 80 年代 | 21 世纪初 | 现在<br>第四次工业革命 |

别的准确率几乎没有差别。这样的结果也再一次证明了深度学习算法在图像识别方面一骑绝尘。2016 年，谷歌公司基于深度学习开发的 AlphaGo 以 4∶1 的比分战胜了国际顶尖围棋高手李世石，深度学习获得了前所未有的热度。深度学习技术正在被推广到其他崭新的应用中。

### 1.2.2 AI 技术三要素之计算资源

人工智能算法的实现需要强大的计算资源，特别是大规模使用深度学习以后，对计算能力提出了很高的要求。2015 年，随着 GPU 的广泛使用，AI 技术迎来真正的大爆发，硬件算力的提升是 AI 快速发展的基础。图 1-5 为通用计算机计算能力的演变。

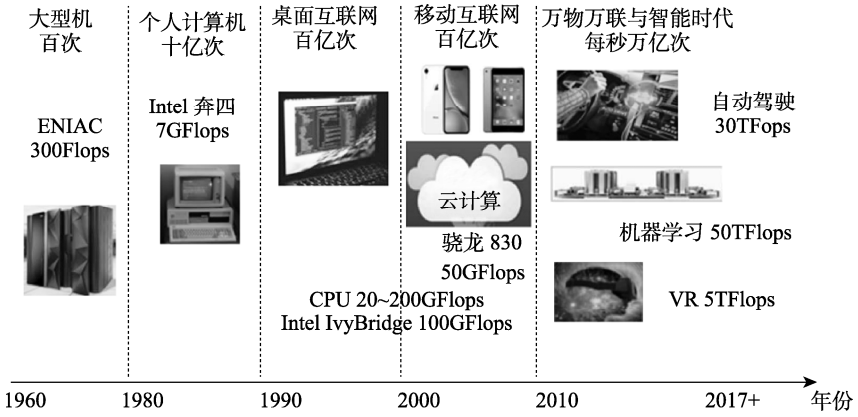


图 1-5 通用计算机计算能力的演变

计算芯片的架构逐渐向深度学习应用优化的趋势发展（如表 1-4 所示），从传统的 CPU 为主 GPU 为辅的英特尔处理器转变为 GPU 为主 CPU 为辅的架构。2017 年，NVIDIA（英伟达公司）推出了图形处理芯片 Tesla V100，谷歌公司为机器学习定制了 TPU。CPU、GPU 和 FPGA 等通用芯片是目前人工智能领域的主要芯片，同时，英特尔、谷歌、NVIDIA 等公司陆续针对神经网络算法开发了专用的 ASIC 芯片，今后可能替代现有的通用芯片成为人工智能芯片的主力。表 1-5 为现有的主要人工智能芯片。

### 1.2.3 AI 三要素之数据

数据是生产材料，通过算法及计算资源完成生产力的升级。大量实验表明，深度学习模型的优化效果也会随着数据量的增大而更加准确。

数据可以分为两种：①公开数据集；②自行采集或购买的行业数据。公开数据集的数据质量较高，但种类和数量较少。为了提高深度学习的泛化能力，一般会采用公开数据集+自采数据的形式。本教程整理了视觉方向的常用公开数据集。

数据处理的步骤分为：①数据清洗；②数据标注；③数据增强。本书将在第 9 章详细介绍数据处理的基本方法。

表 1-4　计算芯片架构的发展趋势

| 类别 | 通用芯片 CPU | 通用芯片 GPU | 半定制化芯片 FPGA | 全定制化芯片 ASIC | 类脑芯片 |
|---|---|---|---|---|---|
| 特点 | 通用性：高<br>性能功耗比：低 | 通用性：高<br>性能功耗比：低 | 通用性：一般<br>可编程<br>性能功耗比：中 | 可定制<br>性能功耗比：高 | 理论阶段<br>性能功耗比高 |

表 1-5　主要的人工智能芯片

| AMD | ARM |
|---|---|
| 云计算：EPYC（霄龙）处理器；Project47 服务器 | CPU 架构：Cortex-A76<br>GPU 架构：Mali G76 |
| NVIDIA<br>新一代处理器架构 VOLTA：新一代 NVIDIA NVLink 高速互联技术<br>云计算：Tesla V100GPU 加速器；DGX-2 全球最大 GPU；GPU 云平台<br>机器人：Jetson Xavier 机器人专用 AI 芯片<br>GPU 工作站：基于 Volta 架构的 GV100<br>无人驾驶：Driver Xavier 首个自动驾驶处理器 | Google<br>云计算：TPU 3.0；Cloud TPU<br>移动端：Pixel Visual Core<br><br>Qualcomm<br>移动端：骁龙 855 处理器<br>无人驾驶：C-V2X 芯片组 |
| Intel<br>云计算：至强可扩展处理 Purley；<br>Xeon+FPGA（云端/设备端低功耗性能计算）；<br>Xeon Phi+Nervana（云端高性能计算）<br>无人驾驶：EyeQ4/EyeQ5 SoC<br>边缘计算：Myriad X VPU.Movidius Myriad X 视觉处理单元（VPU） | Apple<br>移动端：A12 芯片<br><br>Huawei<br>移动端：麒麟 980 芯片 |
| XILINX<br>云计算：可重配置加速堆栈（FPGA-Accelerator Stack）<br>设备端：reVISION 加速堆栈 | NXP<br>跨界处理器：i.MX RT1060 |

**1. 图像分类数据库**

1）MNIST

经典的小型（28×28 像素）灰度手写数字数据集。

2）CIFAR-10

10 个类别，多达 60000 张的 32×32 像素彩色图像（50000 张训练图像和 10000 张测试图像），平均每种类别拥有 6000 张图像。广泛用于测试新算法的性能。类别包括飞机、汽车、鸟、猫、鹿、狗、青蛙、马、船、卡车等。

3）CIFAR-100

与 CIFAR-10 类似，区别在于 CIFAR-100 拥有 100 种类别，每个类别包含 600 张图像（500 张训练图像和 100 张测试图像），然后这 100 个类别又被划分为 20 个超类。因此，数据集里的每张图像自带一个“精细”标签（所属的类）和一个“粗略”标签（所

属的超类）。

4）Caltech-UCSD Birds-200-2011

包含 200 种鸟类（主要为北美洲鸟类）照片的图像数据集，可用于图像识别。类别数量为 200 类；图片数量为 11788 张；平均每张图片含有的标注数量包括 15 个局部位置、312 个二进制属性和 1 个边框。

5）Caltech 101

包含 101 种物品类别的图像数据集，平均每个类别拥有 40~800 张图像，其中很大一部分类别的图像数量为 50 张左右。每张图像的大小约为 300×200 像素。该数据集也可以用于目标检测定位。

6）Oxford-IIIT Pet

包含 37 种宠物类别的图像数据集，每个类别约有 200 张图像。这些图像在比例、姿势和光照方面有着丰富的变化。该数据集也可以用于目标检测定位。

7）Oxford 102 Flowers

包含 102 种花类的图像数据集（主要是一些英国常见的花类），每个类别包含 40~258 张图像。这些图像在比例、姿势和光照方面有着丰富的变化。

8）Food-101

包含 101 种食品类别的图像数据集，共有 101000 张图像，平均每个类别拥有 250 张测试图像和 750 张训练图像。训练图像未经过数据清洗。所有图像都已经重新进行了尺寸缩放，最大边长达到了 512 像素。

9）Stanford Cars

包含 196 种汽车类别的图像数据集，共有 16185 张图像，分别为 8144 张训练图像和 8041 张测试图像，每个类别的图像类型比例基本上都是五五开。本数据集的类别主要基于汽车的牌子、车型和年份进行划分。

**2. 目标检测、定位与分割数据库**

1）Camvid: Motion-based Segmentation and Recognition Dataset

700 张包含像素级别语义分割的道路交通图像分割数据集。类别包括道路、树、墙、建筑物、交通灯、车、行人、天空、人行道、动物、路标等。

2）PASCAL Visual Object Classes（VOC）

用于目标检测与分割的标准图像数据集，同时提供 2007 年与 2012 年两个版本。2012 年的版本拥有 20 个类别，分别包括人、动物、交通工具、室内设施等子类。训练数据的 11530 张图像中包含了 27450 个 ROI 注释对象和 6929 个目标分割数据。

3）COCO 数据集

全称为 Common Objects in Context，用于目标检测与分割。是目前为止语义分割领域最大的数据集，提供的类别有 80 类，超过 33 万张图片，其中 20 万张有标注，整个数据集中个体的数目超过 150 万个。提供的 80 个类别分别包括人、动物、交通工具、室内设施、食物等子类。

4）KITTI 数据集

KITTI 数据集是目前国际上最大的自动驾驶场景下的计算机视觉算法评测数据集。

该数据集用于评测立体图像、光流、视觉测距、3D 物体检测和 3D 跟踪等计算机视觉技术在车载环境下的性能。KITTI 包含市区、乡村和高速公路等场景采集的真实图像数据，每张图像中最多达 15 辆车和 30 个行人，还有各种程度的遮挡与截断。整个数据集由 389 对立体图像和光流图，39.2 km 视觉测距序列以及超过 20 万个 3D 标注物体的图像组成，以 10Hz 的频率采样及同步。

5）Cityscape 数据集

Cityscape 数据集是城市街道场景的语义理解图片数据集，该大型数据集包含来自 50 个不同城市的街道场景中记录的多种立体视频序列，除了 20000 个弱注释帧以外，还包含 5000 帧高质量像素级注释。因此，数据集的数量级要比以前的数据集大得多。Cityscape 数据集共有两套评测标准，前者提供 5000 张精细标注的图像，后者提供 5000 张精细标注外加 20000 张粗糙标注的图像。

## 1.3　视频分析技术的应用案例

计算机视觉真正诞生的时间是 1966 年，在 MIT 人工智能实验室成立了计算机视觉学科，这标志着计算机视觉成为一门人工智能领域中的可研究学科，同时历史的发展也证明了计算机视觉是人工智能领域中增长速度最快的一个学科。视觉是人类获取外界信息的最主要来源，80%的信息获取来源于视觉。人工智能的宗旨是通过机器代替人来完成人类正在做的工作，通过计算机完成视频分析、理解并根据人类的经验做出决策是人工智能系统的重要组成部分。**图像分类**、**目标检测**、**图像分割**和**目标跟踪**是视频分析技术中的关键技术，将以上四种关键技术相互组合可以完成人工智能视觉感知的任务。本书将详细介绍基于深度学习的视频分析主要技术，并通过典型实例介绍通过组合关键技术而完成视觉感知任务的案例。下面介绍几个具体的案例，说明视频分析的应用场景。

### 1.3.1　基于人脸识别技术的罪犯抓捕系统

最近，人脸识别技术抓捕到罪犯的案例层出不穷，它是利用深度学习算法进行视频分析的成功应用，成为公安部抓捕罪犯最有力的工具。罪犯抓捕系统的原理如图 1-6 所示，分为人脸检测及人脸匹配两部分。本书在第 2 章详细介绍图像识别的算法原理及实践案例。

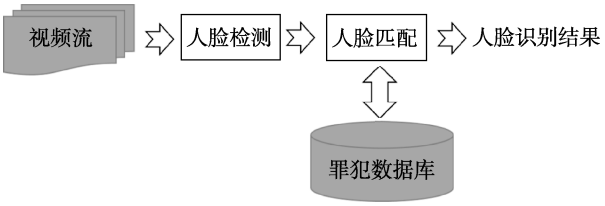


图 1-6　人脸识别系统算法原理图

### 1.3.2 基于文字识别技术的办公自动化系统

在银行、邮局、税务局及政务办事大厅等部门，处理纷繁复杂的手写表格耗费了大量的人力物力。利用视频分析技术将手写表格、资料进行自动识别与理解，分类整理后录入数据库，这是办公自动化的重要组成部分。本书在第 6 章详细介绍文字检测与识别算法的原理及实践案例。

### 1.3.3 基于图像分割及目标检测技术的无人驾驶环境感知系统

无人驾驶是人工智能技术的重要的应用场景。基于视频的环境感知系统是无人驾驶的重要组成部分，它利用视频分析技术通过车载摄像机拍摄的视频流自动感知前方的可行驶区域及障碍物信息。本书在第 7 章详细介绍无人驾驶环境感知算法原理及实践案例。

### 1.3.4 基于目标检测及跟踪技术的电子交警系统

随着视频监控装置的普及，利用视频分析技术自动识别车辆的违规行为是提高城市交通运行效率的关键，包括超速、违规变道、违规压线、闯红灯、违规停车等违法行为，并通过车牌识别技术直接对违规车辆进行处罚。本书在第 11 章详细介绍了目标检测及跟踪技术的实践案例。

### 1.3.5 基于图像比对技术的产品缺陷检测系统

工业产品的质量检测是生产线的重要环节，通过人工智能系统替代质检员，完成高速、准确的产品缺陷检测是自动化生产的必要组成部分。视频可以获取生产产品的外观特征，通过图像对比技术挑选出与正常产品有差异的产品，并进一步分析缺陷的种类。

### 1.3.6 基于行为识别技术的安全生产管理系统

生产安全是大型生产企业生产管理的重中之重，大多数安全事故是由于工作人员的违规行为导致的，例如工作人员进入危险区域、颠倒生产流程、没有观察到工作环境变化等。随着监控设备的普及，通过视频分析技术检测生产人员的违规行为，系统可以及时提醒并预警正是减少生产事故的主要途径。

本书将在第 2~5 章分别介绍计算机视觉的关键技术：图像分类、目标检测、图像分割和目标跟踪。剩下的章节将对计算机视觉领域深度学习算法的其他理论及应用展开介绍。

## 1.4 本 章 小 结

人工智能是 21 世纪最期待的技术之一，人工智能技术的发展将带来第四次产业革命，本章介绍了人工智能的发展历史、发展趋势、应用场景等几个部分。计算机视觉是近年来人工智能技术最热门的应用之一，从第 2 章开始，本书将介绍计算机视觉的关键技术，大家将通过理论学习结合实践操作的形式，逐渐掌握计算机视觉领域的关键技术。

# 第 2 章

# 深度卷积神经网络

微课视频

当前最流行的神经网络是深度卷积神经网络（Deep Convolutional Neural Networks, DCNN），虽然卷积网络也存在浅层结构，但是因为准确度和表现力等原因很少使用。目前提到 DCNN 和卷积神经网络，学术界和工业界不再进行刻意区分，一般都指深层结构的卷积神经网络，层数从“几层”到“几十上百”不定。卷积神经网络目前在多个研究领域均取得了巨大的成功，例如: 语音识别、图像识别、图像分割等。

**本章学习目标**

- 深度卷积神经网络的概念
- 几种典型的深度卷积神经网络框架
- 图像分类项目实战

## 2.1 深度卷积神经网络的概念

**深度学习**（Deep Learning, DL）是机器学习的一个重要分支，源于人工神经网络的研究。深度学习的模型结构是一种含多个隐藏层的神经网络。而多层神经网络目前效果比较好的是卷积神经网络，在图像处理和音频处理上效果较好。

**卷积神经网络**（Convolution Neural Networks，CNN）是一类包含卷积算法且具有深度结构的前馈神经网络，是深度学习的代表算法之一。卷积神经网络仿造生物的视觉机制构建，能够进行平移不变的分类任务。其中，卷积神经网络在计算机视觉中应用广泛，例如图像识别等。本章主要对卷积神经网络的几种典型的结构进行说明。传统神经网络的训练阶段主要包括**特征提取**和**特征映射**。在卷积神经网络中，特征提取是指通过卷积神经网络获得图像的特征图。特征提取阶段通常由卷积层、激活层和池化层构成。其中

卷积和池化的组合可根据模型的不同需求出现多次，组合方式没有限制，如“卷积层+卷积层+池化层”。最常见的深度卷积神经网络结构是“若干个卷积层+池化层”的组合。

**小贴士：为什么卷积神经网络在图像处理领域取得了比其他网络结构更好的效果？**

答：卷积神经网络通过卷积和池化操作自动学习图像在各个层次上的特征，这符合我们理解图像的常识。人在认知图像时是分层抽象的，首先理解的是颜色和亮度，然后是边缘、角点、直线等局部细节特征，接下来是纹理、几何形状等更复杂的信息和结构，最后形成整个物体的概念。卷积层是用卷积核[1]依次与图像的每个像素做乘积，得到特征图。池化处理也叫作降采样处理，是对不同位置的特征进行聚合统计。可以减少参数数量，减小图像尺寸，避免过拟合。

# 2.2 卷积神经网络的构成

## 2.2.1 卷积层

卷积是一种积分变换的数学方法，在许多方面得到了广泛应用。卷积层是 DCNN 的核心结构，由若干个卷积单元构成，目的是提取输入图像的不同特征。对于给定的输入图像，输出特征图中每个像素实际上是输入图像中局部区域中像素的加权平均，其权值由卷积核定义。

如图 2-1 所示，输入图像经过一个 3×3 卷积核矩阵输出特征图。在图 2-1 的局部连接中，右边每个神经元都对应 3×3=9 个参数，这 9 个参数是共享的。卷积核的步长是指卷积核每次移动的像素数，填充像素是卷积前在图像边缘拓展的像素，目的是获得图像边缘特征。在一个 $W \times W$ 的输入图像上，用 $F \times F$ 的卷积核进行卷积操作，卷积核的步长为 $S$，填充的像素数为 $P$，得到的特征图的边长为 $N = (W - F + 2P)/S + 1$。

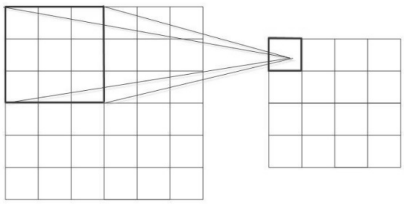

图 2-1 卷积操作

## 2.2.2 激活函数

神经网络中的卷积操作是属于线性操作，因为线性模型的表达能力不够，所以需要在网络中加入非线性因素。深层神经网络中通常使用非线性函数作为网络的激活函数，

1：可以把卷积想象成与矩阵的一个滑动窗口函数。这个窗口函数就是卷积核，又称滤波器或是特征检测器。

通过非线性的组合可以逼近任何函数。如果激活函数是线性函数，那么每一层输出都是上层输入的线性函数，无论神经网络有多少层，输出都是输入的线性组合，加深神经网络的层数就没有什么意义。线性函数的问题在于不管加深层数到多少，总是存在与之等效（无隐藏层）的神经网络。神经网络中常见的激活函数包括 Sigmoid 函数、tanh 函数和 ReLU 函数等。

**1. Sigmoid 函数**

Sigmoid 函数是常用的非线性激活函数，其数学形式见式（2-1）：

$$f(x)=\frac{1}{1+\mathrm{e}^{x}} \tag{2-1}$$

Sigmoid 函数在传统神经网络经常被使用，其作用是将神经元的输出信号映射到[0,1]区间。对于深层卷积神经网络，在进行反向传播时 Sigmoid 函数很容易出现梯度消失的问题。这是因为 Sigmoid 函数存在饱和区间，在饱和区间里函数的梯度接近于零，这样进行反向传播计算得出的梯度也会接近于零。结果是在参数更新的过程中，梯度传播到前几层的时候几乎变为零，导致网络的参数几乎不会再有更新。另外，Sigmoid 函数的输出值始终在 0~1 间，其输出值不是零均值，从而导致上一层输出的非零均值数据作为后一层神经元的输入，产生的结果是如果数据输入神经元是恒正的，那么计算出的梯度也是恒正的，这会产生出锯齿现象而导致网络的收敛速度变慢。虽然使用批处理（Batch）进行训练能够缓和非零均值这个问题，但这仍会给深度网络的训练造成诸多不便。

**2. tanh 函数**

tanh 函数是 Sigmoid 函数的变形，其数学形式见式（2-2）：

$$f(x)=\frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}-\mathrm{e}^{-x}} \tag{2-2}$$

tanh 函数将神经元的输出信号映射到[–1, 1]区间内。tanh 函数的输出是零均值的，在实际的应用中，使用 tanh 函数作为激活函数时，反向传播的收敛速度要优于使用 Sigmoid 作为激活函数时的收敛速度，但也存在梯度消失的问题，会导致训练效率低下。

**3. ReLU 函数**

ReLU 函数是近几年在深度学习领域中非常流行的也是使用最多的一种神经元激活函数，其数学形式见式（2-3）：

$$f(x)=\max(x,0) \tag{2-3}$$

ReLU 函数在 $x>0$ 时的梯度恒等于 1，所以在进行反向传播时，前几层网络的参数也可以得到更新，缓解了梯度消失的问题。另外，Sigmoid 函数和 tanh 函数都需要较大的计算量，而 ReLU 函数能将一部分神经元的输出变为零，等同于对网络参数进行稀疏化处理，减少了网络参数之间的依存关系，缓解过拟合现象的产生。由于 ReLU 函数的线性和非饱和特性，与使用 Sigmoid 函数和 tanh 函数作为激活函数相比较，使用 ReLU 函数能明显加快卷积神经网络的收敛速度。

### 2.2.3 池化层

池化层也被称为采样层，是对不同位置的特征进行聚合统计，通常是取对应位置的最大值（最大池化）、平均值（平均池化）等。

最大池化就是把卷积后函数区域内元素的最大值作为函数输出结果，对输入图像提取局部最大响应，选取最显著的特征，如图 2-2 所示。平均池化就是把卷积后函数区域内元素的算数平均值作为函数输出结果，对输入图像提取局部响应的均值。

池化过程和卷积过程相似，使用不加权参数的采样卷积核，在输入图像的左上角位置按步长向右或向下滑动，对滑动窗口对应区域内的像素进行采样输出。

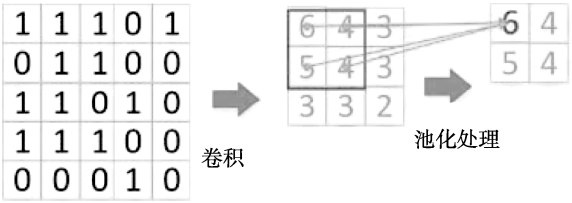


图 2-2 最大池化

池化的优点：①降维；②克服过度拟合；③在图像识别领域，池化还能提供平移和旋转不变性。

通过上述描述，可以将卷积神经网络想象成多个叠加在一起的滤波器，用来识别图像不同位置的特定视觉特征，这些视觉特征在最初的网络层非常简单，随着网络层次的加深变得越来越复杂。

**小贴士：什么是神经网络的过拟合？**

答：过拟合是指为了得到一致假设而使假设变得过度严格，把训练数据学习得太彻底，以至于把误差数据的特征也学习到了。在机器学习中，神经网络的数据一般划分为训练集、验证集和测试集三个部分，用训练集去训练，然后用验证集去验证此阶段神经网络的训练情况。如果在训练集中表现的效果好，但在验证集中的表现忽然变差，之后变得越来越不好时，可能是出现了过拟合。

## 2.3 深度卷积神经网络模型结构

### 2.3.1 常用网络模型

#### 1. LeNet-5

1998 年，纽约大学的 Yann LeCun 等对卷积神经网络改进，该网络模型被称为 LeNet-5。如图 2-3 所示，LeNet-5 卷积神经网络首先将输入图像进行了两次卷积与池化操作，然后是两次全连接层操作，最后使用 Softmax 分类器作为多分类输出。LeNet-5 卷

积神经网络模型对手写数字的识别十分有效，取得了超过人眼的识别精度，被应用于识别邮政编码和支票号码。然而，LeNet-5 卷积神经网络结构简单，难以处理复杂的图像分类问题。实现图 2-3 网络结构的代码清单如下。

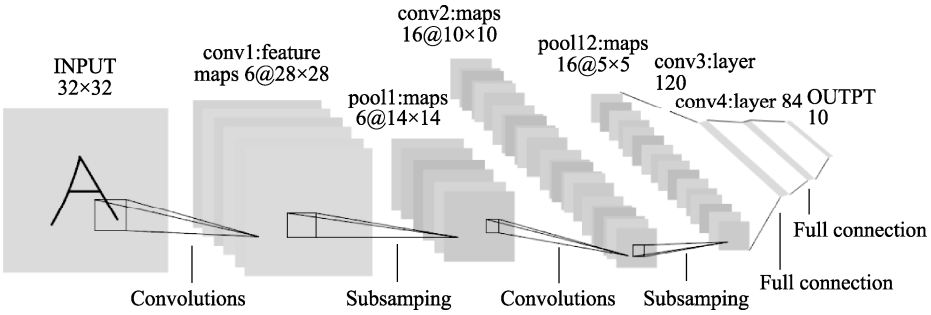


图 2-3　LeNet-5 神经网络结构

**代 码 清 单**

```
#导入各种用到的模块组件

from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.advanced_activations import PReLU
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.optimizers import SGD, Adadelta, Adagrad
from keras.utils import np_utils, generic_utils
from six.moves import range
from data import load_data
import random
import numpy as np

np.random.seed(1024)  # for reproducibility

#加载数据
data, label = load_data()
#打乱数据
index = [i for i in range(len(data))]
random.shuffle(index)
data = data[index]
label = label[index]
print(data.shape[0], ' samples')

#label 为 0~9 共 10 个类别，keras 要求格式为 binary class matrices，需要转化一下，
可以直接调用 keras 提供的这个函数
label = np_utils.to_categorical(label, 10)
###############
#开始建立 CNN 模型
###############
```

```
#生成一个 model
model = Sequential()

#【第一个卷积层】4 个卷积核，每个卷积核大小 5*5。1 表示输入的图片的通道，灰度图为 1 通道
#激活函数用 tanh
#还可以在 model.add(Activation('tanh'))后加上 Dropout 的技巧: mod-
el.add(Dropout(0.5))
model.add(Convolution2D(4, 5, 5, boder_mode='valid',input_shape=(1,28,28)))
model.add(Activation('tanh'))

#【第二个卷积层】8 个卷积核，每个卷积核大小 3*3
#4 表示输入的特征图个数，等于上一层的卷积核个数
#激活函数用 tanh
#采用 maxpooling，pool size 为(2,2)
model.add(Convolution2D(8, 3, 3, border_mode='valid'))
model.add(Activation('tanh'))model.add(MaxPooling2D(pool_size=(2, 2)))

#【第三个卷积层】16 个卷积核，每个卷积核大小 3*3
#激活函数用 tanh
#采用 maxpooling，pool size 为(2,2)
model.add(Convolution2D(16, 3, 3, border_mode='valid'))
model.add(Activation('relu'))model.add(MaxPooling2D(pool_size=(2, 2)))
#【全连接层】先将前一层输出的二维特征图 flatten 为一维的
#Dense 就是隐藏层。16 就是上一层输出的特征图个数
#4 是根据每个卷积层计算出来的: (28-5+1)得到 24,(24-3+1)/2 得到 11，(11-3+1)/2 得
#到 4
#全连接有 128 个神经元节点，初始化方式为 normal
model.add(Flatten())
model.add(Dense(128, init='normal'))
model.add(Activation('tanh'))

#【Softmax 分类】输出是 10 类别
model.add(Dense(10, init='normal'))
model.add(Activation('softmax'))

##############
#开始训练模型
###############
#使用 SGD + momentum
#model.compile 里的参数 loss 就是损失函数(目标函数)
sgd = SGD(lr=0.05, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=
["accuracy"])

#调用 fit 方法，就是一个训练过程，训练的 epoch 数设为 10
#数据经过随机打乱 shuffle=True。verbose=1，训练过程中输出的信息，0、1、2 三种方式
#都可以，无关紧要。show_accuracy=True，训练时每一个 epoch 都输出 accuracy
#validation_split=0.2，将 20%的数据作为验证集
```

```
model.fit(data, label, batch_size=64, nb_epoch=10,shuffle=True,verbose=
1,validation_split=0.2)
```

**2. AlexNet**

随着高效的并行计算处理器（GPU）的兴起，人们建立了更高效的卷积神经网络。2012 年，Hinton 和他的学生 Alex Krizhevsky 设计了深度卷积神经网络 AlexNet，AlexNet 在 ImageNet ILSVRC 比赛中获得冠军，将之前最好的分类错误率 25%降低为 15%。如图 2-4 所示，AlexNet 是一个 8 层的神经网络模型，包括 5 个卷积层及相应的池化层，3 个全连接层。

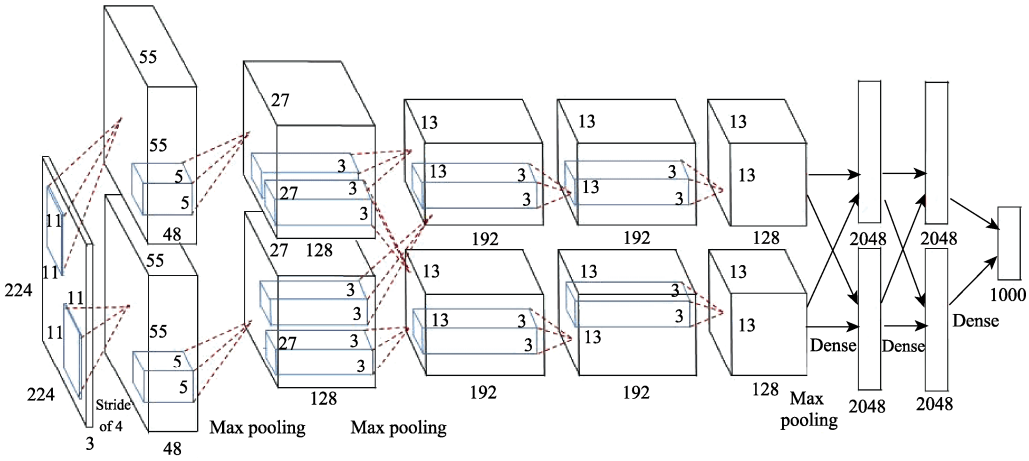


图 2-4　AlexNet 神经网络结构

**3. ZF-Net**

在 2013 年进行的 ImageNet ILSVRC 比赛里，成绩排名前 20 名的小组都是使用深度学习算法，其中 Matt Zeiler 和 Rob Fergus 设计的深度卷积神经网络模型 ZF-Net 获得了比赛的冠军，在不使用额外的训练数据情况下，Top-5 分类错误率达到了 11.7%。ZF-Net 的网络结构如图 2-5 所示，其所使用的卷积神经网络结构是基于 AlexNet 进行了调整，主要的改进是把第一个卷积层的卷积核滤波器的尺寸从 11×11 更改为 7×7 大小，并且步长从 4 减小到 2，这个改进使得输出特征图的尺寸增加到 110×110，相当于增加了网络的

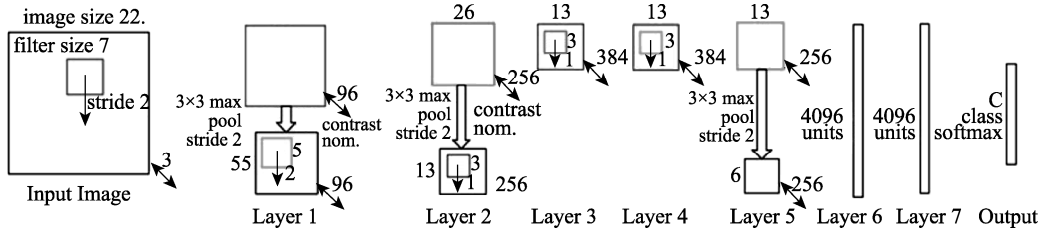


图 2-5　ZF-Net 神经网络结构

宽度，可以保留更多的原始像素信息。

**4. VGG-Net**

在 2014 年进行的 ImageNet ILSVRC 比赛里，来自英国牛津大学的 Karen Simonyan 和 Andrew Zisserman 设计的深度卷积神经网络模型（Visual Geometry Group，VGG）取得了目标定位任务中的第一名与分类任务的第二名。VGG 网络设计的原理是利用增加网络模型的深度来提高网络的性能。VGG 网络的组成可以分为 8 部分，包括 5 个卷积池化组合、2 个全连接特征层和 1 个全连接分类层。每个卷积池化组合是由 1~4 个的卷积层进行串联所组成的，所有卷积层的卷积核的尺寸大小是 3×3。在这当中，利用多个卷积核滤波器大小为 3×3 的卷积层进行串联可以看作是使用一个大尺寸卷积核滤波器的卷积层的分解，例如使用两个卷积核滤波器大小为 3×3 的卷积层的实际有效卷积核大小是 5×5，三个卷积核滤波器大小为 3×3 的卷积层的实际有效卷积核大小是 7×7。这样做的优点是，使用多个小尺寸卷积核的卷积层可以比使用一个大尺寸卷积核的卷积层具有更少的参数，且能在不影响感受野大小的情况下增加网络的非线性，这样使得网络的判别性更强。如图 2-6 所示，VGG 网络根据每个卷积组内卷积层的层数不同，一共有 A~E 五种配置方案（按照列展示）。在图 2-6 中，配置的深度从左（A）到右（E）逐渐增加，增加层的部分用**粗线**标出。根据实际测试的结果显示，随着网络层数的不断加深，VGG 网络

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224×224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

图 2-6　VGG 神经网络结构

的准确率在 16 层时达到性能瓶颈，之后趋于饱和。

**5. GoogLeNet**

来自 Google 公司的 Christian Szegedy 等人设计的 GoogLeNet 网络模型使用的基本结构是利用 Inception 模块进行级联，在实现了扩大卷积神经网络的层数时，网络参数却得到了降低，这样可以对计算资源进行充分使用，使得算法的计算效率大大提高。在 2014 年举行的 ImageNet ILSVRC 比赛中，GoogLeNet 网络模型取得了图像分类任务的第一名。GoogLeNet 由多个 Inception 基本模块级联所构成的，具有更深层的网络结构，其深度超过 30 层。Inception 模块的基本结构如图 2-7 所示，其主要思想是使用 3 个不同尺寸的卷积核对前一个输入层提取不同尺度的特征信息，然后将这些特征信息进行融合操作后作为下一层的输入。Inception 模块使用的卷积核尺寸为 1×1、3×3 以及 5×5，其中 1×1 大小的卷积核较前一层有较低的维度，其作用是对数据进行降维，在传递给后面的卷积核尺寸为 3×3 和 5×5 的卷积层时降低了卷积计算量，避免了由于增加网络规模所带来的巨大计算量。通过对 4 个通道的特征融合，下一层可以从不同尺度上提取到更有用的特征。

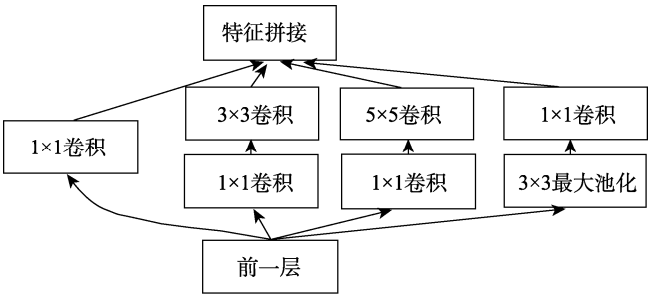


图 2-7　Inception 模块结构

**6. ResNet**

微软亚洲研究院何恺明等人设计的残差网络（Residual Networks，ResNet）在 2015 年举行的 ImageNet ILSVRC 比赛里取得了图像检测、图像定位以及图像分类三个主要项目的第一名，又在同一年的微软 COCO 比赛中取得了检测和分割的第一名。在 ImageNet 比赛中，残差网络的深度达到了 152 层，该深度是 VGG 网络模型深度的 8 倍，但是残差网络的参数量却要比 VGG 网络更少。通常进行训练的网络层数很深时，如果仅仅只是不断叠加标准前馈卷积网络的层数，随着网络深度的增加，深度网络模型训练和测试结果的错误率反而会增加。ResNet 的主要思想就是在标准的前馈卷积网络中，加上一个绕过一些层的跳跃连接。每绕过一层就会产生出一个残差块（Residual Block），卷积层预测添加输入张量的残差，如图 2-8 所示，网络要优化的是残差函数 $F(x)$。ResNet 将网络

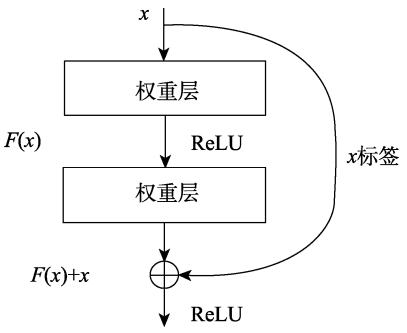


图 2-8　残差结构

层数提高到了 152 层，虽然大幅增加了网络的层数，却将训练更深层神经网络的难度降低了，同时也显著提升了准确率。ResNet 网络一般采用的层数有 18、34、50、101、152，可以根据项目实际的精度及速度要求来选择合适的 ResNet 模型。不同层数的 ResNet 架构如图 2-9 所示。

| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| | | 3×3 max pool, stride 2 | | | | |
| conv2_x | 56×56 | [3×3, 64; 3×3, 64] ×2 | [3×3, 64; 3×3, 64] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 |
| conv3_x | 28×28 | [3×3, 128; 3×3, 128] ×2 | [3×3, 128; 3×3, 128] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×8 |
| conv4_x | 14×14 | [3×3, 256; 3×3, 256] ×2 | [3×3, 256; 3×3, 256] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×23 | [1×1, 256; 3×3, 256; 1×1, 1024] ×36 |
| conv5_x | 7×7 | [3×3, 512; 3×3, 512] ×2 | [3×3, 512; 3×3, 512] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

图 2-9　不同层数的 ResNet 架构

### 2.3.2　网络模型对比

表 2-1 列出了常用深度卷积神经网络的名称、网络的深度、网络的参数量及其在 ImageNet 数据集中的图像分类精度。从表中的数据可以看出，增加网络层数的确能够提升图像分类的精度，从 AlexNet 的 8 层到 VGG 的 19 层，网络参数量从 60M 增加到 144M，图像分类 Top-5 错误率由 15.3%下降到 7.1%。而采用残差结构的 ResNet 在图像分类任务上的 Top-5 错误率降低到了 4.49%。

**表 2-1　常用网络模型对比**

| 模　　型 | 深　　度 | 参　数　量 | Top-5 错误率 |
|---|---|---|---|
| AlexNet | 8 层 | 60M | 15.3% |
| ZF-Net | 8 层 | 60M | 16.0% |
| VGG | 19 层 | 144M | 7.1% |
| GoogLeNet | 31 层 | 8M | 6.6% |
| ResNet | 152 层 | 22M | 4.5% |

## 2.4　图 像 分 类

图像分类是从给定的分类集合中给图像分配一个标签，实现输入图像并返回一个分类标签。标签总是来自预定义的可能类别集。图像分类是将上述的卷积神经网络的最后一层由一个全连接层和 softmax 函数构成，从而达到图像分类的目的。

# 2.5　迁 移 学 习

在大多数情况下，面对某一领域的某一特定问题，很难找到足够充分的训练数据，这是业内一个普遍存在的事实。利用迁移学习的技术，从其他数据源训练得到的模型，经过一定的修改和完善，就可以在类似的领域得到复用，大大缓解了数据源不足引起的问题。

深度学习领域中有超过 50%的高质量论文以某种方式使用了迁移学习或者预训练（Pretraining）技术。迁移学习已经逐渐成为了资源不足（数据或者运算力的不足）的 AI 项目的首选技术。迁移学习的基本思路是利用预训练模型，即已经通过现成的数据集训练好的模型（这里预训练的数据集可以对应完全不同的待解问题，例如具有相同的输入，不同的输出）。开发者需要在预训练模型中找到能够输出可复用特征的层次（Layer），然后利用该层次的输出作为输入特征来训练那些需要参数较少的规模更小的神经网络。由于预训练模型此前已经习得了数据的组织模式，因此这个较小规模的网络只需要学习数据中针对特定问题的特定联系就可以了。

**代 码 清 单**

```
#加载预训练模型
Model.load_weights("model.h5")
#在训练过程中只调整一部分网络层，可以加入以下语句
for layer in model.layers[:10]:
      Layer.trainable = False #冻结前面的 10 层
for layer in model.layers[10:]:#调整 10 层后面的层数
      Layer.trainable = True
```

**小贴士：怎么解决过拟合问题？**

答：过拟合问题的解决方案主要包括以下几点。

**1. L1 和 L2 正则化**

L1 正则化可通过假设权重 $w$ 的先验分布为拉普拉斯分布（Laplace distribution），由最大后验概率估计导出。L2 正则化可通过假设权重 $w$ 的先验分布为高斯分布（Gaussian distribution），由最大后验概率估计导出。L1 正则化更容易获得稀疏解，还可以挑选重要特征。L2 正则有均匀化权重的作用。

**2. 数据增强**

这是最直观也是最有效的方式之一，有了足够的数据支持，DCNN 就会降低发生过拟合的概率。通俗地讲，数据增强即需要得到更多的符合要求的数据，即和已有的数据是独立同分布的，或者近似独立同分布的。一般有以下方法：

（1）从数据源头采集更多数据；

（2）复制原有数据并对其进行添加随机噪声等图像变换处理；

（3）数据合成；

（4）根据当前数据集估计数据分布参数，使用该分布产生更多数据等。

**3. Early stopping**

Early stopping（早停）是一种通过迭代次数截断来防止过拟合的方法，即在模型对训练数据集迭代收敛之前停止迭代来防止过拟合。因为在初始化网络的时候一般都使用初始值较小的权值，训练时间越长，部分网络权值可能越大。如果我们在合适时间停止训练，就可以将网络的能力限制在一定范围。

**4. Dropout**

Dropout（随机失活）是指在深度学习网络的训练过程中，对于神经网络单元，按照一定的概率将其暂时从网络中丢弃。

**5. Batch Normalization**

Batch Normalization（深归一化）是一种非常有用的正则化方法，可以让大型的卷积网络训练速度提高很多倍，同时收敛后分类的准确率也可以有大幅度的提升。

**小贴士：训练的小技巧有哪些？**

答：要时刻注意损失值的变化。在迭代的过程中，损失应该逐渐减少，如果损失长时间不减少，则表示训练已经停止了。解决方案有以下两种：①尝试不同的优化器；②如果训练样本的类别失衡，可以通过加权损失函数来处理。

# 2.6 图像识别项目实例

项目简介：用 ImageNet 库的训练模型，可以实现 2.2 万类的图像分类，下面的例子实现对如图 2-10 所示的图像进行分类。

图 2-10 图像识别数据集实例

**代 码 清 单**

## 2.6.1 下载 ImageNet 的训练模型

```
WEIGHTS_PATH =
'https://github.com/fchollet/deep-learning-models/releases/download/v0
.2/resnet50_weights_tf_dim_ordering_tf_kernels.h5'
```

### 2.6.2　ResNet 模型构建

1）导入第三方库

```
import numpy as np
from keras import layers
from keras.layers import Input, Add, Dense, Activation, ZeroPadding2D,
 BatchNormaliztion, Flatten, Conv2D, AveragePooling2D, MaxPooling2D, G
lobalMaxPooling2D
from keras.models import Model, load_model
from keras.preprocessing import image
from keras.utils import layer_utils
from keras.utils.data_utils import get_file
from keras.applications.imagenet_utils import preprocess_input
import pydot
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
from keras.utils import plot_model
from resnets_utils import *
from keras.initializers import glorot_uniform
import scipy.misc
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import keras.backend as K
K.set_image_data_format('channels_last')
K.set_learning_phase(1)
```

2）定义恒定残差模块

```
def identity_block(X, k_stride, k_size, stage, block):  #定义标识模块
    '''''
    描述:实现偏差单元
    参数:  X – 输入数据
            k_stride – 卷积核步长
            k_size – 卷积核尺寸
            stage – 网络位置
            block – 图层名称
    返回值:X 的激活结果
    '''
    #定义偏差
    conv_name_base = 'res' + str(stage) + block + 'branch'
    bn_name_base = 'bn' + str(stage) + block + 'branch'
    #retrive the filters
    F1, F2, F3 = k_size
    #复制输入数据以供最终添加使用
    X_shortcut = X

    #1 主要路径 卷积->池化->激活
    X = Conv2D(filters = F1,  kernel_size = (1, 1), strides = (1, 1),
```

```
padding = 'valid', name = conv_name_base + '2a', kernel_initializer = glorot_
uniform(seed = 0))(X)  #卷积层 F1，尺寸 1×1，步长(1,1)，补零方法为'VALID'
    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)  #
批标准化
    X = Activation('relu')(X)  #激活函数为 relu

    #2 主要路径 卷积->池化->激活
    X = Conv2D(filters = F2,  kernel_size = (k_stride, k_stride), strides =
(1, 1), padding = 'same', name = conv_name_base + '2b', kernel_initializer
 = glorot_ uniform(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    #3 主要路径 卷积->池化
    X = Conv2D(filters = F3,  kernel_size = (1, 1), strides = (1, 1),
padding = 'valid', name = conv_name_base + '2c', kernel_initializer =
glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)

    #快捷路径
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)
    return X
```

3）定义卷积块模块

```
def convolutional_block(X, k_stride, k_size, stage, block, stride = 2)
:  #定义卷积层
    '''''
    描述：实现卷积操作
    参数：  X – 输入数据
            k_stride – 卷积核步长
            k_size – 卷积核尺寸
            stage – 图层名
            block – 模块名
            stride – 与卷积核不同的步长
    返回值:    X -- X 的卷积结果
    '''
    #定义偏差
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'
    #retrive filters
    F1, F2, F3 = k_size
    #复制输入 X
    X_shortcut = X
    #1 主要路径
    X = Conv2D(F1, (1, 1), strides = (stride, stride), name = conv_nam
e_base + '2a', padding = 'valid',
```

```
    kernel_initializer=glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
    X = Activation('relu')(X)

    #2 主要路径
    X = Conv2D(F2, (k_stride, k_stride), strides = (1, 1), name = conv
_name_base + '2b', padding = 'same',kernel_initializer =glorot_uniform
(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
    X = Activation('relu')(X)

    #3 主要路径
    X = Conv2D(F3, (1, 1), strides = (1, 1), name = conv_name_base +'2
c', padding = 'valid', kernel_initializer = glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)

    #快捷路径
    X_shortcut = Conv2D(F3, (1, 1), strides = (stride, stride),
                            name = conv_name_base + '1',padding =
'valid', kernel_initializer = glorot_uniform(seed = 0))(X_shortcut)
    X_shortcut = BatchNormalization(axis = 3, name = bn_name_base +
'1')(X_shortcut)

    #最后的主要路径
    X = Add()([X, X_shortcut])
    X = Activation('relu')(X)
    return X
```

4）ResNet50 模型构建

```
def resNet50(input_shape = (64, 64 ,3), classes = 6):  #搭建 restnet50 网
络模型
    '''
    描述 : 建立 resNet50 网络
    参数 :  input_shape  -- 输入数据
            classes – 类的数目
    返回值 :模型—keras 模型
    '''
    #将输入定义为具有形状的张量
    X_input = Input(input_shape)
    #补零
    X = ZeroPadding2D((3, 3))(X_input)  #补 3 圈 0
    #Block 1
    X = Conv2D(64, (7, 7), strides = (2, 2), name = 'conv1', kernel_
initializer = glorot_uniform(seed = 0))(X)
    X = BatchNormalization(axis = 3, name = 'bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides = (2, 2))(X) #池化窗口 3×3, 步长 2×2
    # Block 2
```

```
    X = convolutional_block(X, k_stride = 3, k_size = [64, 64, 256],
stage = 2, block = 'a', stride = 1)
    X = identity_block(X, 3, [64, 64, 256], stage = 2, block = 'b')
    X = identity_block(X, 3, [64, 64, 256], stage = 2, block = 'c')
    # Block 3
    X = convolutional_block(X, k_stride = 3, k_size = [128, 128, 512],
 stage = 3, block = 'a', stride = 2)  #卷积模块 a 步长 3，尺寸 128×128×512
    X = identity_block(X, 3, [128, 128, 512], stage = 3, block = 'b')
    X = identity_block(X, 3, [128, 128, 512], stage = 3, block = 'c')
    X = identity_block(X, 3, [128, 128, 512], stage = 3, block = 'd')
    # Block 4
    X = convolutional_block(X, k_stride = 3, k_size = [256, 256, 1024],
stage = 4, block = 'a', stride = 2)
    X = identity_block(X, 3, [256, 256, 1024], stage = 4, block = 'b')
    X = identity_block(X, 3, [256, 256, 1024], stage = 4, block = 'c')
    X = identity_block(X, 3, [256, 256, 1024], stage = 4, block = 'd')
    X = identity_block(X, 3, [256, 256, 1024], stage = 4, block = 'e')
    X = identity_block(X, 3, [256, 256, 1024], stage = 4, block = 'f')
    # Block 5
    X = convolutional_block(X, k_stride = 3, k_size = [512, 512, 2048],
stage = 5, block = 'a', stride = 2)
    X = identity_block(X, 3, [512, 512, 2048], stage = 5, block = 'b')
    X = identity_block(X, 3, [512, 512, 2048], stage = 5, block = 'c')
    #平均池化
    X = AveragePooling2D((2, 2), name = 'avg_pool')(X)  #平均池化
    #输出标签
    X = Flatten()(X)
    X = Dense(classes, activation = 'softmax', name = 'full_connection'+
str(classes), kernel_initializer = glorot_uniform(seed = 0))(X) #全连接层
    #创建模型
    model = Model(inputs = X_input, outputs = X, name = 'resNet50')
    return model
```

### 2.6.3 测试图像

```
img_path = 'images\\myfigure.jpg'
img = image.load_img(img_path, target_size = (64,64))
x = image.img_to_array(img)
x = np.expand_dims(x, axis = 0)
x = preprocess_input(x)
print('Input image shape:', x.shape)
my_image = scipy.misc.imread(img_path)
 model.predict(x)##测试数据
from keras.applications.resnet50
import ResNet50 from keras.preprocessing
import image from keras.applications.resnet50
import preprocess_input, decode_predictions
import numpy as np
 model = ResNet50(weights='imagenet')
```

```
 img_path = 'elephant.jpg'
 img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
preds = model.predict(x) # 测试数据
print('Predicted:', decode_predictions(preds, top=3)[0])
```

## 2.7　本 章 小 结

本章对深度卷积神经网络基础理论进行详细阐述，简单描述 DCNN 的起源以及发展，分析卷积神经网络的网络架构以及相关运算，包括卷积运算、激活函数和池化处理，对常用的卷积神经网络模型进行详细的介绍和分析。并通过项目实战进一步介绍了图像分类任务的实现方法。

## 2.8　习　　题

1. 常用的池化操作有哪些？各有什么特点？
2. 思考 Dropout 为何能防止过拟合？
3. 试计算 ResNet-50 的总参数量。
4. 给定卷积核的尺寸，如何计算特征图大小？
5. 当学习率太高或太低时会怎么样？
6. 激活函数有哪些？各有什么用途？
7. 详解深度学习中的梯度消失原因及其解决方法。

# 第 3 章

# 目标检测

微课视频

目标检测是计算机视觉中一个热门方向，广泛应用于机器人导航、工业检测、航空航天等诸多领域，也是智能监控系统的核心部分，同时目标检测也是泛身份识别领域的一个基础性的算法，对后续的人脸识别、步态识别、人群计数、实例分割等任务起着至关重要的作用。

**本章学习目标**

- 目标检测的概念
- 典型的深度学习目标检测算法
- 目标检测算法的评价指标
- 项目实战

## 3.1 目标检测的概念

目标检测的任务是找出图像中所有感兴趣的目标，并确定它们的**位置**和**类别**（如图 3-1 所示）。由于各类物体有不同的形状、姿态，加上成像时受光照、遮挡等因素的干扰，目标检测一直是计算机视觉领域最严峻的挑战之一。

目标检测与识别在生活中多个领域有着广泛的应用，如图 3-1 所示，它可以将图像或者视频中感兴趣的物体与不感兴趣的部分区分开；判断是否存在目标；确定目标的位置；进一步识别确定的目标等。

2014 年，Girshick 等创新性地提出了 R-CNN（Regions with CNNs Features）算法，开启了深度学习在目标检测方面应用的新纪元。R-CNN 算法分为挑选候选窗口及进一步甄别候选窗口两部分。2015 年，Girshick 在 R-CNN 的基础上提出了 Fast R-CNN 算法，Fast R-CNN 的处理时间比 R-CNN 提高了 25 倍，但由于挑选候选框算法是在 CPU 下运

图 3-1　目标检测的实例

算的，其处理速度仍然无法满足实时的要求。随后发表的 Faster R-CNN 算法提出了通过 RPN 网络替代 R-CNN 与 Fast R-CNN 算法中的挑选候选窗口部分，使得目标检测算法中的所有计算过程都在 GPU 内进行，计算的速度和精度都有了大幅度的提升。Faster R-CNN 的处理速度比 R-CNN 提高了 250 倍，达到了 5 帧/秒。2016 年，Joseph Redmon 等在 Faster R-CNN 的基础上提出了 YOLO（You Only Look Once）算法，将 Faster R-CNN 中挑选候选窗口和甄别候选窗口合二为一进行处理，处理速度达到 45 帧/秒。同年，为了解决 YOLO 算法难以检测小目标的问题，SSD 算法做了两个关键性的改进：①合并深度学习网络的不同层级；②引入与 Faster R-CNN 相似的变形框概念。2017 年，Joseph Redmon 等在前期的基础上做了重大改进，提出了 YOLO v2 版本，YOLO v2 算法在应用变形框的同时，网络结构、输入图像尺寸等方面也做了较大的调整。2018 年，又提出了 YOLO v3 算法，YOLO v3 在 YOLO v2 的基础上兼顾了速度和检测精度，检测速度可以达到 58 帧/秒，检测精度可以达到 73mAP。2020 年，YOLO v4 面世，在 YOLO v3 的基础上，在检测精度及速度方面有了进一步地提升，可以说，YOLO 系列算法已经成为工业界目标检测应用最广泛的算法之一。

目前目标检测算法主要包括**基于候选区域**的卷积神经网络算法及**基于回归方法**的卷积神经网络算法。以下将分别介绍这两种算法。

## 3.2　基于候选区域的目标检测算法

基于候选区域的深度卷积神经网络（Region-based Convolutional Neural Networks）是一种将深度卷积神经网络和区域推荐相结合的物体检测方法，也可以叫作**两阶段目标检测算法**。第一阶段完成**区域框的推荐**，第二阶段是对区域框进行**目标识别**。区域框推

荐算法提供了很好的区域选择方案，使用图像中的颜色、纹理、边缘等图像特征信息作为目标区域推荐的依据，预先在图像中找出可能会出现目标的位置。这种有针对性的选取目标区域，可保证在选取较少区域框的情况下仍然保持很高的召回率，从而降低了时间复杂度。在推荐候选区域框之后对该候选区域框内的图像进行提取特征，最后进行图像的分类工作。下面选取 Faster R-CNN 和 R-FCN 这两种主要的算法进行说明。

### 3.2.1 Faster R-CNN 目标检测算法

由 Ross B. Girshick 等人提出的 R-CNN 算法首先利用选择性搜索（Selective Search）算法在图像中提取数千个候选区域，然后利用卷积神经网络对每个候选区域进行目标特征的提取，接着用每个候选区域提取到的特征来训练支持向量机分类器对候选区域进行分类，最后依据每个区域的分类得分使用非极大值抑制（NMS）算法和线性回归算法优化出最终的目标位置。R-CNN 算法的训练被分成多个阶段，分开训练提取特征的卷积神经网络、用于分类的分类器和边框回归器，步骤非常烦琐，这导致训练过程要耗费大量的时间和存储空间，而且每一个候选区域都需要运行整个前向网络计算，这使得测试的速度非常慢。并且特征提取网络和分类器的训练不相关，这影响了目标检测的准确率。Fast R-CNN 将特征提取和分类融合进一个分类框架，直接使用 softmax 函数替代了 SVM 分类，提高了训练模型的速度，取得了更高的目标检测准确率。但 Fast R-CNN 生成候选区域的方法是使用非常耗时的选择性搜索算法，目标检测时间大多消耗在这上面，使得 Fast R-CNN 无法满足实时应用。Faster R-CNN 算法，在 Fast R-CNN 上增加候选区域推荐网络（RPN）来取代选择性搜索算法用于生成候选区域，大大加快了目标检测速度。

Faster R-CNN 方法中最重要的是使用候选区域推荐网络获得准确的候选区域框，代替了传统的 Selective Search 和 Edge Boxes 等方法，并且将选择区域框的过程嵌入卷积神经网络中，与 Fast R-CNN 网络共享卷积层的参数，从而提高网络的训练和测试速度。候选区域推荐网络的核心思想是使用卷积神经网络直接产生候选区域框，使用的方法本质上就是滑动窗口，因为 Anchor 映射机制和边框回归可以得到多种尺度和多种长宽比的候选区域框。RPN 网络将每个特征图的位置编码成一个特征向量，对每一个位置输出物体得分（Objectness Score）和区域回归边界，即在每个卷积映射位置上输出这个位置的 3 种尺度和 3 种长宽比的共 9 个区域建议的物体得分和回归边界。后边再接入到分类和边框回归这两个全连接层。分类层包含 2 个元素，用于判别前景和背景的估计概率。回归层包含 4 个坐标元素，用于确定目标位置。RPN 网络示意图如图 3-2 所示。

### 3.2.2 基于区域的全卷积网络（R-FCN）目标检测算法

在基于深度卷积神经网络的图像分类及目标检测两项任务中，分类是要增加目标的平移不变性，而检测则要求对目标的平移做出准确响应，即减少目标的平移变化，因为目标检测不仅要对目标进行分类，而且要确定目标具体位置。但是常用的网络模型比如 AlexNet、VGG 和 ResNet 等都是基于 ImageNet 的分类任务所训练的，所以会偏向于平移不变性，这与目标检测任务存在矛盾。Faster R-CNN 算法在网络的卷积层之间插入 ROI

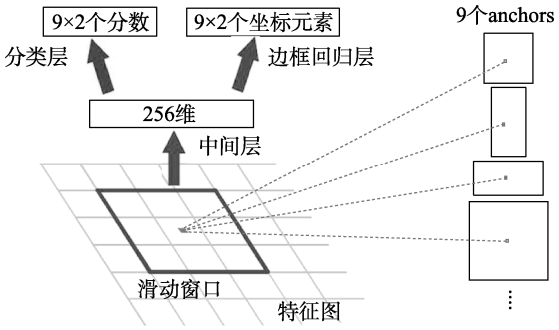


图 3-2　RPN 网络结构

池化层，这种方法在 ROI 池化层之前都是卷积，是具备平移不变性的，但一旦插入 ROI 池化层之后，后面的网络结构就不再具备平移不变性了。R-FCN 方法的整体结构全部由卷积神经网络组成，为了给全卷积神经网络引入平移变化，用专门的卷积层构建了位置敏感分数地图（Position-Sensitive Score Maps）。每一个空间敏感地图对感兴趣区域的相对空间位置的信息进行了编码，并插入感兴趣区域池化层来接收整合信息，用于监管这些分数地图，从而给卷积神经网络加入了平移变化。

R-FCN 网络结构示意图如图 3-3 所示，采用了类似于 R-CNN 的物体检测策略，包括区域推荐和区域分类两部分。使用 Faster R-CNN 中的区域推荐网络进行候选区域的提取，该区域推荐网络为全卷积网络。R-FCN 在与区域推荐网络共享的卷积层后面多增加了 1 个卷积层，最后 1 个卷积层的输出从整幅图像的卷积响应图像中分割出感兴趣区域的卷积响应图像。R-FCN 最后 1 个卷积层在整幅图像上为每类生成 $k^2$个位置敏感分数图，有 $C$ 类物体外加 1 个背景，因此有 $k^2$（$C$+1）个通道的输出层。

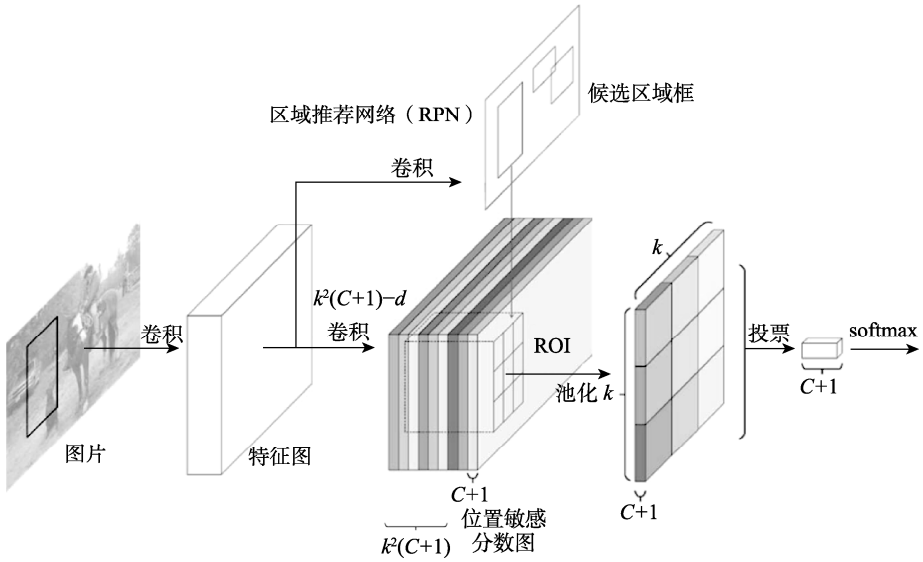


图 3-3　R-FCN 网络结构示意图

# 3.3 基于回归的目标检测算法

目前在深度卷积神经网络的物体检测方面，Faster R-CNN 是应用比较广泛的检测方法之一，但是由于网络结构参数的计算量大，导致其检测速度慢，从而不能达到某些应用领域实时检测的要求。尤其对于嵌入式系统，所需计算时间太长。同样，许多方法都是以牺牲检测精度为代价来换取检测速度。为了解决精度与速度的问题，YOLO 与 SSD 方法应运而生，此类方法使用基于回归方法的思想，直接在输入图像的多个位置中回归出这个位置的区域框坐标和物体类别。

## 3.3.1 YOLO 目标检测算法

YOLO 是端到端的物体检测深度卷积神经网络，与 Faster R-CNN 的区别在于 YOLO 可一次性预测多个候选框，并直接在输出层回归物体位置区域和区域内物体所属类别。而 Faster R-CNN 仍然是采用 R-CNN 那种将物体位置区域框与物体识别分开训练的思想，只是利用 RPN 网络，将提取候选框的步骤放在深度卷积神经网络内部实现。YOLO 最大的优势就是速度快，可满足端到端训练和实时检测要求。

如图 3-4 所示，YOLO 方法的物体检测过程为：首先将输入的图像划分成 7×7 个小网格，在每个小网格子里预测出 2 个区域框，从而可在整张图像上预测出 2×7×7 个目标物体的区域框。利用交并比（IoU）衡量这些区域框与图像上的真实区域框的差距，得到可能性高的候选区域框。最后使用非极大值抑制去掉多余的区域框。YOLO 整体训练方法过程相对简单，不需要中间的推荐区域步骤，直接通过网络回归完成物体的定位与分类。YOLO 将物体检测任务转换为对回归问题的解决，每秒能够检测 45 张图像，在一定程度上提高了物体检测速度。但是 YOLO 也存在一定的精度问题，由于不存在区域推荐机制，只使用 7×7 的网格进行回归会使得 YOLO 不能非常准确地定位出目标的位置，从而导致物体检测的精度较差。

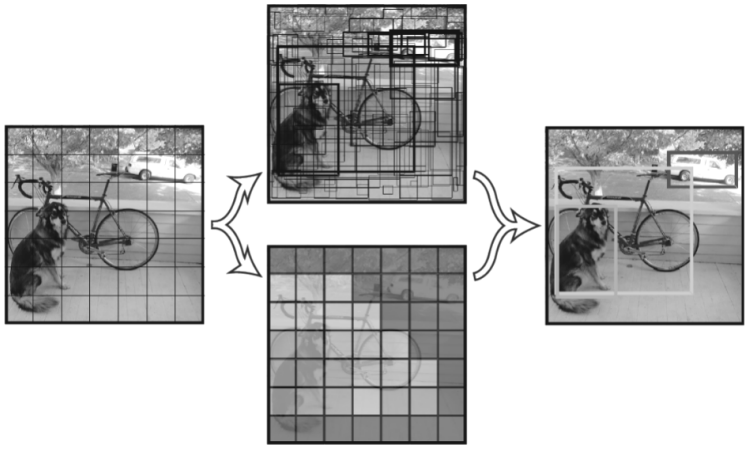

图 3-4 YOLO 目标检测过程

通过对 YOLO 和 Faster R-CNN 的误差比较分析，可以看到 YOLO 造成了大量的定位误差。此外，与基于候选区域的方法相比，YOLO 召回率相对较低。因此，YOLO 的改进版本 YOLO v2 主要改进点在于改进定位精度和提高召回率，同时要保持分类准确性。并且通过在 YOLO 所有卷积层上添加了批标准化（Batch Normalization），网络的收敛性得到了显著的改善，同时消除了对其他形式正则化的需求。原来的 YOLO 以 224×224 的分辨率训练分类器网络，YOLO v2 版本将分辨率提高到 448×448 进行检测，从 YOLO 中移除全连接层，并且使用 Faster R-CNN 中区域推荐机制来预测边界框。这个过程首先是消除了一个池化层，使网络卷积层输出具有更高的分辨率，其次没有预测偏移量，而是按照 YOLO 方法直接预测出相对于网格单元位置的位置坐标。YOLO v3 采用了新的网络结构 DarkNet-53 提取特征，采用了多尺度检测的方法，同时改变了对象分类算法进一步提高了目标检测的精度。2020 年发布的 YOLO v4 在目标检测精度及处理速度方面均有大幅度提升。

### 3.3.2　SSD 目标检测算法

SSD（Single Shot MultiBox Detector）获取目标位置和类别与 YOLO 方法类似，而相比于 YOLO 是在整张特征图上划分 7×7 的网格内进行回归，YOLO 对于目标物体的定位并不精准，所以为解决精度问题，SSD 利用类似 Faster R-CNN 推荐区域得分机制实现精准定位。与 Faster R-CNN 的推荐候选框得分机制不同，SSD 在多个特征图上进行处理。Faster R-CNN 首先提取候选框，然后再进行分类，而 SSD 利用得分机制直接进行分类和区域框回归。在保证速度的同时，SSD 检验结果的精度与 Faster R-CNN 相差不多，从而能够满足实时检测与高精度的要求。

如图 3-5 所示，SSD 网络对输入图像进行卷积处理时，针对尺寸为 8×8 或 4×4 特征图的每个位置上评估出不同长宽比的小集合默认框。对于每个默认框，预测对所有对象类别的形状偏移和置信度。在训练时，首先将这些默认框匹配到真实标签区域框中。例如，两个默认框匹配到猫和狗，这些框为正，其余视为负。模型损失是位置损失和置信损失之间的加权和。SSD 方法基于前馈卷积神经网络，产生固定大小的区域框集合和区域框中物体类别的分数，然后使用非极大值抑制算法产生最终预测。

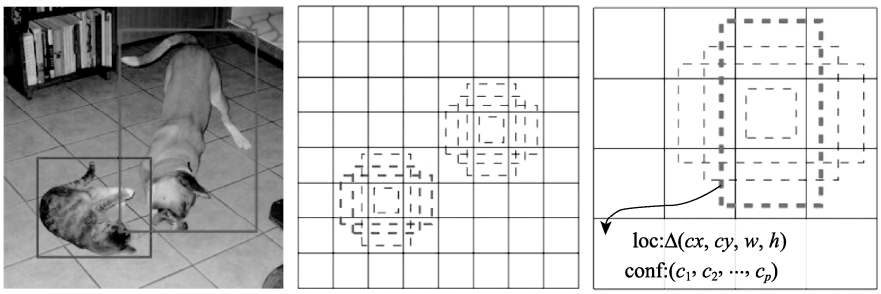


图 3-5　SSD 目标检测过程

# 3.4 目标检测算法评价指标

目标检测需要预测出目标的具体位置以及目标类别，对于一个目标是否检测正确，首先要确定预测类别置信度是否达到阈值，之后确定预测框与实际框的重合度大小是否超过规定阈值。对于多目标检测，分别对每一类进行评价，因此可以由多种目标评价问题转换为多个二分类评价问题。针对重合度的定义，通常采用 IoU（Intersection over Union）来代表。IoU 是指对目标预测框与实际框之间的交集面积与两个框之间并集面积之比，IoU 越大表示预测框与实际框之间重合度越高，检测得越准确，见式（3-1）：

$$\mathrm{IoU}=\frac{\mathrm{area}(B_p \cap B_{gt})}{\mathrm{area}(B_p \cup B_{gt})} \tag{3-1}$$

其中，$B_p$ 为预测框的坐标位置；$B_{gt}$ 表示实际框的位置；area 表示面积。

对于目标检测中某类目标检测算法的最终评价，需要统计在某个阈值条件下，算法对该目标检测的准确率和召回率。准确率（Precision）为对于某个预测类别来说，预测正确的框占所有预测框的比例。而召回率（Recall）为对于某个预测类别来说，预测正确的框占所有真实框的比例。这两个指标的计算方法见式（3-2）和式（3-3）：

$$\mathrm{Precision}=\frac{\mathrm{TP}}{\mathrm{TP+FP}} \tag{3-2}$$

$$\mathrm{Recall}=\frac{\mathrm{TP}}{\mathrm{TP+FN}} \tag{3-3}$$

其中，TP 表示正确预测到的正样本数量；FP 表示错误预测的正样本数量；FN 表示错误预测的负真实样本数量。

以 Recall 值为横轴，Precision 值为纵轴，我们就可以得到 PR 曲线。我们会发现，Precision 与 Recall 的值呈现负相关，在局部区域会上下波动。由于单独使用召回率和准确率无法全面地衡量模型性能，所以一般采用综合上述两个指标的另一个指标 AP。顾名思义 AP 就是平均精准度，简单来说就是对 PR 曲线上的 Precision 值求均值。对于 pr 曲线来说，我们使用积分来进行计算。对于评价多种目标检测算法，采用求多种目标 AP 均值的方法来评价，即 mAP，见式（3-4）~式（3-6）：

$$\mathrm{AP}=\int_0^1 p(r)\,\mathrm{d}r \tag{3-4}$$

在实际应用中，我们并不直接对该 PR 曲线进行计算，而是对 PR 曲线进行平滑处理。即对 PR 曲线上的每个点，Precision 的值取该点右侧最大的 Precision 的值。

$$P_{\mathrm{interp}}(r)=\max_{\tilde{r}:\tilde{r}\geqslant r} p(\tilde{r}) \tag{3-5}$$

在平滑处理的 PR 曲线上，取横轴 0-1 的 10 等分点（包括断点共 11 个点）的 Precision 的值，计算其平均值为最终 AP 的值。

$$\mathrm{AP}=\frac{1}{11}\sum_{r\in\{0,0.1,\cdots,1\}}P_{\mathrm{interp}}(r) \tag{3-6}$$

$$\mathrm{mAP}=\frac{1}{n}\sum_{i\in\{1,2,\cdots,n\}}\mathrm{AP}_i \tag{3-7}$$

## 3.5　深度卷积神经网络目标检测算法性能对比

Girshick 等人设计的 R-CNN、Fast R-CNN 和 Faster R-CNN 等一系列目标检测算法在网络结构上进行不断的改进升级，从而使目标检测在精度和速度上都获得了很大的进步。尤其是 Faster R-CNN 是端到端训练的卷积神经网络架构，其训练和测试的速度相比 R-CNN 提升了数百倍，但是在获取候选区域的计算量仍然很大，无法实现实时的目标检测。与使用候选区域的检测算法相比，YOLO 和 SSD 是非常快速高效的检测算法，YOLO 在目标定位中略有缺陷，但是在其改进版本中解决了这个问题，而 SSD 是一个在检测精度和检测速度上是一个相对均衡的目标检测算法。表 3-1 为几种不同检测算法在检测精度和检测速度方面上的对比，使用的训练数据集是 VOC2007 和 VOC2012，测试数据集为 VOC2007。从表 3-1 中可知在基于候选区域的深度卷积神经网络检测方法中，R-FCN 无论是在检测精度还是检测速度上都领先 Faster R-CNN。而在基于回归方法的深度卷积神经网络目标检测算法中，YOLO v3 在目标检测精度方面表现较好，而 YOLO 在目标检测速度方面表现较好。

**表 3-1　目标检测网络对比**

| 检 测 框 架 | mAP | FPS |
|---|---|---|
| R-FCN | 79.4 | 7 |
| Faster R-CNN | 76.4 | 5 |
| SSD500 | 76.8 | 19 |
| YOLO | 63.4 | 45 |
| YOLO v2 | 78.6 | 40 |
| YOLO v3 | 82.3 | 39 |

**小贴士：可以简单地说明候选区域法和回归法的区别吗？**

答：候选区域法可以认为是两阶段的检测方法，第一阶段专门用来推荐候选区域的，第二阶段判断候选区域中是否有目标及目标的种类是什么？而回归法是一阶段的检测方法，通过一个阶段完成上述的两个功能。从原理上可以看出，回归法的速度快，但候选区域法的定位一般更加准确。可以根据不同的项目需求选择不同的算法。

**小贴士：产业界比较常用的检测方法是什么？为什么？**

答：YOLO 是比较常用的检测方法，截至 2020 年 4 月已经发布了四个版本。YOLO v4 的检测性能最好。YOLO 的作者还提供了 C 语言源码，移植性较好。另外，每个 YOLO 版本里都会提供更小卷积层数的版本 Tiny-YOLO。可以根据不同的项目需求，选择不同的 YOLO 版本。

# 3.6 目标检测项目实战

项目简介：可以识别如图 3-6 所示的图像中 20 类物体。用到的训练集为 VOC 数据集。

图 3-6 VOC 数据集示意图

**代 码 清 单**

## 3.6.1 Faster R-CNN

### 1. 导入第三方库

本实例导入了 Keras 库，keras_frcnn 文件夹里面存放的是实现 Faster R-CNN 所用到的各种类和方法。

```
from __future__ import division
import random
import pprint
import sys
import time
```

```
import numpy as np
import pickle
from keras import backend as K
from keras.optimizers import Adam, SGD, RMSprop
from keras.layers import Input
from keras.models import Model
from keras_frcnn import config, data_generators
from keras_frcnn import losses as losses_fn
import keras_frcnn.roi_helpers as roi_helpers
from keras.utils import generic_utils
import os
from keras_frcnn import resnet as nn
from keras_frcnn.simple_parser import get_data
```

**2. Faster R-CNN 的主干网络——VGG16 的定义**

```
def nn_base(input_tensor=None, trainable=False):
    # 定义适当的输入类型
    if K.image_dim_ordering() == 'th':
        input_shape = (3, None, None)
    else:
        input_shape = (None, None, 3)

    if input_tensor is None:
        img_input = Input(shape=input_shape)
    else:
        if not K.is_keras_tensor(input_tensor):
            img_input = Input(tensor=input_tensor, shape=input_shape)
        else:
            img_input = input_tensor
    if K.image_dim_ordering() == 'tf':
        bn_axis = 3
    else:
        bn_axis = 1
    # Block 1
    x = Conv2D(64, (3, 3), activation='relu', pad-
ding='same',name='block1_conv1')(img_input)
    x = Conv2D(64, (3, 3), activation='relu', padding='same',
name='block1_conv2')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)
    # Block 2
    x = Conv2D(128, (3, 3), activation='relu', padding='same',
name='block2_conv1')(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same',
name='block2_conv2')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)
    # Block 3
    x = Conv2D(256, (3, 3), activation='relu', padding='same',
name='block3_conv1')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same',
name='block3_conv2')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same',
```

```
name='block3_conv3')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)
    # Block 4
    x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block4_conv1')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block4_conv2')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block4_conv3')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)
    # Block 5
    x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block5_conv1')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block5_conv2')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block5_conv3')(x)
    return x
```

**3. RPN 网络的定义**

该网络的输入有以下几项。

（1）base_layers: 也就是前面 VGG 网络的主干网络最后的输出。假设输入主干的图片尺度为 600 × 600 × 3，则该 RPN 输入特征图（Feature map）的 shape 是 38 × 38 × 512。

（2）num_anchors: 这个是值得每个锚点产生的 ROI 的数量。例如，根据论文中锚点（anchors）的尺度为[16, 32, 64]共 3 种，长宽比例为[1∶1, 1∶2, 2∶1]也是 3 种。则 num_anchors=3×3（该值并不固定，可能需要根据具体实验数据以及应用场景做相应的修改）。

网络的输出有以下几项。

（1）x_class: 目标类别

（2）x_regr: bboxes 回归层。bboxes 回归由于是 RCNN 系列的核心部分，所以需要特别说明。

```
def rpn(base_layers, num_anchors):
    x = Conv2D(512, (3, 3), padding='same', activation='relu', ker-
nel_initializer='normal', name='rpn_conv1')(base_layers)
    x_class = Conv2D(num_anchors, (1, 1), activation='sigmoid', ker-
nel_initializer='uniform', name='rpn_out_class')(x)
    x_regr = Conv2D(num_anchors * 4, (1, 1), activation='linear', ker-
nel_initializer='zero', name='rpn_out_regress')(x)
    return [x_class, x_regr, base_layers]
```

**4. 分类部分网络的定义**

网络的输入有以下几项。

（1）base_layer: 也就是前面的 VGG 网络的输出，同样其尺度为（38 × 38 × 512）。

（2）input_rois: 就是 RPN 网络提取的 ROI。

（3）num_rois: 前面 R-CNN 和 Fast R-CNN 提取的 ROI 的数量大约是 2000 个，但是由于 RPN 网络提取的 ROI 是有目的性的，仅仅提取其中不超过 300 个就好。在代码本 Keras 版本的代码中，默认设置的是 32 个，这个参数可以根据实际情况调整。

（4）nb_classes: 指数据集中所有的类别数，VOC 数据集有 20 个前景类别，另外加一个背景，总共 21 类。

网络输出有以下几项。

（1）out_calss: 也就是对应每个 ROI 输出一个包含 21 个类别的输出。

（2）out_regr: 也就是对应每个 ROI 的每个类别有 4 个修正参数。

**5. 整体网络的定义**

```
def classifier(base_layers, input_rois, num_rois, nb_classes = 21,
trainable=False):
    if K.backend() == 'TensorFlow':
        pooling_regions = 7
        input_shape = (num_rois,7,7,512)
    elif K.backend() == 'theano':
        pooling_regions = 7
        input_shape = (num_rois,512,7,7)
    out_roi_pool = RoiPoolingConv(pooling_regions, num_rois)([base_layers,
input_rois])
    out = TimeDistributed(Flatten(name='flatten'))(out_roi_pool)
    out = TimeDistributed(Dense(4096, activation='relu', name='fc1'))(out)
    out = TimeDistributed(Dropout(0.5))(out)
    out = TimeDistributed(Dense(4096, activation='relu', name='fc2'))(out)
    out = TimeDistributed(Dropout(0.5))(out)

    out_class = TimeDistributed(Dense(nb_classes, activation='softmax',
kernel_initializer='zero'),
name='dense_class_{}'.format(nb_classes))(out)
    out_regr = TimeDistributed(Dense(4 * (nb_classes-1), activation= 'lin-
ear', kernel_initializer='zero'),
name='dense_regress_{}'.format(nb_classes))(out)

return [out_class, out_regr]

if K.image_dim_ordering() == 'th':
    input_shape_img = (3, None, None)
else:
    input_shape_img = (None, None, 3)
img_input = Input(shape=input_shape_img)
roi_input = Input(shape=(None, 4))
# 定义基础网络(这里是 ResNet，可以是 VGG, Inception 等)
shared_layers = nn.nn_base(img_input, trainable=True)
# 在基础层上，构建 RPN 网络
num_anchors = len(cfg.anchor_box_scales) * len(cfg.anchor_box_ratios)
rpn = nn.rpn(shared_layers, num_anchors)
classifier=nn.classifier(shared_layers,roi_input,cfg.num_rois, nb_ clas-
ses= len(classes_count), trainable=True)
```

```
model_rpn = Model(img_input, rpn[:2])
model_classifier = Model([img_input, roi_input], classifier)
#这个模型包含 RPN 和分类器，用于为模型加载或保存权重
model_all = Model([img_input, roi_input], rpn[:2] + classifier)

optimizer = Adam(lr=1e-5)
optimizer_classifier = Adam(lr=1e-5)
model_rpn.compile(optimizer=optimizer,
                loss=[losses_fn.rpn_loss_cls(num_anchors),        loss-
es_fn.rpn_loss_regr(num_anchors)])
model_classifier.compile(optimizer=optimizer_classifier,
                         l  oss=[losses_fn.class_loss_cls,        loss-
es_fn.class_loss_regr(len(classes_count) - 1)],
                           metrics={'dense_class_{}'.format(len(classes_
                           count)): 'accuracy'})
model_all.compile(optimizer='sgd', loss='mae')
```

### 3.6.2 用 YOLO 训练自己的模型

Joseph Redmon 等人提供了基于 C 语言编写的 YOLO 系列算法的公开代码，我们可以通过以下方法，完成 YOLO 的训练与测试。本方法默认操作者的使用环境为已经配置好 CUDA 与 CUDNN 的 Ubuntu 系统。

**1. 获取 VOC 数据集**

**2. 在 YOLO 官网下载 YOLOv3 项目**

```
git clone https://github.com/pjreddie/darknet
cd darknet
```

**3. 修改 darknet 目录下的 Makefile**

1）打开文件

```
vi Makefile
```

2）如需使用 GPU 训练，进行如下修改

```
GPU=1  # 使用 GPU 训练
CUDNN=1 #使用 CUDNN
OPENCV=0
OPENMP=0
DEBUG=0
```

3）利用 make 命令编译工程

```
make
```

**4. 准备数据集**

根据自己需要，修改 voc_label.py 中的 sets、classes 和 classes 参数并运行。运行 python voc_label.py，将 VOC 数据转化成 YOLO v3 需要的数据形式。

VOC2007 文件夹中包括 Annotations、ImageSets 和 JPEGImages 三个文件夹。在 ImageSets 下新建 Main 文件夹。文件目录如图 3-7 所示。

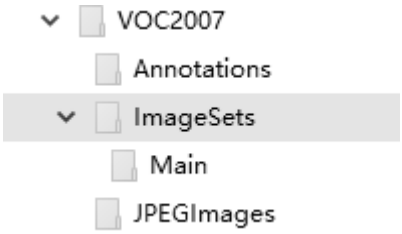


图 3-7　文件目录示例

将自己的数据集图片复制到 JPEGImages 目录下。将数据集 label 文件复制到 Annotations 目录下。在 VOC2007 下新建 MakeFileList.py 文件夹，将下面代码复制进去运行，将生成四个文件：train.txt、val.txt、test.txt 和 trainval.txt。

**数据准备代码清单**

```
import os
import random
trainval_percent = 0.1
train_percent = 0.9
xmlfilepath = 'Annotations'
txtsavepath = 'ImageSets\Main'
total_xml = os.listdir(xmlfilepath)
num = len(total_xml)
list = range(num)
tv = int(num * trainval_percent)
tr = int(tv * train_percent)
trainval = random.sample(list, tv)
train = random.sample(trainval, tr)
ftrainval = open('ImageSets/Main/trainval.txt', 'w')
ftest = open('ImageSets/Main/test.txt', 'w')
ftrain = open('ImageSets/Main/train.txt', 'w')
fval = open('ImageSets/Main/val.txt', 'w')
for i in list:
    name = total_xml[i][:-4] + '\n'
    if i in trainval:
        ftrainval.write(name)
        if i in train:
            ftest.write(name)
        else:
            fval.write(name)
    else:
        ftrain.write(name)

ftrainval.close()
ftrain.close()
```

```
fval.close()
ftest.close()
```

Main 文件夹中的文件分别包括 test.txt（测试集图片路径）、train.txt（训练集图片路径）、val.txt（验证集图片路径）。生成后的目录结构如图 3-8 所示。

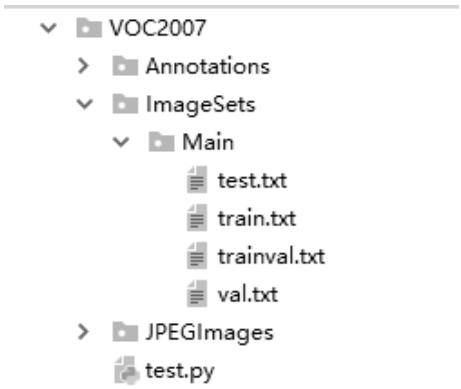


图 3-8　文件目录示例

**5. 修改部分配置文件**

（1）修改 data/voc.name，改成自己所需的类别。

（2）修改 cfg/voc.data。

（3）修改 cfg/yolov3-voc.cfg。

这里需要注意的是如何设置 yolov3-voc.cfg 中的 batch 和 subdivisions 的数值，batch/subdivisions 的值就是每次输入网络进行训练的图片数，batch 和 subdivisions 数值太大会导致内存消耗过高从而导致训练失败。

**6. 下载预训练模型**

```
wget https://pjreddie.com/media/files/darknet53.conv.74
```

将 darknet53.conv.74 放到 scripts 文件夹。

**7. 开始训练**

```
./darknet detector train cfg/voc.data cfg/yolov3-voc.cfg
  scripts/darknet53.conv.74
```

**8. 测试命令**

```
./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
```

这里 yolov3.weights 是训练得到的 YOLO 权重。

## 3.7　本 章 小 结

本章介绍了几种典型的深度学习目标检测算法。深度学习算法主要可以分为以 Faster R-CNN 为代表的基于候选区域的目标检测算法及以 YOLO 为代表的基于回归的目标检测算法。通过项目实例分别介绍两种目标检测算法的构建方法。

## 3.8　习　　题

1. 试用 Python 编程实现 mAP 的计算。
2. 解释非极大值抑制（NMS）算法和 Soft NMS，用 Python 编程实现 NMS 算法。
3. 解释交并比（IoU）的概念。
4. 解释 Faster R-CNN 中 RPN 的作用。
5. 动手实践 YOLO 的训练和测试。