

兼容TensorFlow 1.x与2. x版本

共75个实例，覆盖了TensorFlow的大量接口

提供了大量可重用代码，可应用于真实场景

涉及图像识别、文本分类、数值分析、机器翻译、语音合成等

李大学

力荐

京东终身荣誉技术顾问/燧云科技创始人

深度学习之 TensorFlow

工程化项目实战

李金洪◎编著



中国工信出版集团



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

业界点评

这本书内容全面、针对性强、案例丰富。书中覆盖了TensorFlow 1.x和2.x版本的很多使用细节，介绍了很多项目级的技术解决方案。本书对提升AI开发水平大有帮助。

京东终身荣誉技术顾问/磁云科技创始人 **李大学**

本书特色

- ① 兼容TensorFlow 1.x与2.x版本，提供了大量的编程经验
- ② 覆盖了TensorFlow的大量接口
- ③ 提供了高度可重用代码，公开了大量的商用代码片段
- ④ 书中的实战案例可应用于真实场景
- ⑤ 从工程角度出发，覆盖工程开发全场景
- ⑥ 提供了大量前沿论文链接地址，便于读者进一步深入学习
- ⑦ 注重方法与经验的传授

资源与服务

关注并访问公众号“xiangyuejiqiren”（见下方二维码），在公众号中回复“深2”即可得到相关资源的下载链接。

在阅读过程中，如有不理解的技术点，可以到论坛 <https://bbs.ainacannda.com> 发帖提问。



上架建议：人工智能/TensorFlow

ISBN 978-7-121-36392-4

9 787121 363924 >

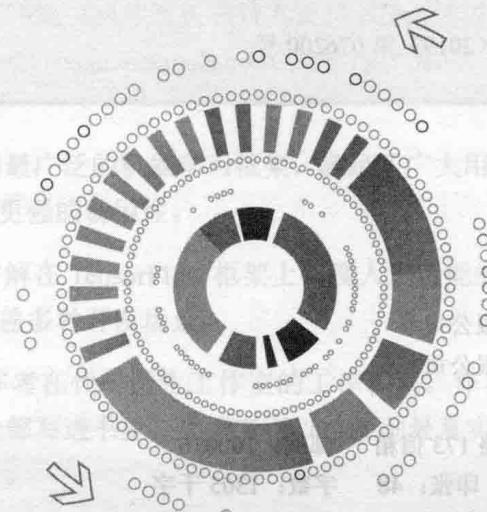
定价：159.00元



策划编辑：吴宏伟
责任编辑：牛 勇
封面设计：吴海燕

深度学习之 TensorFlow 工程化项目实战

李金洪◎编著



电子工业出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

这是一本非常全面的、专注于实战的 AI 图书，兼容 TensorFlow 1.x 和 2.x 版本，共 75 个实例。

全书共分为 5 篇：第 1 篇，介绍了学习准备、搭建开发环境、使用 AI 模型来识别图像；第 2 篇，介绍了用 TensorFlow 开发实际工程的一些基础操作，包括使用 TensorFlow 制作自己的数据集、快速训练自己的图片分类模型、编写训练模型的程序；第 3 篇，介绍了机器学习算法相关内容，包括特征工程、卷积神经网络（CNN）、循环神经网络（RNN）；第 4 篇，介绍了多模型的组合训练技术，包括生成式模型、模型的攻与防；第 5 篇，介绍了深度学习在工程上的应用，侧重于提升读者的工程能力，包括 TensorFlow 模型制作、布署 TensorFlow 模型、商业实例。

本书结构清晰、案例丰富、通俗易懂、实用性强。适合对人工智能、TensorFlow 感兴趣的读者作为自学教程。另外，本书也适合社会培训学校作为培训教材，还适合大中专院校的相关专业作为教学参考书。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

图书在版编目（CIP）数据

深度学习之 TensorFlow 工程化项目实战 / 李金洪编著. —北京：电子工业出版社，2019.5

ISBN 978-7-121-36392-4

I. ①深… II. ①李… III. ①人工智能—算法 IV. ①TP18

中国版本图书馆 CIP 数据核字（2019）第 076200 号

策划编辑：吴宏伟

责任编辑：牛 勇

印 刷：三河市良远印务有限公司

装 订：三河市良远印务有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编：100036

开 本：787×1092 1/16 印张：48 字数：1305 千字

版 次：2019 年 5 月第 1 版

印 次：2019 年 5 月第 1 次印刷

定 价：159.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，
联系及邮购电话：(010) 88254888, 88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819, faq@phei.com.cn。

作者介绍



李金洪

“大蛇智能”网站创始人、“代码医生”工作室主程序员。

精通Python、C、C++、汇编、Java和Go等多种编程语言。擅长神经网络、算法、协议分析、逆向工程和移动互联网安全架构等技术。在深度学习领域，参与过某移动互联网后台的OCR项目、某娱乐节目机器人的语音识别和声纹识别项目，以及人脸识别、活体检测等多个项目。在“代码医生”工作室工作期间，完成过金融、安全、市政和医疗等多个领域的AI算法外包项目。

出版过《Python带我起——入门、进阶、商业实战》《深度学习之TensorFlow——入门、原理与进阶实战》两本书。

本书编辑

吴宏伟

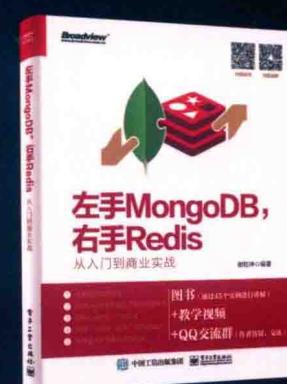
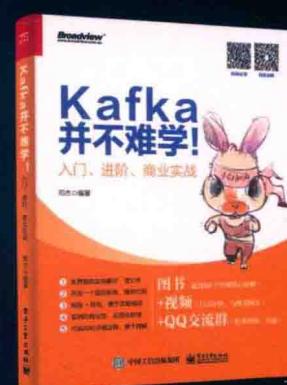
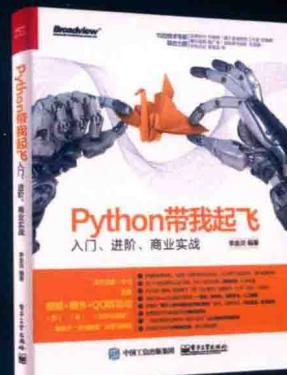
邮箱：wuhongwei@phei.com.cn

Q Q：83744810

欢迎投稿



好书分享



前言

关注并访问公众号“xiangyuejiqiren”，在公众号中回复“深2”得到相关资源的下载链接。

| | | |
|---------------------------------|----------------------------------|------------------------------------|
| 3-1 使用AI模型来识别图像.py | 3-2 使用Inception-Mobile模型来识别图像.py | 4-1 将模拟数据制作成内存对象数据集.py |
| 3-2 带迭代的模型和数据集.py | 3-3 将图片制作成内存对象数据集.py | 4-2 将excel文件制作成内存对象数据集.py |
| 3-4 将图片文件制作成TFRecord数据集.py | 3-4 interleave例子.py | 4-3 Dataset对象的操作方法.py |
| 3-6 将内存数据转成DataSet数据集.py | 3-9 from_tensor_slices的注意事项.py | 4-10 将图片文件制作成Dataset数据集.py |
| 4-1 将TFRecord文件制作成Dataset数据集... | 4-12 在动态图里加载Dataset数据集.py | 4-13 在动态图里加载Dataset数据集_t28.py |
| 4-14 在不同场景区中应用数据集.py | 5-1 mydataset.py | 5-1 model.py |
| 5-3 train.py | 5-4 test.py | 5-5 测试TF-Hub库中的mobilenet_v2模型.py |
| 5-6 使用模型评估人物的年龄.py | 6-1 使用静态图训练一个具有保存检查点功能... | 6-2 使用动态图训练一个具有保存检查点功能... |
| 6-3 动态识别一种帧数方法.py | 6-4 从动态图种获取变量.py | 6-5 静态图中使用动态图.py |
| 6-6 使用估算器框架训练一个回归模型.py | 6-7 为估算器添加钩子.py | 6-8 自定义hook.py |
| 6-9 将估算器模型转为静态图模型.py | 6-10 tf.layers模块.py | 6-11 keras回调模型.py |
| 6-12 使用tf.keras预训练模型.py | 6-13 在静态图中使用tf.keras.py | 6-14 tfserving例子.html |
| 6-15 使用估算器框架进行分布式训练.py | 6-16 使用估算器框架进行分布式训练ps.py | 6-17 使用估算器框架进行分布式训练chief.py |
| 6-18 使用估算器框架进行分布式训练work.py | 6-19 使用ResNet识别桔子和苹果.py | 6-20 在TensorFlow中训练mnist.py |
| 6-21 查看Tf1模型及超参数.py | 6-22_t2code.py | 7-1 用wide and deep模型预测人口收入.py |
| 7-2 用boosting trees模型预测人口收入.py | 7-3 使用feature_column处理连续值特征.py | 7-4 将连续值特征转换为离散特征.py |
| 7-5 将离散文本特征转化为one-hot编码与... | 7-4 提取特征生成交叉表.py | 7-5 特征工程.py |
| 7-8 脸部COCO数据集中的标注框.py | 7-9 minstmeans.py | 7-6 电影推荐系统.py |
| 7-11 用lattice预测收入.py | 7-12 lattice结合dnn.py | 7-7 序列特征工程.py |
| 7-14 MKR.py | 7-15 train.py | 7-8 data_loader.py |
| 8-1 读取fashion-mnist 数据集.py | 8-2 Capsulemodel.py | 8-3 使用 tensorflow 训练白底黑白图中的服装图案.py |
| 8-4 capsnet_ckpt.py | 8-5 train_EM.py | 8-6 NLP文本预处理.py |
| 8-7 TextCnn模型.py | 8-6 使用TextCnn模型进行文本分类.py | 8-7 使用keras注意力机制模型分析评论者情... |
| 8-10 keras注意机制模型.py | 8-11 yolo_v3.py | 8-8 使用YOLOV3模型进行实例检测.py |
| 8-13 annotation.py | 8-14 generator.py | 8-9 box.py |



本书由大蛇智能官网提供内容有关的技术支持。在阅读过程中，如有不理解的技术点，可以到论坛 <https://bbs.ainacannda.com> 发帖进行提问。

TensorFlow 是目前使用最广泛的机器学习框架，满足了广大用户的需求。如今 TensorFlow 已经更新到 2.x 版本，具有更强的易用性。

本书通过大量的实例讲解在 TensorFlow 框架上实现人工智能的技术，兼容 TensorFlow 1.x 与 TensorFlow 2.x 版本，覆盖多种开发场景。

书中的内容主要源于作者在代码医生工作室的工作积累。作者将自己在真实项目中使用 TensorFlow 的经验与技巧全部写进书里，让读者可以接触到最真实的案例、最实战的场景，尽快搭上人工智能的“列车”。

作者将自身的项目实战经验浓缩到三本书里，形成了“深度学习三部曲”。三本书形成一套完善的知识体系，构成了完备的技术栈闭环。

本书是“深度学习三部曲”的最后一本。

- 《Python 带我起飞——入门、进阶、商业实战》，主要讲解了 Python 基础语法。与深度学习关系不大，但包含了开发神经网络模型所必备的基础知识。
- 《深度学习之 TensorFlow——入门、原理与进阶实战》，主要讲解了深度学习的基础

网络模型及 TensorFlow 框架的基础编程方法。

- 《深度学习之 TensorFlow 工程化项目实战》，主要讲解在实战项目中用到的真实模型，以及将 TensorFlow 框架用于各种生产环境的编程方法。

这三本书可以将一个零基础的读者顺利带入深度学习行业，并让其能够成为一名合格的深度学习工程师。

本书特色

1. 兼容 TensorFlow 1.x 与 2.x 版本，提供了大量的编程经验

本书兼顾 TensorFlow 1.x 与 2.x 两个版本，给出了如何将 TensorFlow 1.x 代码升级为 TensorFlow 2.x 可用的代码。

2. 覆盖了 TensorFlow 的大量接口

TensorFlow 是一个非常庞大的框架，内部有很多接口可以满足不同用户的需求。合理使用现有接口可以在开发过程中起到事半功倍的效果。然而，由于 TensorFlow 的代码迭代速度太快，有些接口的配套文档并不是很全。作者花了大量的时间与精力，对一些实用接口的使用方法进行摸索与整理，并将这些方法写到书中。

3. 提供了高度可重用代码，公开了大量的商用代码片段

本书实例中的代码大多都来自代码医生工作室的商业项目，这些代码的易用性、稳定性、可重用性都很强。读者可以将这些代码提取出来直接用在自己的项目中，加快开发进度。

4. 书中的实战案例可应用于真实场景

本书中大部分实例都是当前应用非常广泛的通用任务，包括图片分类、目标识别、像素分割、文本分类、语音合成等多个方向。读者可以在书中介绍的模型的基础上，利用自己的业务数据集快速实现 AI 功能。

5. 从工程角度出发，覆盖工程开发全场景

本书以工程实现为目标，全面覆盖开发实际 AI 项目中所涉及的知识，并全部配有实例，包括开发数据集、训练模型、特征工程、开发模型、保护模型文件、模型防御、服务端和终端的模型部署。其中，特征工程部分全面讲解了 TensorFlow 中的特征列接口。该接口可以使数据在特征处理阶段就以图的方式进行加工，从而保证了在训练场景下和使用场景下模型的输入统一。

6. 提供了大量前沿论文链接地址，便于读者进一步深入学习

本书使用的 AI 模型，大多来源于前沿的技术论文，并在原有论文基础上做了一些结构改进。这些实例具有很高的科研价值。读者可以根据书中提供的论文链接地址，进一步深入学习更多的前沿知识，再配合本书的实例进行充分理解，达到融会贯通。本书也可以帮助 AI 研究者进行学术研究。

7. 注重方法与经验的传授

本书在讲解知识时，更注重传授方法与经验。全书共有几十个“提示”标签，其中的内容都是含金量很高的成功经验分享与易错事项总结，有关于经验技巧的，也有关于风险规避的，可以帮助读者在学习的路途上披荆斩棘，快速进步。

本书读者对象

- 人工智能爱好者
- 人工智能专业的高校学生
- 人工智能专业的教师
- 人工智能初学者
- 人工智能开发工程师
- 使用 TensorFlow 框架的工程师
- 集成人工智能的开发人员

关于作者

本书由李金洪主笔编写，参与本书编写的还有以下作者。

石昌帅

代码医生工作室成员，具有丰富的嵌入式及算法开发经验，参与多款机器人、图像识别等项目开发，擅长机器人定位、导航技术、计算机视觉技术，熟悉 NVIDIA jetson 系列、Raspberry PI 系列等平台软硬件开发、算法优化。从事的技术方向包括机器人导航、图像处理、自动驾驶等。

甘月

代码医生工作室成员，资深 iOS 高级工程师，有丰富的 iOS 研发经验，先后担任 iOS 主管、项目经理、iOS 技术总监等职务，精通 Objective-C、Swift、C 等编程语言，参与过银行金融、娱乐机器人、婚庆、医疗等领域的多个项目。擅长 Mac 系统下的 AI 技术开发。

江枭宇

代码医生工作室成员，是大蛇智能社区成长最快的 AI 学者。半年时间，由普通读者升级为社区的资深辅导员。在校期间曾参加过电子设计大赛（获省级一等奖）、Google 校企合作的 AI 创新项目、省级创新训练 AI 项目。熟悉 Python、C 和 Java 等编程语言。擅长图像处理方向、特征工程方向及语义压缩方向的 AI 任务。

- 4.6.1 如何生成 Dataset 数据集
- 4.6.2 如何使用 Dataset 接口
- 4.6.3 tf.data.Dataset 接口所支持的数据类型转换操作

目 录

第1篇 准备

| | |
|---|----|
| 第1章 学习准备 | 2 |
| 1.1 TensorFlow 能做什么 | 2 |
| 1.2 学习 TensorFlow 的必备知识 | 3 |
| 1.3 学习技巧：跟读代码 | 4 |
| 1.4 如何学习本书 | 4 |
| 第2章 搭建开发环境 | 5 |
| 2.1 准备硬件环境 | 5 |
| 2.2 下载及安装 Anaconda | 6 |
| 2.3 安装 TensorFlow | 9 |
| 2.4 GPU 版本的安装方法 | 10 |
| 2.4.1 在 Windows 中安装 CUDA | 10 |
| 2.4.2 在 Linux 中安装 CUDA | 13 |
| 2.4.3 在 Windows 中安装 cuDNN | 13 |
| 2.4.4 在 Linux 中安装 cuDNN | 14 |
| 2.4.5 常见错误及解决方案 | 16 |
| 2.5 测试显卡的常用命令 | 16 |
| 2.6 TensorFlow 1.x 版本与 2.x 版本共存的解决方案 | 18 |
| 第3章 实例 1：用 AI 模型识别图像是桌子、猫、狗，还是其他 | 21 |
| 3.1 准备代码环境并预训练模型 | 21 |
| 3.2 代码实现：初始化环境变量，并载入 ImgNet 标签 | 24 |
| 3.3 代码实现：定义网络结构 | 25 |
| 3.4 代码实现：载入模型进行识别 | 26 |
| 3.5 扩展：用更多预训练模型完成图片分类任务 | 28 |

第2篇 基础

| | |
|------------------------------------|-----------|
| 第4章 用TensorFlow制作自己的数据集 | 30 |
| 4.1 快速导读 | 30 |
| 4.1.1 什么是数据集 | 30 |
| 4.1.2 TensorFlow的框架 | 31 |
| 4.1.3 什么是TFDS | 31 |
| 4.2 实例2：将模拟数据制作成内存对象数据集 | 32 |
| 4.2.1 代码实现：生成模拟数据 | 32 |
| 4.2.2 代码实现：定义占位符 | 33 |
| 4.2.3 代码实现：建立会话，并获取数据 | 34 |
| 4.2.4 代码实现：将模拟数据可视化 | 34 |
| 4.2.5 运行程序 | 34 |
| 4.2.6 代码实现：创建带有迭代值并支持乱序功能的模拟数据集 | 35 |
| 4.3 实例3：将图片制作成内存对象数据集 | 37 |
| 4.3.1 样本介绍 | 38 |
| 4.3.2 代码实现：载入文件名称与标签 | 39 |
| 4.3.3 代码实现：生成队列中的批次样本数据 | 40 |
| 4.3.4 代码实现：在会话中使用数据集 | 41 |
| 4.3.5 运行程序 | 42 |
| 4.4 实例4：将Excel文件制作成内存对象数据集 | 42 |
| 4.4.1 样本介绍 | 43 |
| 4.4.2 代码实现：逐行读取数据并分离标签 | 43 |
| 4.4.3 代码实现：生成队列中的批次样本数据 | 44 |
| 4.4.4 代码实现：在会话中使用数据集 | 45 |
| 4.4.5 运行程序 | 46 |
| 4.5 实例5：将图片文件制作成TFRecord数据集 | 46 |
| 4.5.1 样本介绍 | 47 |
| 4.5.2 代码实现：读取样本文件的目录及标签 | 47 |
| 4.5.3 代码实现：定义函数生成TFRecord数据集 | 48 |
| 4.5.4 代码实现：读取TFRecord数据集，并将其转化为队列 | 49 |
| 4.5.5 代码实现：建立会话，将数据保存到文件 | 50 |
| 4.5.6 运行程序 | 51 |
| 4.6 实例6：将内存对象制作成Dataset数据集 | 52 |
| 4.6.1 如何生成Dataset数据集 | 52 |
| 4.6.2 如何使用Dataset接口 | 53 |
| 4.6.3 tf.data.Dataset接口所支持的数据集变换操作 | 54 |

| | |
|---|-----------|
| 4.6.4 代码实现：以元组和字典的方式生成 Dataset 对象 | 58 |
| 4.6.5 代码实现：对 Dataset 对象中的样本进行变换操作 | 59 |
| 4.6.6 代码实现：创建 Dataset 迭代器..... | 60 |
| 4.6.7 代码实现：在会话中取出数据..... | 60 |
| 4.6.8 运行程序 | 61 |
| 4.6.9 使用 <code>tf.data.Dataset.from_tensor_slices</code> 接口的注意事项..... | 62 |
| 4.7 实例 7：将图片文件制作成 Dataset 数据集..... | 63 |
| 4.7.1 代码实现：读取样本文件的目录及标签..... | 64 |
| 4.7.2 代码实现：定义函数，实现图片转换操作 | 64 |
| 4.7.3 代码实现：用自定义函数实现图片归一化 | 65 |
| 4.7.4 代码实现：用第三方函数将图片旋转 30° | 65 |
| 4.7.5 代码实现：定义函数，生成 Dataset 对象 | 66 |
| 4.7.6 代码实现：建立会话，输出数据..... | 67 |
| 4.7.7 运行程序 | 68 |
| 4.8 实例 8：将 TFRecord 文件制作成 Dataset 数据集 | 69 |
| 4.8.1 样本介绍 | 69 |
| 4.8.2 代码实现：定义函数，生成 Dataset 对象..... | 70 |
| 4.8.3 代码实现：建立会话输出数据..... | 71 |
| 4.8.4 运行程序 | 72 |
| 4.9 实例 9：在动态图中读取 Dataset 数据集..... | 72 |
| 4.9.1 代码实现：添加动态图调用..... | 72 |
| 4.9.2 制作数据集 | 73 |
| 4.9.3 代码实现：在动态图中显示数据..... | 73 |
| 4.9.4 实例 10：在 TensorFlow 2.x 中操作数据集..... | 74 |
| 4.10 实例 11：在不同场景中使用数据集 | 77 |
| 4.10.1 代码实现：在训练场景中使用数据集..... | 78 |
| 4.10.2 代码实现：在应用模型场景中使用数据集 | 79 |
| 4.10.3 代码实现：在训练与测试混合场景中使用数据集 | 80 |
| 4.11 <code>tf.data.Dataset</code> 接口的更多应用 | 81 |
| 第 5 章 10 分钟快速训练自己的图片分类模型 | 82 |
| 5.1 快速导读 | 82 |
| 5.1.1 认识模型和模型检查点文件 | 82 |
| 5.1.2 了解“预训练模型”与微调（Fine-Tune） | 82 |
| 5.1.3 学习 TensorFlow 中的预训练模型库——TF-Hub 库 | 83 |
| 5.2 实例 12：通过微调模型分辨男女 | 83 |
| 5.2.1 准备工作 | 84 |

| | |
|--|------------|
| 5.2.2 代码实现：处理样本数据并生成 Dataset 对象 | 85 |
| 5.2.3 代码实现：定义微调模型的类 MyNASNetModel | 88 |
| 5.2.4 代码实现：构建 MyNASNetModel 类中的基本模型 | 88 |
| 5.2.5 代码实现：实现 MyNASNetModel 类中的微调操作 | 89 |
| 5.2.6 代码实现：实现与训练相关的其他方法 | 90 |
| 5.2.7 代码实现：构建模型，用于训练、测试、使用 | 92 |
| 5.2.8 代码实现：通过二次迭代来训练微调模型 | 94 |
| 5.2.9 代码实现：测试模型 | 96 |
| 5.3 扩展：通过摄像头实时分辨男女 | 100 |
| 5.4 TF-slim 接口中的更多成熟模型 | 100 |
| 5.5 实例 13：用 TF-Hub 库微调模型以评估人物的年龄 | 100 |
| 5.5.1 准备样本 | 101 |
| 5.5.2 下载 TF-Hub 库中的模型 | 102 |
| 5.5.3 代码实现：测试 TF-Hub 库中的 MobileNet_V2 模型 | 104 |
| 5.5.4 用 TF-Hub 库微调 MobileNet_V2 模型 | 107 |
| 5.5.5 代码实现：用模型评估人物的年龄 | 109 |
| 5.5.6 扩展：用 TF-Hub 库中的其他模型处理不同领域的分类任务 | 113 |
| 5.6 总结 | 113 |
| 5.7 练习题 | 114 |
| 5.7.1 基于 TF-slim 接口的练习 | 115 |
| 5.7.2 基于 TF-Hub 库的练习 | 115 |
| 第 6 章 用 TensorFlow 编写训练模型的程序 | 117 |
| 6.1 快速导读 | 117 |
| 6.1.1 训练模型是怎么一回事 | 117 |
| 6.1.2 用“静态图”方式训练模型 | 117 |
| 6.1.3 用“动态图”方式训练模型 | 118 |
| 6.1.4 什么是估算器框架接口（Estimators API） | 119 |
| 6.1.5 什么是 tf.layers 接口 | 120 |
| 6.1.6 什么是 tf.keras 接口 | 121 |
| 6.1.7 什么是 tf.js 接口 | 122 |
| 6.1.8 什么是 TFLearn 框架 | 123 |
| 6.1.9 该选择哪种框架 | 123 |
| 6.1.10 分配运算资源与使用分布策略 | 124 |
| 6.1.11 用 tfdbg 调试 TensorFlow 模型 | 127 |
| 6.1.12 用钩子函数（Training_Hooks）跟踪训练状态 | 127 |
| 6.1.13 用分布式运行方式训练模型 | 128 |

| | |
|--|------------|
| 6.1.14 用 T2T 框架系统更方便地训练模型 | 128 |
| 6.1.15 将 TensorFlow 1.x 中的代码移植到 2.x 版本 | 129 |
| 6.1.16 TensorFlow 2.x 中的新特性——自动图 | 130 |
| 6.2 实例 14：用静态图训练一个具有保存检查点功能的回归模型 | 131 |
| 6.2.1 准备开发步骤 | 131 |
| 6.2.2 生成检查点文件 | 131 |
| 6.2.3 载入检查点文件 | 132 |
| 6.2.4 代码实现：在线性回归模型中加入保存检查点功能 | 132 |
| 6.2.5 修改迭代次数，二次训练 | 135 |
| 6.3 实例 15：用动态图（eager）训练一个具有保存检查点功能的回归模型 | 136 |
| 6.3.1 代码实现：启动动态图，生成模拟数据 | 136 |
| 6.3.2 代码实现：定义动态图的网络结构 | 137 |
| 6.3.3 代码实现：在动态图中加入保存检查点功能 | 138 |
| 6.3.4 代码实现：按指定迭代次数进行训练，并可视化结果 | 139 |
| 6.3.5 运行程序，显示结果 | 140 |
| 6.3.6 代码实现：用另一种方法计算动态图梯度 | 141 |
| 6.3.7 实例 16：在动态图中获取参数变量 | 142 |
| 6.3.8 小心动态图中的参数陷阱 | 144 |
| 6.3.9 实例 17：在静态图中使用动态图 | 145 |
| 6.4 实例 18：用估算器框架训练一个回归模型 | 147 |
| 6.4.1 代码实现：生成样本数据集 | 147 |
| 6.4.2 代码实现：设置日志级别 | 148 |
| 6.4.3 代码实现：实现估算器的输入函数 | 148 |
| 6.4.4 代码实现：定义估算器的模型函数 | 149 |
| 6.4.5 代码实现：通过创建 config 文件指定硬件的运算资源 | 151 |
| 6.4.6 代码实现：定义估算器 | 152 |
| 6.4.7 用 tf.estimator.RunConfig 控制更多的训练细节 | 153 |
| 6.4.8 代码实现：用估算器训练模型 | 153 |
| 6.4.9 代码实现：通过热启动实现模型微调 | 155 |
| 6.4.10 代码实现：测试估算器模型 | 158 |
| 6.4.11 代码实现：使用估算器模型 | 158 |
| 6.4.12 实例 19：为估算器添加日志钩子函数 | 159 |
| 6.5 实例 20：将估算器代码改写成静态图代码 | 161 |
| 6.5.1 代码实现：复制网络结构 | 161 |
| 6.5.2 代码实现：重用输入函数 | 163 |
| 6.5.3 代码实现：创建会话恢复模型 | 163 |
| 6.5.4 代码实现：继续训练 | 163 |

| | |
|---|-----|
| 6.6 实例 21：用 tf.layers API 在动态图上识别手写数字 | 165 |
| 6.6.1 代码实现：启动动态图并加载手写图片数据集 | 165 |
| 6.6.2 代码实现：定义模型的类 | 166 |
| 6.6.3 代码实现：定义网络的反向传播 | 167 |
| 6.6.4 代码实现：训练模型 | 167 |
| 6.7 实例 22：用 tf.keras API 训练一个回归模型 | 168 |
| 6.7.1 代码实现：用 model 类搭建模型 | 168 |
| 6.7.2 代码实现：用 sequential 类搭建模型 | 169 |
| 6.7.3 代码实现：搭建反向传播的模型 | 171 |
| 6.7.4 代码实现：用两种方法训练模型 | 172 |
| 6.7.5 代码实现：获取模型参数 | 172 |
| 6.7.6 代码实现：测试模型与用模型进行预测 | 173 |
| 6.7.7 代码实现：保存模型与加载模型 | 173 |
| 6.7.8 代码实现：将模型导出成 JSON 文件，再将 JSON 文件导入模型 | 175 |
| 6.7.9 实例 23：在 tf.keras 接口中使用预训练模型 ResNet | 176 |
| 6.7.10 扩展：在动态图中使用 tf.keras 接口 | 178 |
| 6.7.11 实例 24：在静态图中使用 tf.keras 接口 | 178 |
| 6.8 实例 25：用 tf.js 接口后方训练一个回归模型 | 180 |
| 6.8.1 代码实现：在 HTTP 的头标签中添加 tfjs 模块 | 180 |
| 6.8.2 代码实现：用 JavaScript 脚本实现回归模型 | 181 |
| 6.8.3 运行程序：在浏览器中查看效果 | 181 |
| 6.8.4 扩展：tf.js 接口的应用场景 | 182 |
| 6.9 实例 26：用估算器框架实现分布式部署训练 | 182 |
| 6.9.1 运行程序：修改估算器模型，使其支持分布式 | 182 |
| 6.9.2 通过 TF_CONFIG 进行分布式配置 | 183 |
| 6.9.3 运行程序 | 185 |
| 6.9.4 扩展：用分布策略或 KubeFlow 框架进行分布式部署 | 186 |
| 6.10 实例 27：在分布式估算器框架中用 tf.keras 接口训练 ResNet 模型， 识别图片中是橘子还是苹果 | 186 |
| 6.10.1 样本准备 | 186 |
| 6.10.2 代码实现：准备训练与测试数据集 | 187 |
| 6.10.3 代码实现：制作模型输入函数 | 187 |
| 6.10.4 代码实现：搭建 ResNet 模型 | 188 |
| 6.10.5 代码实现：训练分类器模型 | 189 |
| 6.10.6 运行程序：评估模型 | 190 |
| 6.10.7 扩展：全连接网络的优化 | 190 |

| | |
|---|-----|
| 6.11 实例 28：在 T2T 框架中用 tf.layers 接口实现 MNIST 数据集分类..... | 191 |
| 6.11.1 代码实现：查看 T2T 框架中的数据集（problems）..... | 191 |
| 6.11.2 代码实现：构建 T2T 框架的工作路径及下载数据集..... | 192 |
| 6.11.3 代码实现：在 T2T 框架中搭建自定义卷积网络模型..... | 193 |
| 6.11.4 代码实现：用动态图方式训练自定义模型..... | 194 |
| 6.11.5 代码实现：在动态图中用 metrics 模块评估模型..... | 195 |
| 6.12 实例 29：在 T2T 框架中，用自定义数据集训练中英文翻译模型..... | 196 |
| 6.12.1 代码实现：声明自己的 problems 数据集..... | 196 |
| 6.12.2 代码实现：定义自己的 problems 数据集..... | 197 |
| 6.12.3 在命令行下生成 TFRecord 格式的数据..... | 198 |
| 6.12.4 查找 T2T 框架中的模型及超参，并用指定的模型及超参进行训练..... | 199 |
| 6.12.5 用训练好的 T2T 框架模型进行预测..... | 201 |
| 6.12.6 扩展：在 T2T 框架中，如何选取合适的模型及超参..... | 202 |
| 6.13 实例 30：将 TensorFlow 1.x 中的代码升级为可用于 2.x 版本的代码..... | 203 |
| 6.13.1 准备工作：创建 Python 虚环境..... | 203 |
| 6.13.2 使用工具转换源码..... | 204 |
| 6.13.3 修改转换后的代码文件..... | 204 |
| 6.13.4 将代码升级到 TensorFlow 2.x 版本的经验总结..... | 205 |

第 3 篇 进阶

| | |
|--------------------------------------|-----|
| 第 7 章 特征工程——会说话的数据..... | 208 |
| 7.1 快速导读 | 208 |
| 7.1.1 特征工程的基础知识 | 208 |
| 7.1.2 离散数据特征与连续数据特征 | 209 |
| 7.1.3 了解特征列接口 | 210 |
| 7.1.4 了解序列特征列接口 | 210 |
| 7.1.5 了解弱学习器接口——梯度提升树（TFBT 接口） | 210 |
| 7.1.6 了解特征预处理模块（tf.Transform） | 211 |
| 7.1.7 了解因子分解模块 | 212 |
| 7.1.8 了解加权矩阵分解算法 | 212 |
| 7.1.9 了解 Lattice 模块——点阵模型 | 213 |
| 7.1.10 联合训练与集成学习 | 214 |
| 7.2 实例 31：用 wide_deep 模型预测人口收入 | 214 |
| 7.2.1 了解人口收入数据集 | 214 |
| 7.2.2 代码实现：探索性数据分析 | 217 |
| 7.2.3 认识 wide_deep 模型 | 218 |

| | |
|--|-----|
| 7.2.4 部署代码文件 | 219 |
| 7.2.5 代码实现：初始化样本常量 | 220 |
| 7.2.6 代码实现：生成特征列 | 220 |
| 7.2.7 代码实现：生成估算器模型 | 222 |
| 7.2.8 代码实现：定义输入函数 | 223 |
| 7.2.9 代码实现：定义用于导出冻结图文件的函数 | 224 |
| 7.2.10 代码实现：定义类，解析启动参数 | 225 |
| 7.2.11 代码实现：训练和测试模型 | 226 |
| 7.2.12 代码实现：使用模型 | 227 |
| 7.2.13 运行程序 | 228 |
| 7.3 实例 32：用弱学习器中的梯度提升树算法预测人口收入 | 229 |
| 7.3.1 代码实现：为梯度提升树模型准备特征列 | 230 |
| 7.3.2 代码实现：构建梯度提升树模型 | 230 |
| 7.3.3 代码实现：训练并导出梯度提升树模型 | 231 |
| 7.3.4 代码实现：设置启动参数，运行程序 | 232 |
| 7.3.5 扩展：更灵活的 TFBT 接口 | 233 |
| 7.4 实例 33：用 feature_column 模块转换特征列 | 233 |
| 7.4.1 代码实现：用 feature_column 模块处理连续值特征列 | 234 |
| 7.4.2 代码实现：将连续值特征列转化成离散值特征列 | 237 |
| 7.4.3 代码实现：将离散文本特征列转化为 one-hot 与词向量 | 239 |
| 7.4.4 代码实现：根据特征列生成交叉列 | 246 |
| 7.5 实例 34：用 sequence_feature_column 接口完成自然语言处理任务的数据预处理工作 | 248 |
| 7.5.1 代码实现：构建模拟数据 | 248 |
| 7.5.2 代码实现：构建词嵌入初始值 | 249 |
| 7.5.3 代码实现：构建词嵌入特征列与共享特征列 | 249 |
| 7.5.4 代码实现：构建序列特征列的输入层 | 250 |
| 7.5.5 代码实现：建立会话输出结果 | 251 |
| 7.6 实例 35：用 factorization 模块的 kmeans 接口聚类 COCO 数据集中的标注框 | 253 |
| 7.6.1 代码实现：设置要使用的数据集 | 253 |
| 7.6.2 代码实现：准备带聚类的数据样本 | 253 |
| 7.6.3 代码实现：定义聚类模型 | 255 |
| 7.6.4 代码实现：训练模型 | 256 |
| 7.6.5 代码实现：输出图示化结果 | 256 |
| 7.6.6 代码实现：提取并排序聚类结果 | 258 |
| 7.6.7 扩展：聚类与神经网络混合训练 | 258 |

| | |
|---|------------|
| 7.7 实例 36：用加权矩阵分解模型实现基于电影评分的推荐系统..... | 259 |
| 7.7.1 下载并加载数据集 | 259 |
| 7.7.2 代码实现：根据用户和电影特征列生成稀疏矩阵 | 260 |
| 7.7.3 代码实现：建立 WALS 模型，并对其进行训练 | 261 |
| 7.7.4 代码实现：评估 WALS 模型 | 263 |
| 7.7.5 代码实现：用 WALS 模型为用户推荐电影 | 264 |
| 7.7.6 扩展：使用 WALS 的估算器接口 | 265 |
| 7.8 实例 37：用 Lattice 模块预测人口收入 | 265 |
| 7.8.1 代码实现：读取样本，并创建输入函数 | 266 |
| 7.8.2 代码实现：创建特征列，并保存校准关键点 | 267 |
| 7.8.3 代码实现：创建校准线性模型 | 270 |
| 7.8.4 代码实现：创建校准点阵模型 | 270 |
| 7.8.5 代码实现：创建随机微点阵模型 | 271 |
| 7.8.6 代码实现：创建集合的微点阵模型 | 271 |
| 7.8.7 代码实现：定义评估与训练函数 | 272 |
| 7.8.8 代码实现：训练并评估模型 | 273 |
| 7.8.9 扩展：将点阵模型嵌入神经网络中 | 274 |
| 7.9 实例 38：结合知识图谱实现基于电影的推荐系统 | 278 |
| 7.9.1 准备数据集 | 278 |
| 7.9.2 预处理数据 | 279 |
| 7.9.3 搭建 MKR 模型 | 279 |
| 7.9.4 训练模型并输出结果 | 286 |
| 7.10 可解释性算法的意义 | 286 |
| 第 8 章 卷积神经网络 (CNN) —— 在图像处理中应用最广泛的模型 | 287 |
| 8.1 快速导读 | 287 |
| 8.1.1 认识卷积神经网络 | 287 |
| 8.1.2 什么是空洞卷积 | 288 |
| 8.1.3 什么是深度卷积 | 290 |
| 8.1.4 什么是深度可分离卷积 | 290 |
| 8.1.5 了解卷积网络的缺陷及补救方法 | 291 |
| 8.1.6 了解胶囊神经网络与动态路由 | 292 |
| 8.1.7 了解矩阵胶囊网络与 EM 路由算法 | 297 |
| 8.1.8 什么是 NLP 任务 | 298 |
| 8.1.9 了解多头注意力机制与内部注意力机制 | 298 |
| 8.1.10 什么是带有位置向量的词嵌入 | 300 |
| 8.1.11 什么是目标检测任务 | 300 |

| | | |
|--------|------------------------------------|-----|
| 8.1.12 | 什么是目标检测中的上采样与下采样 | 301 |
| 8.1.13 | 什么是图片分割任务 | 301 |
| 8.2 | 实例 39：用胶囊网络识别黑白图中服装的图案 | 302 |
| 8.2.1 | 熟悉样本：了解 Fashion-MNIST 数据集 | 302 |
| 8.2.2 | 下载 Fashion-MNIST 数据集 | 303 |
| 8.2.3 | 代码实现：读取及显示 Fashion-MNIST 数据集中的数据 | 304 |
| 8.2.4 | 代码实现：定义胶囊网络模型类 CapsuleNetModel | 305 |
| 8.2.5 | 代码实现：实现胶囊网络的基本结构 | 306 |
| 8.2.6 | 代码实现：构建胶囊网络模型 | 309 |
| 8.2.7 | 代码实现：载入数据集，并训练胶囊网络模型 | 310 |
| 8.2.8 | 代码实现：建立会话训练模型 | 311 |
| 8.2.9 | 运行程序 | 313 |
| 8.2.10 | 实例 40：实现带有 EM 路由的胶囊网络 | 314 |
| 8.3 | 实例 41：用 TextCNN 模型分析评论者是否满意 | 322 |
| 8.3.1 | 熟悉样本：了解电影评论数据集 | 322 |
| 8.3.2 | 熟悉模型：了解 TextCNN 模型 | 322 |
| 8.3.3 | 数据预处理：用 preprocessing 接口制作字典 | 323 |
| 8.3.4 | 代码实现：生成 NLP 文本数据集 | 326 |
| 8.3.5 | 代码实现：定义 TextCNN 模型 | 327 |
| 8.3.6 | 代码实现：训练 TextCNN 模型 | 330 |
| 8.3.7 | 运行程序 | 332 |
| 8.3.8 | 扩展：提升模型精度的其他方法 | 333 |
| 8.4 | 实例 42：用带注意力机制的模型分析评论者是否满意 | 333 |
| 8.4.1 | 熟悉样本：了解 tf.keras 接口中的电影评论数据集 | 333 |
| 8.4.2 | 代码实现：将 tf.keras 接口中的 IMDB 数据集还原成句子 | 334 |
| 8.4.3 | 代码实现：用 tf.keras 接口开发带有位置向量的词嵌入层 | 336 |
| 8.4.4 | 代码实现：用 tf.keras 接口开发注意力层 | 338 |
| 8.4.5 | 代码实现：用 tf.keras 接口训练模型 | 340 |
| 8.4.6 | 运行程序 | 341 |
| 8.4.7 | 扩展：用 Targeted Dropout 技术进一步提升模型的性能 | 342 |
| 8.5 | 实例 43：搭建 YOLO V3 模型，识别图片中的酒杯、水果等物体 | 343 |
| 8.5.1 | YOLO V3 模型的样本与结构 | 343 |
| 8.5.2 | 代码实现：Darknet-53 模型的 darknet 块 | 344 |
| 8.5.3 | 代码实现：Darknet-53 模型的下采样卷积 | 345 |
| 8.5.4 | 代码实现：搭建 Darknet-53 模型，并返回 3 种尺度特征值 | 345 |
| 8.5.5 | 代码实现：定义 YOLO 检测模块的参数及候选框 | 346 |
| 8.5.6 | 代码实现：定义 YOLO 检测块，进行多尺度特征融合 | 347 |

| | |
|---|------------|
| 8.5.7 代码实现：将 YOLO 检测块的特征转化为 bbox attrs 单元 | 347 |
| 8.5.8 代码实现：实现 YOLO V3 的检测部分 | 349 |
| 8.5.9 代码实现：用非极大值抑制算法对检测结果去重 | 352 |
| 8.5.10 代码实现：载入预训练权重 | 355 |
| 8.5.11 代码实现：载入图片，进行目标实物的识别 | 356 |
| 8.5.12 运行程序 | 358 |
| 8.6 实例 44：用 YOLO V3 模型识别门牌号 | 359 |
| 8.6.1 工程部署：准备样本 | 359 |
| 8.6.2 代码实现：读取样本数据，并制作标签 | 359 |
| 8.6.3 代码实现：用 tf.keras 接口构建 YOLO V3 模型，并计算损失 | 364 |
| 8.6.4 代码实现：在动态图中训练模型 | 368 |
| 8.6.5 代码实现：用模型识别门牌号 | 372 |
| 8.6.6 扩展：标注自己的样本 | 374 |
| 8.7 实例 45：用 Mask R-CNN 模型定位物体的像素点 | 375 |
| 8.7.1 下载 COCO 数据集及安装 pycocotools | 376 |
| 8.7.2 代码实现：验证 pycocotools 及读取 COCO 数据集 | 377 |
| 8.7.3 拆分 Mask R-CNN 模型的处理步骤 | 383 |
| 8.7.4 工程部署：准备代码文件及模型 | 385 |
| 8.7.5 代码实现：加载数据构建模型，并输出模型权重 | 385 |
| 8.7.6 代码实现：搭建残差网络 ResNet | 387 |
| 8.7.7 代码实现：搭建 Mask R-CNN 模型的骨干网络 ResNet | 393 |
| 8.7.8 代码实现：可视化 Mask R-CNN 模型骨干网络的特征输出 | 396 |
| 8.7.9 代码实现：用特征金字塔网络处理骨干网络特征 | 400 |
| 8.7.10 计算 RPN 中的锚点 | 402 |
| 8.7.11 代码实现：构建 RPN | 403 |
| 8.7.12 代码实现：用非极大值抑制算法处理 RPN 的结果 | 405 |
| 8.7.13 代码实现：提取 RPN 的检测结果 | 410 |
| 8.7.14 代码实现：可视化 RPN 的检测结果 | 412 |
| 8.7.15 代码实现：在 MaskRCNN 类中对 ROI 区域进行分类 | 415 |
| 8.7.16 代码实现：金字塔网络的区域对齐层（ROIAlign）中的区域框与特征的匹配算法 | 416 |
| 8.7.17 代码实现：在金字塔网络的 ROIAlign 层中按区域边框提取内容 | 418 |
| 8.7.18 代码实现：调试并输出 ROIAlign 层的内部运算值 | 421 |
| 8.7.19 代码实现：对 ROI 内容进行分类 | 422 |
| 8.7.20 代码实现：用检测器 DetectionLayer 检测 ROI 内容，得到最终的实物矩形 | 426 |
| 8.7.21 代码实现：根据 ROI 内容进行实物像素分割 | 432 |
| 8.7.22 代码实现：用 Mask R-CNN 模型分析图片 | 436 |

| | |
|--|------------|
| 8.8 实例 46：训练 Mask R-CNN 模型，进行形状的识别 | 439 |
| 8.8.1 工程部署：准备代码文件及模型 | 440 |
| 8.8.2 样本准备：生成随机形状图片 | 440 |
| 8.8.3 代码实现：为 Mask R-CNN 模型添加损失函数 | 442 |
| 8.8.4 代码实现：为 Mask R-CNN 模型添加训练函数，使其支持微调与全网训练 | 444 |
| 8.8.5 代码实现：训练并使用模型 | 446 |
| 8.8.6 扩展：替换特征提取网络 | 449 |
| 第 9 章 循环神经网络（RNN）——处理序列样本的神经网络 | 450 |
| 9.1 快速导读 | 450 |
| 9.1.1 什么是循环神经网络 | 450 |
| 9.1.2 了解 RNN 模型的基础单元 LSTM 与 GRU | 451 |
| 9.1.3 认识 QRNN 单元 | 451 |
| 9.1.4 认识 SRU 单元 | 451 |
| 9.1.5 认识 IndRNN 单元 | 452 |
| 9.1.6 认识 JANET 单元 | 453 |
| 9.1.7 优化 RNN 模型的技巧 | 453 |
| 9.1.8 了解 RNN 模型中多项式分布的应用 | 453 |
| 9.1.9 了解注意力机制的 Seq2Seq 框架 | 454 |
| 9.1.10 了解 BahdanauAttention 与 LuongAttention | 456 |
| 9.1.11 了解单调注意力机制 | 457 |
| 9.1.12 了解混合注意力机制 | 458 |
| 9.1.13 了解 Seq2Seq 接口中的采样接口（Helper） | 460 |
| 9.1.14 了解 RNN 模型的 Wrapper 接口 | 460 |
| 9.1.15 什么是时间序列（TFTS）框架 | 461 |
| 9.1.16 什么是梅尔标度 | 461 |
| 9.1.17 什么是短时傅立叶变换 | 462 |
| 9.2 实例 47：搭建 RNN 模型，为女孩生成英文名字 | 463 |
| 9.2.1 代码实现：读取及处理样本 | 463 |
| 9.2.2 代码实现：构建 Dataset 数据集 | 466 |
| 9.2.3 代码实现：用 tf.keras 接口构建生成式 RNN 模型 | 467 |
| 9.2.4 代码实现：在动态图中训练模型 | 468 |
| 9.2.5 代码实现：载入检查点文件并用模型生成名字 | 469 |
| 9.2.6 扩展：用 RNN 模型编写文章 | 471 |
| 9.3 实例 48：用带注意力机制的 Seq2Seq 模型为图片添加内容描述 | 471 |
| 9.3.1 设计基于图片的 Seq2Seq | 471 |
| 9.3.2 代码实现：图片预处理——用 ResNet 提取图片特征并保存 | 472 |

| | |
|---|------------|
| 9.3.3 代码实现：文本预处理——过滤处理、字典建立、对齐与向量化处理 | 475 |
| 9.3.4 代码实现：创建数据集 | 477 |
| 9.3.5 代码实现：用 tf.keras 接口构建 Seq2Seq 模型中的编码器 | 477 |
| 9.3.6 代码实现：用 tf.keras 接口构建 Bahdanau 类型的注意力机制 | 478 |
| 9.3.7 代码实现：搭建 Seq2Seq 模型中的解码器 Decoder..... | 478 |
| 9.3.8 代码实现：在动态图中计算 Seq2Seq 模型的梯度 | 480 |
| 9.3.9 代码实现：在动态图中为 Seq2Seq 模型添加保存检查点功能 | 480 |
| 9.3.10 代码实现：在动态图中训练 Seq2Seq 模型..... | 481 |
| 9.3.11 代码实现：用多项式分布采样获取图片的内容描述 | 482 |
| 9.4 实例 49：用 IndRNN 与 IndyLSTM 单元制作聊天机器人 | 485 |
| 9.4.1 下载及处理样本 | 486 |
| 9.4.2 代码实现：读取样本，分词并创建字典 | 487 |
| 9.4.3 代码实现：对样本进行向量化、对齐、填充预处理 | 489 |
| 9.4.4 代码实现：在 Seq2Seq 模型中加工样本..... | 489 |
| 9.4.5 代码实现：在 Seq2Seq 模型中，实现基于 IndRNN 与 IndyLSTM 的 动态多层 RNN 编码器 | 491 |
| 9.4.6 代码实现：为 Seq2Seq 模型中的解码器创建 Helper | 491 |
| 9.4.7 代码实现：实现带有 Bahdanau 注意力、dropout、OutputProjectionWrapper 的解码器.... | 492 |
| 9.4.8 代码实现：在 Seq2Seq 模型中实现反向优化..... | 493 |
| 9.4.9 代码实现：创建带有钩子函数的估算器，并进行训练 | 494 |
| 9.4.10 代码实现：用估算器框架评估模型 | 496 |
| 9.4.11 扩展：用注意力机制的 Seq2Seq 模型实现中英翻译 | 498 |
| 9.5 实例 50：预测飞机发动机的剩余使用寿命..... | 498 |
| 9.5.1 准备样本 | 499 |
| 9.5.2 代码实现：预处理数据——制作数据集的输入样本与标签 | 500 |
| 9.5.3 代码实现：构建带有 JANET 单元的多层次动态 RNN 模型..... | 504 |
| 9.5.4 代码实现：训练并测试模型..... | 505 |
| 9.5.5 运行程序 | 507 |
| 9.5.6 扩展：为含有 JANET 单元的 RNN 模型添加注意力机制..... | 508 |
| 9.6 实例 51：将动态路由用于 RNN 模型，对路透社新闻进行分类 | 509 |
| 9.6.1 准备样本 | 509 |
| 9.6.2 代码实现：预处理数据——对齐序列数据并计算长度 | 510 |
| 9.6.3 代码实现：定义数据集 | 510 |
| 9.6.4 代码实现：用动态路由算法聚合信息 | 511 |
| 9.6.5 代码实现：用 IndyLSTM 单元搭建 RNN 模型..... | 513 |
| 9.6.6 代码实现：建立会话，训练网络..... | 514 |
| 9.6.7 扩展：用分级网络将文章（长文本数据）分类 | 515 |

| | |
|--|------------|
| 9.7 实例 52：用 TFTS 框架预测某地区每天的出生人数 | 515 |
| 9.7.1 准备样本 | 515 |
| 9.7.2 代码实现：数据预处理——制作 TFTS 框架中的读取器 | 515 |
| 9.7.3 代码实现：用 TFTS 框架定义模型，并进行训练 | 516 |
| 9.7.4 代码实现：用 TFTS 框架评估模型 | 517 |
| 9.7.5 代码实现：用模型进行预测，并将结果可视化 | 517 |
| 9.7.6 运行程序 | 518 |
| 9.7.7 扩展：用 TFTS 框架进行异常值检测 | 519 |
| 9.8 实例 53：用 Tacotron 模型合成中文语音（TTS） | 520 |
| 9.8.1 准备安装包及样本数据 | 520 |
| 9.8.2 代码实现：将音频数据分帧并转为梅尔频谱 | 521 |
| 9.8.3 代码实现：用多进程预处理样本并保存结果 | 523 |
| 9.8.4 拆分 Tacotron 网络模型的结构 | 525 |
| 9.8.5 代码实现：搭建 CBHG 网络 | 527 |
| 9.8.6 代码实现：构建带有混合注意力机制的模块 | 529 |
| 9.8.7 代码实现：构建自定义 wrapper | 531 |
| 9.8.8 代码实现：构建自定义采样器 | 534 |
| 9.8.9 代码实现：构建自定义解码器 | 537 |
| 9.8.10 代码实现：构建输入数据集 | 539 |
| 9.8.11 代码实现：构建 Tacotron 网络 | 542 |
| 9.8.12 代码实现：构建 Tacotron 网络模型的训练部分 | 545 |
| 9.8.13 代码实现：训练模型并合成音频文件 | 546 |
| 9.8.14 扩展：用 pypinyin 模块实现文字到声音的转换 | 551 |
| 第 4 篇 高级 | |
| 第 10 章 生成式模型——能够输出内容的模型 | 554 |
| 10.1 快速导读 | 554 |
| 10.1.1 什么是自编码网络模型 | 554 |
| 10.1.2 什么是对抗神经网络模型 | 554 |
| 10.1.3 自编码网络模型与对抗神经网络模型的关系 | 555 |
| 10.1.4 什么是批量归一化中的自适应模式 | 555 |
| 10.1.5 什么是实例归一化 | 556 |
| 10.1.6 了解 SwitchableNorm 及更多的归一化方法 | 556 |
| 10.1.7 什么是图像风格转换任务 | 557 |
| 10.1.8 什么是人脸属性编辑任务 | 558 |
| 10.1.9 什么是 TFgan 框架 | 558 |

| | |
|---|-----|
| 10.2 实例 54：构建 DeblurGAN 模型，将模糊相片变清晰..... | 559 |
| 10.2.1 获取样本..... | 559 |
| 10.2.2 准备 SwitchableNorm 算法模块..... | 560 |
| 10.2.3 代码实现：构建 DeblurGAN 中的生成器模型..... | 560 |
| 10.2.4 代码实现：构建 DeblurGAN 中的判别器模型..... | 562 |
| 10.2.5 代码实现：搭建 DeblurGAN 的完整结构..... | 563 |
| 10.2.6 代码实现：引入库文件，定义模型参数..... | 563 |
| 10.2.7 代码实现：定义数据集，构建正反向模型..... | 564 |
| 10.2.8 代码实现：计算特征空间损失，并将其编译到生成器模型的训练模型中..... | 566 |
| 10.2.9 代码实现：按指定次数训练模型..... | 568 |
| 10.2.10 代码实现：用模型将模糊相片变清晰..... | 569 |
| 10.2.11 练习题 | 572 |
| 10.2.12 扩展：DeblurGAN 模型的更多妙用 | 572 |
| 10.3 实例 55：构建 AttGAN 模型，对照片进行加胡子、加头帘、加眼镜、变年轻等修改 | 573 |
| 10.3.1 获取样本 | 573 |
| 10.3.2 了解 AttGAN 模型的结构 | 574 |
| 10.3.3 代码实现：实现支持动态图和静态图的数据集工具类 | 575 |
| 10.3.4 代码实现：将 CelebA 做成数据集 | 577 |
| 10.3.5 代码实现：构建 AttGAN 模型的编码器 | 581 |
| 10.3.6 代码实现：构建含有转置卷积的解码器模型 | 582 |
| 10.3.7 代码实现：构建 AttGAN 模型的判别器模型部分 | 584 |
| 10.3.8 代码实现：定义模型参数，并构建 AttGAN 模型 | 585 |
| 10.3.9 代码实现：定义训练参数，搭建正反向模型 | 587 |
| 10.3.10 代码实现：训练模型 | 592 |
| 10.3.11 实例 56：为人脸添加不同的眼镜 | 595 |
| 10.3.12 扩展：AttGAN 模型的局限性 | 597 |
| 10.4 实例 57：用 RNN.WGAN 模型模拟生成恶意请求 | 597 |
| 10.4.1 获取样本：通过 Panabit 设备获取恶意请求样本 | 597 |
| 10.4.2 了解 RNN.WGAN 模型 | 600 |
| 10.4.3 代码实现：构建 RNN.WGAN 模型 | 601 |
| 10.4.4 代码实现：训练指定长度的 RNN.WGAN 模型 | 607 |
| 10.4.5 代码实现：用长度依次递增的方式训练模型 | 612 |
| 10.4.6 运行代码 | 613 |
| 10.4.7 扩展：模型的使用及优化 | 614 |

| | |
|---|-----|
| 第 11 章 模型的攻与防——看似智能的 AI 也有脆弱的一面 | 616 |
| 11.1 快速导读 | 616 |
| 11.1.1 什么是 FGSM 方法 | 616 |
| 11.1.2 什么是 cleverhans 模块 | 616 |
| 11.1.3 什么是黑箱攻击 | 617 |
| 11.1.4 什么是基于雅可比矩阵的数据增强方法 | 618 |
| 11.1.5 什么是数据中毒攻击 | 620 |
| 11.2 实例 58：用 FGSM 方法生成样本，并攻击 PNASNet 模型， 让其将“狗”识别成“盘子” | 621 |
| 11.2.1 代码实现：创建 PNASNet 模型 | 621 |
| 11.2.2 代码实现：搭建输入层并载入图片，复现 PNASNet 模型的预测效果 | 623 |
| 11.2.3 代码实现：调整参数，定义图片的变化范围 | 624 |
| 11.2.4 代码实现：用梯度下降方式生成对抗样本 | 625 |
| 11.2.5 代码实现：用生成的样本攻击模型 | 626 |
| 11.2.6 扩展：如何防范攻击模型的行为 | 627 |
| 11.2.7 代码实现：将数据增强方式用在使用场景，以加固 PNASNet 模型，防范攻击 | 627 |
| 11.3 实例 59：击破数据增强防护，制作抗旋转对抗样本 | 629 |
| 11.3.1 代码实现：对输入的数据进行多次旋转 | 629 |
| 11.3.2 代码实现：生成并保存鲁棒性更好的对抗样本 | 630 |
| 11.3.3 代码实现：在 PNASNet 模型中比较对抗样本的效果 | 631 |
| 11.4 实例 60：以黑箱方式攻击未知模型 | 633 |
| 11.4.1 准备工程代码 | 633 |
| 11.4.2 代码实现：搭建通用模型框架 | 634 |
| 11.4.3 代码实现：搭建被攻击模型 | 637 |
| 11.4.4 代码实现：训练被攻击模型 | 638 |
| 11.4.5 代码实现：搭建替代模型 | 639 |
| 11.4.6 代码实现：训练替代模型 | 639 |
| 11.4.7 代码实现：黑箱攻击目标模型 | 641 |
| 11.4.8 扩展：利用黑箱攻击中的对抗样本加固模型 | 645 |

第 5 篇 实战——深度学习实际应用

| | |
|--|-----|
| 第 12 章 TensorFlow 模型制作——一种功能，多种身份 | 648 |
| 12.1 快速导读 | 648 |
| 12.1.1 详细分析检查点文件 | 648 |
| 12.1.2 什么是模型中的冻结图 | 649 |
| 12.1.3 什么是 TF Serving 模块与 saved_model 模块 | 649 |

| | |
|--|------------|
| 12.1.4 用编译子图 (defun) 提升动态图的执行效率..... | 649 |
| 12.1.5 什么是 TF_Lite 模块 | 652 |
| 12.1.6 什么是 TFjs-converter 模块..... | 653 |
| 12.2 实例 61：在源码与检查点文件分离的情况下，对模型进行二次训练..... | 653 |
| 12.2.1 代码实现：在线性回归模型中，向检查点文件中添加指定节点 | 654 |
| 12.2.2 代码实现：在脱离源码的情况下，用检查点文件进行二次训练 | 657 |
| 12.2.3 扩展：更通用的二次训练方法..... | 659 |
| 12.3 实例 62：导出/导入冻结图文件 | 661 |
| 12.3.1 熟悉 TensorFlow 中的 freeze_graph 工具脚本 | 661 |
| 12.3.2 代码实现：从线性回归模型中导出冻结图文件 | 662 |
| 12.3.3 代码实现：导入冻结图文件，并用模型进行预测 | 664 |
| 12.4 实例 63：逆向分析冻结图文件 | 665 |
| 12.4.1 使用 import_to_tensorboard 工具 | 666 |
| 12.4.2 用 TensorBoard 工具查看模型结构 | 666 |
| 12.5 实例 64：用 saved_model 模块导出与导入模型文件..... | 668 |
| 12.5.1 代码实现：用 saved_model 模块导出模型文件 | 668 |
| 12.5.2 代码实现：用 saved_model 模块导入模型文件 | 669 |
| 12.5.3 扩展：用 saved_model 模块导出带有签名的模型文件 | 670 |
| 12.6 实例 65：用 saved_model_cli 工具查看及使用 saved_model 模型..... | 672 |
| 12.6.1 用 show 参数查看模型 | 672 |
| 12.6.2 用 run 参数运行模型 | 673 |
| 12.6.3 扩展：了解 scan 参数的黑名单机制..... | 674 |
| 12.7 实例 66：用 TF-Hub 库导入、导出词嵌入模型文件..... | 674 |
| 12.7.1 代码实现：模拟生成通用词嵌入模型 | 674 |
| 12.7.2 代码实现：用 TF-Hub 库导出词嵌入模型 | 675 |
| 12.7.3 代码实现：导出 TF-Hub 模型 | 678 |
| 12.7.4 代码实现：用 TF-Hub 库导入并使用词嵌入模型 | 680 |
| 第 13 章 部署 TensorFlow 模型——模型与项目的深度结合..... | 681 |
| 13.1 快速导读 | 681 |
| 13.1.1 什么是 gRPC 服务与 HTTP/REST API | 681 |
| 13.1.2 了解 TensorFlow 对移动终端的支持 | 682 |
| 13.1.3 了解树莓派上的人工智能 | 683 |
| 13.2 实例 67：用 TF_Serving 部署模型并进行远程使用..... | 684 |
| 13.2.1 在 Linux 系统中安装 TF_Serving | 684 |
| 13.2.2 在多平台中用 Docker 安装 TF_Serving | 685 |
| 13.2.3 编写代码：固定模型的签名信息 | 686 |

| | |
|---|------------|
| 13.2.4 在 Linux 中开启 TF_Serving 服务 | 688 |
| 13.2.5 编写代码：用 gRPC 访问远程 TF_Serving 服务 | 689 |
| 13.2.6 用 HTTP/REST API 访问远程 TF_Serving 服务 | 691 |
| 13.2.7 扩展：关于 TF_Serving 的更多例子 | 694 |
| 13.3 实例 68：在安卓手机上识别男女 | 694 |
| 13.3.1 准备工程代码 | 694 |
| 13.3.2 微调预训练模型 | 695 |
| 13.3.3 搭建安卓开发环境 | 698 |
| 13.3.4 制作 lite 模型文件 | 701 |
| 13.3.5 修改分类器代码，并运行 App | 702 |
| 13.4 实例 69：在 iPhone 手机上识别男女并进行活体检测 | 703 |
| 13.4.1 搭建 iOS 开发环境 | 703 |
| 13.4.2 部署工程代码并编译 | 704 |
| 13.4.3 载入 Lite 模型，实现识别男女功能 | 706 |
| 13.4.4 代码实现：调用摄像头并采集视频流 | 707 |
| 13.4.5 代码实现：提取人脸特征 | 710 |
| 13.4.6 活体检测算法介绍 | 712 |
| 13.4.7 代码实现：实现活体检测算法 | 713 |
| 13.4.8 代码实现：完成整体功能并运行程序 | 714 |
| 13.5 实例 70：在树莓派上搭建一个目标检测器 | 717 |
| 13.5.1 安装树莓派系统 | 718 |
| 13.5.2 在树莓派上安装 TensorFlow | 721 |
| 13.5.3 编译并安装 Protobuf | 725 |
| 13.5.4 安装 OpenCV | 726 |
| 13.5.5 下载目标检测模型 SSDLite | 726 |
| 13.5.6 代码实现：用 SSDLite 模型进行目标检测 | 727 |
| 第 14 章 商业实例——科技源于生活，用于生活 | 730 |
| 14.1 实例 71：将特征匹配技术应用在商标识别领域 | 730 |
| 14.1.1 项目背景 | 730 |
| 14.1.2 技术方案 | 730 |
| 14.1.3 预处理图片——统一尺寸 | 731 |
| 14.1.4 用自编码网络加夹角余弦实现商标识别 | 731 |
| 14.1.5 用卷积网络加 triplet-loss 提升特征提取效果 | 731 |
| 14.1.6 进一步的优化空间 | 732 |
| 14.2 实例 72：用 RNN 抓取蠕虫病毒 | 732 |
| 14.2.1 项目背景 | 733 |

| | |
|---|------------|
| 14.2.2 判断是否恶意域名不能只靠域名..... | 733 |
| 14.2.3 如何识别恶意域名..... | 733 |
| 14.3 实例 73：迎宾机器人的技术关注点——体验优先..... | 734 |
| 14.3.1 迎宾机器人的产品背景..... | 734 |
| 14.3.2 迎宾机器人的实现方案..... | 734 |
| 14.3.3 迎宾机器人的同类产品..... | 736 |
| 14.4 实例 74：基于摄像头的路边停车场项目..... | 737 |
| 14.4.1 项目背景..... | 737 |
| 14.4.2 技术方案..... | 738 |
| 14.4.3 方案缺陷..... | 738 |
| 14.4.4 工程化补救方案..... | 738 |
| 14.5 实例 75：智能冰箱产品——硬件成本之痛..... | 739 |
| 14.5.1 智能冰箱系列的产品背景..... | 739 |
| 14.5.2 智能冰箱的技术基础..... | 740 |
| 14.5.3 真实的非功能性需求——低成本..... | 740 |
| 14.5.4 未来的技术趋势及应对策略..... | 741 |
| 第15章 机器视觉：通过 TensorFlow 实现人脸识别与车牌识别器..... | 667 |
| 15.1 机器视觉入门..... | 667 |
| 15.2 识别车牌：从图像采集到识别并输出识别结果..... | 668 |
| 15.3 识别行人：从行人检测到识别并输出行人识别结果..... | 673 |
| 15.4 识别车辆：从车辆检测到识别并输出车辆识别结果..... | 673 |
| 15.5 识别水果：从水果检测到识别并输出水果识别结果..... | 674 |
| 15.6 识别垃圾：从垃圾检测到识别并输出垃圾识别结果..... | 674 |
| 15.7 识别面部：从面部检测到识别并输出面部识别结果..... | 675 |
| 15.8 识别商品：从商品检测到识别并输出商品识别结果..... | 675 |
| 第16章 部署 TensorFlow 模型：如何将运行的深度模型部署到生产环境..... | 681 |
| 16.1 部署完成..... | 681 |
| 16.1.1 什么是TensorFlow服务与TensorFlow代理..... | 681 |
| 16.1.2 了解TensorFlow服务与TensorFlow代理..... | 682 |
| 16.1.3 在线部署部署上的人工智能领域识别流程及经验分享..... | 683 |
| 16.2 定制API：将TF-Serving集成到自己的应用中..... | 684 |
| 16.2.1 将TensorFlow模型部署到TF-Serving..... | 684 |
| 16.2.2 在Docker环境中部署TF-Serving..... | 685 |
| 16.2.3 在Docker上部署进阶的签名体系..... | 685 |

建议读者使用 TensorFlow 1.x 版本开发实际项目，一方面 2.x 技术所更新的技术，将 2.x 版本迭代到 2.3 以上，再考虑使用 2.x 版本开发实际项目。

另外，由于在 TensorFlow 1.x 版本中开发的神经网络在 TensorFlow 2.x 版本中不能直接运行，所以本节会展示了具体的转化方法。可以将 1.x 版本的代码转化为 2.x 版本的代码，同时分析了 2.x 与 1.x 版本的使用差异。

4. Python 和 TensorFlow

第 1 篇 准备

随着人工智能的发展，Python 语言越来越受关注。到目前为止，使用 Python 语言开发 AI 算法自己完成，已经成为一种趋势。

通过本篇内容，读者不仅可以对 TensorFlow 有一个初步的了解，还可以对如何用 TensorFlow 开发进行 AI 模型有一个大致的了解，为后面的学习做准备。

- 第 1 章 学习准备
- 第 2 章 搭建开发环境
- 第 3 章 实例 1：用 AI 模型来识别图像是桌子、猫、狗，还是其他

1. 对于读者：笔者是 Python 初学者，水平一般，曾学过 MTA，会写 Python 脚本而已，对深度学习一知半解。

推荐先从 Python 基础开始，有关基础知识请见 [第 2 章](#)，具体在前面开发章节学习即可。

① 第 1 章 学习准备

本章将向读者介绍 TensorFlow 1.x 未出世前的代表作“WolfPhoneT”相关内容。该部分将从 WolfPhoneT 的诞生背景入手，分析其成功的原因，以及 WolfPhoneT 在当时的影响力。同时，将对比 WolfPhoneT 与 TensorFlow 1.x 的区别，帮助读者理解 TensorFlow 1.x 的核心思想。最后，对 WolfPhoneT 的未来前景进行展望。

② 第 2 章 搭建开发环境

本章将向读者介绍 TensorFlow 1.x 的安装方法，以及如何搭建 TensorFlow 1.x 的开发环境。首先，将介绍 TensorFlow 1.x 的安装方法，包括通过 pip 安装 TensorFlow 1.x，以及通过源码安装 TensorFlow 1.x。其次，将介绍 TensorFlow 1.x 的搭建方法，包括通过 Anaconda 安装 TensorFlow 1.x，以及通过 Docker 容器安装 TensorFlow 1.x。最后，将通过一个简单的例子，帮助读者理解 TensorFlow 1.x 的核心思想。

本章 WolfPhoneT 的简介及版本

本章将向读者介绍 WolfPhoneT 的简介及版本。首先，将介绍 WolfPhoneT 的简介，包括 WolfPhoneT 的诞生背景、主要功能、以及 WolfPhoneT 的特点。其次，将介绍 WolfPhoneT 的版本，包括 WolfPhoneT 的 1.0 版本、2.0 版本、3.0 版本等。最后，将通过一个简单的例子，帮助读者理解 WolfPhoneT 的核心思想。

本章 WolfPhoneT 的原理及实现

本章将向读者介绍 WolfPhoneT 的原理及实现。首先，将介绍 WolfPhoneT 的原理，包括 WolfPhoneT 的核心思想、主要功能、以及 WolfPhoneT 的特点。其次，将介绍 WolfPhoneT 的实现，包括 WolfPhoneT 的核心思想、主要功能、以及 WolfPhoneT 的特点。

第 1 章

学习准备上菜

本章将介绍一些基本概念和常识，以及学习本书的方法。

1.1 TensorFlow 能做什么

TensorFlow 框架可以支持多种开发语言，可以在多种平台上部署。

- 在代码领域：可以支持 C、JavaScript、Go、Java、Python 等多种编程语言。
- 在应用平台领域：可以支持 Windows、Linux、Android、Mac 等。
- 在硬件应用领域：可以支持 X86 平台、ARM 平台、MIPS 平台、树莓派、iPhone、Android 手机平台等。
- 在应用部署领域：可以支持 Hadoop、Spark、Kubernetes 等大数据平台。

1. TensorFlow 的应用领域

从应用角度来看，用 TensorFlow 几乎可以搭建出来 AI 领域所能触及的各种网络模型。其中包括：

- NLP（自然语言处理）领域的分类、翻译、对话、摘要生成、模拟生成等。
- 图片处理领域的图片识别、像素语义分析、实物检测、模拟生成、压缩、超清还原、图片搜索、跨域生成等。
- 数值分析领域的异常值监测、模拟生成、时间序列预测、分类等。
- 语音领域的语音识别、声纹识别、TTS（语音合成）模拟合成等。
- 视频领域的分类识别、人物跟踪、模拟生成等。
- 音乐领域的生成音乐、识别类型等。

甚至还可以实现跨领域的文本转图像、图像转文本、根据视频生成文本摘要等。

2. 本书所介绍的 TensorFlow 内容

作为深度学习领域应用广泛的框架，TensorFlow 集成了多种高级接口，可以方便地进行开发、调试和部署。有的接口，甚至只通过命令行操作便可以实现定制化的 AI 模型。

本书将花大量篇幅来介绍这些高级接口的使用方法与技巧。

3. 本书所用到的 TensorFlow 版本

在书中，大部分的代码都是以 TensorFlow 1.x 版本来实现的。TensorFlow 1.x 目前比较稳定，

建议读者使用 TensorFlow 1.x 版本开发实际项目，并跟进 2.x 版本所更新的技术。待 2.x 版本迭代到 2.3 以上，再考虑使用 2.x 版本开发实际项目。

另外，由于在 TensorFlow 1.x 版本中开发的部分代码在 TensorFlow 2.x 版本中不能直接运行。所以本书也介绍了具体的转化方法，可以将 1.x 版本的代码转化为 2.x 版本的代码。同时还介绍了 2.x 与 1.x 版本的使用区别，并配有相关例子。

4. Python 和 TensorFlow 的关系

随着人工智能的兴起，Python 语言越来越受关注。到目前为止，使用 Python 语言开发 AI 项目已经成为一种行业趋势。

综合来看，在 TensorFlow 框架中用 Python 进行开发，是保持自己技术不被淘汰的上选。

1.2 学习 TensorFlow 的必备知识

随着智能化时代的到来，AI 的工程化与理论化逐渐分离的特点越来越明显。所以，如果想学好 TensorFlow，则需要先搞清楚自己的定位——是偏工程应用，还是偏理论研究。

1. 对于偏工程应用的读者

如果是偏工程应用的读者，就目前的各种集成 API 来看，主要需要编程技术与调试能力。推荐先从 Python 基础开始，将基础知识掌握扎实，可以让后面的开发事半功倍。



提示：

推荐作者的《Python 带我起飞——入门、进阶、商业实战》一书。该书涵盖了在 TensorFlow 开发过程中可能会遇到的各种 Python 语法，同时又去除了在深度学习中不常用的知识点，并配有 47 段教学视频，可以让零基础的读者以最少的时间迅速掌握语法。

接下来就是 TensorFlow 的基本 API 和基本网络模型的实现。



提示：

这里推荐作者的另一本书《深度学习之 TensorFlow——入门、原理与进阶实践》。该书有 96 个实例，涵盖了 AI 开发中所需的基础网络模型与基础技巧。全书无公式，所有理论都用大白话讲解，可以让读者顺畅地跟着实例做出真实结果。通过练习 96 个实例，读者可以成为一个中级的 AI 工程师。

最后就是对本书的学习了。本书中的例子和知识更偏重于端到端的工程交付，几乎涵盖了 AI 领域的各大主流应用，也分享了许多来源于实际项目的经验与技巧。读者在打牢基础之后，将有能力修改本书中的例子，并将它们运用到真实项目中。学会本书中的内容，可以让自己的职场身价有一个质的飞跃。

2. 对于偏理论研究的读者

研究工作者推动了社会的进步、行业的发展，值得人们尊敬。要想成为一名优秀的研究人

员，付出的精力会远远大于工程应用人员。本书并不能引导读者如何成为一个研究人员，但是可以在工作中起到催化器的作用。

本书中把深度学习实践过程中的很多细节和各种情况都进行了拆分和归类并用代码实现。研究者可以通过将这些代码拼凑起来，迅速地将自己的理论转化为代码实现，并验证结果。本书可以大大提升研究者将理论落地的进度。

如果是刚入行的研究者，同样也是建议先把编程基础打扎实。这个过程与偏工程应用的读者是一样的，没有捷径可走。另外，在《深度学习之 TensorFlow——入门、原理与进阶实战》一书中，还介绍了许多底层 API 的原理及实现。读者可以重点关注这部分，对于开发相对底层的神经网络算法，以及开发自己的深度学习框架会很有帮助。

1.3 学习技巧：跟读代码

要掌握好深度学习的知识，需要理论与实践相结合。但一定要目标明确，要知道自己花时间和精力做这件事情的目标是什么。

花了大量的时间研究理论、推导公式、阅读海量的论文，只能使自己更透彻地了解技术原理，但还需要配合一定的编程能力将理论知识转化成代码，这样才能真正体现出技术的价值。

与其学好理论再去研究代码，还不如直接从代码入手，将理论与编码能力同时提高。而其中的捷径就是跟读代码。因为它源于实践，用于实践。

在跟读代码过程中，有以下几点值得注意：

- 先从代码的语句来了解技术的原理。
- 如果遇到不懂的逻辑，则再去有针对性地查阅相关文献。
- 如果遇到已经封装好的底层代码，只要弄明白其输入、输出、能完成什么功能即可。
- 在没有阅读大量的代码之前，切记少去自己编写代码。从作者个人经验来看，提升自己快速编码能力的捷径确实是跟读代码。因为代码里包含了别人思考的成果、遇到的陷阱和凝聚的经验。这是提升自己编码能力的快捷通道。否则，你只有把前人经历过的事情再做一遍，才能到达同样的水平。

1.4 如何学习本书

本书从实用角度讲述了用 TensorFlow 开发人工智能项目。本书配有大量的实例，从样本制作到网络模型的导入、导出，覆盖了日常工作中的所有环节。每个实例都有对应的知识点。

每章都可以分为“快速导读”与“实例”两部分。

- “快速导读”部分介绍了本章实例所对应的理论知识。
- “实例”部分注重一步一步完成具体实例。对于希望快速上手的读者，直接使用书中的实例即可。

如果想了解更多相关的原理，推荐先阅读作者的另一本书《深度学习之 TensorFlow——入门、原理与进阶实战》，然后再学习每章的“快速导读”部分，这样会有一个透彻的理解。

第 2 章

搭建开发环境

本章主要介绍了搭建 TensorFlow 框架的方法。讲解用集成化的 Python 开发工具 Anaconda 来完成 Python 环境的整体部署，以及选择硬件配置、软件版本的相关知识。

2.1 准备硬件环境

本书中的实例大都是相对较大的模型，所以建议读者准备一个带有 GPU 的机器，并使用和 GPU 相配套的主板及电源。



提示：

在已有的主机上直接添加 GPU(尤其是在原有服务器上添加 GPU)，需要考虑以下问题：

- 主板的插槽是否支持。例如，需要 PCIE x16 (16 倍数) 的插槽。
- 芯片组是否支持。例如，需要 C610 系列或是更先进的芯片组。
- 电源是否支持。GPU 的功率一般都会很大，必须采用配套的电源。如果检查驱动已安装正常，但在系统中却找不到 GPU，则可以考虑是否是由于电源供电不足导致的。

如果不准备硬件，则可以用云服务的方式训练模型。云服务是需要单独购买的，且按使用时间收费。如果不需要频繁训练模型，则推荐使用这种方式。

读者在学习本书的过程中，需要频繁训练模型。如果使用云服务，则会花费较高的成本。建议直接购买一台带有 GPU 卡的机器会好一些。

1. 如何选择 GPU

(1) 如果是个人学习使用。

推荐选择英伟达公司生产的 GPU，型号最好高于 GTX1070。选择 GPU 还需要考虑显存的大小。推荐选择显存大于 8GB 的 GPU。这一点很重要，因为在运行大型神经网络时，系统默认将网络节点全部载入显存。如果显存不足，则会显示资源耗尽提示，导致程序不能正常运行。

(2) 如果企业级使用。

应根据运算需求量、具体业务，以及公司资金情况来综合考虑。

2. 是否需要安装多块 GPU

(1) 如果是个人学习使用。

不建议在一台机器上安装多块 GPU。可以直接用两块卡的资金购买一块高性能的 GPU，这种方式会更为划算。

(2) 如果是用于企业级使用。

如果一块高配置的 GPU 无法满足运算需求，则可以使用多块 GPU 协同计算。不过 TensorFlow 多卡协同机制并不能完全智能地将整体性能发挥出来。有时会出现只有一个 GPU 的运算负荷较大，其他卡的运算不饱和的情况（这种问题在 TensorFlow 较新的版本中，也逐步得到了改善）。可以通过定义运算策略或是手动分配运算任务的方式，让多 GPU 协同的运算效率更高（见 6.1.10 小节）。

如果一台服务器上的多卡协同计算仍然满足不了需求，则可以考虑分布式并行运算。当然，根据自身具体的硬件资源，也可以将现有的机器集群起来，进行分布式运算。

2.2 下载及安装 Anaconda

下面来详细介绍 Anaconda 的下载及安装方法。

1. 下载 Anaconda 开发工具

- (1) 通过 <https://www.anaconda.com> 来到 Anaconda 官网。
- (2) 单击右上角的 Download 按钮，如图 2-1 所示。



图 2-1 单击 Download 按钮

- (3) 在新弹出的页面中，单击文字链接“Anaconda Distribution”，如图 2-2 所示。

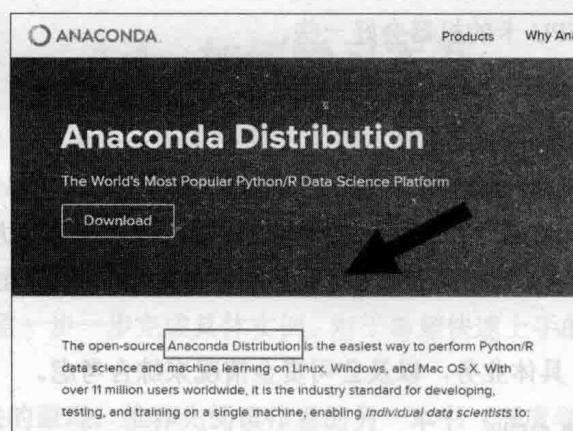


图 2-2 单击“Anaconda Distribution”链接

(4) 进入 Anaconda Distribution 页，单击页面中的链接“Old package lists”，如图 2-3 所示。

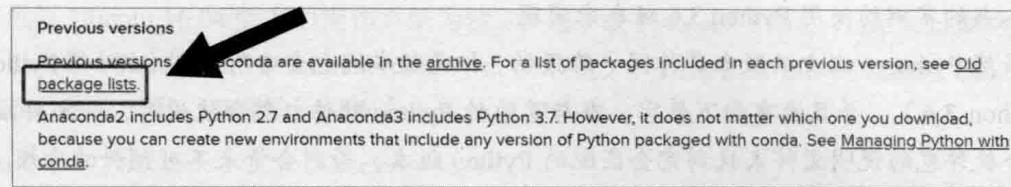


图 2-3 单击链接“Old package lists”

(5) 进入 Old package lists 页面，单击图中的链接“Anaconda installer archive”（如图 2-4 所示），下载完全版本。

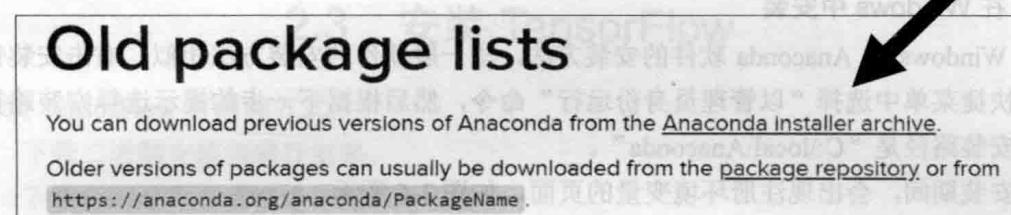


图 2-4 下载链接

(6) 完全版本的安装文件如图 2-5 所示。其中有 Linux、Windows、Mac OSX 的多种版本可供选择。以 Windows 64 位下的 Python 3.6 版本为例，对应的安装包为 Anaconda3-5.0.1-Windows-x86_64.exe（见图 2-5 中的标注）。

Anaconda installer archive

| Filename | Size | Last Modified | MD5 |
|---|--------|---------------------|----------------------------------|
| Anaconda2-5.1.0-Linux-ppc64le.sh | 267.3M | 2018-02-15 09:08:49 | e894dcc547a1c7d67deb04f6bba7223a |
| Anaconda2-5.1.0-Linux-x86.sh | 431.3M | 2018-02-15 09:08:51 | e26fb9d3e53049f6e32212270af6b987 |
| Anaconda2-5.1.0-Linux-x86_64.sh | 533.0M | 2018-02-15 09:08:50 | 5b1b5784cae93cf696e11e66983d8756 |
| Anaconda2-5.1.0-MacOSX-x86_64.pkg | 588.0M | 2018-02-15 09:08:52 | 4f9c197dfe6d3dc7e50a8611b4d3cfa2 |
| Anaconda2-5.1.0-MacOSX-x86_64.sh | 505.9M | 2018-02-15 09:08:53 | e9845ccf67542523c5be09552311666e |
| Anaconda2-5.1.0-Windows-x86.exe | 419.8M | 2018-02-15 09:08:55 | a09347a53e04a15ee965300c2b95dfde |
| Anaconda2-5.1.0-Windows-x86_64.exe | 522.6M | 2018-02-15 09:08:54 | b16dd6858fc7decf671ac71e6d7cfdb |
| Anaconda3-5.1.0-Linux-ppc64le.sh | 285.7M | 2018-02-15 09:08:56 | 47b5b21b7dbac0d4d0f0a4653f5b1c |
| Anaconda3-5.1.0-Linux-x86.sh | 449.7M | 2018-02-15 09:08:58 | 793a94ee85baf64d0ebb67a0c49af4d7 |
| Anaconda3-5.1.0-Linux-x86_64.sh | 551.2M | 2018-02-15 09:08:57 | 966406059cf7ed89cc82eb475ba506e5 |
| Anaconda3-5.1.0-MacOSX-x86_64.pkg | 594.7M | 2018-02-15 09:09:06 | 6ed496221b843d1b5fe8463d3136b649 |
| Anaconda3-5.1.0-MacOSX-x86_64.sh | 511.3M | 2018-02-15 09:10:24 | 047e12523fd287149ecd80c803598429 |
| Anaconda3-5.1.0-Windows-x86.exe | 435.5M | 2018-02-15 09:10:28 | 7a2291ab99178a4cdec530861494531f |
| Anaconda3-5.1.0-Windows-x86_64.exe | 537.1M | 2018-02-15 09:10:26 | 83a8b1edb21fa0ac481b23f65b604c6 |
| Anaconda2-5.0.1-Linux-x86.sh | 413.2M | 2017-10-24 12:13:07 | ae155b192027e23189d723a897782fa3 |
| Anaconda2-5.0.1-Linux-x86_64.sh | 507.7M | 2017-10-24 12:13:52 | dc13fe5502cd78dd03e8a727bb9be63f |
| Anaconda2-5.0.1-Windows-x86.exe | 403.4M | 2017-10-24 12:08:14 | 623e8d9ca2270cb9823a897dd0e9bfce |
| Anaconda3-5.0.1-Windows-x86.exe | 420.4M | 2017-10-24 12:37:10 | 9d2ffb0ac1f8a72ef4a5c535f3891f2 |
| Anaconda3-5.0.1-Windows-x86_64.exe | 514.8M | 2017-10-24 12:37:59 | 3dde7dbbef158db6dc44fce495671c92 |
| Anaconda2-5.0.1-MacOSX-x86_64.pkg | 562.8M | 2017-10-23 20:01:12 | 46fc99d1cf1e27f3b2a3eb63feela532 |
| Anaconda2-5.0.1-MacOSX-x86_64.sh | 486.5M | 2017-10-23 19:51:04 | 17314016dced36614a3bef8ff3db7066 |
| Anaconda2-5.0.1-Windows-x86_64.exe | 499.8M | 2017-10-23 21:57:22 | b8d9bc02edd61af3f7ece3d07e726e91 |

图 2-5 下载列表（部分）

**提示：**

本书的实例均使用 Python 3.6 版本来实现。

虽然 Python 3 以上的版本算作同一阶段的，但是版本间也会略有区别（例如：Python 3.5 与 Python 3.6），并且没有向下兼容。在与其他的 Python 软件包整合使用时，一定要按照所要整合软件包的说明文件来找到完全匹配的 Python 版本，否则会带来不可预料的麻烦。

另外，不同版本的 Anaconda 默认支持的 Python 版本是不一样的：支持 Python 2 的版本 Anaconda，统一以“Anaconda 2”为开头来命名；支持 Python 3 的版本 Anaconda，统一以“Anaconda 3”为开头来命名。当前最新的版本为 Anaconda 5.1.0，可以支持 Python 3.6 版本。

2. 在 Windows 中安装

在 Windows 中 Anaconda 软件的安装方法，与一般软件的安装方法相似。右击安装包，在弹出的快捷菜单中选择“以管理员身份运行”命令，然后根据下一步的提示选择安装路径。这里假设安装路径是“C:\local\Anaconda”。

在安装期间，会出现注册环境变量的页面，如图 2-6 所示。

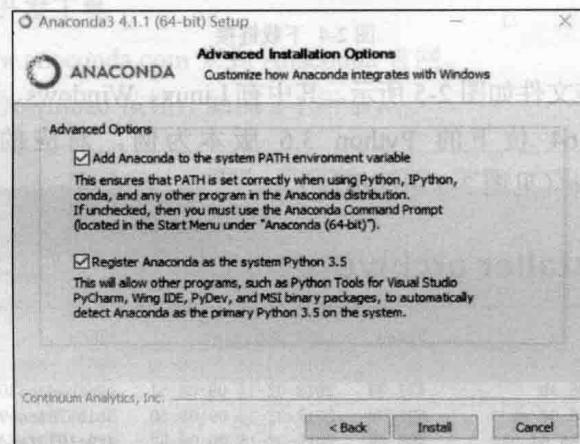


图 2-6 注册环境变量的页面

图 2-6 中有两个复选框，建议全部都勾选上，表示要注册环境变量。只有注册好环境变量，才可以在命令行下通过 Python 命令运行程序。

在安装 Anaconda 时，Python 常用的第三方库也会一起被安装了，路径如下：

```
C:\local\Anaconda3\Lib\site-packages
```

如果想要再安装其他的第三方库，可以使用 Anaconda 中自带的 pip 命令——在命令行下直接输入“`pip + 空格 + 第三方安装包名称`”。运行 pip 命令之后，系统会自动从网上下载相关的安装包，并安装到本机。例如，下面是在本机上安装 TensorFlow 的命令：

```
C:\Users\Administrator>pip install tensorflow
```

如果要卸载某个第三方安装包，直接将上一行命令中的 `install` 替换成 `uninstall` 即可。

3. 在 Linux 中安装

这里以 Ubuntu 16.04 版本的操作系统为例。首先下载 Python 3.6 版的 Anaconda 集成开发工具（可以下载 Anaconda3-5.1.0-Linux-x86_64.sh 安装包），然后在命令行终端通过 chmod 命令为其增加可执行权限，接着输入以下命令运行该安装包：

```
chmod u+x Anaconda3-5.1.0-Linux-x86_64.sh
./Anaconda3-5.1.0-Linux-x86_64.sh
```

在安装过程中，会有各种交互性提示。有的需要按 Enter 键，有的需要输入“yes”，按照提示来即可。

2.3 安装 TensorFlow

安装 TensorFlow 有两种方式：

- 下载二进制安装包进行安装。
- 下载源码进行手动编译，然后再安装。

第一种方式比较简单、稳定，适用于大多数的情况。第二种方式相对较难，容易出错，但灵活度更高，适用于定制化场景。

为了让读者可以快速上手，本节将介绍第一种安装方法。第二种安装方法见本书 13.5.2 小节。

1. 了解 TensorFlow 的 Nightly 版本与 Release 版本

在 GitHub 网站上，TensorFlow 项目的主页（<https://github.com/tensorflow/tensorflow>）中介绍了 TensorFlow 两种版本的安装包：Nightly 版本与 Release 版本。这两个版本的含义以下。

- **Nightly 版本：**TensorFlow 的源码更新非常活跃。为此，TensorFlow 开发团队搭建了一个自动构建版本的平台。该平台会定期（一般是一天一次）将最新的 TensorFlow 源码编译成二进制安装包。这个安装包被称为 Nightly 版本。
- **Release 版本：**当 Nightly 更新到一定程度，根据更新功能的完成量与当前版本的 BUG 情况，会推出一个阶段性的发布版本。这个版本被称为 Release 版本。

在 Nightly 版本中包含了 TensorFlow 的最新功能，但稳定性不如 Release 版本，所以它常用于提升自我技术的研究场景；Release 版本的稳定性更好，但功能相对滞后，常用于开发工程项目。

2. 下载 TensorFlow 的二进制安装包，并进行安装

在装好 Anaconda 之后，可以用 pip 命令安装 TensorFlow 了。这个步骤与系统无关。保持电脑联网状态即可。

(1) 安装 TensorFlow 的 Release 版本。

在命令行里输入以下命令：

```
pip install tensorflow-gpu
```

上面命令执行后，系统会将支持 GPU 的 TensorFlow Release 版本安装包下载到机器上，并进行安装。

如果是想安装 CPU 版本，则可以输入下列命令：

```
pip install tensorflow
```

如果想安装指定版本，则可以直接在命令后面加上版本号：

```
pip install tensorflow-gpu==1.13.1
```

该命令执行后，系统会将 1.13.1 版本的 TensorFlow 安装到本机。

(2) 安装 TensorFlow 的 Nightly 版本，可以使用以下命令：

```
pip install tf-nightly-gpu
```

安装 Nightly 的 gpu 版本

```
pip install tf-nightly
```

安装 Nightly 的 cpu 版本

还有更多关于 TensorFlow 的安装、卸载、更新方法，可以参考《深度学习之 TensorFlow——入门、原理与进阶实战》一书的 2.2 节。这里不再详述。



提示：

如果安装的是 GPU 版本，还需要按照 2.4 节的方法安装配套的开发包，才可以正常使用。

还有一种更简单的方式安装 GPU 版本的 TensorFlow。在安装完 Anaconda 软件后，直接使用以下命令：

```
conda install tensorflow-gpu
```

系统会自动把 TensorFlow 的 GPU 版本及对应的 NVIDIA 驱动安装到本机，不再需要按照 2.4 节的描述进行手动安装。

用 conda 命令安装虽然方便，但这不属于 TensorFlow 官方支持的安装方式。用这种方式只能安装比最新发布的版本滞后一些。如果想及时安装最新发布的 TensorFlow，还得用 pip 命令。

如果想查看 Anaconda 软件中集成的 TensorFlow 安装包版本，可以通过以下命令：

```
anaconda search -t conda tensorflow
```

2.4 GPU 版本的安装方法

如果用 pip 命令安装 TensorFlow 框架的 GPU 版本，还需要安装 CUDA 软件包和 CuDnn 库。如果是用 conda 命令安装 TensorFlow，则可以跳过此节。

2.4.1 在 Windows 中安装 CUDA

来到官方网站：<https://developer.nvidia.com/cuda-downloads>，如图 2-7 所示。

根据自己的环境选择对应的版本。以 Windows 为例，exe 文件分为网络版和本地版：

- 网络版安装包比较小，但是在安装过程中需要联网下载其他文件。
- 本地版安装包是直接下载完整安装包，下载之后就可以正常安装了。

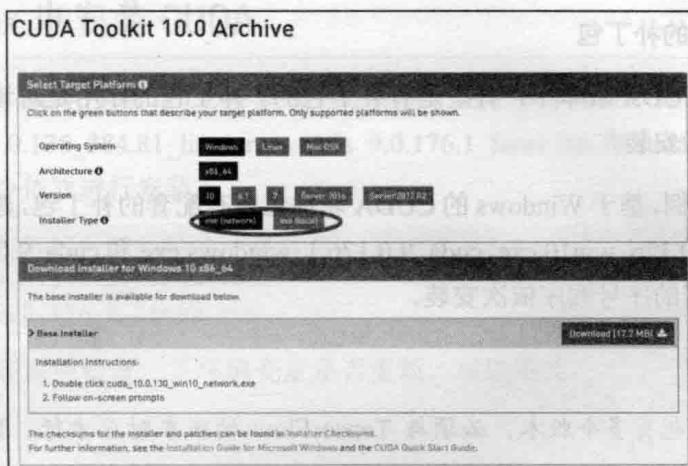


图 2-7 CUDA 页面

1. 安装 Visual Studio 以支持 CUDA 的更多工具包

CUDA 中的部分工具需要运行在 Visual Studio 之上。Visual Studio 是微软开发的集成化开发工具包。如果需要以源码编译的方式安装 TensorFlow，则建议安装 Visual Studio。否则也可以跳过该步骤。在安装 CUDA 过程中，如果出现如图 2-8 所示界面，则表明本机没有安装 Visual Studio。单击图 2-8 中的链接“Visual Studio”，即可下载 Visual Studio 工具包。

单击图 2-9 中的“免费下载”按钮，将“vs_community_1673162104.1537510790.exe”安装文件下载到本地。以管理员方式运行该安装文件进行安装。

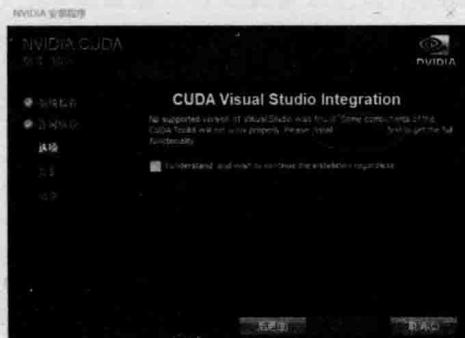


图 2-8 CUDA 提示页面

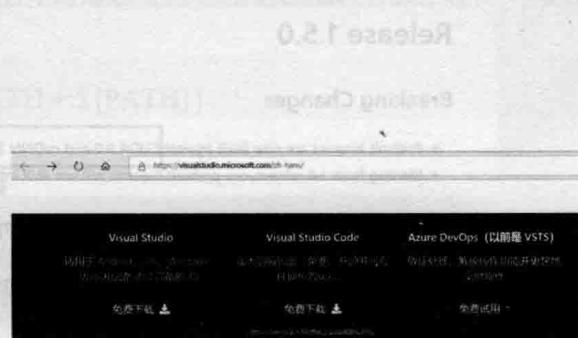


图 2-9 Visual Studio 的下载页面

安装过程需要保持网络畅通，系统需要从网络下载数据，如图 2-10 所示。



图 2-10 Visual Studio 的安装界面

2. 安装 CUDA 的补丁包

在已经发布的 CUDA 版本中，有些是有补丁包的。补丁包的作用是对该版本的功能扩充和问题修复。建议读者安装。

以 CUDA 9.0 为例，基于 Windows 的 CUDA 软件包带有配套的补丁包，建议一起下载下来。共 3 个文件：cuda_9.0.176_win10.exe、cuda_9.0.176.1_windows.exe 和 cuda_9.0.176.2_windows.exe，需要按照版本、补丁的序号顺序依次安装。



提示：

CUDA 软件包也有多个版本，必须与 TensorFlow 的版本对应才行。TensorFlow 版本与 CUDA 版本的对应关系如下：

- TensorFlow 1.0 至 1.4 版本只支持 CUDA 8.0。
- TensorFlow 1.5 至 1.12 版本支持支持 CUDA 9.0。
- TensorFlow 1.13 之后的版本，支持 CUDA10.0。

读者可以根据以下链接找到 CUDA 的更多版本：<https://developer.nvidia.com/cuda-toolkit-archive>。

另外，还可以根据以下网址找到 TensorFlow 版本对应的 CUDA 版本：<https://github.com/tensorflow/tensorflow/blob/master/RELEASE.md>。

图 2-11 显示的是 TensorFlow 1.5 版本支持 CUDA 9.0 和 cuDNN 7 版本。

Release 1.5.0

Breaking Changes

- Prebuilt binaries are now built against CUDA 9.0 and cuDNN 7.
- Starting from 1.6 release, our prebuilt binaries will use AVX instructions. This may break TF on older CPUs.

图 2-11 TensorFlow 发布页面

当然，如果选择编译源码的方式安装 TensorFlow，则可以随意指定所需要的 CUDA 版本。



提示：

如果要安装 TensorFlow 的 1.13 版本，则需要下载 CUDA10.0 进行安装（CUDA 的版本必须严格匹配，比如使用 10.1 的版本会报错误）。CUDA10.0 没有补丁包，直接安装即可。

如果本机已经装有 CUDA9.0，想要升级到 CUDA10.0，则可以在控制面板里将 CUDA9.0 相关的软件包卸载，再进行 CUDA10.0 的安装即可。

2.4.2 在 Linux 中安装 CUDA

以 Ubuntu 16.04 版本为例，CUDA 软件包还提供了两个补丁文件，建议一起下载下来。一共 3 个文件：cuda_9.0.176_384.81_linux.run、cuda_9.0.176.1_linux.run 和 cuda_9.0.176.2_linux.run。

然后用以下命令依次进行安装：

```
sudo sh cuda_9.0.176_384.81_linux.run
sudo sh cuda_9.0.176.1_linux.run
sudo sh cuda_9.0.176.2_linux.run
```

执行命令后，还需要检查一下环境变量是否更新。可以通过以下命令查看环境变量：

```
echo $PATH
```

执行后，会输出当前环境中的可执行目录，如下所示：

```
/root/anaconda3/bin:/root/anaconda3/bin:/usr/local/cuda-9.0/bin:/usr/local/sbin:
/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

从上面的信息中可以看到，新安装的 CUDA 生效的路径是：/usr/local/cuda-9.0/bin，表示安装正确。



提示：

执行“echo \$PATH”命令后，如果在本机输出的信息中没有安装好的 CUDA 文件夹，则需要手动在环境变量里添加。具体做法是：用 vim 编辑~/.bashrc 文件，将 CUDA 文件的路径添加到最后一行的变量 PATH 中。假设 CUDA 的路径是/usr/local/cuda-9.0/bin，则在~/.bashrc 文件中添加以下内容：

```
export PATH=/usr/local/cuda-9.0/bin${PATH:+:$PATH}
```

2.4.3 在 Windows 中安装 cuDNN

通过以下网址来到下载页面。需要注册并且填写问卷才能下载这个安装包。

<https://developer.nvidia.com/cudnn>

cuDNN 库的版本选择也是有规定的。以 Windows 10 操作系统为例，具体如下：

- TensorFlow 1.0 到 1.2 版本使用的是 cuDNN 5.1 版本（安装包为 cudnn-8.0-windows10-x64-v5.1.zip）。
- TensorFlow 1.3 和 1.4 版本使用的是 cuDNN 6.0 版本（安装包为 cudnn-8.0-windows10-x64-v6.0.zip）。
- TensorFlow 1.5 到 1.10 版本使用的是 cuDNN 7.0 版本（安装包为 cudnn-9.0-windows10-x64-v7.rar）。
- TensorFlow 1.11 和 1.12 版本使用的是 cuDNN 7.2 版本（安装包为 cudnn-9.0-windows10-x64-v7.2.1.38.zip）。

- TensorFlow 1.13 之后的版本使用的是 cuDNN 7.5 版本（安装包为 cudnn-10.0-windows10-x64-v7.5.0.56.zip）。

得到相关包后将其解压缩，并复制到 CUDA 路径对应的文件夹下，覆盖原有文件，如图 2-12 所示。

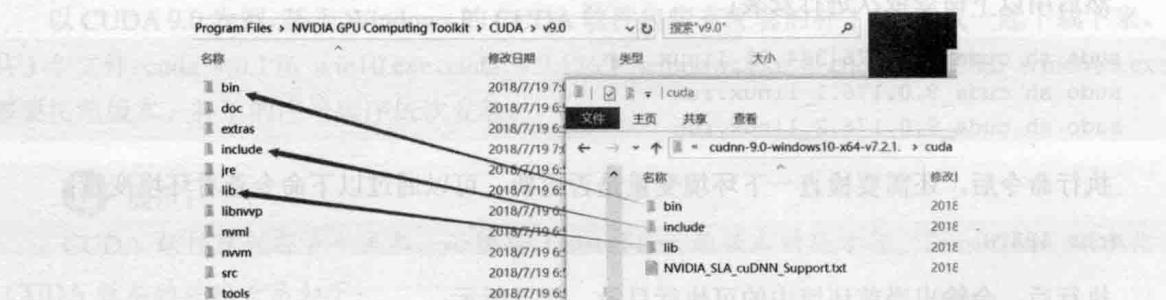


图 2-12 安装 cuDNN 库

2.4.4 在 Linux 中安装 cuDNN

这里介绍两种安装方法：自动安装与手动安装。

自动安装比较简单。不过由于 Linux 系统配置过于灵活，在某种特定的环境下，有可能会失败。而手动安装相对麻烦，但是不会出现失败问题。

(1) 自动安装。

使用自动安装时，需要下载 Deb 安装包。如图 2-13 所示，一定要选择开发库（Developer Library）的安装包，而不能选择运行时库（Runtime Library）的安装包。

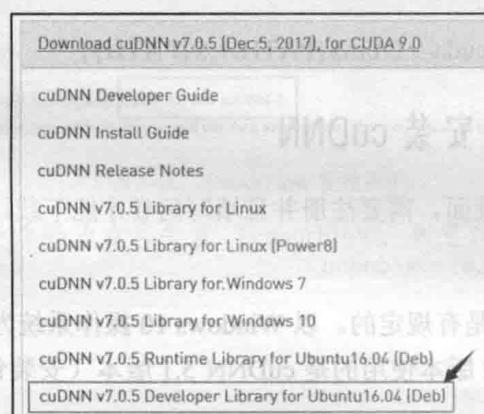


图 2-13 选择 cuDNN

下载完之后，输入以下命令即可进行安装：

```
sudo dpkg -i libcudnn7_7.0.5.15-1+cuda9.0_amd64.deb
sudo apt-get update
sudo apt-get install libcudnn7-dev
```

(2) 手动安装。

手动安装的方法与 Windows 中的安装方法非常相似，需要直接下载 cuDNN 的“Library for Linux”安装包，并将安装包里的文件手动复制到指定路径即可。以下载一个 CUDA 9.0 版本的 cuDNN 7.2.1 安装包为例，具体操作如下。



提示：

在 GitHub 上发布的 TensorFlow 新版本说明中用到的 cuDNN 版本，有时会在 NVIDIA 官网上找不到。例如，TensorFlow 1.11.0 版本使用的是 cuDNN 7.2.1，而在 NVIDIA 的官方网站上找不到 CUDA 9.0 版本的 cuDNN 7.2.1 下载链接，只有 CUDA 9.2 版本的 cuDNN 7.2.1。这时，可以将 9.2 版本对应的链接（如图 2-14 所示）复制下来，得到以下网址：

https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v7.2.1/prod/9.2_20180806/cudnn-9.2-linux-x64-v7.2.1.38

手动将 9.2 全部变成 9.0，一样可以下载。改后的网址如下：

https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v7.2.1/prod/9.0_20180806/cudnn-9.0-linux-x64-v7.2.1.38

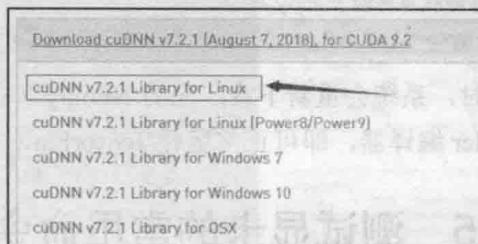


图 2-14 下载 cuDNN 库在 Linux 系统中的安装包

单击图 2-14 中箭头所指的链接，会将“cudnn-9.0-linux-x64-v7.2.1.38.jigsawpuzzle8”文件下载到本地。

下载完成后，将“cudnn-9.0-linux-x64-v7.2.1.38.jigsawpuzzle8”文件的扩展名改为 zip 并解压缩，会得到一个 cudnn-9.0-linux-x64-v7.2.1.38 文件。再继续将该文件的扩展名改为 zip 并解压缩，会得到真正的 cuDNN 库文件，如图 2-15 所示。

将其中的内容全部复制到 Linux 系统中 cuda-9.0 安装目标中对应的文件夹里，如图 2-16 所示。

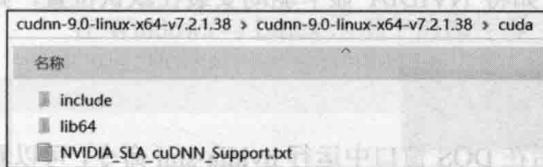


图 2-15 cuDNN 解压缩后的内容

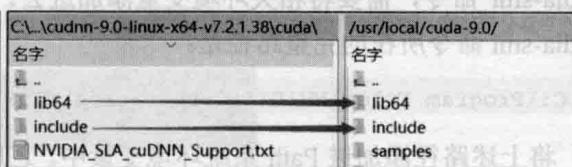


图 2-16 复制 cuDNN

当复制完成后，还要对库文件的权限进行修改。可以使用以下命令：

```
sudo chmod a+r /usr/local/cuda-9.0/include/cudnn.h /usr/local/cuda-9.0/lib64/libcudnn*
```

2.4.5 常见错误及解决方案

安装好 TensorFlow 的 GPU 版及配套的软件包后，在运行 TensorFlow 的代码时有时会出现 Numpy 库冲突的情况，如图 2-17 所示。

```
ModuleNotFoundError: No module named
'numpy.core._multiarray_umath'

Traceback (most recent call last):
  File "<frozen importlib._bootstrap>", line 968, in
    _find_and_load
    SystemError: <class '_frozen_importlib._ModuleLockManager'>
        returned a result with an error set

    ImportError: numpy.core._multiarray_umath failed to import
    ImportError: numpy.core.umath failed to import
```

图 2-17 Numpy 库冲突

出现这种情况是因为本地 Numpy 库的版本与 TensorFlow 所依赖的版本库不兼容。可以先将其卸载，再重新安装一次 TensorFlow 即可。具体命令如下：

```
conda uninstall numpy #卸载当前的Numpy库
pip install tensorflow-gpu==1.13.1 #重新安装TensorFlow
```

在重新安装 TensorFlow 时，系统会重新下载匹配的 Numpy 库并进行安装。待安装完成之后，重启 Anaconda 中的 Spyder 编译器，即可正常运行 TensorFlow 的代码。

2.5 测试显卡的常用命令

这里介绍几个小命令，它可以帮助读者定位在安装过程产生的问题。

1. 用 nvidia-smi 命令查看显卡信息

nvidia-smi 指的是 NVIDIA System Management Interface。该命令用于查看显卡的信息及运行情况。

(1) 在 Windows 系统中使用 nvidia-smi 命令。

在安装完成 NVIDIA 显卡驱动之后，对于 Windows 用户而言，DOS 窗口中还无法识别 nvidia-smi 命令，需要将相关环境变量添加进去。如将 NVIDIA 显卡驱动安装在默认位置，则 nvidia-smi 命令所在的完整路径是：

```
C:\Program Files\NVIDIA Corporation\NVSMI
```

将上述路径添加进 Path 系统环境变量中。之后在 DOS 窗口中运行 nvidia-smi 命令，可以看到如图 2-18 所示界面。

图中第 1 行是作者的驱动信息，第 3 行是显卡信息“GeForce GTX 1070”，第 4 行和第 5 行是当前使用显卡的进程。

如果这些信息都存在，则表示当前的安装是成功的。



提示：

在安装 CUDA 时，建议本机 NVIDIA 的显卡驱动更新到最新版本。否则，在执行 nvidia-smi 命令时有可能出现如下错误：

```
C:\Program Files\NVIDIA Corporation\NVSMI>nvidia-smi.exe
```

NVIDIA-SMI has failed because it couldn't communicate with the NVIDIA driver. Make sure that the latest NVIDIA driver is installed and running. This can also be happening if non-NVIDIA GPU is running as primary display, and NVIDIA GPU is in WDDM mode.

该错误表明本机 NVIDIA 的显卡驱动版本过老，不支持当前的 CUDA 版本。将驱动更新之后再运行“nvidia-smi”命令即可恢复正常。

(2) 在 Linux 系统中使用“nvidia-smi”命令。

在 Linux 系统中，可以通过在命令行里输入“nvidia-smi”来显示显卡信息，显示的信息如图 2-19 所示。

图 2-18 Windows 系统的显卡信息

图 2-19 Linux 系统的显卡信息



提示：

还可以用“nvidia-smi -l”命令实时查看显卡状态。

2. 查看 CUDA 的版本

在装完 CUDA 之后，可以通过以下命令来查看具体的版本：

```
nvcc -V
```

在 Windows 与 Linux 系统中的操作都一样，直接在命令行里输入命令即可，如图 2-20 所示。



图 2-20 查看 CUDA 版本

3. 查看 cuDNN 的版本

在装完 cuDNN 之后，可以通过查看 include 文件夹下的 cudnn.h 文件的代码找到具体的版本：

(1) 在 Windows 系统中查看 cuDNN 版本。

在 Windows 系统中找到 CUDA 安装路径下的 include 文件夹，打开 cudnn.h 文件，在里面

如果找到以下代码，则代表当前是 7 版本。

```
#define CUDNN_MAJOR 7
```

(2) 在 Linux 系统中查看 cuDNN 版本。

在 Linux 系统中，默认的安装路径是“/usr/local/cuda/include/cudnn.h”，在该路径下打开文件即可查看。

也可以使用以下命令：

```
root@user-NULL:~# cat /usr/local/cuda/include/cudnn.h | grep CUDNN_MAJOR -A 2
```

显示内容如图 2-21 所示。

```
root@user-NULL:~# cat /usr/local/cuda/include/cudnn.h | grep CUDNN_MAJOR -A 2
#define CUDNN_MAJOR 7
#define CUDNN_MINOR 0
#define CUDNN_PATCHLEVEL 5
...
#define CUDNN_VERSION (CUDNN_MAJOR * 1000 + CUDNN_MINOR * 100 + CUDNN_PATCHLEVEL)
```

图 2-21 查看 cuDNN 版本

在 Linux 和 MAC 系统中的安装方法可以参考以下网址：

http://www.tensorfly.cn/tfdoc/get_started/os_setup.html

2.6 TensorFlow 1.x 版本与 2.x 版本共存的解决方案

由于 TensorFlow 框架的 1.x 版本与 2.x 版本差异较大。在 1.x 版本上实现的项目，有些并不能直接运行在 2.x 版本上。而新开发的项目推荐使用 2.x 版本。这就需要解决 1.x 版本与 2.x 版本共存的问题。

如用 Anaconda 软件创建虚环境的方法，则可以在同一个主机上安装不同版本的 TensorFlow。

1. 查看 Python 虚环境及 Python 的版本

在装完 Anaconda 软件之后，默认会创建一个虚环境。该虚环境的名字是“base”是当前系统的运行主环境。可以用“conda info --envs”命令进行查看。

(1) 在 Linux 系统中查看所有的 Python 虚环境。

以 Linux 系统为例，查看所有的 Python 虚环境。具体命令如下：

(base) root@user-NULL:~# conda info --envs 该命令执行后，会显示如下内容：

```
# conda environments:
#
base          * /root/anaconda3
```

在显示结果中可以看到，当前虚环境的名字是“base”，是 Anaconda 默认的 Python 环境。

(2) 在 Linux 系统中查看当前 Python 的版本

可以通过“python --version”命令查看当前 Python 的版本。具体命令如下：

```
(base) root@user-NULL:~# python --version
```

执行该命令后会显示如下内容：

```
Python 3.6.4 :: Anaconda, Inc.
```

在显示结果中可以看到，当前 Python 的版本是 3.6.4。

2. 创建 Python 虚环境

创建 Python 虚环境的命令是“`conda create`”。在创建时，应指定好虚环境的名字和需要使用的版本。

(1) 在 Linux 系统中创建 Python 虚环境。

下面以在 Linux 系统中创建一个 Python 版本为 3.6.4 的虚环境为例（在 Windows 系统中，创建方法完全一致）。具体命令如下：

```
(base) root@user-NULL:~# conda create --name tf2 python=3.6.4
```

该命令创建一个名为“tf2”的 Python 虚环境。具体步骤如下：

① 在创建过程中会提示是否安装对应软件包，如图 2-22 所示。输入“Y”，则下载及安装软件包。



图 2-22 提示是否安装对应的软件包

② 安装完软件包后，系统将会自动进行其他配置。如果出现如图 2-23 所示的界面，则表示创建 Python 虚拟环境成功。



图 2-23 Python 虚拟环境创建成功

在图 2-23 中显示了使用虚拟环境的命令：

```
conda activate tf2          #将虚拟环境 tf2 作为当前的 Python 环境
conda deactivate          #使用默认的 Python 环境
```



提示：

在 Windows 中，激活和取消激活虚拟环境的命令如下：

```
activate tf2
deactivate
```

(2) 检查 Python 虚环境是否创建成功。

再次输入“`conda info --envs`”命令，查看所有的 Python 虚环境。具体命令如下：

```
(base) root@user-NULL:~# conda info -envs
该命令执行后，会显示如下内容：# conda environments:
#
base            * /root/anaconda3
tf2              /root/anaconda3/envs/tf2
```

可以看到，相比 2.6 节，虚环境中多了一个“tf2”，表示创建成功。

(3) 删除 Python 虚环境。

如果想删除已经创建的虚环境，则可以使用“`conda remove`”命令。具体命令如下：

```
(base) root@user-NULL:~# conda remove --name tf2 --all
```

该命令执行后没有任何显示。可以再次通过“`conda info --envs`”命令查看 Python 虚环境是否被删除。

3. 在 Python 虚环境中安装 TensorFlow

激活新创建的虚环境“tf2”，然后按照 2.3 节中介绍的方法安装 TensorFlow。具体命令如下：

```
(base) root@user-NULL:~# conda activate tf2          激活 tf2 虚拟环境
(tf2) root@user-NULL:~# pip install tf-nightly-2.0-preview 安装 TensorFlow 2.0 版
```

在显示结果中可以清楚地看到，TensorFlow 已经安装到了刚刚创建的 Python 环境。

(2) 在 Linux 系统中安装 TensorFlow

可以通过“`python -m pip install tensorflow`”命令安装 TensorFlow。

第 3 章

实例1：用AI模型识别图像是桌子、猫、狗，还是其他

本章用训练好的模型去识别图像，让读者对模型的应用有一个直观的感受。

实例描述

用代码载入一个训练好的 AI 模型。调用该模型，让其对输入的任意图片进行分类识别，并观察识别结果。

本实例使用的是在 ImgNet 数据集上训练好的 PNASNet 模型。PNASNet 模型是一个很优秀的图片识别模型，可以识别出 1000 种类别的物体。

3.1 准备代码环境并预训练模型

本实例要用到 TensorFlow 1.x 版本的 TF-slim 接口。在具体操作之前，需要确保本机已经安装了 TensorFlow 1.x 版本。



提示：

本书的代码环境以 TensorFlow 1.13.1 版本为主。因为 TensorFlow 2.x 版本的代码是基于 TensorFlow 1.13.1 转化而来。TensorFlow 1.13.1 版本可以部分支持 TensorFlow 2.0 版本的代码，详情可见 4.9.4 小节的实例。

在本书中基于 TensorFlow 其他版本（例如：2.x 版本）的实例会有特殊说明。建议读者按照 2.6 节内容在本机建立一个虚环境，实现 TensorFlow 1.x 与 2.x 两个版本共存。

1. 下载 TensorFlow 的 models 模块

models 模块中有许多成熟模型，可以直接拿来使用。在项目中，用 models 模块进行二次开发可以大大提升工作效率。

models 模块独立于 TensorFlow 项目，在使用时需要额外下载。下载地址如下：

<https://github.com/tensorflow/models/>

打开上述的网址链接，可以将 models 模块的源码下载到本地。

**提示：**

下载 models 模块的源码，可以手动直接下载，也可以使用 Git 工具进行下载。Git 工具的使用方法见 13.5.5 小节“（1）下载 models 代码”中的介绍。

2. 部署 TensorFlow 的 slim 模块

将下载的 models 模块解压缩之后，将其\models-master\research 路径下的 slim 文件夹（如图 3-1 所示），复制到本地代码的同级路径下。

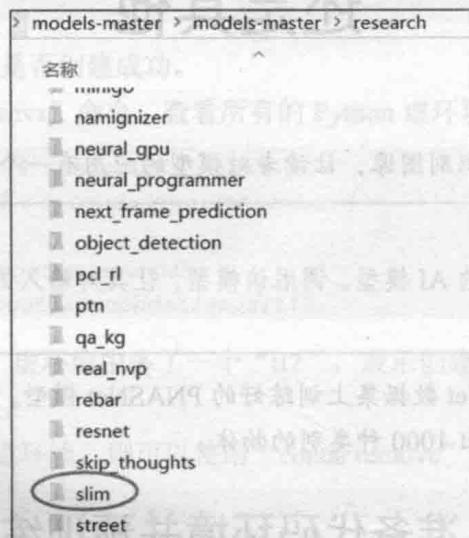


图 3-1 slim 模块的路径

在 slim 文件夹中，有许多成熟模型的代码实现。这些代码都是使用 TF-slim 接口来实现的。TF-slim 接口是 TensorFlow 1.0 之后推出的一个新的轻量级高级 API。该接口将很多常见的 TensorFlow 函数做了二次封装，使代码变得更加简洁。

本实例将使用 slim 文件夹中的 PNASNet 模型。

**提示：**

本书以 TF-slim 接口的应用作为第一个实例，意在让熟悉 TensorFlow 1.x 版本的读者更容易上手，可以快速进入学习状态。

在 TensorFlow 1.x 版本中，TF-slim 接口非常稳定、实用，适用于开发处理图像方面的模型。但在 TensorFlow 2.x 版本中，TF-slim 接口被边缘化了。

如果读者已经用 TF-slim 接口开发了部分项目，建议一直在 TensorFlow 1.x 版本中运行。

如果要开发新的模型，则建议少用 TF-slim 接口。推荐使用 tf.keras 接口（见 6.1.6 小节）。该接口可以兼容 TensorFlow 1.x 与 2.x 两个版本。

3. 下载 PNASNet 模型

（1）访问以下网站，下载训练好的 PNASNet 模型：

<https://github.com/tensorflow/models/tree/master/research/slim>

打开该链接后，可以在网页中找到模型文件“pnasnet-5_large_2017_12_13.tar.gz”的下载地址，如图3-2所示。

| VGG 16 | Code | vgg_16_2016_08_28.tar.gz | 71.5 | 89.8 |
|-----------------------------------|------|-----------------------------------|------|------|
| VGG 19 | Code | vgg_19_2016_08_28.tar.gz | 71.1 | 89.8 |
| MobileNet_v1_1.0_224 | Code | mobilenet_v1_1.0_224.tgz | 70.9 | 89.9 |
| MobileNet_v1_0.50_160 | Code | mobilenet_v1_0.50_160.tgz | 59.1 | 81.9 |
| MobileNet_v1_0.25_128 | Code | mobilenet_v1_0.25_128.tgz | 41.5 | 66.3 |
| MobileNet_v2_1.4_224 [*] | Code | mobilenet_v2_1.4_224.tgz | 74.9 | 92.5 |
| MobileNet_v2_1.0_224 [*] | Code | mobilenet_v2_1.0_224.tgz | 71.9 | 91.0 |
| NASNet-A_Mobile_224# | Code | nasnet-a_mobile_04_10_2017.tar.gz | 74.0 | 91.6 |
| NASNet-A_Large_331# | Code | nasnet-a_large_04_10_2017.tar.gz | 82.7 | 96.2 |
| PNASNet-5_Large_331 | Code | pnasnet-5_large_2017_12_13.tar.gz | 82.9 | 96.2 |

图3-2 PNASNet模型的下载页面

(2) 将预训练模型下载到本地并进行解压缩，得到如图3-3所示的文件结构。

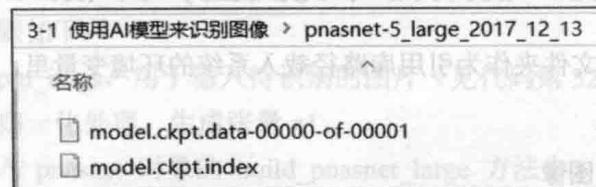


图3-3 PNASNet模型文件

(3) 将整个pnasnet-5_large_2017_12_13文件夹放到本地代码的同级目录下。



提示：

在图3-2中可以看到，除本实例要用的PNASNet模型外，还有好多其他的模型。其中倒数第4行的mobilenet_v2_1.0_224.tgz模型也是比较常用的。该模型体积小、运算快，常用在移动设备中。

4. 准备ImgNet数据集标签

预训练模型PNASNet是在ImgNet数据集上训练好的。在用该模型进行分类时，还需要配合与其对应的标签文件一起使用。在slim文件夹中，将获得标签文件的操作封装到了代码里，在使用时直接调用即可。



提示：

由于标签文件采用英文进行分类，读起来不太直观。书籍同步的配套资源中提供了一个翻译好的中文标签分类文件“中文标签.csv”。读者可以将该中文标签文件下载到本地进行加载。

将预训练模型文件、slim文件夹、代码文件、中文标签都准备好后，目录结构如图3-4所示。

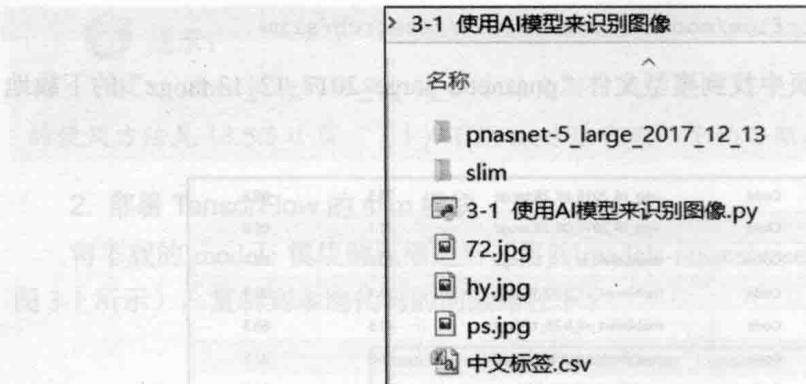


图 3-4 实例 1 文件结构

在图 3-4 中有三个图片文件“72.jpg”“hy.jpg”“ps.jpg”，它们是用来测试的图片，读者可以将其替换为自己所要识别的文件。

3.2 代码实现：初始化环境变量，并载入 ImgNet 标签

首先将本地的 slim 文件夹作为引用库路径载入系统的环境变量里，然后载入 ImgNet 标签并显示出来。

代码 3-1 用 AI 模型识别图像

```

01 import sys                                # 初始化环境变量
02 nets_path = r'slim'
03 if nets_path not in sys.path:
04     sys.path.insert(0,nets_path)
05 else:
06     print('already add slim')
07
08 import tensorflow as tf                    # 引入模块
09 from PIL import Image
10 from matplotlib import pyplot as plt
11 from nets.nasnet import pnasnet
12 import numpy as np
13 from datasets import imagenet
14 slim = tf.contrib.slim
15
16 tf.reset_default_graph()
17
18 image_size = pnasnet.build_pnasnet_large.default_image_size # 获得图片的尺寸
19 labels = imagenet.create_readable_names_for_imagenet_labels() # 获得标签
20 print(len(labels),labels)                                # 显示输出标签
21
22 def getone(onestr):
23     return onestr.replace(',', ' ')
24

```

```

25 with open('中文标签.csv', 'r+') as f:          # 打开文件
26     labels = list(map(getone, list(f)))    #
27     print(len(labels), type(labels), labels[:5])  # 输出中文标签

```

在代码中读取了英文和中文两种标签，并将其输出。程序运行后，输出结果如下：

```

1001 {0: 'background', 1: 'tench, Tinca tinca', 2: 'goldfish, Carassius auratus',
3: 'great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias',
4: 'tiger shark, Galeocerdo cuvieri', 5: 'hammerhead, hammerhead shark', ..., 994:
'gyromitra', 995: 'stinkhorn, carrion fungus', 996: 'earthstar', 997: 'hen-of-the-woods,
hen of the woods, Polyporus frondosus, Grifola frondosa', 998: 'bolete', 999: 'ear, spike,
capitulum', 1000: 'toilet tissue, toilet paper, bathroom tissue'}
1001 <class 'list'> ['背景 known \n', '丁鲷 \n', '金鱼 \n', '大白鲨 \n', '虎鲨 \n']

```

结果中一共输出了两行信息：第1行是英文标签，第2行是中文标签。

3.3 代码实现：定义网络结构

定位网络结构的步骤如下：

- (1) 定义占位符 input_imgs，用于输入待识别的图片（见代码第32行）。
- (2) 对占位符进行归一化处理，生成张量 x1。
- (3) 将张量 x1 传入 pnasnet 对象的 build_pnasnet_large 方法中，生成处理结果 logits 与 end_points。其中 end_points 是字典类型，里面是模型输出的具体结果。
- (4) 从字典 end_points 中取出关键字“Predictions”所对应的值 prob。prob 是一个包含 1000 个元素的数组，数组中的元素表示被预测图片在这 1000 个分类中的概率。
- (5) 用 tf.argmax 函数在数组 prob 中找到数值最大的索引，该索引便是该图片的分类。

具体代码如下：

代码 3-1 用 AI 模型识别图像（续）

```

28 sample_images = ['hy.jpg', 'ps.jpg', '72.jpg']      # 定义待测试图片的名称
29
30 input_imgs = tf.placeholder(tf.float32, [None, image_size, image_size, 3]) # 定义占位符
31
32 x1 = 2 * (input_imgs / 255.0) - 1.0                # 归一化图片
33
34 arg_scope = pnasnet.pnasnet_large_arg_scope()        # 获得模型的命名空间
35 with slim.arg_scope(arg_scope):
36     logits, end_points = pnasnet.build_pnasnet_large(x1, num_classes = 1001,
37     is_training=False)
38     prob = end_points['Predictions']                  # 获得结果的输出节点
39     y = tf.argmax(prob, axis = 1)

```

代码第28行指定了待识别图片的名称。如果想识别自己的图片，直接修改这里的图片名称

即可。

在代码第 34 行中，`arg_scope` 是命名空间的意思。在 TensorFlow 中，相同名称的不同张量是通过命名空间来标识的。关于命名空间的更多知识可以参考《深度学习之 TensorFlow——入门、原理与进阶实战》一书的 4.3 节。

3.4 代码实现：载入模型进行识别

本节的代码步骤如下：

- (1) 定义要加载的预训练模型的路径。
- (2) 建立会话。
- (3) 在会话中载入预训练模型。
- (4) 将图片输入预训练模型进行识别。

具体代码如下：

代码 3-1 用 AI 模型识别图像（续）

```

39 checkpoint_file = r'pnasnet-5_large_2017_12_13\model.ckpt' # 定义预训练模型
40 saver = tf.train.Saver() # 定义 saver, 用于加载模型
41 with tf.Session() as sess: # 建立会话
42     saver.restore(sess, checkpoint_file) # 载入模型
43
44     def preimg(img): # 定义图片预处理函数
45         ch = 3
46         if img.mode=='RGBA': # 兼容 RGBA 图片
47             ch = 4
48
49         imgnp = np.asarray(img.resize((image_size,image_size)),
50                             dtype=np.float32).reshape(image_size,image_size,ch)
51         return imgnp[:, :, :3]
52
53     # 获得原始图片与预处理图片
54     batchImg = [ preimg( Image.open(imgfilename) ) for imgfilename in
55     sample_images ]
56     # 输入模型
57     yv, img_norm = sess.run([y,x1], feed_dict={input_imgs: batchImg}) # 输出结果
58
59     print(yv,np.shape(yv))
60     def showresult(yy,img_norm,img_org): # 定义显示图片的函数
61         plt.figure()
62         p1 = plt.subplot(121)
63         p2 = plt.subplot(122)
64         p1.imshow(img_org) # 显示图片
65         p1.axis('off')

```

```

66 p1.set_title("organization image")
67
68     p2.imshow((img_norm * 255).astype(np.uint8)) #显示图片
69     p2.axis('off')
70     p2.set_title("input image")
71
72 plt.show()
73     print(yy,labels[yy])
74
75 for yy,img1,img2 in zip(yv,batchImg,orgImg): #显示每条结果及图片
76     showresult(yy,img1,img2)

```

在TensorFlow的静态图中，运行模型时有一个“图”的概念。在本实例中，原始的网络结构会在静态图中定义好，接着通过建立一个会话（见代码第41行）让当前代码与静态图连接起来，然后调用sess中的run函数将数据输入静态图中并返回结果，从而实现图片的识别。

在进行模型识别之前，所有的图片都要统一成固定大小（见代码第49行），并进行归一化处理（见代码第32行）。这个过程被叫作图片预处理。将经过预处理后的图片放到模型中，才能够得到准确的结果。

代码运行后，输出以下结果：

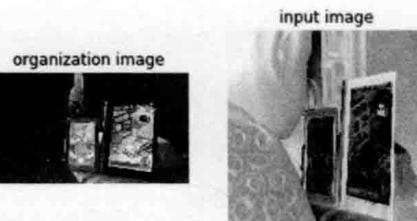


图3-5 PNASNet识别结果(a)

621 笔记本电脑

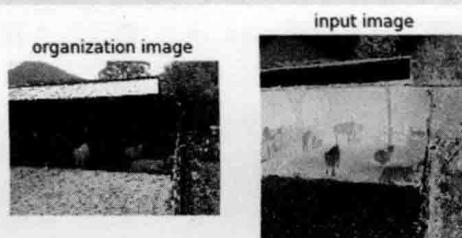


图3-5 PNASNet识别结果(b)

342 猪

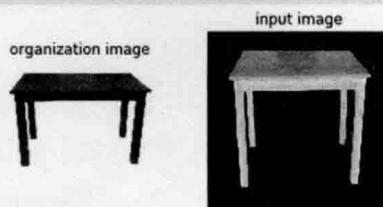


图3-5 PNASNet识别结果(c)

533 餐桌板

结果一共显示了 3 幅图和 3 段文字。每幅图片下一行的文字是模型识别出来的结果。在每幅图中，左侧是原始图片，右侧是预处理后的图片。

3.5 扩展：用更多预训练模型完成图片分类任务

在本书的配套资源中提供了一个用 NASNet-Mobile 模型来识别图像的例子（详见代码文件“3-2 用 nasnet-mobile 模型识别图像.py”），有兴趣的读者可以自行研究。

还可以在 tf.keras 接口中使用预训练模型（见 6.7.9 小节的实例）。用 tf.keras 接口编写的代码可以直接运行在 TensorFlow 1.x 版本和 2.x 版本中。用 tf.keras 接口编写代码是 TensorFlow 2.x 版本主推的代码编写方式。其实现起来更为简洁，可以使代码量大大减少，比 TF-slim 接口还要方便。

结果一共显示了 3 张图片和 3 行文字。每张图片下一行的文字是模型识别出来的结果。左侧图片中，左侧是原始图片，右侧是预处理后的图片。

第 4 章 拓展：用更多预训练模型完成图片分类任务

用TensorFlow制作自己的数据集

还可以在 `tf.keras` 接口中使用预训练模型（见 6.7.9 小节的实例）。用 `tf.keras` 接口编写的代码可以在原生 TensorFlow 1.x 和 2.x 中运行。

本章来学习数据集的创建和使用。其中，创建部分包括将内存对象、文件对象、`TFRecord` 对象、`Dataset` 对象制作成数据集；使用部分包括用生成器、队列、`TFRecordReader`、`Dataset` 迭代器等方法从数据集中读取数据。

4.1 快速导读

在学习实例之前，有必要了解一下数据集的基础知识。

4.1.1 什么是数据集

数据集是样本的集合。深度学习离不了样本的学习。在用 TensorFlow 框架开发深度学习模型之前，需要为模型准备好数据集。在训练模型环节，程序需要从数据集中不断地将数据注入模型中，模型通过对注入数据的计算来学习特征。

1. TensorFlow 的数据集格式

TensorFlow 中有 4 种数据集格式：

- 内存对象数据集：直接用字典变量 `feed_dict`，通过注入模式向模型输入数据。该数据集适用于少量的数据集输入。
- `TFRecord` 数据集：用队列式管道（`tfRecord`）向模型输入数据。该数据集适用于大量的数据集输入。
- `Dataset` 数据集：通过性能更高的输入管道（`tf.data`）向模型输入数据。该数据集适用于 TensorFlow 1.4 之后的版本。
- `tf.keras` 接口数据集：支持 `tf.keras` 语法的数据集接口。该数据集适用于 TensorFlow 1.4 之后的版本。

2. 学习建议

本章会通过多个实例介绍前 3 种数据集的使用方法。建议读者：

- 简单了解前两种数据集（内存对象数据集、`TFRecord` 数据集）的使用方法，达到能读懂代码的程度即可。
- 重点掌握第 3 种数据集（`Dataset` 数据集）。在 TensorFlow 2.x 之后，主要推荐使用 `Dataset`

数据集。

- tf.keras 接口数据集对数据预处理的一些方法进行了封装，并集成了许多常用的数据集，这些数据集都有对应的载入函数，可以直接调用它们。所以，这种数据集使用起来非常方便。

4.1.2 TensorFlow 的框架

数据集的使用方法，跟框架的模式有关。在TensorFlow中，大体可以分为5种框架。

- 静态图框架：是一种“定义”与“运行”相分离的框架，是TensorFlow最原始的框架，也是最灵活的框架。定义的张量，必须要在会话(session)中调用run方法才可以获得其具体值。
- 动态图框架：更符合Python语言的框架。即在代码被调用的同时，便开始计算具体值，不需要再建立会话来运行代码。
- 估算器框架：是一个集成了常用操作的高级API。在该框架中进行开发，代码更为简单。
- Keras框架：是一个支持Keras接口的框架。
- Swift框架：是一个可以在苹果系统中使用Swift语言开发TensorFlow模型的框架，使用了动态图机制。

本章重点讲解的是数据集的制作。为了配合数据集，还需要用到框架方面的知识。静态图框架是TensorFlow中最早的框架，也是最基础的框架，本书中的大多实例都是基于该框架实现的。当然，在少数实例中也会使用其他框架。每个框架的具体使用方法，会伴随实例进行详细讲解。

另外，Swift框架不在本书的介绍范围之内。有兴趣的读者可以在以下链接中找到相关资料自行研究：

<https://github.com/tensorflow/swift>

4.1.3 什么是TFDS

TFDS是TensorFlow中的数据集集合模块。该模块将常用的数据集封装起来，实现自动下载与统一的调用接口，为开发模型提供了便利。

1. 安装TFDS

TFDS模块要求当前的TensorFlow版本在1.12或者1.12之上。在满足这个条件之后，可以使用以下命令进行安装：

```
pip install tensorflow-datasets
```

2. 用TFDS加载数据集

在装好TFDS模块后，可以编写代码从该模块中加载数据集。以MNIST数据集为例，具体代码如下：

```
import tensorflow_datasets as tfds
```

```

tf.enable_eager_execution()          # 启动动态图
print(tfds.list_builders())         # 查看有效的数据集
ds_train, ds_test = tfds.load(name="mnist", split=["train", "test"]) # 加载数据集
ds_train = ds_train.shuffle(1000).batch(128).prefetch(10) # 用 tf.data.Dataset 接口加工数据集
for features in ds_train.take(1):
    image, label = features["image"], features["label"]

```

在上面代码中，用 `tfds.load` 方法实现数据集的加载。还可以用 `tfds.builder` 方法实现更灵活的操作。具体可以参考以下链接：

<https://github.com/tensorflow/datasets>
https://www.tensorflow.org/datasets/api_docs/python/tfds

在该链接中还介绍了 `tfds.as_numpy` 方法，该方法会将数据集以生成器对象的形式进行返回，该生成器对象的类型为 Numpy 数组。更多应用请参考 6.6 节实例。

3. 在 TFDS 中添加自定义数据集

TFDS 模块还支持自定义数据集的添加。具体方法可以参考如下链接：

https://github.com/tensorflow/datasets/blob/master/docs/add_dataset.md

4.2 实例 2：将模拟数据制作成内存对象数据集

本实例将用内存中的模拟数据来制作成数据集。生成的数据集被直接存放在 Python 内存对象中。这种做法的好处是——让数据集的制作独立于任何框架。

当然，由于本实例没有使用 TensorFlow 中的任何框架，所以，所有需要特征变换的代码都得手动编写，这会增加很大的工作量。

实例描述

生成一个模拟 $y \approx 2x$ 的数据集，并通过静态图的方式显示出来。

为了演示一套完整的操作，在生成数据集之后，还要在静态图中建立会话，将数据显示出来。本实例的实现步骤如下：

- (1) 生成模拟数据。
- (2) 定义占位符。
- (3) 建立会话（session），获取并显示模拟数据。
- (4) 将模拟数据可视化。
- (5) 运行程序。

4.2.1 代码实现：生成模拟数据

在样本制作过程中，最忌讳的一次性将数据都放入内存中。如果数据量很大，这样容易造成内存用尽。即使是模拟数据，也不建议一次性将数据全部生成后一次放入内存。

一般常用的做法是：

(1) 创建一个模拟数据生成器。

(2) 每次只生成指定批次的样本（见4.2.2小节）。

这样在迭代过程中，就可以用“随用随制作”的方式来获得样本数据。

下面定义GenerateData函数来生成模拟数据，并将GenerateData函数的返回值设为生成器方式。这种做法使内存被占用得最少。具体代码如下：

代码4-1 将模拟数据制作成内存对象数据集

```
01 import tensorflow as tf
02 import numpy as np
03 import matplotlib.pyplot as plt
04
05 #在内存中生成模拟数据
06 def GenerateData(batchsize = 100):
07     train_X = np.linspace(-1, 1, batchsize)      #生成-1~1之间的100个浮点数
08     train_Y = 2 * train_X + np.random.randn(*train_X.shape) * 0.3 #y=2x,
    但是加入了噪声
09     yield train_X, train_Y                      #以生成器的方式返回
```

代码第9行，用关键字yield修饰函数GenerateData的返回方式，使得函数GenerateData以生成器的方式返回数据。生成器对象只使用一次，之后便会自动销毁。这样做可以为系统节省大量的内存。



提示：

有关生成器的更多知识，请参考《Python带我起飞——入门、进阶、商业实战》一书中5.8节“迭代器”与6.8节“生成器”部分的内容。

4.2.2 代码实现：定义占位符

在正常的模型开发中，这个环节应该是定义占位符和网络结构。在训练模型时，系统会将数据集的输入数据用占位符来代替，并使用静态图的注入机制将输入数据传入模型，进行迭代训练。

因为本实例只需要从数据集中获取数据，所以只定义占位符，不需要定义其他网络节点。具体代码如下：

代码4-1 将模拟数据制作成内存对象数据集（续）

```
10 #定义模型结构部分，这里只有占位符张量
11 Xinput = tf.placeholder("float", (None))          #定义两个占位符，用来接收参数
12 Yinput = tf.placeholder("float", (None))
```

代码第11行的Xinput用于接收GenerateData函数的train_X返回值。

代码第12行的Yinput用于接收GenerateData函数的train_Y返回值。

**提示：**

关于静态图和注入机制的更多内容，建议参考《深度学习之 TensorFlow——入门、原理与进阶实战》一书的第 4 章内容。

4.2.3 代码实现：建立会话，并获取数据

首先定义数据集的迭代次数，接着建立会话（session）。在 session 中，使用了两层 for 循环：第 1 层是按照迭代次数来循环；第 2 层是对 GenerateData 函数返回的生成器对象进行循环，并将数据打印出来。

因为 GenerateData 函数返回的生成器对象只有一个元素，所以第 2 层循环也只运行了一次。

代码 4-1 将模拟数据制作成内存对象数据集（续）

```

13 #建立会话，获取并输出数据
14 training_epochs = 20          #定义需要迭代的次数
15 with tf.Session() as sess:    #建立会话(session)
16     for epoch in range(training_epochs): #迭代数据集20遍
17         for x, y in GenerateData():        #通过for循环打印所有的点
18             xv, yv = sess.run([Xinput, Yinput], feed_dict={Xinput: x, Yinput: y}) #通过静态图注入的方式传入数据
19             #打印数据
20             print(epoch, "| x.shape:", np.shape(xv), "| x[:3]:", xv[:3])
21             print(epoch, "| y.shape:", np.shape(yv), "| y[:3]:", yv[:3])

```

代码第 14 行，定义了数据集的迭代次数。这个参数在训练模型时才会用到。本实例中，变量 training_epochs 代表读取数据的次数。

4.2.4 代码实现：将模拟数据可视化

为了使本实例的结果更加直观，下面把取出的数据以图的方式显示出来。具体代码如下：

代码 4-1 将模拟数据制作成内存对象数据集（续）

```

22 #显示模拟数据点
23 train_data = list(GenerateData())[0]          #获取数据
24 plt.plot(train_data[0], train_data[1], 'ro', label='Original data') #生成图像
25 plt.legend()                                     #添加图例说明
26 plt.show()                                       #显示图像

```

图像显示部分不是本实例重点，读者了解一下即可。

4.2.5 运行程序

代码运行后，输出以下结果：

```

0 | x.shape: (100,) | x[:3]: [-1.         -0.97979796 -0.959596 ]
0 | y.shape: (100,) | y[:3]: [-2.0518072 -1.7162607 -1.9215399]
1 | x.shape: (100,) | x[:3]: [-1.         -0.97979796 -0.959596 ]
1 | y.shape: (100,) | y[:3]: [-1.7399402 -1.8851279 -1.8028339]
...
18 | x.shape: (100,) | x[:3]: [-1.         -0.97979796 -0.959596 ]
18 | y.shape: (100,) | y[:3]: [-2.1623547 -2.1738577 -2.6779299]
19 | x.shape: (100,) | x[:3]: [-1.         -0.97979796 -0.959596 ]
19 | y.shape: (100,) | y[:3]: [-2.2008154 -1.9220618 -1.3616668]

```

程序循环运行了 20 次，每次都会生成 100 个 x 与 y 对应的数据。

输出结果的第 1、2 行可以看到，在第 1 次循环时，取出了 x 与 y 的内容。每行数据的内容被“|”符号被分割成三段，依次为：迭代次数、数据的形状、前 3 个元素的值。

同时，程序又生成了数据的可视化结果，如图 4-1 所示。

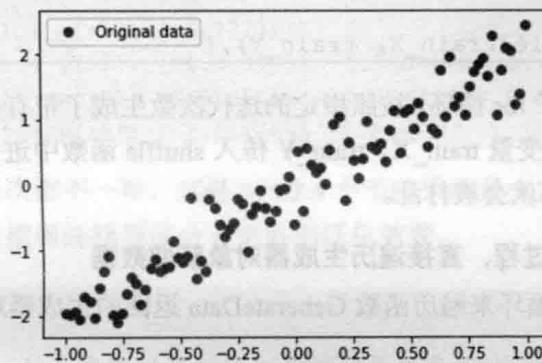


图 4-1 模拟数据集的可视化结果

4.2.6 代码实现：创建带有迭代值并支持乱序功能的模拟数据集

下面对本实例的代码做更进一步的优化：

- (1) 将数据集与迭代功能绑定在一起，让代码变得更简洁。
- (2) 对数据集进行乱序操作，让生成的 x 数据无规则。

通过对数据集的乱序，可以消除样本中无用的特征，从而大大提升模型的泛化能力。下面详细介绍具体实现方法。

1. 修改 GenerateData 函数，生成带有多个元素的生成器对象，并对其进行乱序操作

在函数 GenerateData 的定义中传入参数 `training_epochs`，并按照 `training_epochs` 的循环次数生成带有多个元素的生成器对象。具体代码如下：



提示：

在乱序操作部分使用的是 `sklearn.utils` 库中的 `shuffle` 方法。要使用该方法需要先安装 `sklearn` 库。具体命令如下：

```
pip install sklearn
```

在本书的 4.4.3 和 4.6.5 小节中，还会介绍一些打乱数据集中样本顺序的方法。

代码 4-2 带迭代的模拟数据集

```

01 import tensorflow as tf
02 import numpy as np
03 import matplotlib.pyplot as plt
04 from sklearn.utils import shuffle #导入 sklearn 库
05
06 #在内存中生成模拟数据
07 def GenerateData(training_epochs ,batchsize = 100):
08     for i in range(training_epochs):
09         train_X = np.linspace(-1, 1, batchsize)      #train_X 是-1~1之间连续
          的 100 个浮点数
10         train_Y = 2 * train_X + np.random.randn(*train_X.shape) * 0.3 #y=2x,
          但是加入了噪声
11         yield shuffle(train_X, train_Y),i

```

在代码第 8 行，加入了 for 循环，按照指定的迭代次数生成了带有多个元素的迭代器对象。在代码第 11 行，将生成的变量 train_X、train_Y 传入 shuffle 函数中进行乱序。这样所得到的样本 train_X、train_Y 的顺序就会被打乱。

2. 修改 session 处理过程，直接遍历生成器对象获取数据

在 session 中，用 for 循环来遍历函数 GenerateData 返回的生成器对象（见代码第 18 行）。具体代码如下：

代码 4-2 带迭代的模拟数据集（续）

```

12 Xinput = tf.placeholder("float", (None))      #定义两个占位符，用来接收参数
13 Yinput = tf.placeholder("float", (None))
14
15 training_epochs = 20                         #定义需要迭代的次数
16
17 with tf.Session() as sess:                   #建立会话(session)
18     for (x, y) ,ii in GenerateData(training_epochs):    #用一个for循环来遍历
          生成器对象
19         xv,yv = sess.run([Xinput,Yinput],feed_dict={Xinput: x, Yinput: y}) #通过静态图注入的方式传入数据
20         print(ii,"| x.shape:",np.shape(xv),"| x[:3]:",xv[:3]) #输出数据
21         print(ii,"| y.shape:",np.shape(yv),"| y[:3]:",yv[:3])

```

3. 获得并可视化只有一个元素的生成器对象

再次调用函数 GenerateData，并传入参数 1。函数 GenerateData 会返回只有一个元素的生成器，生成器中的元素为一个批次的模拟数据。获得数据后，将其以图的方式显示出来。

代码 4-2 带迭代的模拟数据集（续）

```
22 #显示模拟数据点
```

```

23 train_data = list(GenerateData(1))[0] # 获取数据
24 plt.plot(train_data[0][0], train_data[0][1], 'ro', label='Original data')
# 生成图像
25 plt.legend() # 添加图例说明
26 plt.show() # 显示图像

```

代码第23行，用函数 GenerateData 返回了一个生成器对象，该生成器对象具有一个元素。

4. 该数据集运行程序

整个代码改好后，运行效果如下：

```

0 | x.shape: (100,) | x[:3]: [-0.8787879  0.97979796  0.8787879]
0 | y.shape: (100,) | y[:3]: [-1.4220259  1.4639419   1.8528527]
1 | x.shape: (100,) | x[:3]: [-0.97979796  0.83838385  0.7171717]
1 | y.shape: (100,) | y[:3]: [-1.5776895  2.3976982   1.0726162]
...
18 | x.shape: (100,) | x[:3]: [ 0.7777778   1.          -0.03030303]
18 | y.shape: (100,) | y[:3]: [ 1.3839471   1.7204176   -0.62857807]
19 | x.shape: (100,) | x[:3]: [ 0.8181818   0.01010101  0.61616164]
19 | y.shape: (100,) | y[:3]: [ 2.1516888  -0.2165111   1.3852897]

```

可以看到 x 的数据每次都不一样，这是与 4.2.4 小节结果的最大区别。原因是，x 的值已经被打乱顺序了。这样的数据训练模型还会有更好的泛化效果。



总结：

通过本实例的学习，读者在掌握基础的制作模拟数据集方法的同时，更需要记住两个知识点：生成器与乱序。

生成器语法在 TensorFlow 底层的数据集处理中应用得非常广泛。在实际应用中，它可以为系统节省很大的内存。要学会使用生成器语法。

对数据集进行乱序是深度学习中的一个重要知识点，但很容易被开发者忽略。这一点值得注意。

4.3 实例3：将图片制作成内存对象数据集

本实例将使用图片样本数据来制作成数据集。在数据集的实现中，使用了 TensorFlow 的队列方式。这样做的好处是：能充分使用 CPU 的多线程资源，让训练模型与数据读取以并行的方式同时运行，从而大大提升效率。

实例描述

有一套从 1~9 的手写图片样本。

首先将这些图片样本做成数据集，输入静态图中；然后运行程序，将数据从静态图中输出，并显示出来。

在读取图片过程中，最需要考虑的一个因素是内存。如果样本比较少，则可以采用较简单的方式——直接将图片一次性全部读入系统。如果样本足够大，则这种方法会将内存全部占满，使程序无法运行。

所以，一般建议使用“边读边用”的方式：一次只读取所需要的图片，用完后再读取下一批。这种方式能够满足程序正常执行。但是，频繁的 I/O 读取操作也会使性能受到影响。

最好的方式是——以队列的方式进行读取。即使用至少两个线程并发执行：一个线程用于从队列里取数据并训练模型，而另外一个线程用于读取文件放入缓存。这样既保证了内存不会被占满，又赢得了效率。

4.3.1 样本介绍

本实例使用的是 MNIST 数据集，该数据集以图片的形式存放。在随书配套的资源中，找到文件夹为“mnist_digits_images”的样本，并将其复制到本地代码的同级路径下。

打开文件夹“mnist_digits_images”可以看到 10 个子文件夹，如图 4-2 所示。

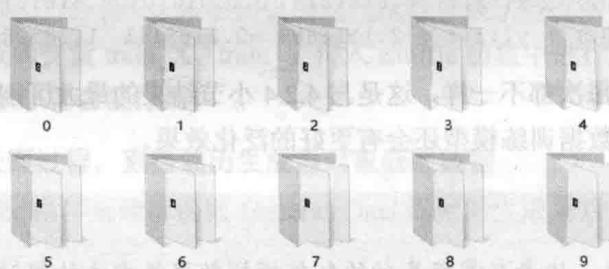


图 4-2 MNIST 图片文件夹

每个子文件夹里放的图片内容都与该文件夹的名称一致。例如，打开名字为“0”的文件夹，会看到各种数字是“0”的图片，如图 4-3 所示。

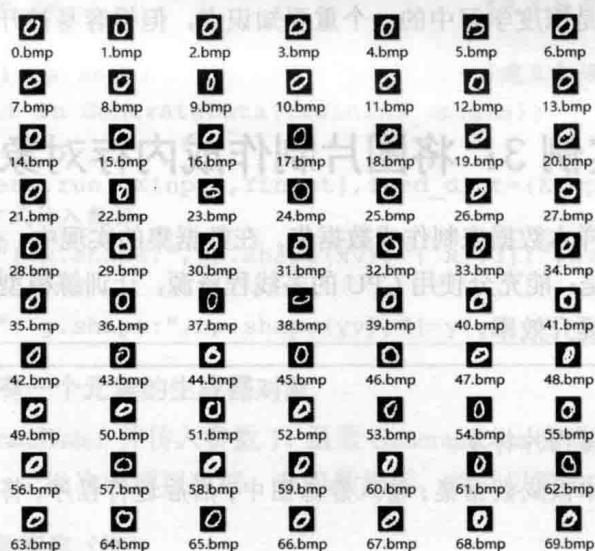


图 4-3 MNIST 图片文件

4.3.2 代码实现：载入文件名称与标签

编写函数 `load_sample` 载入指定路径下的所有文件的名称载入，并将文件所属目录的名称作为标签。

`load_sample` 函数会返回 3 个对象。

- `lfilenames`: 文件名称数组。将根据文件名称来读取图片数据。
- `labels`: 数值化后的标签。与每一个文件的名称一一对应。
- `lab`: 数值化后的标签与字符串标签的对应关系。用于显示使用。

因为标签 `labels` 对象主要用于模型的训练，所以这里将其转化为数值型。待需要输出结果时，再通过 `lab` 将其转化为字符串。

载入文件名称与标签的具体代码如下：

代码 4-3 将图片制作成内存对象数据集

```

01 import tensorflow as tf
02 import os
03 from matplotlib import pyplot as plt
04 import numpy as np
05 from sklearn.utils import shuffle
06
07 def load_sample(sample_dir):
08     '''递归读取文件。只支持一级。返回文件名、数值标签、数值对应的标签名'''
09     print ('loading sample dataset..')
10     lfilenames = []
11     labelsnames = []
12     for (dirpath, dirnames, filenames) in os.walk(sample_dir):#遍历文件夹
13         for filename in filenames:#遍历所有文件名
14             filename_path = os.sep.join([dirpath, filename])
15             lfilenames.append(filename_path) #添加文件名
16             labelsnames.append( dirpath.split('\\')[-1] ) #添加文件名对应的
17             的标签
18
19             lab= list(sorted(set(labelsnames))) #生成标签名称列表
20             labdict=dict( zip( lab ,list(range(len(lab)))) ) #生成字典
21
22             labels = [labdict[i] for i in labelsnames]
23             return
24             shuffle(np.asarray( lfilenames),np.asarray( labels)),np.asarray(lab)
25
26     (image,label),labelsnames = load_sample(data_dir) #载入文件名称与标签
27     print(len(image),image[:2],len(label),label[:2])#输出 load_sample 返回的结果
28     print(labelsnames[ label[:2] ],labelsnames) #输出 load_sample 返回的标签字符串

```

代码运行后，输出以下结果：

```

loading sample dataset..
8000  ['data\\mnist_digits_images\\2\\520.bmp'    'data\\mnist_digits_images\\2\\
1.bmp'] 8000 [2 2]
['2' '2'] ['0' '1' '2' '3' '4' '5' '6' '7' '8' '9']

```

输出结果的第 2 行共分为 4 部分，依次是：图片的长度（8000）、头两个图片的文件名、标签的长度（8000）、前两个标签的具体值（[2 2]）。

因为函数 `load_sample` 已经将返回值的顺序打乱（见代码第 23 行），所以该函数返回数据的顺序是没有规律的。

4.3.3 代码实现：生成队列中的批次样本数据

编写函数 `get_batches`，返回批次样本数据。具体步骤如下：

- (1) 用 `tf.train.slice_input_producer` 函数生成一个输入队列。

- (2) 按照指定路径读取图片，并对图片进行预处理。

- (3) 用 `tf.train.batch` 函数将预处理后的图片变成批次数据。

在第(3)步调用函数 `tf.train.batch` 时，还可以指定批次(`batch_size`)、线程个数(`num_threads`)、队列长度(`capacity`)。该函数的定义如下：

```

def batch(tensors, batch_size, num_threads=1, capacity=32,
          enqueue_many=False, shapes=None, dynamic_pad=False,
          allow_smaller_final_batch=False, shared_name=None, name=None)

```

在实际使用时，按照对应的参数进行设置即可。

函数 `get_batches` 的完整实现及调用代码如下：

代码 4-3 将图片制作成内存对象数据集（续）

```

29 def get_batches(image,label,resize_w,resize_h,channels,batch_size):
30
31     queue = tf.train.slice_input_producer([image ,label]) #实现一个输入队列
32     label = queue[1]                                     #从输入队列里读取标签
33
34     image_c = tf.read_file(queue[0])                      #从输入队列里读取image路径
35
36     image = tf.image.decode_bmp(image_c,channels)        #按照路径读取图片
37
38     image = tf.image.resize_image_with_crop_or_pad(image,resize_w,resize_h)
#修改图片的大小
39
40     #将图像进行标准化处理
41     image = tf.image.per_image_standardization(image)
42     image_batch,label_batch = tf.train.batch([image,label], #生成批次数据
43                                         batch_size,
44                                         num_threads = 64)
45
46     images_batch = tf.cast(image_batch,tf.float32) #将数据类型转换为 float32

```

```

47 #修改标签的形状
48 labels_batch = tf.reshape(label_batch, [batch_size])
49 return images_batch, labels_batch
50 batch_size = 16
51 image_batches, label_batches = get_batches(image, label, 28, 28, 1, batch_size)

```

代码第50、51行定义了批次大小，并调用get_batches函数生成两个张量（用于输入数据）。

4.3.4 代码实现：在会话中使用数据集

首先，定义showresult和showimg函数，用于将图片数据进行可视化输出。

接着，建立session，准备运行静态图。在session中启动一个带有协调器的队列线程，通过session的run方法获得数据并将其显示。

具体代码如下：

代码4-3 将图片制作成内存对象数据集（续）

```

52 def showresult(subplot, title, thisimg):          #显示单个图片
53     p = plt.subplot(subplot)
54     p.axis('off')
55     #p.imshow(np.asarray(thisimg[0], dtype='uint8'))
56     p.imshow(np.reshape(thisimg, (28, 28)))
57     p.set_title(title)
58
59 def showimg(index, label, img, ntop):             #显示批次图片
60     plt.figure(figsize=(20,10))                   #定义显示图片的宽和高
61     plt.axis('off')
62     ntop = min(ntop, 9)
63     print(index)
64     for i in range(ntop):
65         showresult(100+10*ntop+i, label[i], img[i])
66     plt.show()
67
68 with tf.Session() as sess:                      #初始化
69     init = tf.global_variables_initializer()
70     sess.run(init)
71
72     coord = tf.train.Coordinator()               #建立列队协调器
73     threads = tf.train.start_queue_runners(sess = sess, coord = coord) #启动
    队列线程
74
75     try:
76         for step in np.arange(10):
77             if coord.should_stop():
78                 break
79             images, label = sess.run([image_batches, label_batches]) #注入数据
80             showimg(step, label, images, batch_size) #显示图片

```

```

81         print(label) # 打印数据
82
83     except tf.errors.OutOfRangeError:
84         print("Done!!!")
85     finally:
86         coord.request_stop()
87
88 coord.join(threads) # 关闭线队

```

关于线程、队列及队列协调器方面的知识，属于 Python 的基础知识。如果读者不熟悉这部分知识，可以参考《Python 带我起飞——入门、进阶、商业实战》一书的 10.2 节。

4.3.5 运行程序

程序运行后，输出以下结果：



图 4-4 MNIST 图片输出 (1)

[4 1 3 5 2 4 3 0 7 6 3 5 5 8 6 8]



图 4-5 MNIST 图片输出 (2)

[0 1 1 5 6 2 9 7 6 0 8 2 7 7 0 5]

图 4-4 的内容为一批次图片数据的前 9 张输出结果。

在图 4-5 的上面有一个数字“9”，代表第 9 次输出的结果。

在图 4-5 的下面有一个数组，代表这一批次数据对应的标签。因为批次大小为 16（见代码第 50 行），所以图 4-5 下面的数组元素个数为 16。

4.4 实例 4：将 Excel 文件制作成内存对象数据集

用 TensorFlow 中的队列方式，将 Excel 文件格式的样本数据制作成数据集。

实例描述

有两个 Excel 文件：一个代表训练数据，一个代表测试数据。

现在需要做的是：（1）将训练数据的样本按照一定批次读取并输出；（2）将测试数据的样本按照顺序读取并输出。

在制作数据集时，习惯将数据分成 2 或 3 部分，这样做的主要目的是，将训练模型使用的

数据与测试模型使用的数据分开，使得训练模型和评估模型各自使用不同的数据。这样做可以很好地反应出模型的泛化性。

4.4.1 样本介绍

本实例的样本是两个 csv 文件——“iris_training.csv”和“iris_test.csv”。这两个文件的内部格式完全一样，如图 4-6 所示。

| | A | B | C | D | E | F |
|----|----|---------------|--------------|---------------|--------------|---------|
| 1 | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
| 2 | 1 | 5.9 | 3 | 4.2 | 1.5 | 1 |
| 3 | 2 | 6.9 | 3.1 | 5.4 | 2.1 | 2 |
| 4 | 3 | 5.1 | 3.3 | 1.7 | 0.5 | 0 |
| 5 | 4 | 6 | 3.4 | 4.5 | 1.6 | 1 |
| 6 | 5 | 5.5 | 2.5 | 4 | 1.3 | 1 |
| 7 | 6 | 6.2 | 2.9 | 4.3 | 1.3 | 1 |
| 8 | 7 | 5.5 | 4.2 | 1.4 | 0.2 | 0 |
| 9 | 8 | 6.3 | 2.8 | 5.1 | 1.5 | 2 |
| 10 | 9 | 5.6 | 3 | 4.1 | 1.3 | 1 |

图 4-6 iris_training 和 iris_test 文件的数据格式

在图 4-6 中，样本一共有 6 列：

- 第 1 列 (Id) 是序号，可以不用关心。
 - 第 2~4 列 (SepalLengthCm、SepalWidthCm、PatalLengthCm、PatalWidthCm) 是数据样本列。
 - 最后一列 (Species) 是标签列。
- 下面就通过代码读取样本。

4.4.2 代码实现：逐行读取数据并分离标签

定义函数 `read_data` 用于读取数据，并将数据中的样本与标签进行分离。在函数 `read_data` 中，实现以下逻辑：

- (1) 调用 `tf.TextLineReader` 函数，对单个 Excel 文件进行逐行读取。
- (2) 调用 `tf.decode_csv`，将 Excel 文件中的单行内容按照指定的列进行分离。
- (3) 将 Excel 单行中的多个属性列按样本数据列与标签数据列进行划分：将样本数据列 (`featurecolumn`) 放到第 2~4 列，用 `tf.stack` 函数将其组合到一起；将标签数据列 (`labelcolumn`) 放到最后 1 列。

具体代码如下：

代码 4-4 将 Excel 文件制作成内存对象数据集

```

01 import tensorflow as tf
02
03 def read_data(file_queue):
04     reader = tf.TextLineReader(skip_header_lines=1) #tf.TextLineReader 可以每次读取一行
05     key, value = reader.read(file_queue)
06

```

```

07     defaults = [[0], [0.], [0.], [0.], [0.], [0]]      #为每个字段设置初始值
08     cvscolumn = tf.decode_csv(value, defaults)          #对每一行进行解析
09
10    featurecolumn = [i for i in cvscolumn[1:-1]]        #划分出列中的样本数据列
11    labelcolumn = cvscolumn[-1]                          #划分出列中的标签数据列
12
13    return tf.stack(featurecolumn), labelcolumn         #返回结果

```

4.4.3 代码实现：生成队列中的批次样本数据

编写 `create_pipeline` 函数，用于返回批次数据。具体步骤如下：

- (1) 用 `tf.train.string_input_producer` 函数生成一个输入队列。
- (2) 用 `read_data` 函数读取 csv 文件内容，并进行样本与标签的分离处理。
- (3) 在获得数据的样本（feature）与标签（label）之后，用 `tf.train.shuffle_batch` 函数生成批次数据。

其中，`tf.train.shuffle_batch` 函数的具体定义如下：

```

def shuffle_batch(tensors, batch_size, capacity, min_after_dequeue,
                  num_threads=1, seed=None, enqueue_many=False, shapes=None,
                  allow_smaller_final_batch=False, shared_name=None, name=None)

```

在 `tf.train.shuffle_batch` 函数中，可以指定批次（`batch_size`）、线程个数（`num_threads`）、队列的最小的样本数（`min_after_dequeue`）、队列长度（`capacity`）等。



提示：

`min_after_dequeue` 的值不能超过 `capacity` 的值。`min_after_dequeue` 的值越大，则样本被打乱的效果越好。

具体代码如下。

代码 4-4 将 Excel 文件制作成内存对象数据集（续）

```

14 def create_pipeline(filename, batch_size, num_epochs=None):      #创建队列数据集函数
15     #创建一个输入队列
16     file_queue = tf.train.string_input_producer([filename],
17           num_epochs=num_epochs)
18
19     feature, label = read_data(file_queue)                         #载入数据和标签
20
21     min_after_dequeue = 1000 #在队列里至少保留 1000 条数据
22     capacity = min_after_dequeue + batch_size                      #队列的长度
23
24     feature_batch, label_batch = tf.train.shuffle_batch(#生成乱序的批次数据
25               [feature, label], batch_size=batch_size, capacity=capacity,
26               min_after_dequeue=min_after_dequeue

```

```

26 5.1 ) 样本介绍
27
28     return feature_batch, label_batch          #返回指定批次数据
29 #读取训练集
30 x_train_batch, y_train_batch = create_pipeline('iris_training.csv', 32,
31 num_epochs=100)                                #从文件中读取训练集
31 x_test, y_test = create_pipeline('iris_test.csv', 32) #读取测试集

```

程序的最后两行（第 30、31 行）代码，分别用 `create_pipeline` 函数生成了训练数据集和测试数据集。其中，训练数据集的迭代次数为 100 次。

4.4.4 代码实现：在会话中使用数据集

建立 `session`，准备运行静态图。在 `session` 中，先启动一个带有协调器的队列线程，然后通过 `run` 方法获得数据并将其显示。

具体代码如下：

代码 4-4 将 Excel 文件制作成内存对象数据集（续）

```

32 with tf.Session() as sess:
33
34     init_op = tf.global_variables_initializer()          #初始化
35     local_init_op = tf.local_variables_initializer()    #初始化本地变量
36     sess.run(init_op)
37     sess.run(local_init_op)
38
39     coord = tf.train.Coordinator()                      #创建协调器
40     threads = tf.train.start_queue_runners(coord=coord) #开启线程队列
41
42     try:
43         while True:
44             if coord.should_stop():
45                 break
46             example, label = sess.run([x_train_batch, y_train_batch])#注入训练数据
47             print ("训练数据: ",example)                         #打印数据
48             print ("训练标签: ",label)                          #打印标签
49     except tf.errors.OutOfRangeError:                   #定义取完数据的异常处理
50         print ('Done reading')
51         example, label = sess.run([x_test, y_test])      #注入测试数据
52         print ("测试数据: ",example)                      #打印数据
53         print ("测试标签: ",label)                        #打印标签
54     except KeyboardInterrupt:                         #定义按 ctrl+c 键对应的异常处理
55         print("程序终止...")
56     finally:
57         coord.request_stop()                           #停止线程

```

```

59     coord.join(threads)
60     sess.close()

```

在代码第 46 行, 用 `sess.run` 方法从训练集里不停地取数据。当训练集里的数据被取完之后, 会触发 `tf.errors.OutOfRangeError` 异常。

在代码第 49 行, 捕获了 `tf.errors.OutOfRangeError` 异常, 并接着将测试数据输出。

更多异常的知识请参考《Python 带我起飞——入门、进阶、商业实战》一书的第 7 章。



提示:

代码第 35 行, 初始化本地变量是必要的。如果不进行初始化则会报错。

4.4.5 运行程序

程序运行后, 输出以下结果:

```

.....
[5.7 4.4 1.5 0.4]
[6.2 2.8 4.8 1.8]
[5.7 3.8 1.7 0.3]]
训练标签: [0 2 0 1 0 2 2 2 0 1 2 1 1 1 0 2 2 0 2 2 2 0 0 2 1 2 0 0 1 1 0 0]
训练数据: [[5.1 3.8 1.6 0.2]
[6. 2.9 4.5 1.5]
.....
[7.6 3. 6.6 2.1]
训练标签: [0 1 2 0 2 0 1 0 1 2 0 2 0 1 2 0 0 0 0 1 0 1 0 2 1 0 2 2 0 1 2 0]
Done reading
测试数据: [[6.3 2.8 5.1 1.5]
[6.7 3.1 4.7 1.5]
.....
[6. 3.4 4.5 1.6]]
测试标签: [2 1 1 1 0 1 2 0 1 1 1 0 0 1 0 1 0 1 1 1 0 1 2 2 2 1 1 2 1 1 0 1]

```

4.5 实例 5: 将图片文件制作成 TFRecord 数据集

实例描述

有两个文件夹, 分别放置男人与女人的照片。

现要求: (1) 将两个文件夹中的图片制作成 TFRecord 格式的数据集; (2) 从该数据集中读数据, 将得到的图片数据保存到本地文件中。

TFRecord 格式是与 TensorFlow 框架强绑定的格式, 通用性较差。

但是, 如果不考虑代码的框架无关性, TFRecord 格式还是很好的选择。因为它是一种非常高效的数据持久化方法, 尤其对需要预处理的样本集。

将处理后的数据用 TFRecord 格式保存并进行训练, 可以大大提升训练模型的运算效率。

4.5.1 样本介绍

本实例的样本为两个文件夹——man 和 woman，其中分别存放着男人和女人的图片，各 10 张，共计 20 张，如图 4-7 所示。

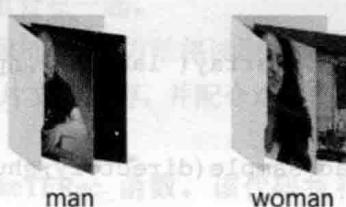


图 4-7 man 和 woman 图片样本

从图 4-7 可以看出，样本被分别存放在两个文件夹下。

- 文件夹的名称可以被当作样本标签（man 和 woman）。
- 文件夹中的具体图片文件可以被当作具体的样本数据。

下面通过代码完成本实例的功能。

4.5.2 代码实现：读取样本文件的目录及标签

定义函数 load_sample，用来将图片路径及对应标签读入内存。具体代码如下：

代码 4-5 将图片文件制作成 TFRecord 数据集

```

01 import os
02 import tensorflow as tf
03 from PIL import Image
04 from sklearn.utils import shuffle
05 import numpy as np
06 from tqdm import tqdm
07
08 def load_sample(sample_dir, shuffleflag = True):
09     '''递归读取文件。只支持一级。返回文件名、数值标签、数值对应的标签名'''
10     print ('loading sample dataset..')
11     lfilenames = []
12     labelsnames = []
13     for (dirpath, dirnames, filenames) in os.walk(sample_dir): #递归遍历文件夹
14         for filename in filenames: #遍历所有文件名
15             #print(dirnames)
16             filename_path = os.sep.join([dirpath, filename])
17             lfilenames.append(filename_path) #添加文件名
18             labelsnames.append( dirpath.split('\\')[-1] ) #添加文件名对应的标签
19
20     lab= list(sorted(set(labelsnames))) #生成标签名称列表
21     labdict=dict( zip( lab ,list(range(len(lab)))) ) #生成字典

```

```

22
23     labels = [labdict[i] for i in labelsnames]
24     if shuffleflag == True:
25         return
26     shuffle(np.asarray(lfilenames), np.asarray(labels)), np.asarray(lab)
27     else:
28         return
29     (np.asarray(lfilenames), np.asarray(labels)), np.asarray(lab)
30     directory='man_woman\\' # 定义样本路径
31     (filenames,labels),_ = load_sample(directory,shuffleflag=False) # 载入文件名称与标签

```

在代码第 6 行中引入了第三方库——`tqdm`，以便在批处理过程中显示进度。如果运行时提示找不到该库，则可以在命令行中用以下命令进行安装：

```
pip install tqdm
```

`load_sample` 函数的返回值有三个，分别是：图片文件的名称列表（`filenames`）、每个图片文件对应的标签列表（`labels`）、具体的标签数值对应的字符串列表（`lab`）。

在代码的最后两行（第 29、30 行），用 `load_sample` 函数返回具体的文件目录信息。

4.5.3 代码实现：定义函数生成 TFRecord 数据集

定义函数 `makeTFRec`，将图片样本制作成 `TFRecord` 格式的数据集。具体代码如下：

代码 4-5 将图片文件制作成 `TFRecord` 数据集（续）

```

31 def makeTFRec(filenames,labels): # 定义生成 TFRecord 的函数
32     # 定义 writer，用于向 TFRecords 文件写入数据
33     writer= tf.python_io.TFRecordWriter("mydata.tfrecords")
34     for i in tqdm( range(0,len(labels) ) ):
35         img=Image.open(filenames[i])
36         img = img.resize((256, 256))
37         img_raw=img.tobytes()    # 将图片转化为二进制格式
38         example = tf.train.Example(features=tf.train.Features(feature={
39             # 存放图片的标签 label
40             "label":
41                 tf.train.Feature(int64_list=tf.train.Int64List(value=[labels[i]])),
42             # 存放具体的图片
43             'img_raw':
44                 tf.train.Feature(bytes_list=tf.train.BytesList(value=[img_raw])),
45             }))          # 用 example 对象对 label 和 image 数据进行封装
46         writer.write(example.SerializeToString())    # 序列化为字符串
47     writer.close()                                # 数据集制作完成
48 makeTFRec(filenames,labels)

```

代码第34行调用了第三方库——tqdm，实现进度条的显示。

函数makeTFRec接收的参数为文件名列表(filenames)、标签列表(labels)。内部实现的流程是：

- (1) 按照filenames中的路径读取图片。
- (2) 将读取的图片与标签组合在一起。
- (3) 用TFRecordWriter对象的write方法将读取的图片与标签数据写入文件。

依次读取filenames中的图片文件内容，并配合对应的标签一起，调用TFRecordWriter对象的write方法进行写入操作。

代码第48行调用了makeTFRec函数。该代码执行后，可以在本地文件路径下找到mydata.tfrecords文件。这个文件就是制作好的TFRecord格式样本数据集。

4.5.4 代码实现：读取TFRecord数据集，并将其转化为队列

定义函数read_and_decode，用来将TFRecord格式的数据集转化为可以输入静态图的队列格式。

函数read_and_decode支持两种模式的队列格式转化：训练模式和测试模式。

- 在训练模式下，会对数据集进行乱序(shuffle)操作，并将其按照指定批次组合起来。
- 在测试模式下，会按照顺序读取数据集一次，不需要乱序操作和批次组合操作。

具体代码如下：

代码4-5 将图片文件制作成TFRecord数据集(续)

```

49 def read_and_decode(filenames, flag = 'train', batch_size = 3):
50     #根据文件名生成一个队列
51     if flag == 'train':
52         filename_queue = tf.train.string_input_producer(filenames) #乱序操作,
并循环读取
53     else:
54         filename_queue =
55             tf.train.string_input_producer(filenames, num_epochs = 1, shuffle = False)
56     reader = tf.TFRecordReader()
57     _, serialized_example = reader.read(filename_queue)      #返回文件名和文件
58     features = tf.parse_single_example(serialized_example, #取出包含image
和label的feature
59                                         features={
60                                             'label': tf.FixedLenFeature([], tf.int64),
61                                             'img_raw' : tf.FixedLenFeature([], tf.string),
62                                         })
63     #tf.decode_raw可以将字符串解析成图像对应的像素数组
64     image = tf.decode_raw(features['img_raw'], tf.uint8)

```