深度学习
计算机视觉实战
卷积神经网络、Python、TensorFlow 和 Kivy

[埃及] 艾哈迈德 • 法齐 • 迦得 (Ahmed Fawzy Gad) 著
林　赐 译

清华大学出版社
北　京

Practical Computer Vision Applications Using Deep Learning with CNNs: With Detailed Examples in Python Using TensorFlow and Kivy

Ahmed Fawzy Gad

EISBN：978-1-4842-4166-0

北京市版权局著作权合同登记号　图字：01-2019-6106

图书在版编目(CIP)数据

深度学习计算机视觉实战：卷积神经网络、Python、TensorFlow 和 Kivy / (埃及)艾哈迈德 • 法齐 • 迦得(Ahmed Fawzy Gad)著；林赐 译. —北京：清华大学出版社，2020.7

书名原文：Practical Computer Vision Applications Using Deep Learning with CNNs: With Detailed Examples in Python Using TensorFlow and Kivy

ISBN 978-7-302-55822-4

Ⅰ. ①深… Ⅱ. ①艾… ②林… Ⅲ. ①计算机视觉 ②机器学习 Ⅳ. ①TP302.7②TP181

中国版本图书馆 CIP 数据核字(2020)第 106142 号

责任编辑：王　军
装帧设计：孔祥峰
责任校对：成凤进
责任印制：沈　露

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：010-62786544
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
印 装 者：三河市吉祥印务有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　**印　　张：**21.25　　**字　　数：**491 千字
版　　次：2020 年 9 月第 1 版　　**印　　次：**2020 年 9 月第 1 次印刷
定　　价：98.00 元

产品编号：084277-01

译 者 序

目前市面上，诸如深度学习与计算机视觉的书籍汗牛充栋，让人眼花缭乱，难以选择。大部分专业书籍公式艰深，理论深奥，极其容易沦为“睡前读物”。本书不希冀成为高大全式的百科全书，而是另辟蹊径，着重实践，以一种“随风潜入夜，润物细无声”的方式，深入浅出地告诉读者使用卷积神经网络进行图像处理的原理。

本书将深度学习应用到不同平台上，使用卷积神经网络(CNN)和 Python 编写计算机视觉应用，解释了传统的机器学习对图像进行处理的原理，强调了传统的人工为计算机视觉选择特征的局限性，顺理成章地提出了使用卷积神经网络自动化处理图像的优点并使用遗传算法优化这个神经网络。

计算机视觉处理博大精深，是一门融图像、数学、统计、编程于一体的科学。学习新理论是一种抽丝剥茧的过程，学习者很难通过一次学习或简单地通过一本书，就能够透彻理解一种理论，熟练掌握一项技术。所谓“博学之，审问之，慎思之，明辨之，笃行之”，《中庸》里的这句名言高度概括了学习的过程。要想完全掌握计算机视觉与 CNN，仅靠阅读本书是远远不够的。要成为 AI 从业人员，光有编码实践，而没有深厚的数学基础和理论知识也是不行的。其结果将是对人工智能一知半解，不能够融会贯通。因此，我建议读者搭配理论书籍，一同阅读理解此书。当理解到达一定深度后，自会有“山重水复疑无路，柳暗花明又一村”的感觉。

在此，特别感谢清华大学出版社的领导和编辑们，感谢他们对我的信任和理解，把这样一本好书交给我翻译。同时我也要感谢他们为本书的翻译投入的巨大热情，可谓呕心沥血。没有他们的耐心和帮助，本书不可能顺利付梓。

林赐

2020 年 4 月 5 日于渥太华大学

作 者 简 介

Ahmed Fawzy Gad 是一名助教，来自埃及，2015 年在埃及梅努菲亚大学计算机与信息学院获得信息技术荣誉理学学士学位，2018 年获得硕士学位。Ahmed 对深度学习、机器学习、计算机视觉和 Python 饶有兴趣。他曾担任机器学习项目的软件工程师和顾问。通过分享著作并在 YouTube 频道上录制教程，为数据科学界添砖加瓦是 Ahmed 的奋斗目标。

Ahmed 发表了多篇研究论文，撰写了 *TensorFlow: A Guide to Build Artificial Neural Networks Using Python* (Lambert，2017)一书。Ahmed 一直希望在其所感兴趣的领域与其他专家分享经验。

技术审校者简介

Leonardo De Marchi 拥有人工智能硕士学位，曾在体育界担任数据科学家，并与纽约尼克斯和曼联等客户合作，还曾与 Justgiving 等大型社交网络合作。

现在，他在 Badoo(全球最大的约会网站，拥有超过 3.6 亿用户)担任首席数据科学家。他还是 ideai.io(一家专门从事深度学习和机器学习培训的公司)的首席讲师并受雇于欧盟委员会。

Lentin Joseph 是来自印度的作家和机器人企业家。他在印度经营着机器人软件公司 Qbotics Labs (http://qboticslabs.com)。

在机器人领域，他拥有 8 年的经验，尤其擅长使用机器人操作系统(ROS)、Open-CV 和 PCL 进行机器人软件开发。

他撰写了 7 本关于 ROS 的书籍：*Learning Robotics Using Python*(两个版本，Packt), *Mastering ROS for Robotics Programming*(两个版本，Packt), *ROS Robotics Projects*(Packt), *Robot Operating System for Absolute Beginners*(Apress)和 *ROS Programming: Building Powerful Robots*(Packt)。

他还审校了三本与机器人技术和 ROS 相关的书籍。第一本是 *Effective Robotics Programming Using ROS*(Packt)，接着是 *Raspberry Pi Image Processing*(Apress)和 *Raspberry Pi Supercomputer*(Apress)。

Lentin 及其团队是 HRATC 2016 挑战赛(ICRA 2016 的一部分)的获胜者。他还是 ICRA 2015 的 HRATC 挑战赛的决赛入围者。

他在印度获得了机器人技术和自动化的硕士学位，曾在卡内基-梅隆大学的机器人学院做过研究。

前　言

人工智能(Artificial Intelligence，AI)是将人类思维嵌入计算机的一个领域。换句话说，就是创建模仿生物大脑功能的人工大脑。现在，人们需要将使用智能可做的所有事情转移到机器中。第一代 AI 专注于人类可以规范描述的问题。进行智能操作的步骤以机器必须遵循的指令形式来描述。机器遵循人类给出的步骤，而不做任何改变。这些是第一代 AI 的特征。

人类可以完整描述一些简单问题，如井字游戏甚至是国际象棋，但无法描述更复杂的问题。在国际象棋中，将棋盘表示为 8×8 大小的矩阵可以简单地解释问题，描述每个棋子及其走法。机器将仅限于完成人类规范描述的那些任务。通过编写此类指令，机器可智能地下棋，此时机器智能是人工的。机器本身不是智能的，但人类能以几条静态代码行的形式将其智能传递给机器。所谓“静态”，这表示在所有情况下机器行为都是相同的。

这种状况下，机器与人类紧密相连，不能单独工作。这就像是主仆关系。人类是主人，机器是仆人，后者仅遵循人类的指令，而不能做其他事情。

将智能行为嵌入代码块并不能处理人类的所有智能行为。一些简单任务(如排序数字)可以由人类描述，然后交由机器处理，它们具有 100%的人类智能。但某些复杂任务，例如语音转文本、图像识别、情感分析等，不能仅通过代码来解决。此类问题无法像国际象棋那样被人类描述。我们不可能编写代码识别猫等图片对象。由于不存在分类物体的单一规则，因此此类识别物体的智能行为不能简单地使用静态代码解决。例如，不存在识别猫的规则。即使某条规则能够成功创建，可以识别某个环境中的猫，但当应用到另一个环境中时，也必然失败。那么，对于此类任务，我们如何使机器拥有智能呢？这就要实现机器学习(Machine Learning，ML)，即由机器学习规则。

为了使机器能够识别对象，我们可以按照机器可理解的方式提供来自专家的先验知识。这种基于知识的系统构成第二代 AI。此类系统的其中一个挑战在于如何处理不确定性和未知知识。人类可以在不同的复杂环境中识别对象，能够智能地处理不确定性和未知知识，但机器不能。

在 ML 中，人类负责完成研究数据的复杂任务，要找出哪些类型的特征能够准确地对对象进行分类。遗憾的是，找到最佳类型的特征是一项艰巨的任务。这是研究人员针对不同应用正尝试进行回答的问题。例如，为诊断疾病，专业人士首先要收集患者和未患病者的数据，对这些数据进行正确标记，然后找到可以明确区分他们的某些类型的特征。这类特征可能是年龄、性别、血糖和血压。数据集越大，人类找到适用于所有样本的特征的难度就越大，因此这是一项非常具有挑战性的任务。

但如今，我们可以训练 ML 模型来了解如何区分不同的类别。ML 算法可以找到合适的数学函数，在输入和输出之间建立最健壮的关系。

机器学习算法并不能解决所有问题。关键的智能仍然存在于人类专家的头脑中，而非机器中。人类收集和标记数据，提取最合适的特征，并选择最佳的 ML 算法。在这之后，机器学习算法仅学习人类所讲的内容。在寻找将输入映射到输出的规则的过程中，机器仍然起着重要的作用。

通常，使用来自特定环境的数据进行训练的 ML 算法不能用在其他环境中。这是一个关键的限制。整个世界存在着大量数据，数据每天都在增加，传统的机器学习技术不适合对其进行处理。例如，使用一组工程化的特征描述图像是非常复杂的，这是因为即使在同一环境中，情况也是富于变化的。我们应该重复这项工作(即特征工程)，以使 ML 算法能够适用于其他环境。

随着类别数量的增加，人类找到好的区别性特征的能力会降低，因此我们不应该依赖于人类，而要将这些工作留给机器。机器本身会尝试探索数据，找到合适的特征，以区分不同的类别。我们仅需要提供数据给机器即可。这就是深度学习(Deep Learning，DL)。人们正趋于使用卷积神经网络(CNN)这个 DL 模型处理大量图像。

DL 领域着重于学习如何从原始数据中得出结论，而无须进行特征工程之类的中间步骤。这就是实际上 DL 可以被称为“自动化特征工程”的原因。它对处理器和内存的要求比较高，可能需要数周的时间区分不同的类别。

本书针对的是未来的数据科学家，这些科学家正逐步开始了解 DL 的基本概念，以用于计算机视觉。读者应该对图像处理和 Python 有一个基本了解。以下是各章内容的概述。

第 1 章基于计算机视觉中一些常用特征描述子的介绍，选择了最适合的特征集对 Fruits 360 数据集进行分类。此类特征使用 Python 实现。通过在预处理步骤中过滤此类特征，可以使用最少数目的特征进行分类。该章得出的结论是，传统的手工特征不适用于复杂问题。DL 是处理大量样本及类别的替代方法。

第 2 章讨论人工神经网络(Artificial Neural Network，ANN)，这是 DL 模型的基础。它首先解释了 ANN 仅是线性模型的组合。我们通过指定最佳层数和神经元为一些简单示例设计 ANN 架构。基于数值示例和 Python 示例，我们可以清楚地知道 ANN 如何进行前向和后向传递。

第 3 章使用第 2 章中的特征集实现 ANN，对 Fruits 360 数据集的子集进行

分类。由于在实现过程中未使用任何优化技术，因此分类正确率较低。

第 4 章简单介绍单目标和多目标优化技术。它使用基于随机技术的遗传算法优化 ANN 权重。这将分类正确率提高到 97%以上。

第 5 章讨论识别多维信号的 CNN。该章从强调全连接神经网络(FCNN)和 CNN 之间的差别以及 CNN 如何从 FCNN 中衍生出来开始。基于数值示例，CNN 中的两个基本操作(即卷积和池化)的概念将逐渐变得清晰。我们可以使用 NumPy 实现 CNN 层，从而详细了解 CNN 如何工作。

第 6 章介绍 DL 库 TensorFlow，我们使用这个库构建用于并行和分布式处理大量数据的 DL 模型。通过构建简单的线性模型和模拟 XOR 门的 ANN 等示例，我们讨论了 TensorFlow 占位符、变量、数据流图和 TensorBoard。在该章结束时，使用 tensorflow.nn 模块创建了 CNN，用于分类 CIFAR10 数据集。

第 7 章将训练过的模型部署到 Web 服务器上，以便让互联网用户使用 Web 浏览器进行访问。我们使用 Flask 微框架创建 Web 应用，使用 HTML、CSS 和 JavaScript 构建前端页面来访问 Web 服务器。HTML 页面发送带有一张图像的 HTTP 请求到服务器，服务器使用预测类别对请求进行响应。

第 8 章使用 Kivy 开源库构建跨平台应用。通过将 Kivy 链接到 NumPy，我们可以构建数据科学应用，不作任何改变地在不同平台上进行工作。这消除了对特定平台自定义代码的开销。我们创建了一个 Android 应用，它读取图像并执行第 5 章中使用 NumPy 实现的 CNN。

为了让他人从所创建的项目中受益，我们可以将其在线发布。附录 A 讨论了如何打包 Python 项目，将它们发布到 Python 包索引(PyPI)仓库。

在开始学习之前，让我们先简单了解本书中使用的 Python 环境。

本书中的所有代码都使用 Python 实现。由于使用原生 Python 处理图像的工作非常复杂，因此我们在各章中使用了多个库帮助生成高效的实现。

首先，我们可以从 www.python.org/downloads 这个链接下载原生 Python 代码。本书使用 Python 3(请安装适用于你的系统的 Python 版本)。下一步是准备本书需要的所有库。我们不建议单独安装各个库，而推荐使用 Anaconda Python 发行版(可以通过链接 www.anaconda.com/download 进行下载)。它支持 Windows、Mac 和 Linux 并打包了超过 1400 个的数据科学库。我们可以从 https://repo.anaconda. com/pkgs 这个页面访问所有受支持的软件包的列表。只要在计算机上安装了 Anaconda，所有支持的库就都可以使用。这有助于快速准备好 Python 环境。

本书所需的库为 NumPy、SciPy、Matplotlib、scikit-image、scikit-learn、TensorFlow、Flask、werkzeug、Jinja、pickle、Pillow 和 Kivy。除了 Kivy 外，Anaconda 支持所有这些库。在本书的各个章节中，我们可以看到各个库的函数。注意，此类库很容易安装。在安装了原生 Python 后，可使用 pip 安装程序下载并安装库，所基于的指令为 pip install <lib-name>。我们仅需要输入库的名称。有些安装并不简单，可能随着系统的改变而改变。因此，我们无法涵盖不同的

安装。出于这个原因，比起分别安装各个库，Anaconda 是更好的选择。让我们继续讨论所需的库。

Python 支持许多内置的数据结构：列表、元组、字典、集合和字符串。但在数据科学应用中，没有一种数据结构可提供灵活性。

这些数据结构支持同时使用不同的数据类型工作。相同的数据结构可能包含数字、字符、对象等。字符串是个例外，它只支持字符。此外，字符串和元组是不可变的，这意味着在创建后，我们不可能改变它们的值。使用字典保存图像像素要求为每个像素添加键，但这会增大所保存数据的量。集合仅限于集合操作，而图像不限于此类操作。

谈到图像，这是本书的主要内容，列表是合适的数据结构。这是一种可变的数据类型，能够存放矩阵。不过，使用列表会使过程变得复杂。由于不同的数值数据类型可以保存在相同的列表中，因此我们必须确定每一项都是某一特定类型的数值类型。为应用简单的操作(如将一个数字加到图像上)，我们必须写一个循环访问每个元素，单独地应用此种操作。在数据科学应用中，我们推荐使用工具让操作变得简单。在构建应用时，我们需要克服一些富有挑战性的任务，而在编写此类任务时，我们没有必要增加另一个挑战。

出于这种原因，我们使用了 NumPy 库。它的基本作用是在 Python 语言中支持一种新的数据结构，即数组。使用 NumPy 数组比使用列表简单。例如，在将图像转换为 NumPy 数组后，仅使用加法操作，我们就可以将某个数字添加到图像中的每个元素上。许多其他的库也有相应函数接收并返回 NumPy 数组。

虽然在 NumPy 数组内部支持某些操作，但这不意味着要应用这些操作。SciPy 库支持 NumPy 数组中的相同操作。它也支持使用 scipy.ndimage 子模块处理 *n* 维 NumPy 数组(例如图像)。对于有关图像的更多高级操作，我们使用 scikit-image 库。例如，使用此库可以提取图像特征。

在读取图像并应用一些操作后，我们使用 Matplotlib 显示图像。我们仅将其用于 2D 可视化，但是它也支持一些 3D 特征。

在读取图像、提取特征并进行可视化后，我们可以使用 scikit-learn 库构建 ML 模型。它支持备用的不同类型的模型。只需要提供输入、输出及其参数，即可获得已训练的模型。

训练完 ML 模型后，可使用 pickle 库将其保存，供以后使用。pickle 库可以序列化和反序列化对象。此时，我们可构建和保存 ML 模型。然后，我们可以转向使用 TensorFlow 构建和保存 DL 模型。这是最常使用的 DL 库，它支持不同的 API，可同时满足专业人士和初学者的需求。TensorFlow 用自己的方式保存已训练的模型。

Flask 是用于构建 Web 应用程序的微框架，我们使用它部署已训练的模型。通过将已训练的模型部署到 Web 服务器上，客户端可以使用 Web 浏览器进行访问。它们可以上传测试图像到服务器，接收类别标签。Flask 使用 Jinja2 模板引擎和 WSGI 构建应用。因此，我们必须安装 Jinja 和 werkzeug 库。

为构建可以在设备上运行的数据科学移动应用，我们要使用 Kivy。这是支持 Python 代码跨平台运行的 Python 库。在本书中，我们使用 Kivy 构建适用于 Android 设备的丰富数据科学应用。在市面上，我们可以使用 Kivy 生成的 APK，它与通常使用 Android Studio 创建的 APK 一模一样。

Kivy 使用 python-for-android 打包器，它允许添加所需的依赖包到 Android 应用程序中。由于 scikit-image 不支持 python-for-android，因此我们使用 Pillow 读取图像，这个库支持在 Android 设备上运行。

目　录

第 1 章

计算机视觉识别

大多数计算机科学研究都在尝试建造一个类似于人类的机器人，使其能够充当人类。此类机器人甚至有可能具有情绪属性。通过传感器，机器人能感知周围的环境温度。通过面部表情，机器人能够知道一个人是悲伤还是快乐。现在看起来不可能的事情在未来可能只是一种挑战。

目前，非常具有挑战性的一种应用是物体识别。它是指可以基于不同类型的数据(如音频、图像和文本)进行识别。由于图像信息的丰富性有助于任务的完成，因此图像识别是一种非常有效的方式。因而，它被认为是最流行的计算机视觉应用。

世界上存在大量的物体，区分它们是一项复杂的任务。不同的物体可能具有相似的外观，而仅有微妙的细节差别。此外，根据物体的周边环境，同一个物体也会有不同的表现。例如，根据光、视角、畸变和阻塞，相同的物体在图片上具有不同的表现。对于图像识别而言，由于每个像素的微小变化会导致图像中的巨大变化，从而导致系统不能正确地识别物体，因此像素可能不是一个好选择，但这取决于原生图片。我们的目标是找到一组独特属性或特征，只要物体的结构出现在图片中的某处，这组属性或特征就不会随着像素位置或值的改变而改变。手动从图片中提取特征是图像识别面临的一个重大挑战，这正是特征提取的自动方法替代像素方法的原因所在。

当前，由于在任何环境中对物体进行识别都是非常复杂的，因此替代方法是限制目标环境或目标物体。例如，我们可以只针对一组动物，而不是识别所有类型的动物。我们可以将环境局限于室内图片，而不是室内外的图片。我们仅使用一些视图进行物体识别，而不是在不同的视图中进行物体识别。一般而言，在较小领域内创建人工智能应用虽然具有挑战性，但它比通用的人工智能应用更简单，困难也较少。

本章将讨论如何构建对水果图像进行分类的识别应用。首先介绍一些类型的特征(一般来说，这些特征对不同类型的应用都有用)，然后找出此类特征的最佳特征，用于目标应用。在本章结束前，我们将明白为什么手动提取特征具有挑战性，以及为什么首选使用卷积神经网络(Convolutional Neural Network，CNN)进行自动特征挖掘。

1.1　图像识别步骤

类似于大多数传统的识别应用，图像识别可能要遵循一些预定义的步骤，接收输入，返回所期望的结果。图 1-1 总结了此类步骤。

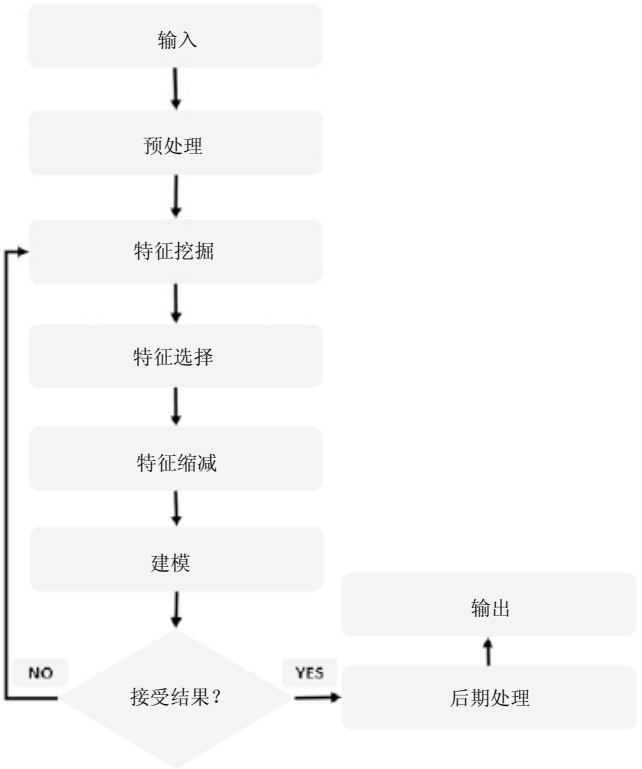


图 1-1　总体识别步骤

有时，所输入的图像不适合以当前的格式进行处理。例如，如果要创建一个面部识别应用，在复杂的环境中捕捉图像，识别图像中的人，那么最好在开始识别目标物体之前移除背景。这种情况下，背景移除是一种预处理。一般来说，在实际工作之前的任何步骤都被称为预处理。预处理是最大限度地提高成功识别概率的一个步骤。

输入图像准备就绪后，我们就完成实际工作，开始进行特征提取或特征挖掘。在大部分识别应用中，这是关键步骤。目标是找出能够精确描述每个输入图像的一组代表性特征。这样一组特征应该可以最大化每个输入图像映射到其正确输出的概率，最小化分配给每个输入图像错误标签的概率。所得到的结果就是可以对要使用的特征类型进行分析。

应用与所使用的特征集息息相关。可以基于应用选择特征。通过理解应用的本

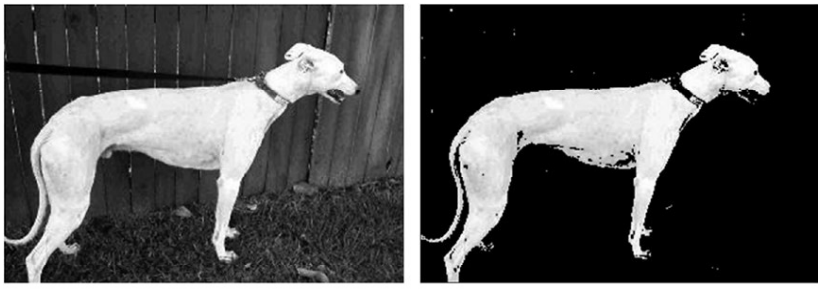
质，可以很容易地检测出所需要的特征类型。对于如人脸检测的应用，所要提取的特征是什么？人脸具有不同颜色的皮肤，因此我们可以确定肤色是所要使用的特征。了解到应用是在室外灰度图像、较暗和移动的环境中检测人脸有助于选择合适的特征。如果被要求创建应用识别橙子和香蕉，那么你可以受益于橙子和香蕉具有不同的颜色这个事实，仅确定颜色特征就足以识别二者。但这不足以识别不同类型的皮肤癌。我们需要做更多的工作，找出最适合的特征集。1.2 节将讨论有助于图像识别应用的一些特征。

在创建特征向量并获得在识别应用中很可能有用的特征后，我们要通过其他步骤以进一步增强，即进行特征选择和缩减。我们将特征选择和缩减的主要目标定义为从特征集中获得最优特征子集，通过减小不相关的特征、相关联的特征和噪声特征增强学习算法性能或精度。1.3 节将讨论通过移除不相关、相关联和噪声特征缩减特征向量长度的方法。

1.2 特征提取

将原始格式的图片作为训练模型的输入并不常见。我们有不同理由证明提取特征是一种更好的方式。其中一个理由是，即使是小图片，也具有大量像素。如果每个像素都作为模型的输入，那么对于尺寸为 100×100 像素的灰度图像而言，就有 100×100=10 000 个输入变量输入模型中。对于 100 个样本的小数据集而言，整个数据集共有 100×10 000=1 000 000 个输入。如果图片是 RGB 格式，那么要将总数乘以 3。这需要大量的内存，此外也是计算密集型的。

在训练前，倾向于进行特征提取的另一个原因是输入图像中有不同属性且不同类型的物体，而我们只想针对单个物体。例如，图 1-2(a)显示了来自“狗与猫 Kaggle”竞赛的狗图像。我们的目标是检测狗，并不关心木头和草。如果将完整的图像用作模型的输入，木头和草将影响结果。只使用专属于狗的特征是更好的做法。很明显，图 1-2(b)显示了狗的颜色与图像中的其他颜色不同。



(a) (b)

图 1-2 在使用特征的情况下来针对图像内的特定物体相对容易

一般来说，对问题的成功建模与最佳特征的选择紧密相关。数据科学家应选择

最有代表性的特征集用于问题求解。可以使用不同类型的特征描述图像。这些特征可以不同方式进行分类。一种方式是检测是全局还是局部(从图像的特定区域)提取这些特征。局部特征指边和关键点等特征。全局特征指颜色直方图或像素数特征。所谓全局，意味着特征描述整张图像。例如，颜色直方图居于左侧区域中间表明整张图片偏暗。这种描述并不只针对图片的特定区域。局部特征则聚焦于图像内的特定内容，如边。

下面将讨论以下特征：

- 颜色直方图
- 边缘
  - HOG
- 纹理
  - GLCM
  - GLGCM
  - LBP

1.2.1　颜色直方图

颜色直方图表示的是整幅图上颜色的分布。它通常用于灰度图像，但是也可以进行修改，将其与彩色图像一起使用。为简单起见，我们计算图 1-3 中的 5×5 的 2 位图像的颜色直方图。这幅图像是使用 NumPy 随机生成的，仅有 4 个灰度等级。

3	2	2	0	3
1	3	0	2	2
2	2	2	2	3
3	3	3	2	3
0	2	3	2	2

图 1-3　尺寸为 5×5 的 2 位灰度图像

通过计算每个灰度等级的频度，所得到的直方图如图 1-4 所示。根据直方图，可以很明显地看到高频率的柱状区间位于右边，因此这幅图大部分的像素的值较高，图像较亮。

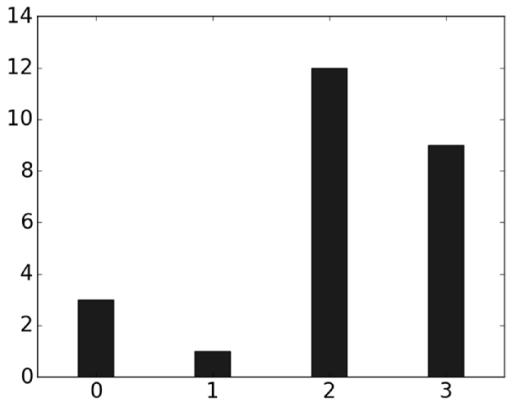


图 1-4　2 位 5×5 灰度图像的直方图

代码清单 1-1 给出了 Python 代码，用于随机生成先前的微小图像，同时计算并显示直方图。

代码清单 1-1　随机生成微小图像的直方图

```
import matplotlib.pyplot
import numpy

rand_img = numpy.random.uniform(low=0, high=3, size=(5,5))
rand_img = numpy.uint8(rand_img)
hist = numpy.histogram(rand_img, bins=4)
matplotlib.pyplot.bar(left=[0,1,2,3], height=hist[0], align="center",
width=0.3)
matplotlib.pyplot.xticks([0,1,2,3], fontsize=20)
matplotlib.pyplot.yticks(numpy.arange(0, 15, 2), fontsize=20)
```

numpy.random.uniform()接收要返回的数组的大小，以及图像像素所分配值区间的上下限。由于我们要创建 2 位的图像，因此下限为 0，上限为 3。numpy.uint8()用于将值从浮点值转换为整数值。然后，使用 numpy.histogram()计算直方图，这个函数接收图像和柱状区间的数目，返回每个等级的频率。最终使用 matplotlib.pyplot.bar()返回柱状图，在 *x* 轴上显示每个等级，在 *y* 轴上显示频率。matplotlib.pyplot.xticks()和 matplotlib.pyplot.yticks()用于改变 *x* 轴和 *y* 轴的范围，以及所显示的字体大小。

1. 真实世界图像的直方图

让我们计算图 1-2(a)所示的真实世界图像在转换为黑白图像后的直方图，如图 1-5 所示。直方图看起来大部分集中在左侧，这意味着图片整体较暗。由于狗的身体是白色的，因此直方图的某些部分位于柱状图分布的最右侧。

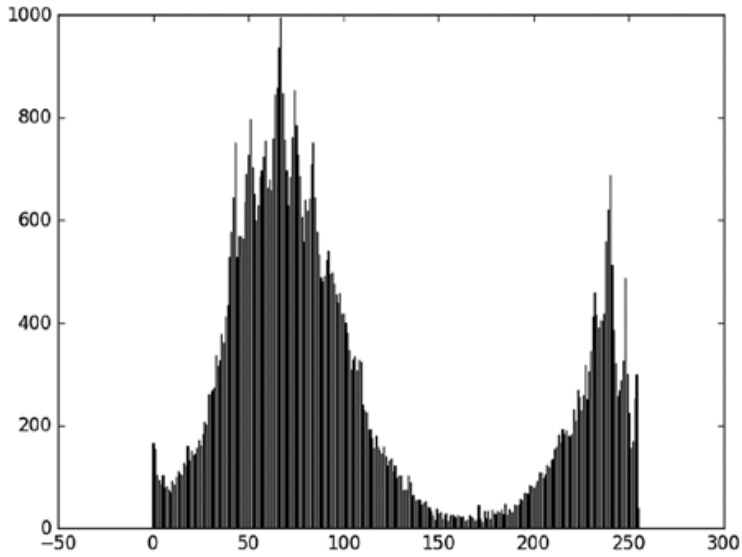


图 1-5　灰度图像的直方图

代码清单 1-2 给出了 Python 代码，用于读取彩色图像，将其转换为灰度图，计算其直方图，最后将直方图绘制成柱状图。

代码清单 1-2　真实世界图像的直方图

```
import matplotlib.pyplot
import numpy
import skimage.io

im = skimage.io.imread("69.jpg", as_grey=True)
im = numpy.uint8(im*255)

hist = numpy.histogram(im, bins=256)
matplotlib.pyplot.bar(left=numpy.arange(256), height= hist[0],
align="center", width=0.1)
```

通过 skimage.io.imread()函数，使用 as_grey 属性读取图像并将其转换为灰度图。当 as_grey 设置为 True 时，返回的图像为灰度图。所返回的图像数据类型为 float64。使用 numpy.uint8()将其转换为 0~255 的无符号整数。由于 numpy.uint8()不重新调整输入，因此在转换之前先将图像中的每个像素值乘以 255，这就可以保证数字为使用 8 位表示的整数。例如，应用数字 0.7，结果为 0。我们希望重新调整 0.4，从 0~1 的区间转换为 0~255 的区间，然后将其转换为 uint8。如果不将输入乘以 255，所有的值要么为 0，要么为 1。注意，由于图像使用了 8 个位表示，因此将直方图柱状

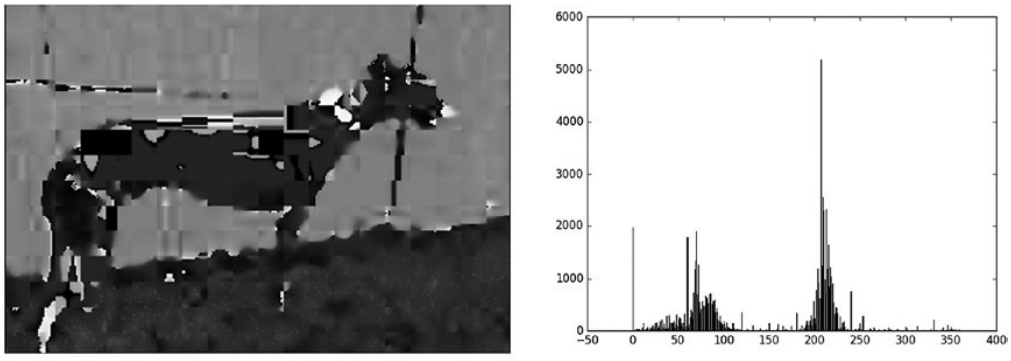
区间的数目设置为 256，而不是先前示例中的 4。

2. HSV 色彩空间

彩色直方图意味着图像像素为色彩空间中的一个像素，然后计数在此彩色空间中存在的等级的频率。此前，图像使用 RGB 色彩空间表示，每个通道的区间为 0~255。但这并不是仅有的色彩空间。

我们将介绍的另一种色彩空间为 HSV(色调-饱和度-值)。此种色彩空间的优点是将色彩信息和亮度信息分离。色调通道持有色彩信息，其他通道(饱和度和值)指定了色彩的亮度。针对色彩而不是亮度创建亮度不变的特征是非常有用的。在本书中，我们不讨论 HSV 色彩空间，但是扩展阅读如何使用 HSV 生成色彩也是不错的。值得一提的是，色调通道表示为圆形，其值的范围为 0~360，其中 0 表示红色，120 表示绿色，240 表示蓝色，360 又回到表示红色。因此，这个色调通道始于红色，终于红色。

对于图 1-2(a)中所述的图像，其色调通道和直方图如图 1-6 所示。当色调通道使用灰度图像表示时，如图 1-6(a)所示，红色会得到较高的值(白色)，如狗的项圈所示。由于蓝色也会获得较高的色调值 240，因此在灰度图像中，它显得较浅。具有色调值 140 的绿色相较于 360 更接近于 0，因此它的颜色较暗。注意，狗的身体使用 RGB 色彩空间表示为白色，在 HSV 空间中看起来是黑色的。这个原因在于，HSV 不关注强度，仅关注颜色。在 V 通道中，这将表示为白色。



(a) HSV 色调通道　　(b) 直方图

图 1-6　使用 HSV 表示的彩色图像色调通道及其直方图

根据代码清单 1-3，将 RGB 的图像转换为 HSV 色彩空间并显示其色调通道的直方图。

代码清单 1-3　使用 Matplotlib 显示图像直方图

```
import matplotlib.pyplot
import numpy
import skimage.io
```

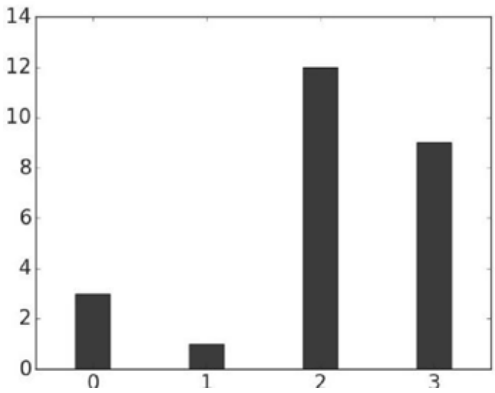
```
import skimage.color
im = skimage.io.imread("69.jpg", as_grey=False)

im_HSV = skimage.color.rgb2hsv(im)
Hue = im_HSV[:, :, 0]

hist = numpy.histogram(Hue, bins=360)
matplotlib.pyplot.bar(left=numpy.arange(360), height=hist[0],
align="center", width=0.1)
```

由于色调通道是 HSV 色彩空间中的第一个通道，因此它被赋予索引 0 并返回。

我们期望不同图像的特征是独一无二的。如果不同的图像具有相同的特征，则会造成结果不准确。颜色直方图就具有这样一个缺点，因为不同图像的颜色直方图可能完全相同。其原因是，颜色直方图只是计数色彩的频率，而不关注它们的排列。图 1-7(a)转置了图 1-3 中的图像。如图 1-7(b)所示，尽管像素的位置不同，转置前后的图像直方图是完全相同的。

3	2	2	0	3
1	3	0	2	2
2	2	2	2	3
3	3	3	2	3
0	2	3	2	2

(a) 转置的图像



(b) 直方图

图 1-7　更改像素位置不改变彩色直方图

即使对图像进行更改(如旋转和缩放)，一个良好的特征描述子也应该保持持续性，因此有人可能会认为这不是一个问题。颜色直方图不能满足这种属性，因为即使图片完全改变了，依然返回相同的颜色直方图。为解决这个问题，我们期望像素强度和位置可返回一个更具代表性的特征。此类特征的示例是纹理特征，如 GLCM。

1.2.2 GLCM

其中一种流行的统计纹理分析方法依赖于从像素对之间的空间关系中提取的二阶统计。最流行的此类特征是从共生矩阵(CM)中提取出来的。其中一种共生矩阵为灰度共生矩阵(GLCM)。顾名思义，这种方法的输入为灰度图像，输出为返回的 GLCM 矩阵。

我们可将 GLCM 描述为二维直方图，根据每个灰度对等级之间的距离计数它们之间的共生个数。GLCM 和一阶直方图不同的是，GLCM 不只依赖于强度，也依赖于像素的空间关系。对于每两个像素，一个称之为参考像素，另一个称之为邻居像素。当两个强度等级之间的距离为 D 且角度为 θ 时，GLCM 可找到两个强度等级共生的次数。$GLCM_{(1,3),D=1,\ \theta=0^\circ}$ 指的是当强度值为 1 的参考像素与强度为 3 的邻居像素之间的分隔距离为 D=1 且角度 $\theta=0^\circ$ 时共生的次数。当 $\theta=0^\circ$ 时，这意味着它们处在同一水平线上。 θ 指定了方向，D 指定了在此方向上的距离。注意，参考像素存在于邻居像素的左侧。

计算 GLCM 的步骤如下所示：

(1) 如果输入图像为灰度格式或二进制格式，那么可以直接使用。如果这是彩色图像，则将其转换为灰度图像或者(如果合适)仅使用其中一个通道。

(2) 找出此图像中的强度等级总数。如果数目为 L，将这些等级从 0 到 L−1 进行计数。

(3) 创建 $L\times L$ 矩阵，其中将行和列从 0 到 L−1 进行计数。

(4) 选择适当的 GLCM 参数(D、θ)。

(5) 找出每两对强度等级之间的共生的次数。

1. D 值

调查研究表明，最好的 D 值范围为 1~10。较大的值将使得 GLCM 不能捕捉到详细的纹理信息。因此，对于 D=1,2,4,8 来说，结果是精确的；对于 D=1,2 来说是最好的。通常情况下，一个像素与邻近它的像素更相关。比起较大的距离，缩短距离可取得更好的结果。

2. θ 值

对于一个 3×3 的矩阵，中心像素具有 8 个相邻像素。在此中心像素与所有其他 8 个像素之间，θ 有 8 个可能的值，如图 1-8 所示。

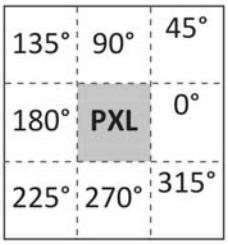


图 1-8　中心像素及其 8 个相邻像素之间的 θ 值

由于将 θ 设定为 0° 和 180° 时共生对是相同的(例如 $GLCM_{(1,3),\theta=0^\circ}=GLCM_{(3,1),\theta=180^\circ}$)，因此用一个角度就足够了。总体说来，相隔 180° 的两个角度值返回的是相同的结果。这可以应用在角度(45°，225°)，(135°，315°)和(90°，270°)上。

当 D=1° 且 θ=0° 时，让我们从对先前图 1-3 中的矩阵计算 GLCM 开始，然后为下面的矩阵重复相同的过程。由于图像具有 4 个强度等级，因此当参考强度为 0 时，可用的对为(0,0)、(0,1)、(0,2)和(0,3)。当参考强度为 1 时，可用的对为(1,0)、(1,1)、(1,2)和(1,3)。对于 2 和 3，也是如此。

3	2	2	0	3
1	3	0	2	2
2	2	2	2	3
3	3	3	2	3
0	2	3	2	2

如果计算 $GLCM_{(0,0),D=1,\theta=0^\circ}$，则值应该为 0。这是因为对于强度为 0 的像素，在水平上没有另一个强度为 0 的像素距离为 1 个像素。对于(0,1)、(1,0)、(1,1)、(1,2)、(2,1)和(3,1)对，结果也为 0。

对于 $GLCM_{(0,2),D=1,\theta=0^\circ}$，则结果为 2，这是因为在水平方向上，强度为 3 的像素与强度为 0 的像素(即 θ=0°)的距离为 1 个像素，这种像素出现了 3 次。对于 $GLCM_{(3,3),D=1,\theta=0^\circ}$，结果也为 2。对于 $GLCM_{(0,3),D=1,\theta=0^\circ}$，结果为 1，这是因为在距离为 1 且角度为 0 的情况下，强度为 3 的像素与强度为 0 的像素对只出现一次。这位于原矩阵的右上方。

得到的完整 GLCM 如图 1-9 所示。这个矩阵的大小为 4×4，因为它具有 4 个强度等级，编号从 0 到 3。我们添加了行和列的标签到矩阵中，这样更容易了解哪个强度等级与另一个强度等级共生。

	0	1	2	3
0	0	0	2	1
1	0	0	0	1
2	1	0	6	3
3	1	0	3	2

图 1-9　图 1-3 中矩阵的 GLCM，其中距离为 1，角度为 0

代码清单 1-4 中给出了返回先前 GLCM 所使用的 Python 代码。

代码清单 1-4　GLCM 矩阵计算

```
import numpy
import skimage.feature

arr = numpy.array([[3, 2, 2, 0, 3],
                   [1, 3, 0, 2, 2],
                   [2, 2, 2, 2, 3],
                   [3, 3, 3, 2, 3],
                   [0, 2, 3, 2, 2]])

co_mat = skimage.feature.greycomatrix(image=arr, distances=[1],
angles=[0],levels=4)
```

skimage.feature.greycomatrix()用于计算 GLCM。这个方法接收输入图像、用于矩阵计算的距离和角度、所使用的等级数目。等级数目是最重要的，默认值为 256。

注意，对于每对特定的角度和距离，都有一个矩阵。由于仅使用一个距离和角度，因此仅返回单个 GLCM 矩阵。所返回的输出形状有 4 个数字，如下所示：

```
co_mat.shape = (4, 4, 1, 1)
```

前两个数字分别表示行数和列数。第三个数表示所使用距离的数目。最后一个数字表示角度的数目。如果要基于更多距离和角度对矩阵进行计算，那么在 skimage.feature.greycomatrix()中指定它们。下一行代码使用两个距离和三个角度计算 GLCM。

```
co_mat = skimage.feature.greycomatrix(image=arr, distances=[1, 4],
angles=[0, 45, 90], levels=4)
```

返回的矩阵的形状为：

```
co_mat.shape = (4, 4, 2, 3)
```

由于有两个距离和三个角度，因此返回的 GLCM 总数为 2×3=6。为返回距离为 1 且角度为 0 的 GLCM，索引如下所示：

```
co_mat[:, :, 0, 0]
```

这返回了完整的 4×4 的 GLCM，但是根据其在 skimage.feature.greycomatrix()函数中的顺序，这仅是第一个距离(1)和第一个角度(0)返回的 GLCM。为返回对应于距离为 4 且角度为 90 的 GLCM，索引如下所示：

```
co_mat[:, :, 1, 2]
```

3. GLCM 归一化

先前计算的 GLCM 有助于学习每个强度等级与其他强度等级共生的次数。我们可以从这样的信息中获益，预测任意两个强度等级共生的概率。GLCM 可以转换为概率矩阵，因此我们可以知道分隔距离为 D 且角度为 θ 的任意两个强度等级 l_1 和 l_2 共生的概率。将矩阵中的每个元素除以矩阵元素的总和可以得到这个概率。我们称所得到的矩阵为归一化矩阵或概率矩阵。基于图 1-9，所有元素的总和为 20。将每个元素除以这个数后，归一化矩阵如图 1-10 所示。

	0	1	2	3
0	0.0	0.0	0.1	0.05
1	0.0	0.0	0.0	0.05
2	0.05	0.0	0.3	0.15
3	0.05	0.0	0.15	0.1

图 1-10　距离为 1 且角度为 0 的归一化 GLCM 矩阵

归一化灰度共生矩阵的其中一个益处是输出矩阵中的所有元素都处在同一个范围内(从 0.0 到 1.0)。此外，结果与图像大小无关。例如，根据图 1-9 的 5×5 大小的图像，对于(2,2)对，最高频率为 6。如果新的图像较大(例如 100×100)，则最高的频率不会是 6，而是更大的一个值(如 2000)。由于这个数值与图像大小相关，因此我们不能比较 6 与 2000。通过归一化矩阵，灰度共生矩阵(GLCM)与图像大小无关，因此我们可以正确地比较它们。在图 1-10 中，(2,2)对的概率为 0.3，这可以与任意大小的任意图像的共生概率进行比较。

使用 Python 归一化 GLCM 非常简单。根据称为 normed 的布尔参数，如果将这个参数设置为 True，那么结果就是归一化的。这个参数默认设置为 False。根据下列这行代码计算归一化矩阵：

```
co_mat_normed = skimage.feature.greycomatrix(image=arr, distances=[1],
angles=[0], levels=4, normed=True)
```

由于我们使用 2 位的图像，仅有 4 个等级，因此灰度共生矩阵的大小为 4×4。对于图 1-2(a)所示的 8 位灰度图像，有 256 个等级，因此矩阵大小为 256×256。归一化 GLCM 如图 1-11 所示。在两个区域内概率较大。由于背景具有较深的颜色，因此第一个区域在左上(强度较低)。由于狗的身体颜色为白色，因此另一个区域在右下(强度较高)。

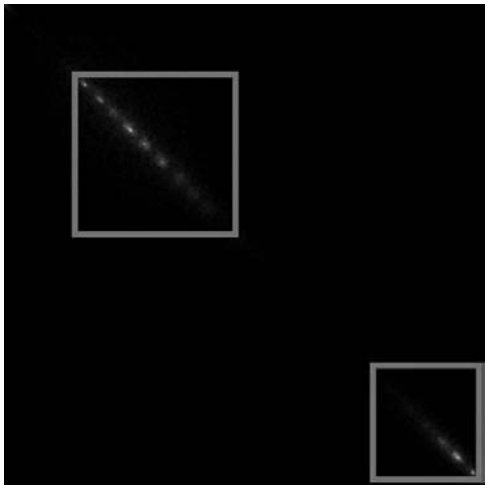


图 1-11　距离为 6 且角度为 0 的具有 256 个等级的灰度图像的灰度共生矩阵

随着等级数目的增加，矩阵的大小也在增加。如图 1-11 所示，GLCM 具有 256×256=65 536 个元素。在特征向量中使用矩阵中的所有元素将会极大地增加其长度。我们可以通过从矩阵中提取特征减小此数字，包括相异度、相关性、均匀性、能量、对比度和 ASM(角二阶矩)。代码清单 1-5 给出了提取此类特征所需的 Python 代码。

代码清单 1-5　提取灰度共生矩阵的特征

```
import skimage.io, skimage.feature
import numpy

img = skimage.io.imread('im.jpg', as_grey=True);

img = numpy.uint8(img*255)
```

```
glcm = skimage.feature.greycomatrix(img, distances=[6], angles=[0],
levels=256, normed=True)

dissimilarity = skimage.feature.greycoprops(P=glcm, prop='dissimilarity')
correlation = skimage.feature.greycoprops(P=glcm, prop='correlation')
homogeneity = skimage.feature.greycoprops(P=glcm, prop='homogeneity')
energy = skimage.feature.greycoprops(P=glcm, prop='energy')
contrast = skimage.feature.greycoprops(P=glcm, prop='contrast')
ASM = skimage.feature.greycoprops(P=glcm, prop='ASM')

glcm_props = [dissimilarity, correlation, homogeneity, energy,
contrast, ASM]

print('Dissimilarity',dissimilarity,'\nCorrelation',correlation,
'\nHomogeneity',homogeneity,'\nEnergy',energy,'\nContrast',contrast,
'\nASM',ASM)
```

灰度共生矩阵的其中一个缺点是依赖于灰度值。只在亮度上的微小改变也会影响所得到的灰度共生矩阵。其中一个解决方法是使用梯度而不是强度构建共生矩阵。这种矩阵称为基于梯度的灰度共生矩阵(GLGCM)。通过使用梯度，GLGCM 对于亮度变化可以保持不变。

无论是 GLCM 还是 GLGCM，都会随着图像的变换而改变。也就是说，如果相同灰度图像受到转换的影响(如旋转)，那么描述子会获得不同的特征。好的特征描述子应该不随着此类效果而变化。

1.2.3　HOG

我们可使用 GLCM 描述图像纹理，但不能用它描述图像强度的突然变化(如边缘)。有时在处理某些问题时，纹理不是一种合适的特征，我们必须寻找另一种特征。我们使用其中一类特征描述子描述图像边缘。此类特征可以描述边缘的不同方面，如边缘方向或取向、边缘位置、边缘强度或幅度。

本节将讨论称为取向梯度直方图(HOG)的描述子，这个描述子描述了边缘取向。有时，目标对象具有唯一的运动方向，因此 HOG 是一种合适的特征。HOG 创建了一些单元条(表示边缘取向的频率)的直方图。让我们来了解 HOG 的工作机制。

1. 图像梯度

图像内的每对相邻像素之间的强度会有变化。为测量这种变化，可计算每个像素的梯度向量来测量这个像素的强度如何变化到其相邻像素的强度。向量的大小是两个像素之间的强度差值。向量也通过 *X* 方向和 *Y* 方向反映出变化的方向。基于图 1-12 中的灰度图像，让我们来计算第 3 行第 4 列像素 21 在 *X* 和 *Y* 方向上强度的变化。

164	43	222	22	58
97	68	50	53	12
91	23	83	21	98
0	88	0	63	162
92	42	32	23	11

图 1-12　进行梯度计算的灰度图像

可以使用掩模找出 *X* 和 *Y* 方向的梯度大小，如图 1-13 所示。让我们开始计算梯度。

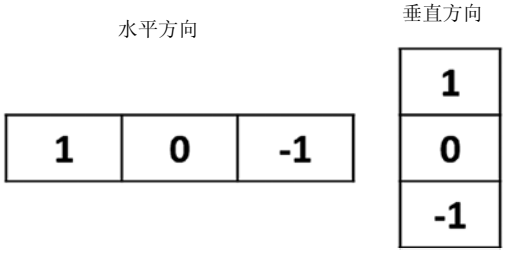


图 1-13　计算水平方向和垂直方向梯度的掩模

将水平掩模对中目标像素，我们可以计算出 *X* 方向的梯度。这里相邻像素是 83 和 98。将这两个值相减(无论是将左边像素的值减去右边像素的值，还是将右边像素的值减去左边像素的值都可以，但是整幅图要保持一致)，则这个像素的变化量为 98−83=15。这种情况下，所使用的角度为 0° 。

为获得此像素在 *Y* 方向上变化的量，将垂直掩模对中目标像素。然后，该像素上方和下方的像素进行相减，返回 63−53=10。这种情况下，使用的角度为 90° 。

计算了 *X* 和 *Y* 方向的变化后，下一步根据公式 1-1 计算最终的梯度大小，以及根据公式 1-2 计算梯度方向。

$$Z = \sqrt{X^2 + Y^2} \qquad \text{(式 1-1)}$$

$$\text{Angle} = \tan^{-1}\frac{Y}{X} \qquad \text{(式 1-2)}$$

梯度大小等于 $\sqrt{15^2 + 10^2} \approx 18.03$ 。

2. 梯度方向

关于梯度方向，有人可能会说，该像素改变方向是在 0°，因为 0°向量的大小比 90°向量的大小大。但是，另一些人可能会说，该像素在 0°和 90°方向都没有改变，而是在中间角度方向发生改变。可以将 X 和 Y 方向都纳入考虑，计算这个角度。该向量的方向为 $\tan^{-1}\frac{15}{10}=56.31°$。因此，该像素改变方向是在 56.31°。

计算完所有图像的角度后，下一步是创建此类角度的直方图。为了让直方图较小，我们并不使用所有的角度，而仅使用预定义角度的集合。通常使用的角度为水平(0°)、垂直(90°)和对角(45°和 135°)。每个角度所贡献的值等于根据公式 1-3 计算得出的梯度大小。例如，如果当前的像素对 Z 单元条做出了贡献，那么它就为 Z 单元条添加值 18.03。

3. 对直方图单元条的贡献

我们先前计算的角度为 56.31°，这不是之前所选中的其中一个角度。解决方法是分配此角度到最近的直方图单元条。56.31°位于单元条 45°和 90°之间。因为比起 90°，56.31°更接近 45°，所以这个角度被分配给单元条 45°。一种更好的做法是将该像素的贡献值拆分给两个角度，即 45°和 90°。

角度 45°和 90°之间的距离是 45°。角度 56.31°和 45°之间的距离为 $|56.31°-45°|=11.31$。这意味着角度 56.31°距 45°的百分比等于 $\frac{11.32}{45°}\%\approx 25\%$。换句话说，56.31°是 75%临近 45°。类似地，角度 56.31°和 95°之间的距离为 $|56.31°-90°|=33.69$。这意味着角度 56.31°距 90°的百分比等于 $\frac{33.69}{45°}\%\approx 75\%$。根据公式 1-3，计算该角度添加到单元条的值。

$$\text{contribution}_{\text{value}}=\frac{\text{abs}(\text{pixel}_{\text{angle}}-\text{bin}_{\text{angle}})}{\text{bin}_{\text{spacing}}}(\text{pixel}_{\text{gradientMagnitude}}) \qquad \text{(式 1-3)}$$

其中 $\text{pixel}_{\text{angle}}$ 是当前像素的方向，$\text{pixel}_{\text{gradientMagnitude}}$ 是当前像素的梯度大小，$\text{bin}_{\text{angle}}$ 是直方图单元条值，$\text{bin}_{\text{spacing}}$ 是每两个单元条之间的间隔大小。

因此，角度 56.31°添加给 45°的值为其梯度大小的 75%，这等于 $\frac{75}{100}\times 18.03\approx 13.5$。它仅添加了其梯度大小的 25%给 90°，等于 $\frac{25}{100}\times 18.03\approx 4.5$。

更实际的直方图包含从 0°开始到 180°结束的 9 个角度。每对角度的差值为 180/9=20。因此，所使用的角度为 0°、20°、40°、60°、80°、100°、120°、140°、160°和 180°。单元条不是这些角度，而是每个范围的中心值。对于 0°~20°这个范围，所使用的单元条为 10°。对于 20°～40°这个范围，单元条为 30°。最终的直方图为 10°、30°、50°、70°、90°、110°、130°、150°和 170°。如果角度为 25°，那么它添加值到其左右两边的单元条。也就是说，它添加 0.25 给

10°，添加 0.75 给 30°。

在位于第 2 行第 2 列处的像素 68 上重复上一步骤，应用水平掩模所得到的结果为 97−50=47，这是在 X 方向上的梯度变化。应用了垂直掩模后，结果为 43−23=20。根据公式 1-2 计算改变的方向，如下所示。

$$\text{Angle} = \tan^{-1}\frac{Y}{X} = \tan^{-1}\frac{20}{47} \approx 23^{\circ}$$

同样，所得到的结果角度不等于任何直方图单元条。因此，该角度的贡献可以拆分给其左右两边的单元条，也就是 15°和 45°。它添加 0.27 给 15°，0.73 给 45°。

位于第 4 行第 2 列的像素的强度值为 88，在 X 方向上的变化为 0。如果应用公式 1-2，结果将会除以 0。为避免结果除以 0，在分母上加上一个非常小的值，如 0.0000001。

4. HOG 步骤

至此，我们已经学会如何计算任何像素的梯度大小和方向。但在计算这些值前后，我们还有一些工作要完成。HOG 步骤汇总如下：

(1) 将输入图片按照长宽比 1:2 分成小片。例如，小片大小可能为 64×128、100×200，以此类推。

(2) 将小片分成小块(例如 4 个块)。

(3) 将每个块分成小单元格。块内单元格的大小是不固定的。例如，如果块的大小为 16×16，我们决定把它分成四个单元格，那每个单元格的尺寸为 8×8。注意，块可能会互相重叠，每个单元格可能多个块可用。

(4) 对于每个块内的每个单元格，计算所有像素的梯度大小和方向。

- 基于图 1-13 中的掩模计算梯度。
- 分别根据公式 1-1 和 1-2 计算梯度大小和方向。

(5) 基于梯度大小和方向构建每个单元格的直方图。如果用于构成直方图的角度数目为 9，那么每个单元格返回 9×1 的特征向量。根据先前的讨论计算直方图。

(6) 串联相同块内所有单元格的所有直方图，返回整个块的单个直方图。如果每个单元格直方图使用 9 个单元条表示，每个块有 4 个单元格，那么串联的直方图长度为 4×9=36。36×1 为向量，是每个块的结果。

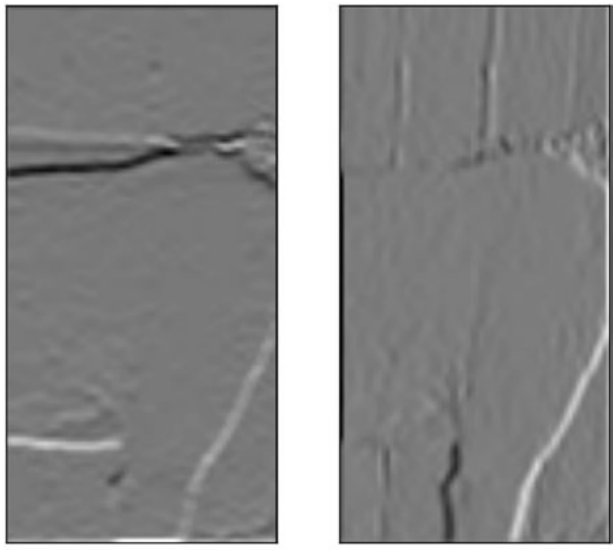
(7) 归一化向量，使其对亮度变化具有健壮性。

(8) 串联小图片内所有块的归一化向量，返回最终的特征向量。

如图 1-14 所示，这是图 1-5(a)中图片的一个小片，大小为 64×128。

图 1-14　计算 HOG 的小图片

创建直方图前，根据垂直和水平掩模计算垂直和水平梯度。梯度如图 1-15 所示。



(a) (垂直梯度)　　(b) (水平梯度)

图 1-15　64×128 小图片的垂直和水平梯度

代码清单 1-6 给出用于计算此类梯度的 Python 代码。

代码清单 1-6　计算梯度

```
import skimage.io, skimage.color
```

```
import numpy
import matplotlib

def calculate_gradient(img, template):
    ts = template.size #Number of elements in the template (3).
    #New padded array to hold the resultant gradient image.
    new_img = numpy.zeros((img.shape[0]+ts-1,
                           img.shape[1]+ts-1))
    new_img[numpy.uint16((ts-1)/2.0):img.shape[0]+numpy.uint16((ts
    -1)/2.0),
            numpy.uint16((ts-1)/2.0):img.shape[1]+
            numpy.uint16((ts-1)/2.0)] = img
    result = numpy.zeros((new_img.shape))
for r in numpy.uint16(numpy.arange((ts-1)/2.0,
img.shape[0]+(ts-1)/2.0)):
    for c in numpy.uint16(numpy.arange((ts-1)/2.0,
                          img.shape[1]+(ts-1)/2.0)):
    curr_region = new_img[r-numpy.uint16((ts-1)/2.0):r+numpy.
    uint16((ts-1)/2.0)+1,
                              c-numpy.uint16((ts-1)/2.0):c+numpy.
                              uint16((ts-1)/2.0)+1]
    curr_result = curr_region * template
    score = numpy.sum(curr_result)
    result[r, c] = score
#Result of the same size as the original image after removing the
padding.
result_img = result[numpy.uint16((ts-1)/2):result.shape[0]-numpy.
uint16((ts-1)/2),numpy.uint16((ts-1)/2):result.shape[1]-numpy.
uint16((ts-1)/2)]
return result_img
```

基于 calculate_gradient(img，template)函数(这个函数接收灰度图像和掩模作为输入)并基于掩模过滤图像，然后返回结果。使用不同的掩模调用图像两次，返回垂直和水平梯度。

之后，根据代码清单 1-7 中的 gradient_magnitude()函数，使用垂直和水平梯度计算梯度大小。

代码清单 1-7　梯度大小

```
def gradient_magnitude(horizontal_gradient, vertical_gradient):
```

```
    horizontal_gradient_square = numpy.power(horizontal_gradient, 2)
    vertical_gradient_square = numpy.power(vertical_gradient, 2)
    sum_squares = horizontal_gradient_square + vertical_gradient_square
    grad_magnitude = numpy.sqrt(sum_squares)
    return grad_magnitude
```

该函数就是应用先前计算水平和垂直梯度的公式 1-1。小图片的梯度大小如图 1-16 所示。

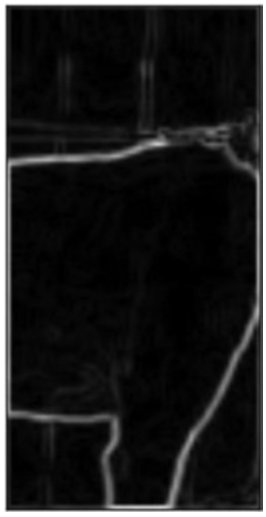


图 1-16　基于先前为 64×128 小图片计算的垂直和水平梯度的梯度大小

下面使用代码清单 1-8 中的 gradient_direction()函数计算梯度方向。

代码清单 1-8　梯度方向

```
def gradient_direction(horizontal_gradient, vertical_gradient):
    grad_direction = numpy.arctan(vertical_gradient/(horizontal_
    gradient+0.00000001))
    grad_direction = numpy.rad2deg(grad_direction)
    # Some angles are outside the 0-180 range. Next line makes all results
    fall within the 0-180 range.
    grad_direction = grad_direction % 180
    return grad_direction
```

注意，有一个小值(0.00000001)被加到分母中，这避免了除以 0 的错误。如果忽略这一点，一些输出结果将为 NaN(非数字)。

图 1-17 显示了被划分成 16×8 个单元格后的小图片。每个单元格都有 8×8 个像素，并且每个块具有 4 个单元格(即每个块具有 16×16 个像素)。

图 1-17　图像被划分为 16×8 个单元格

基于先前所计算的梯度大小和方向，我们可以返回小图片中的第一个 8×8 单元格(左上角的单元格)的结果，如图 1-18 所示。

44.41	88.69	91.57	89.74	84.94	84.98	88.62	91.62
0.26	15.95	165.96	63.43	1.97	178.15	173.66	15.26
0.77	29.74	159.44	116.57	2.05	0.0	0.0	168.69
1.02	45.0	161.57	153.43	0.0	0.0	0.0	146.31
0.75	38.66	160.02	135.0	4.4	1.97	172.87	153.43
0.5	36.87	165.96	135.0	4.57	2.05	171.87	53.13
0.25	14.04	0.0	0.0	0.0	0.0	0.0	33.69
179.25	135.0	10.3	26.57	2.29	5.91	35.54	158.96

(a) 梯度方向

207.19	219.06	219.08	219.	226.88	239.92	249.07	248.1
222.0	7.28	8.25	2.24	29.02	31.02	9.06	11.4
223.02	8.06	8.54	2.24	28.02	30.0	9.0	10.2
225.04	7.07	9.49	2.24	27.0	30.0	9.0	7.21
228.02	6.4	11.7	4.24	26.08	29.02	8.06	2.24
230.01	5.0	12.37	4.24	25.08	28.02	7.07	5.0
231.0	4.12	12.0	3.0	24.0	27.0	6.0	10.82
230.02	5.66	11.18	2.24	25.02	29.15	8.6	13.93

(b) 梯度大小

图 1-18　左上角 8×8 单元格的梯度大小和方向

基于先前讨论的简单示例，我们创建直方图。直方图有 9 个单元条，覆盖了从 0 到 180 的角度范围。仅使用有限的单元条表示这个范围会使每个单元条覆盖多个角度。如果仅使用 9 个单元条，那每个单元条会覆盖 20 个角度。第一个单元条覆盖的角度从 0(包括 0)到 20(不包括 20)。第二个单元条从 20(包括 20)到 40(不包括 40)，直到最后一个单元条，覆盖的范围从 160(包括 160)到 180(包括 180)。

每个区间单元条所赋予的数字等于每个区间的中心值。也就是说，第一个单元条为 10，第二个为 20，以此类推，直到最后一个单元条为 170。我们可以说，单元

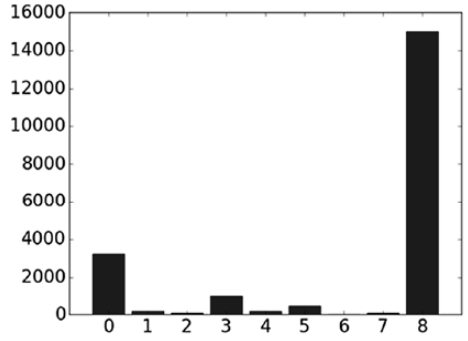
条从 10 开始到 170，每步为 20。对于图 1-18(a)所示的每个角度，可以找到其落点处的左右两个单元条。从左上元素(值为 44.41)开始，它位于单元条 30 和 50 之间。根据公式 1-3，计算出其为两个单元条所贡献的值。单元条 30 的贡献值计算如下：

$$\text{contribution}_{\text{value}} = \frac{\text{abs}(44.41-30)}{20}(207.19) = 0.72 \times 207.19 \approx 149.28$$

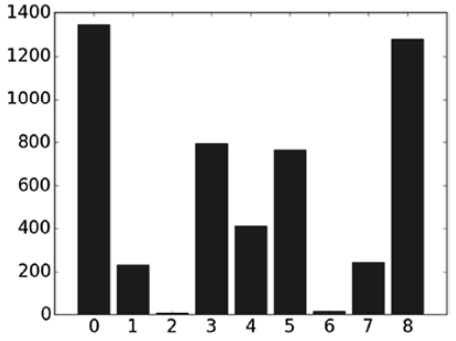
关于单元条 50，其贡献值计算如下：

$$\text{contribution}_{\text{value}} = \frac{\text{abs}(44.41-50)}{20}(207.19) = 0.28 \times 207.19 \approx 57.91$$

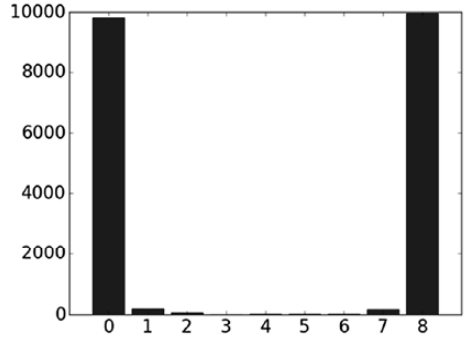
对于当前单元中的所有 8×8 个像素继续此过程。左上单元的直方图如图 1-19(a)所示。假设每个块包含 2×2 个单元格，图 1-17 中用亮色标记的左上方块中剩余的 3 个单元格的 9 单元条直方图如图 1-19 所示。如果计算给定块的所有直方图，则其特征向量为这 4 个 9 单元条直方图串联的结果。特征向量的长度为 9×4 = 36。



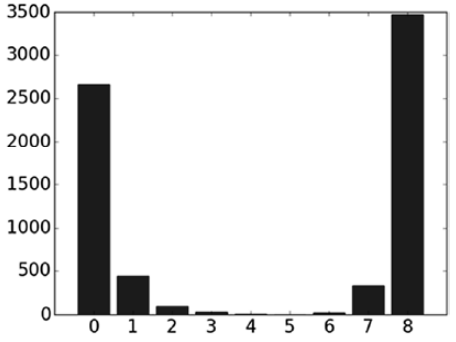
位置(0,0)处的第一个单元格



位置(0,1)处的第二个单元格



位置(1,0)处的第三个单元格



位置(1,1)处的第四个单元格

图 1-19　当前小图片左上块内部的 4 个单元格的 9 单元条直方图

在计算第一个块的特征向量后，选择具有 4 个单元格的下一块，使用亮色标记，

如图 1-20 所示。

同样，计算此块内 4 个单元格的 9 单元条直方图，如图 1-21 所示，其结果将会被串联，返回 36×1 的特征向量。

图 1-20　小图片内的第二个块使用亮色突出显示

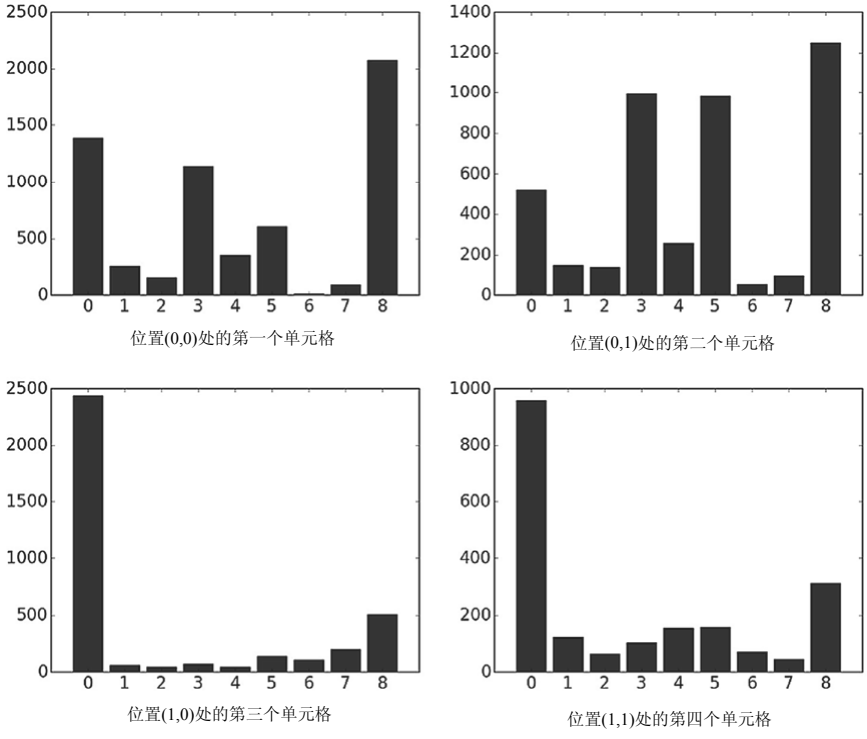


图 1-21　图 1-20 中当前小图片使用亮色标记的第二个块内 4 个单元格的 9 单元条直方图

使用代码清单 1-9 中的 HOG_cell_histogram()函数计算每个单元格的直方图。这个函数接收给定单元格的方向和大小，返回其直方图。

代码清单 1-9　单元格直方图

```
def HOG_cell_histogram(cell_direction, cell_magnitude):
    HOG_cell_hist = numpy.zeros(shape=(hist_bins.size))
    cell_size = cell_direction.shape[0]

    for row_idx in range(cell_size):
        for col_idx in range(cell_size):
        curr_direction = cell_direction[row_idx, col_idx]
        curr_magnitude = cell_magnitude[row_idx, col_idx]

        diff = numpy.abs(curr_direction - hist_bins)

        if curr_direction < hist_bins[0]:
            first_bin_idx = 0
            second_bin_idx = hist_bins.size-1
        elif curr_direction > hist_bins[-1]:
            first_bin_idx = hist_bins.size-1
            second_bin_idx = 0
        else:
            first_bin_idx = numpy.where(diff == numpy.min(diff))[0][0]
            temp = hist_bins[[(first_bin_idx-1)%hist_bins.size, (first_
            bin_idx+1)%hist_bins.size]]
            temp2 = numpy.abs(curr_direction - temp)
            res = numpy.where(temp2 == numpy.min(temp2))[0][0]
            if res == 0 and first_bin_idx != 0:
                second_bin_idx = first_bin_idx-1
            else:
                second_bin_idx = first_bin_idx+1

        first_bin_value = hist_bins[first_bin_idx]
        second_bin_value = hist_bins[second_bin_idx]
        HOG_cell_hist[first_bin_idx] = HOG_cell_hist[first_bin_idx] +
        (numpy.abs(curr_direction - first_bin_value)/(180.0/hist_bins.
        size)) * curr_magnitude
        HOG_cell_hist[second_bin_idx] = HOG_cell_hist[second_bin_idx] +
```

```
        (numpy.abs(curr_direction - second_bin_value)/(180.0/hist_bins.
        size)) * curr_magnitude
    return HOG_cell_hist
```

代码清单 1-10 给出了用于读取小图片并返回第一块中左上角单元格直方图的完整代码。注意，代码使用灰度图像。如果输入图片为灰度图，那么它仅有两个维度。如果输入图像为彩色图像，将有第三个维度表示通道。在这里，仅使用一个灰度通道。NumPy 数组的维度数目使用 ndim 属性返回。

代码清单 1-10　计算左上角单元格直方图的完整实现

```
import skimage.io, skimage.color
import numpy
import matplotlib.pyplot

def calculate_gradient(img, template):
    ts = template.size #Number of elements in the template (3).
    #New padded array to hold the resultant gradient image.
    new_img = numpy.zeros((img.shape[0]+ts-1,
                           img.shape[1]+ts-1))
    new_img[numpy.uint16((ts-1)/2.0):img.shape[0]+numpy.uint16((ts
             -1)/2.0),
            numpy.uint16((ts-1)/2.0):img.shape[1]+numpy.uint16((ts
            -1)/2.0)]
            = img
    result = numpy.zeros((new_img.shape))

    for r in numpy.uint16(numpy.arange((ts-1)/2.0,
    img.shape[0]+(ts-1)/2.0)):
        for c in numpy.uint16(numpy.arange((ts-1)/2.0,
                        img.shape[1]+(ts-1)/2.0)):
            curr_region=new_img[ r-numpy.uint16((ts-1)/2.0):r+numpy.
                                uint16((ts-1)/2.0)+1,
                                c-numpy.uint16((ts-1)/2.0):c+numpy.
                                uint16((ts-1)/2.0)+1]
            curr_result = curr_region * template
            score = numpy.sum(curr_result)
            result[r, c] = score
    #Result of the same size as the original image after removing the
    padding.
    result_img=result[numpy.uint16((ts-1)/2.0):result.shape[0]-numpy.
               uint16((ts-1)/2.0), numpy.uint16((ts-1)/2.0):result.
               shape[1]-numpy.uint16((ts-1)/2.0)]
    return result_img
```

```
def gradient_magnitude(horizontal_gradient, vertical_gradient):
    horizontal_gradient_square = numpy.power(horizontal_gradient, 2)
    vertical_gradient_square = numpy.power(vertical_gradient, 2)
    sum_squares = horizontal_gradient_square + vertical_gradient_square
    grad_magnitude = numpy.sqrt(sum_squares)
    return grad_magnitude

def gradient_direction(horizontal_gradient, vertical_gradient):
    grad_direction = numpy.arctan(vertical_gradient/(horizontal_
    gradient+0.00000001))
    grad_direction = numpy.rad2deg(grad_direction)
    grad_direction = grad_direction%180
    return grad_direction

def HOG_cell_histogram(cell_direction, cell_magnitude):
    HOG_cell_hist = numpy.zeros(shape=(hist_bins.size))
    cell_size = cell_direction.shape[0]

    for row_idx in range(cell_size):
        for col_idx in range(cell_size):
            curr_direction = cell_direction[row_idx, col_idx]
            curr_magnitude = cell_magnitude[row_idx, col_idx]

            diff = numpy.abs(curr_direction - hist_bins)

            if curr_direction < hist_bins[0]:
                first_bin_idx = 0
                second_bin_idx = hist_bins.size-1
            elif curr_direction > hist_bins[-1]:
                first_bin_idx = hist_bins.size-1
                second_bin_idx = 0
            else:
                first_bin_idx = numpy.where(diff == numpy.min(diff))[0][0]
                temp = hist_bins[[(first_bin_idx-1)%hist_bins.size, (first_
                bin_idx+1)%hist_bins.size]]
                temp2 = numpy.abs(curr_direction - temp)
                res = numpy.where(temp2 == numpy.min(temp2))[0][0]
                if res == 0 and first_bin_idx != 0:
                    second_bin_idx = first_bin_idx-1
                    else:
                        second_bin_idx = first_bin_idx+1

            first_bin_value = hist_bins[first_bin_idx]
            second_bin_value = hist_bins[second_bin_idx]
            HOG_cell_hist[first_bin_idx] = HOG_cell_hist[first_bin_idx] +
```

```
            (numpy.abs(curr_direction - first_bin_value)/(180.0/hist_bins.
            size)) * curr_magnitude
            HOG_cell_hist[second_bin_idx] = HOG_cell_hist[second_bin_idx] +
            (numpy.abs(curr_direction - second_bin_value)/(180.0/hist_bins.
            size)) * curr_magnitude
    return HOG_cell_hist

img = skimage.io.imread("im_patch.jpg")
if img.ndim >2:
    img = img[:, :, 0]

horizontal_mask = numpy.array([-1, 0, 1])
vertical_mask = numpy.array([[-1],
                             [0],
                             [1]])

horizontal_gradient = calculate_gradient(img, horizontal_mask)
vertical_gradient = calculate_gradient(img, vertical_mask)

grad_magnitude = gradient_magnitude(horizontal_gradient, vertical_gradient)
grad_direction = gradient_direction(horizontal_gradient, vertical_gradient)

grad_direction = grad_direction % 180
hist_bins = numpy.array([10,30,50,70,90,110,130,150,170])

cell_direction = grad_direction[:8, :8]
cell_magnitude = grad_magnitude[:8, :8]
HOG_cell_hist = HOG_cell_histogram(cell_direction, cell_magnitude)

matplotlib.pyplot.bar(left=numpy.arange(9), height=HOG_cell_hist,
align="center", width=0.8)
matplotlib.pyplot.show()
```

计算了块的特征向量后，下一步是归一化向量。特征归一化的动机是特征向量依赖于图像强度等级，因此最好使其对亮度变化具有健壮性。将向量中的每个元素除以根据公式 1-4 计算得出的向量长度进行归一化。

$$\text{vector}_{\text{length}} = \sqrt{X_1 + X_2 + \cdots + X_i} \qquad \text{(式 1-4)}$$

其中 X_i 表示索引为 i 的向量元素。这是第一个块归一化向量的结果。继续这个过程，直到返回所有块的所有 36×1 的特征向量。然后，串联所处理的整个小图片

的向量。

基于前面的讨论，在计算之前，HOG 需要指定以下参数。

- 取向的数目
- 每个单元格的像素数
- 每块单元格数

skimage.feature 模块中已经使用 Python 实现了 HOG，我们可以很容易地根据 skimage.feature.hog()函数使用它。先前的三个参数具有默认值，可改变这些参数来实现目标。如果将 normalise 参数设置为 True，那么可以返回归一化的 HOG。

```
skimage.feature.hog(image, orientations=9, pixels_per_cell=(8, 8),
cells_per_block=(3, 3), visualise=False, transform_sqrt=False, feature_
vector=True, normalise=None)
```

1.2.4　LBP

LBP 表示局部二值模式，是另一种二阶纹理描述子。提取 LBP 特征的步骤如下所示：

(1) 将图像划分成块(例如 16×16 个块)。

(2) 对于每个块，在每个像素上对中一个 3×3 的窗口。

根据公式 1-5，将所选择的中心像素 P_{central} 与其周围相邻的 8 个像素 P_{neighbor} 一一比较。从 8 次比较中，我们将得到 8 个二进制数字。

$$P_{\text{neighbor}} = \begin{cases} 1, P_{\text{neighbor}} > P_{\text{central}} \\ 0, \text{其他情形} \end{cases} \quad \text{(式 1-5)}$$

(3) 将 8 位的二进制代码转换为整数。整数范围从 0 到 2^8-1 (=255)。

(4) 使用计算得到的整数替换 P_{central} 值。

(5) 计算得到同一块内所有元素的新值后，计算直方图。

(6) 在计算了所有块的直方图后，将它们串联起来。

假设我们现在计算的块如图 1-12 所示，基于此开始计算基本的 LBP。

计算第 3 行第 4 列的像素，将中心像素与邻近的 8 个像素进行一一比较。图 1-22 显示了比较结果。

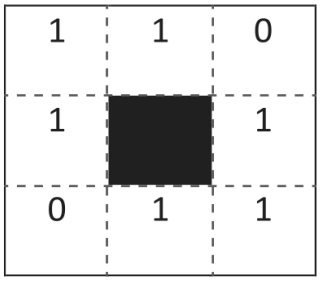


图 1-22　中心像素与邻近 8 个像素相比较的结果

接下来是返回二进制代码。我们可从 3×3 矩阵中的任何位置开始，但是必须在整个图像中保持一致。例如，从左上方位置开始顺时针移动，代码为 11011101。我们可以顺时针或逆时针移动，但必须一致。

之后，把二进制代码中每个二进制数与其位置对应的权重相乘得到的积加起来，将二进制代码转换为十进制。结果为 128 + 64 + 16 + 8 + 4 + 1 = 221。

在计算了块中每个像素的二进制代码后，返回其十进制数，创建直方图。对所有图像块重复此过程，将所有块的直方图串联起来(与在 HOG 情况下一样)。

这是 LBP 的基本实现，但是此类的特征描述子具有多种变化，可使其对亮度变化、缩放和旋转具有健壮性。

通过 Python 语言，我们可以使用接收 3 个参数的 skimage.feature.local_binary_pattern()函数轻松地实现它。

- 输入图像。
- 圆圈中相邻点的数目(*P*)。这个参数有助于实现旋转不变性。
- 圆圈的半径(*R*)。这个参数有助于实现缩放不变形。

此处是将 LBP 应用于图 1-2(a)中灰度图像的示例。

```
import skimage.feature
import skimage.io
import matplotlib.pyplot

im = skimage.io.imread("69.jpg", as_grey=True)
lbp = skimage.feature.local_binary_pattern(image=im, P=9, R=3)
matplotlib.pyplot.imshow(lbp, cmap="gray")
matplotlib.pyplot.xticks([])
matplotlib.pyplot.yticks([])
```

图 1-23 给出了输出图像。

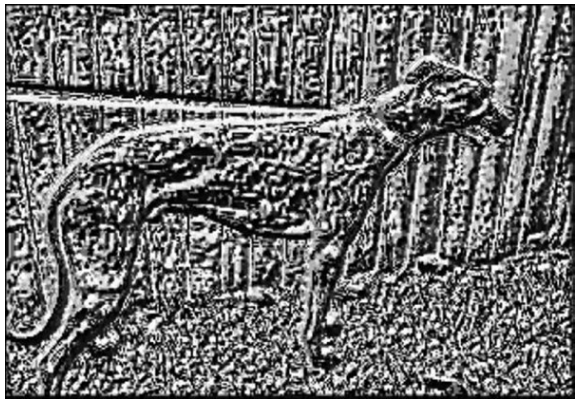


图 1-23　在灰度图像上应用 LBP 的输出

1.3　特征选择和缩减

假设与某一领域专家讨论后，推断出特征 X、Y 和 Z 是合适的特征。虽然初始选择了这些特征，但是也存在其中的一些特征可能于事无补的概率。我们可以基于一些实验确定某个特征是好还是差。在基于这三个特征训练模型后，识别率还是比较低，因而必须改变特征向量中的一些内容。为找到原因，我们可以针对单个特征训练模型，进行一些实验。我们发现特征 Z 与目标之间的相关性较低，因此决定不使用这个特征。从特征向量中完全消除某个特征而留住其他特征的做法被称为特征选择。选择技术将特征分为好和差。一般完全消除差的特征，而只使用好的特征。

事实上，每个特征内部有一些元素可能不适合分析中的应用类型。假设在特征向量内使用特征 X，这个特征是具有 10 个元素的集合。其中一个或多个元素可能对任务没有帮助。例如，一些特征是冗余的。也就是说，一些特征可能彼此优化相关，因此仅使用一个特征就足以描述数据，没必要使用多个特征进行相同的工作。由于有相关特征，因此也可能不存在唯一的最优特征子集。这是因为有多个完美相关的特征，所以某个特征可以替代其他特征，构建出新的特征子集。

另一种类型的特征为无关特征。一些特征与所需求的预测不相关，我们视其为噪声。此类特征不会增强(反而会降低)识别率。因此，我们最好检测出这些特征，将它们移除，使它们不会影响学习过程。仅移除元素子集而保留其他元素的做法被称为特征缩减。

缩减特征子集长度和尽可能移除不良特征背后的另一个动机是，特征向量长度越长，在训练和测试模型中消耗的时间就越多。

为了将特征分类为相关或不相关，我们需要一些度量来显示每个特征与输出类别的相关性，或者说显示单个特征预测所需输出的好坏程度。特征相关性是特征区分不同类别的能力。在选定度量后，我们使用这些度量创建好的特征子集，消除不良特征。我们将特征消除方法分为监督方法和无监督方法。监督方法包括过滤器方法和包装器方法，无监督方法包括嵌入式方法。

1.3.1　过滤器方法

过滤器方法添加了额外的预处理步骤，基于为每个特征计算得到的不同标准，应用排序技术对特征进行排序，测量特征的相关性。这些标准包括标准偏差(STD)、能量、熵、相关性和互信息(MI)。基于某个阈值，选中高排名的特征以训练模型。比起其他选择方法，过滤器方法快速且不耗时。它们计算简单、可扩展、能避免过拟合，与学习模型无关。

对于训练不同的模型，只需要完成选择一次，然后训练模型就可以使用所选中的特征。但比起其他特征选择的方法，过滤器方法有一些关键缺点，会严重影响它们的性能。过滤器/排序方法不能建模特征的关联性。它独立于同一子集内的其他特征/变量选择每个变量/特征。根据选择的标准，当某个特征排名较高时，就可以选

中这个特征。用于排名的标准不考虑不同特征之间的关系。虽然多个特征自身能显著增加学习率，但是当它们与其他特征结合时，并不能保证情况还是如此，因此忽略特征的关联性可能会破坏整个选中的子集。虽然有时有用的变量本身与其他变量结合在一起时依然有用，但不一定总是这样。

如果两个特征 f_1 和 f_2 的效用分别为 x_1 和 x_2，这并不意味着将其组合在一起时，其效用为 $x_1 + x_2$。另外，由于可能存在两个或多个特征完全满足标准，但是每个特征都是执行相同任务的所有其他特征的完美反映，因此忽略特征的关联性容易产生冗余特征和相关特征。因而，无须使用多个相同的特征，仅一个特征足以。相关性是另一种形式的冗余，此时特征不是完全相同，却有关联，通常可能完成相同的工作(可以表示为两条平行线)。由于完全相关的变量没有增加任何额外信息，因此它们是真正的冗余。冗余和相关特征的使用导致长长的特征向量，使得进行特征选择来缩减特性向量的长度的益处没有达到。由于特征选择与性能的解耦合，过滤器方法不依赖于学习算法的性能，因此它不与学习模型交互。相反，仅考虑特征和类别标签之间的单一标准，没有信息指示特征与学习算法一起工作的好坏程度。优良的特征选择子应该考虑学习算法和训练数据集之间如何交互。最终，计算用于确定是否选择某个特征的阈值并不是一件容易的任务。所有这些原因导致需要选用其他特征选择方法来克服这些问题。

1.3.2 包装器方法

包装器方法是第二种方法，它解决了过滤器方法中的一些问题。包装器方法通过创建最大化性能的所选特征子集，尽可能与学习模型交互。该方法创建了所有可能的特征子集，以找到最好的子集。包装器方法之所以称为包装器，是因为它将学习算法包裹在内。它使用感应或学习算法作为黑盒子，使用所选子集训练算法，测试所选特征子集工作的好坏程度，然后使用最大化其性能的子集。当谈到包装器方法时，涉及多个点，包括选择特征子集的长度、创建特征子集空间、搜索特征子集空间、评估学习算法的性能、停止搜索的标准和确定使用哪个学习算法。特征缩减/选择算法的目标是根据最初长度为 N 的完整特征向量创建最大化性能的长度为 L 的特征的所有可能组合，其中 $L<N$。对于长度为 30 的普通特征向量，存在大量的组合来创建长度 $L=10$ 的子集。正因为此，我们应用搜索策略，使用评估函数惩罚不良子集，搜索出最佳子集。这种情况下，目标函数为模型的性能。因此，关注点就从学习问题转变为搜索问题。对于这个问题，由于穷尽搜索使用所有的子集访问训练学习模型，这是计算密集型的，因此不适用。我们使用进化算法(EA)避免探索所有子集，这可被归类为两种类型：顺序选择算法(确定型)和启发式搜索算法(随机型)。

可以进一步将确定型搜索算法分为两类。事实上，这两类相当类似，即前向选择和后向选择。在前向选择中，算法从表示空特征集的根开始，然后一方面逐个添加特征，另一方面为每种变化训练模型。在后向选择中，搜索的根为完整的特征集，

算法一方面为每种变化训练学习模型，另一方面逐个移除特征。此类算法的示例包括顺序特征选择(SFS)、顺序后向选择(SBS)、顺序前向浮点选择(SFFS)、自适应SFFS(ASFFS)、波束搜索。为防止穷尽搜索，其中加入了停止标准，以避免探索所有可能的组合。对于前向选择，标准可为最大的特征向量长度；对于后向选择，标准可为最小的特征向量长度。这也可以是最大化性能，因此在达到所选定的性能后搜索停止。

第二类搜索算法(随机算法)被告知搜索使用启发式评估函数，生成启发值，说明每个子集距离最大性能有多远。此类算法的示例包括遗传算法(GA)、粒子群优化(PSO)、模拟退火(SA)和随机爬山法。我们将在第 4 章中详细讨论 GA。

必须回答的一个问题是 L(特征数)的最优值是什么？包装器方法有不同的方式回答这个问题。一种方式是选择固定数目的特征创建特征向量。通过使用组合，我们能够得到从 N 个特征中选择 L 个特征的所有可能性。但遗憾的是，选定的固定值 L 可能不是最优的，我们不能保证 L 个选中的特征是能够给出学习模型最佳性能的那些特征。因此，另一种方式是将选定特征的数目作为变量并动态变化。我们可在顺序选择算法中通过尝试不同的特征数目，选定能够最大化性能的最佳特征数目来做到这一点。使得特征数目动态变化的缺点是，由于创建不同长度的特征组合训练模型，因此增加了计算时间。为缩短时间，我们可以添加标准，使得在达到目标性能后或选定特征数目达到最大长度后，提早停止学习。

包装器方法与过滤器方法相比存在许多优点。包装器方法与学习模型交互，它使用学习算法性能作为指标，选择最佳特征子集。这种方法也建模了特征的关联性(特征并不是单独或互相独立进行选择)，它监控了组合特征如何影响性能。最后，这对于冗余特征和相关特征具有健壮性。但包装器方法也有大量缺点。由于单个模型要被训练多次，因此比较耗时。有些模型甚至可能要耗费多个小时才能训练一次。其结果是，对于此类模型，包装器方法不是一种选择。此外，由于选择依赖于学习模型，包装器方法会受到过拟合的影响，因此对于不同的模型，它不能一般化选中的特征。

1.3.3　嵌入式方法

使用过滤器方法和包装器方法进行特征选择有各自的优缺点。嵌入式方法尝试结合过滤器方法和包装器方法的优点。由于此类方法避免了重新训练学习算法，因此不会耗时。它还通过与学习模型交互最大化性能。该方法选择特征的依据仅因为其与输出类标签 m 相关。由于这种方法在训练步骤中嵌入特征选择，因此我们称之为嵌入式方法。

嵌入式方法的分类如下：修剪、内置、正则化(处罚)。修剪方法从使用完整的特征集训练模型开始，然后为单个特征计算相关系数。根据所使用的模型，使用此类系数对特征的重要性进行排名。系数值高反映出强的相关性。嵌入式特征选择的内置方法计算每个特征的信息增益，与在决策树学习(ID3)中一样。

在机器学习(ML)中，一些模型可得到良好训练，对训练数据中的任何样本做出正确预测，但遗憾的是，它们不能对训练样本之外的其他样本做出正确预测。我们称这个问题为过拟合。正则化是用来避免这个问题的技术。该技术调整或选择最佳的模型复杂性来拟合训练数据，同时能够预测未见过的样本。没有正则化，模型可能非常简单而欠拟合(对训练和测试样本都不能做出正确的预测)，或者非常复杂而过拟合(对训练样本可以做出正确预测，但对测试样本做出错误预测)。欠拟合和过拟合都会使模型太弱，不能一般化到任何样本。因此，正则化是一般化模型来预测任何样本(无论是训练样本还是测试样本)的一种方式。

1.3.4 正则化

为找到最好的模型，机器学习的常用方法是定义损失函数或成本函数，描述模型拟合数据的好坏程度。目标是找到最小化损失函数的模型。通常情况下，在任何学习模型中，目标函数只有一个标准，即最大化性能。正则化方法为目标函数添加了另一个标准来控制复杂度等级，如公式 1-6 所示。

$$L = \min \ \text{error}(Y_{\text{predict}}, Y_{\text{correct}}) + \lambda \ \text{penalty}(W_i) \qquad \text{(式 1-6)}$$

其中 Y_{predict} 是所预测的类别标签，Y_{correct} 是正确的类别标签，error()计算预测误差，W_i 是特征元素 X_i 的权重，λ 是控制模型复杂度的正则化参数。我们使用这个参数控制目标函数和处罚之间的权衡。根据公式 1-7 定义处罚。

$$\text{penalty}(w) = \sum_{i=1} |W_i| \qquad \text{(式 1-7)}$$

通过改变 λ 值，模型的复杂度改变了。这是通过设置权重接近于 0 来处罚某些特征。系数的量级是决定模型复杂度的显著因子。特征选择可以通过选择具有高权重的特征间接完成。权重越高，预测正确类别的特征就越相关。这是正则化方法称为处罚的原因。

正则化参数 λ 的目标是尽量减少损失 L，使它保持最小值。对于 λ 接近∞的大值而言，系数必须非常小，接近于 0，使总值尽可能小。这使得大多数系数为 0，因此可以移除它们。对于 0<λ<∞的值，有一些系数等于 0，这也会被移除，但是它们中的多数不为 0。λ 的最佳值是多少？对于 λ，没有固定值，它的值可以高效地使用交叉验证(CV)计算得到。

嵌入式方法结合了过滤器方法和包装器方法的优点，成为特征选择的最新研究趋势。由于这种方法使用训练模型性能作为度量，因此它与学习模型交互，又因为它不需要重新训练模型，所以不费时，同时它也对特征依赖进行建模，避免了冗余和相关的特征。选择特征同时进行训练，这样就不需要将数据拆分成训练集和验证集，因此在数据使用方面是非常有效的。不过，虽然对于用于选择特征的学习模型而言，使用嵌入式方法选择特征表现得很好，但是所选择的特征也许会依赖于此类模型，不能在不同的模型上表现良好，与使用过滤器方法所生成的结果不一样。