浙江工业大学

本科毕业设计说明书（论文）

（2019 届）

论文题目　基于 TensorFlow 的图片分类软件开发

作者姓名　黄金凯

指导教师　黄洪

学科(专业)　计算机科学与技术+计算机实验班1501

所在学院　计算机科学与技术学院

提交日期　2019 年 6 月

摘要

使用卷积神经网络进行图片分类是深度学习领域中的一个重要方面，卷积神经网络能够提取图像特征，对图像的细节进行分析，非常近似于人类大脑的处理方式。与全连接神经网络相比，卷积神经网络大大减少了所需的权重参数，提高了神经网络的训练效率。

本论文主要围绕卷积神经网络的设计与优化展开，目的是实现通过卷积神经网络对图像进行分类。卷积神经网络的性能取决于如何根据输入和输出数据的实际情况进行参数和结构调整，卷积核的大小、隐藏层数、梯度下降优化函数的选择等等都会对卷积神经网络造成重要影响。

本论文使用 Google 公司的 TensorFlow 框架进行神经网络的搭建，并实现了一个图片分类程序。首先建立了 10 层神经网络，包括输入层、卷积层、池化层、全连接层、Softmax 层。使用的数据集是 cifar-100，将数据集的数据输入神经网络，经过训练，得到结果计算损失，调整参数，直到损失值降低到一个合理的范围。之后使用训练完的模型对图片进行标签添加与分类放置操作。

完成训练的模型的在 cifar-100 验证集测试准确度达到 99%，普通图片分类问题也能基本得到解决，能够通过卷积神经网络的计算得到图片的正确标签。对于较多的图片，能够分别识别后加入到对应文件夹内。同时本程序还被移植到了 Orange Pi 3 上实现了在 ARM 架构 CPU 和 Linux 系统上的应用。

关键词：人工智能，卷积神经网络，图像分类，TensorFlow，深度学习

Abstract

The use of convolutional neural networks for image classification is an important aspect in the field of deep learning. Convolutional neural networks can extract image features and analyze the details of images, which is very similar to the processing of human brain. Compared with the fully connected neural network, the convolutional neural network can greatly reduce the required weight parameters and improve the training efficiency of the neural network.

This thesis focuses on the design and optimization of convolutional neural networks. The purpose is to classify images by convolutional neural networks. The performance of the convolutional neural network depends on how the parameters and structure are adjusted according to the actual conditions of the input and output data. The size of the convolution kernel, the number of hidden layers, the selection of the gradient descent optimization function, etc. all have an important impact on the convolutional neural network. .

This thesis uses Google's TensorFlow framework to build neural networks and implement a picture classification program. First, a 10-layer neural network is established, including an input layer, a convolution layer, a pooling layer, a fully connected layer, and a Softmax layer. The data set used is cifar-100. The data of the data set will be imported into the neural network. After training, the loss are calculated, and the parameters will be adjusted continuously until the loss value is reduced to a reasonable range. Then use the trained model to add label to images.

The accuracy of the cifar-100 verification set test is 99%, and the general picture classification problem can be basically solved. The correct label of the picture can be obtained by the calculation of the convolutional neural network. For more pictures, they can be identified and added to the corresponding folder. The program has been ported to the Orange Pi 3，which has an ARM CPU and running Linux system.

Keywords： Artificial intelligence, convolutional neural networks, image classification, TensorFlow, deep learning

目录

摘要 ..... **I**

ABSTRACT ..... **II**

第一章 绪论 **1**

1.1 研究工作的背景与意义 1
1.2 国内外研究现状 2
1.3 本论文的主要工作 3
1.4 本论文的组织结构 3
1.5 本章小结 4

第二章 相关理论与技术 **5**

2.1 全连接神经网络 5
2.1.1 深度前馈神经网络 5
2.1.2 去线性化 6
2.1.3 损失函数 7
2.1.4 Softmax 回归 7
2.2 神经网络的优化 8
2.2.1 梯度下降算法 8
2.2.2 反向传播 8
2.2.3 学习率的优化 9
2.2.4 滑动平均 9
2.2.5 过拟合与欠拟合 10
2.2.6 Dropout 方法 11
2.3 卷积神经网络 CNN 11
2.3.1 卷积操作 11
2.3.2 卷积运算的特性 12
2.3.3 多卷积核 14
2.3.4 池化操作 14
2.3.5 局部响应归一化（LRN） 15
2.3.6 基本的卷积神经网络模型 16
2.4 图像处理 17
2.4.1 图像预处理 17
2.4.2 图像的标准化 17
2.5 TensorFlow 介绍 18
2.5.1 基本情况 18
2.5.2 TensorFlow 的优点 18
2.5.3 TensorBoard 19
2.6 使用 GPU 加速训练 19
2.6.1 Nvdia CUDA 介绍 19
2.6.2 Tensorflow-gpu 与 CuDNN 20

第三章 图片分类系统的设计与实现 **21**

3.1 整体框架设计 21
3.1.1 实验环境 21
3.1.2 框架设计 21
3.2 图片数据集获取与处理 22
3.2.1 数据集的获取 22
3.2.2 数据集的处理 23
3.3 搭建神经网络模型与训练 24
3.3.1 搭建神经网络模型 24
3.3.2 神经网络的优化 25
3.3.3 神经网络的训练 26
3.3.4 神经网络的可视化 27
3.4 训练结果验证与分类系统实现 27
3.4.1 训练结果正确率验证 27
3.4.2 单张图片标签获取 28
3.4.3 图片批量分类系统 29
3.5 在 ORANGE PI 3 上实现图片分类 31

第四章 实验结果与分析 34

4.1 训练结果损失与分析 34
4.2 验证集正确率与分析 34
4.3 单张图片标签获取结果与分析 35
4.4 图片批量分类结果与分析 36
4.5 在 ORANGE PI 3 上实现图片分类 38

第五章 总结 40

5.1 完成的工作 40
5.2 存在的问题及下一步工作 40

第六章 参考文献 42

第七章 致谢 45

第八章 附录 46

8.1.1 附件 1 毕业设计文献综述 46
8.1.2 附件 2 毕业设计开题报告 46
8.1.3 附件 3 毕业设计外文翻译（中文译文与外文原文） 46

图目录

图 2-1 全接连神经网络基本结构 5
图 2-2 ReLU 函数图像 6
图 2-3 鞍点图像 8
图 2-4 Dropout 方法示意图 11
图 2-5 卷积运算 12
图 2-6 稀疏连接 13
图 2-7 池化操作 15
图 3-1 卷积神经网络框架设计 21
图 3-2 图片批量分类系统流程图 30
图 3-3 Orange Pi 3 硬件参数图 32
图 3-4 实验用 Orange Pi 3 运行图 32
图 4-1 loss 值变化图 34
图 4-2 验证集正确率验证运行结果 35
图 4-3 图像预处理后的输出 36
图 4-4 单张图片标签获取结果 36
图 4-5 作为样本的图片 37
图 4-6 批量分类结果 37
图 4-7 Orange Pi 3 上的批量分类结果 38
图 4-8 Orange Pi 3 上的程序运行截图 39

表目录

表 4-1 不同训练阶段时的损失值和验证正确率 35
表 4-2 多次运行分类的结果 37

第一章 绪论

1.1 研究工作的背景与意义

人工智能智能技术在近年来越来越流行，已然成为了整个计算机行业的重点[1]，各类基于人工智能技术的应用如雨后春笋般不断涌现，融入到我们生活中的方方面面，如人工智能围棋、人工智能美颜、自动驾驶等等[2]。同时，各大科技公司纷纷推出了自己的人工智能开发套件[3]，使得人工智能程序的开发难度大大降低，让更多初级开发者参与到人工智您的开发中来。

将人工智能技术应用于生活、服务于人类是该技术的最终目标。当人们拍摄照片越来越多，拥有一个能够自动分类、整理图片的图片分类系统将会大大减少人工筛选图片所带来的麻烦[4]。将人工智能技术应用于图片分类将会大大提高分类的成功率，通过不断自我学习，分类程序的准确率也会随时间大大增加。所以一个使用人工智能技术实现的图片自动分类系统将会是大多数人所需要的[5]。

如今人们使用手机等便携式设备拍摄照片越来越频繁，照片的数量越来越多，质量也越来越参差不齐。平时人们使用手机摄像头拍摄图片时并不会每拍一张就给它分个类，比如拍了一张花就加个植物标签，拍了个人就加一个人物标签。这一切都太过繁琐，所以大多数人们的相册都是由一大堆无序的图片构成，浏览和查询都非常麻烦。

使用人工智能技术将图片进行大致的分类，比如根据风景、人物、动物、课件等进行分类[6]，又或者使用人脸识别技术将不同人物的照片区分来将为大大提高相册整理的效率[7]。同时深度学习技术将会有效地改正错误地分类，在一次次的分类中不断根据用户对于分类正确性的反馈提高自己的准确率[8]。

在本项目中将使用卷积神经网络技术分类图片，解决相册浏览时杂乱无序的痛点。

1.2 国内外研究现状

对于深度学习的研究从很早就开始了，深度学习的提出为整个人工智能领域画出了浓墨重彩的一笔，彻底颠覆了整个人工智能的研究领域。深度学习不管在研究领域还是应用领域都绽放出自己的光彩，让人工智能制作的应用或产品真正融入到我们的生活之中[9]。

2006年，加拿大多伦多大学的Geoffrey Hinton教授和他的学生Ruslan Salakhutdinov在《SCIENCE》上发表了一篇关于深度神经网络的突破性文章，使得深度学习理念成为人工智能的主要研究方向[10]，这是整个人工智能领域革命性的一步。2011年以来在语音识别领域，微软和谷歌的研究人员使用深度神经网络技术降低了高达20%到30%的错误率，取得的历史性的重大突破。2012年，基于深度学习的卷积神经网络（CNN）在ImageNet图像数据集测评上将错误率由26%降低到15%[11]。2017年5月，AlphaGo智能围棋程序在乌镇围棋峰会上击败当时世界排名第一的人类棋手柯洁，并与八段棋手搭档完胜五位顶尖九段棋手组合[12]。

近年来，深度神经网络的发展更是快马加鞭，计算机行业以外其他行业也纷纷加入进来，将深度神经网络技术应用于自己的流水线或是产品上。比如，在人工智能图像识别方面，有很多关于汽车和车牌识别的，这也是由于自动驾驶技术越来越热门，而汽车和车牌识别正是自动驾驶技术的基础[13]。不过，有关汽车的技术毕竟直接关系到生命安全，世界各国对于自动驾驶技术依然持保守态度，所以以研究工作居多。

国内方面，百度在深度学习方面的投入了大量资金，2013年，百度成立了深度学习研究院[14]，目前引进推出了数款基于深度学习的产品。在图像识别方面，有百度识图应用，通过神经网络算法，百度识图对于图片的检索正确率达到了国际领先水平。还有百度语音助手等应用，也是通过神经网络技术来实现的。

目前主流的深度学习开发框架有TensorFlow、Caffe、Theano、Keras等[3]，每种框架都有自己的特点和面向的人群，其中TensorFlow是目前最为流行的框架。2016年4月，谷歌发布了宣布TensorFlow支持分布式训练，大大提高了开发和训练

的效率。

1.3 本论文的主要工作

本论文通过 TensorFlow 框架实现了一个图片分类系统，主要工作如下：

第一，设计整个神经网络结构，本论文共设计了 5 层卷积神经网络，当然这 5 层是宏观层面上的，具体层数会有所增加。这 5 层分别是输入层、卷积层、池化层、全连接层、Softmax 层。

第二，使用 TensorFlow 提供的函数与模块设计并搭建整个神经网络，为了达到需要的效果，需要对比不同的函数和模块，以找到最适合的一部分。

第三，针对数据集和训练过程中遇到的问题，对整个神经网络进行参数调优，并使用合适的函数处理数据，主要目标是在不减少神经网络性能的情况下最优化整个神经网络的训练效率和训练结果的准确度。

第四，对于神经网络的输出结果进行分析，并实现一个通过分析结果分类图片的系统，用于展示本论文的具体成果。本系统实现自动通过结果对应的图片标签将一组图片分类到对应标签的文件夹内。

第五，将本程序移植到 Orange Pi 3 上，实现了在 ARM 架构 CPU 和 Linux 系统下的应用。

1.4 本论文的组织结构

本主要围绕卷积神经网络的设计与优化两方面展开，共分为五章，各章内容如下：

第一章，阐述了本论文的研究背景与研究意义，国内外相关领域的研究现状及应用，课题研究的主要任务和本文的主要工作。

第二章，介绍了在本次实验中用到的较为重要的相关理论与基础：全连接神经网络相关术语、神经网络的优化、卷积神经网络的主要结构和相关算法，图像的处理、TensorFlow 的介绍以及使用 GPU 来加速运算的原理与所用工具。

第三章，详细介绍了本次实验所用的卷积神经网络的设计与实现，以及图片

分类系统的设计与实现。

第四章，对实验的结果进行统计与分析。

第五章，总结完成的所有工作并提出存在的问题和要改进的方向。

1.5 本章小结

本章简要分析项目的研究背景、在国内外相关领域的开发和应用现状以及项目的研究的任务和意义。最后，给出了本文的主要工作及本文的组织结构。

第二章 相关理论与技术

2.1 全连接神经网络

2.1.1 深度前馈神经网络

深度前馈神经网络，简称前馈神经网络，是具有前馈特征的神经网络模型，其中最具代表性的样例是多层感知机制（MLP）模型[15]。

神经网络的连接方式分为全连接和稀疏连接，顾名思义，全连接神经网络就是每个神经元都与上一层的神经元有一条连接，而稀疏连接则是部分连接。

全部使用全连接方式进行连接的网络，称之为全连接神经网络。

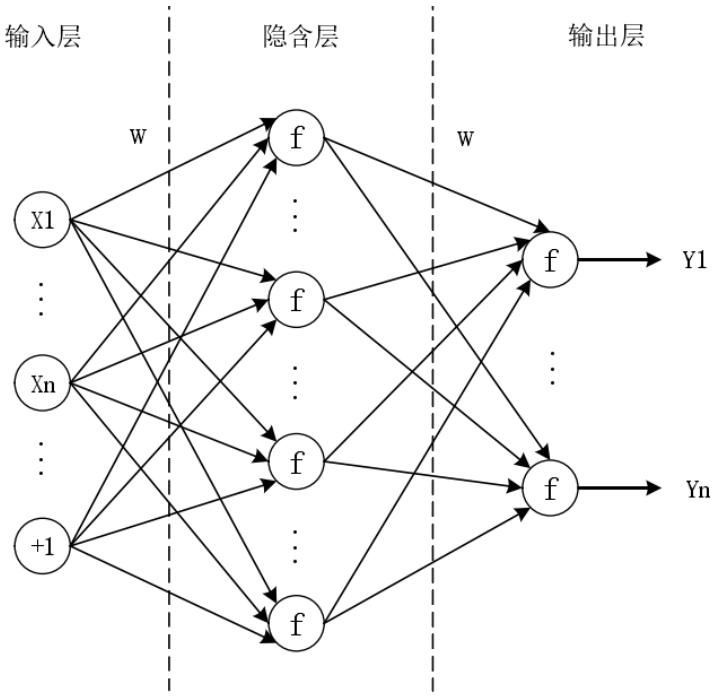

图 2-1 全接连神经网络基本结构

在图 2.1 中，全连接神经网络可以分为输入层，隐藏层，输出层。每个圆圈表示一个神经元，每个神经元可以有多个输入与一个输出，w 表示每条连接的权重，每条连接的权重相互独立。X 表示输入，f 表示隐藏层和输出层的处理函数。Y 表示输出的结果。

每一个节点的取值都是由输入层的取值的加权和与偏置值的和，

$$f=\sum_i w_i x_i+b \tag{2-1}$$

数据从输入层触发，一层层不断向前推进，最终输出结果，这就是所谓的前向传播算法。

2.1.2 去线性化

上一节介绍了基本的全连接神经网络，但该网络存在很大的缺点，就是不管神经网络的层数有多少，只要经过线性变换，最终的模型还是可以表示成单层模型。而且这些模型都是线性模型，线性模型所能解决的问题是有限的，当数据分布较为复杂时，线性模型很难做出合理的分类。举个简单的例子，一根直线不能将一个圆分为圆内和圆外两部分，只有一条曲线才能做到。

在TensorFlow的官方教学平台有一个神经网络测试平台——TensorFlow游乐场，它通过可视化的方法，将神经网络的构造展示出来，用它可以模拟在线性模型下解决复杂的二分问题，结果显示完全无法区分。

所以要对神经网络进行去线性化。这里就要用到激活函数，在这里介绍一种激活函数——线性整流函数（ReLU）。

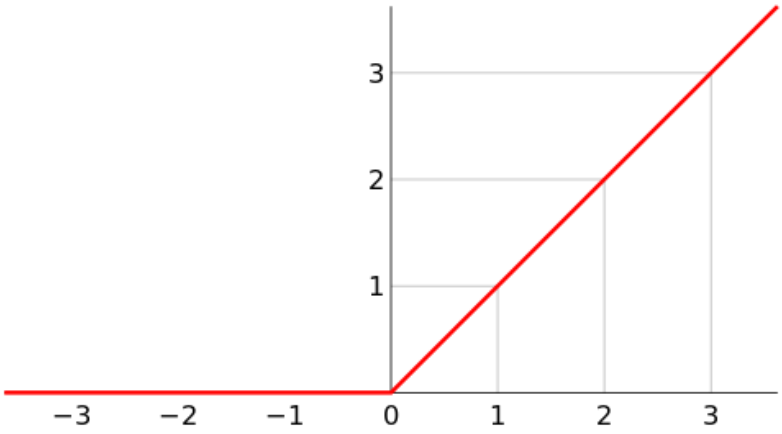

图 2-2 ReLU 函数图像

ReLU函数的定义是$f(x)=\max(0,x)$，当函数处于激活状态时（输出不为0），它的一阶导数恒为1，二阶导数恒为0，这些特性对于神经网络的优化有很大的帮助。

将激活函数用于除了输入层外的每个神经节点，即可使得整个神经网络去线性化。

2.1.3 损失函数

为了解决分类问题，通常会定义一个损失函数来描述预测结果与真实答案之间的区别。损失越小，模型与真实情况的符合度也就越高。

这里将会介绍一种经典损失函数——交叉熵（Cross Entropy）损失函数[16]，交叉熵用于描述两个概率分布向量之间的差距，在分类问题中使用较为广泛，可以用以下式子表示：

$$H(P,Q) = -\sum_{x} P(x)\log Q(x) \qquad (2\text{-}2)$$

在该式子中，P 表示正确答案，Q 表示预测的结果值，H 表示交叉熵，损失值越小，就代表 Q 越接近 P。而神经网络要做的就是尽可能减少损失，这样就能让预测的结果更加接近真实答案。

2.1.4 Softmax 回归

任意事件的概率分布都应该在 0-1 之间，所有事件发生概率的总和应为 1。在神经网络的分类问题中，一般希望最后的输出结果也应为概率值，且总和为 1，这样才方便计算交叉熵。这里就要用到 softmax 函数进行回归处理[17]，公式如下：

$$soft\max(y)_i = \frac{\exp(y_j)}{\sum_{j=1}^{n}\exp(y_j)} \qquad (2\text{-}3)$$

其中，i 表示经过 softmax 后的每一个单元，y_j 表示输入到 softmax 之前的每一个单元值，通过 softmax 函数处理使得输入的数值向量变成了概率分布向量。Tensorflow 提供了 tf.nn.softmax()函数来实现 softmax 回归功能。

2.2 神经网络的优化

2.2.1 梯度下降算法

梯度下降算法和数学中的导数有很大的联系，对于一个函数$l=J(\omega)$，其导数为$J'(\omega)$，如果输出发生微小变化，输出也会产生变化：

$$J(\omega+\sigma)\approx J(\omega)+\sigma J'(\omega) \tag{2-4}$$

沿着函数导数增大的反方向移动 ω 而获得更小的$J(\omega)$的技术就被称为梯度下降。可以将其应用到神经网络中，通过梯度下降算法更新 ω 也就是各边的值，就能找到损失函数的最小值。

但是通过梯度下降算法得到最小值只是一种理想情况，根据学习率的设置不同，损失函数最终可能停留在鞍点或只到达局部最小值，而且无法到达全局最小值。所以需要设置合理的学习率以及使用合适的方法来获取全局最小值[18]。

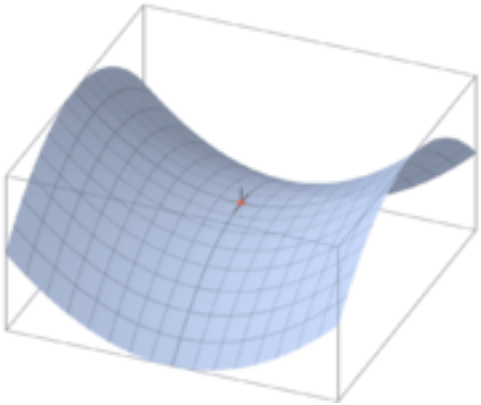

图 2-3 鞍点图像

使用随机梯度下降算法可以在一定程度上优化鞍点与局部最小值问题。随机梯度下降算法会使用选取一批次的样本获取损失函数并求导，获取平均后的梯度，这样可以以较低的性能消耗获得一定的最优解方向。在神经网络训练中会用到进阶的版本就是小批量梯度下降算法[19]，使用这种算法时会选取一定数量批次的样本，来获取梯度。

2.2.2 反向传播

反向传播（Backpropagation，缩写 BP）算法是与梯度下降算法结合使用的常

用方法，使用反向传播算法可以对神经网络中的所有权重计算损失函数的梯度，而使用该梯度产生的梯度下降结果会反馈给最优化方法，用于更新权值。

同样，在训练过程中，学习率的设置也可以通过反向传播算法进行优化，一般希望学习率能在偏导值较大时被设置为较大的值，以加快训练，而在后期偏导值较小时被设置为较小的值，来更加精确地找到损失函数的最小值，一些自适应学习率算法可以帮助我们更好的设置学习率，如 AdaGrad 算法、RMSProp 算法、Adam 算法等[20]。

在 TensorFlow 中，提供了一些封装好的优化器，如 Optimizer，GradientDescentOptimizer 等。

2.2.3 学习率的优化

之前在梯度下降算法中提到了学习率，学习率用于控制梯度下降中的参数更新速度。学习率过大，有可能造成损失值无法收敛；如果学习率过小，就会造成大大降低模型的训练速度。

所以需要一个合理的学习率，通常使用指数衰减法来动态更新学习率[21]，指数衰减公式如下：

$$decayed\_learning\_rate = learning\_rate \times dacay\_rate^{\frac{global\_step}{decay\_steps}} \quad (2\text{-}5)$$

其中 decayed_learning_rate 表示每一轮优化时所使用的学习率，learning_rate 表示预设的初始学习率，decay_rate 表示衰减系数，decay_steps 表示衰减的速度。

首先设置学习率为一个较大的值，但不大于 1，使得开始时能够较快的收敛，之后根据指数衰减公式更新学习率，这样就能使学习率平滑下降，找到更精确的最小值。

2.2.4 滑动平均

滑动平均(exponential moving average)，或者叫指数加权平均(exponentially weighted moving average)，可以用来估计变量的局部均值，该均值被称为影子变量[22]，影子变量的更新与一段时间内的历史取值有关。滑动平均的公式为：

$$v_t = \beta \times v_{t-1} + (1-\beta) \times \theta_t \quad (2\text{-}6)$$

其中v_t和θ_t表示变量v在 t 时刻的值，前者是滑动平均后的值，后者是滑动平均前的值，β为自定义值，且β在 0 到 1 之间。

滑动平均变量可以反应一段时间内值的均值，虽然准确度不如直接所有值求平均，但能减少需要保存的变量的数量，在训练过程中对权重参数使用滑动平均可以使得模型更为健壮。

2.2.5 过拟合与欠拟合

能在未知的新输入数据上表现良好的能力被称为泛化（Generalization），模型在未知的新输入数据上得到的误差称为泛化误差（Generalization Error），在训练中，总希望泛化误差尽可能的低。

当使用训练集进行神经网络训练时会出现结果过度趋向与训练集，导致训练模型在训练集上表现良好，而对于新输入的数据则表现较差，这就是过拟合现象。而欠拟合现象指的是不能在训练集上获得足够低的误差[23]。

解决过拟合问题可以使用正则化方法，正则化（Regularization）将会在损失函数中加入正则化项的惩罚。假设 R(w)表示在损失函数中加入的正则化项，λ 表示自己决定的偏好程度，λR(w)就表示要加入损失函数的损失值。

常用的 R(w)函数有两种，L1 正则化(对权重参数 w 求L^1范数)：

$$R(w) = \|w\|_1 = \sum_i |w_i| \quad (2\text{-}7)$$

以及 L2 正则化：

$$R(w) = \|w\|_2{}^2 = \sum_i |w_i|^2 \quad (2\text{-}8)$$

其中 L1 正则化会使得参数更加稀疏（会有更多参数变为 0），而 L2 正则化不会。同时 L1 正则化公式不可导，而 L2 正则化公式可导，使得在计算 L2 损失函数的偏导数时会比 L1 更加简洁。

在 TensorFlow 中提供了 L2 正则化函数——tf.nn.l2_loss()，用于计算 L2 正则化损失。

2.2.6 Dropout 方法

除了正则化外，还有一种方法能缓解过拟合问题，那就是 Dropout 方法。Dropout 方法会随机丢弃神经网络除了输入层外某一层的部分神经元。

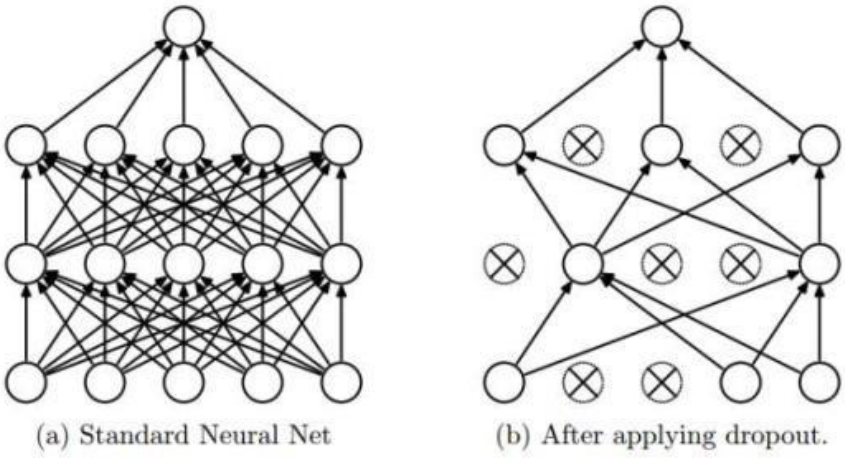

图 2-4 Dropout 方法示意图

如图所示，每一轮训练都会有部分神经元被舍弃，但是舍弃只是暂时的，下一轮时节点会恢复原样并再次随机舍弃，这样可以防止部分过拟合化的节点影响了整个网络，通过不同的神经网络取平均的方法，解决过拟合问题。

在 TensorFlow 中可以使用 tf.nn.dropout()函数来使用该方法。

2.3 卷积神经网络 CNN

卷积神经网络一般用于处理具有网格结构的数据，而图片可以视作由一个个像素点构成的矩阵，所以卷积神经网络在解决图像相关问题上表现优秀[24]。

相比于全连接神经网络，卷积神经网络加入了卷积层与池化层，两者分别代表卷积操作与池化操作[25]。

2.3.1 卷积操作

卷积运算是指使用卷积核去处理一个输入的矩阵，输出会被成为特征映射。其中卷积核是一个小于输入数据的矩阵，假设输入的数据为 I(m, n)，其中 m，n 表示坐标，卷积核为 K，S(i, j)表示特征映射，则卷积操作的公式可以表示为：

$$S(i,j)=(K\times I)(i,j)=\sum_{m}\sum_{n}I(i+m,j+n)K(m,n) \quad (2\text{-}9)$$

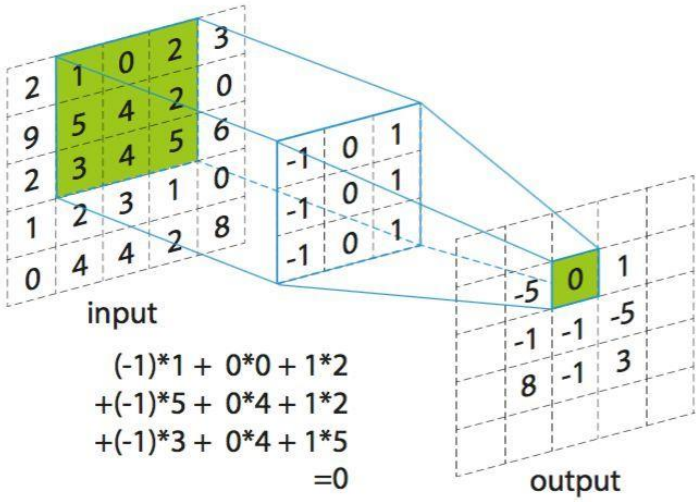

图 2-5 卷积运算

结合图 2-5 和公式 2-9，可以清晰的表示卷积操作，也就是将卷积核大小的一块输入区域对应位相乘之和输出成一个数，之后卷积核可以按照步长进行左右及上下移动，直到映射外所用输入数据。

当矩阵的深度大于 1 时，卷积核的深度应与输入相同，从而可以对每一层进行卷积操作，得到的结果进行相加就能得到卷积操作后的结果。

2.3.2 卷积运算的特性

（1） 卷积运算的稀疏连接

卷积运算具有稀疏连接特性，这也是区别于全连接的方面。稀疏连接指的是每一层的神经元都与上一层的神经元有连接，在数据特别庞大特别是图像处理方面，使用全连接将会带来大量的参数，大大加大了神经网络的负担，而且绝大部分都是无用的，此时就应该使用稀疏连接提取重要特征。

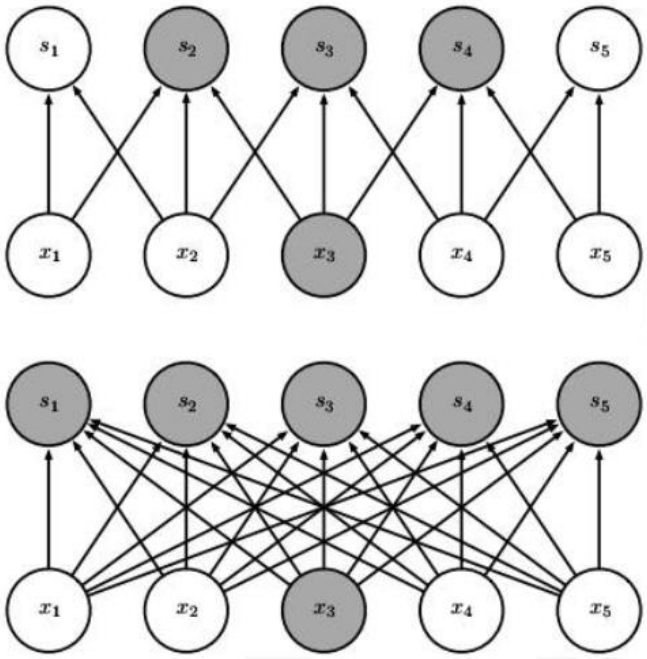

图 2-6 稀疏连接

卷积运算的稀疏连接借鉴了动物识别图像的方式：从局部到全局，每一个感受野之接受一小块特定区域的信号，在这块区域中，像素的相关性较大。比如我们看到一个盛着苹果和橘子的盘子的图像，我们不会一个一个像素点去看整张图像，而是看到一个苹果，一个橘子，一个盘子，假设有三个神经元，就只需要对应三样物体的图像像素而不用每一个都对应所有图片中像素点，这就是稀疏连接对应局部特征的例子。

稀疏连接最大的作用就是减少权重参数，从而减少了计算的复杂度以及过多权重参数带来的过拟合问题。

（2） 卷积运算的参数共享

参数共享指的是同一层的每个神经元共享相同的参数。在全连接神经网络中，假设一个神经元与上一层有 100 个连接，也就有 100 个权重参数，如果有 100 个神经元，就有 10000 个权重参数。在卷积运算中，100 个神经元共享所有权重参数，且数量为卷积核的大小，这就是参数共享。

参数共享的最大作用就是大大减少了权重参数的数量，结合稀疏连接的特性，在实际的训练过程中，大大减轻了计算负担，提高了训练效率。

（3） 卷积运算的平移等变

等变是数学中的一个概念，指的是某个函数的输入改变，输出也会以同样的方式改变，如$f(g(x)) = g(f(x))$，就可以称为 f(x)对 g 具有等变性。

应用于卷积运算中就是，如果对图像进行平移后再进行卷积，结果和卷积后再进行平移得到的结果是一样的。这就意味着，当输入图像内的某些部分移动时，输出的对应部分也会产生相同的移动，当然这是在单卷积核的条件下，当卷积核有多个时，某些变换可能并不能等变变换。

2.3.3 多卷积核

使用卷积核的目的在于提取图中的基本特征（点、线、面），当卷积核只有一个时，提取到的可能只是图中的某个特征（如特定朝向的边）。所以，如果需要提取图片中更多的特征信息，就需要多个独立的卷积核分别对图片进行过滤[29]。多卷积核的好处就是能提高提取出的特征的数量，对于后续的图片分类有很大的提升，而且由于卷积核包含的参数本身就较少，即使使用多卷积核，对于总体权重参数的增加也是有限的，并不会加重神经网络的计算量，但是多层叠加后依然会出现一些问题，这就需要之后的池化操作。

在 TensorFlow 中，卷积核被称为“过滤器（Filter）”，相比于卷积核，过滤器的称法更加容易理解，在实际编程中，卷积核的大小、数量都是可以通过 TensorFlow 提供的方法进行设置的。

2.3.4 池化操作

使用多卷积核后参数的增长虽然有限，但是多个隐藏层叠加之后，权重参数依然会到达一个恐怖的数字，将这么多的权重参数全部加入到分类器（Softmax 层）中时，显然是不合适的。这时候就需要精简权重参数，这就需要池化操作，池化操作会将平面内某一位置及其相邻位置的结果进行汇总，并将汇总后的结果作为当前区域的值。池化函数分为最大池化（max pooling）函数与平均池化（average pooling）函数，最大池化函数会将区域内最大的值作为当前区域的值，平均池化函数会将区域内所有值的平均值作为当前区域的值，

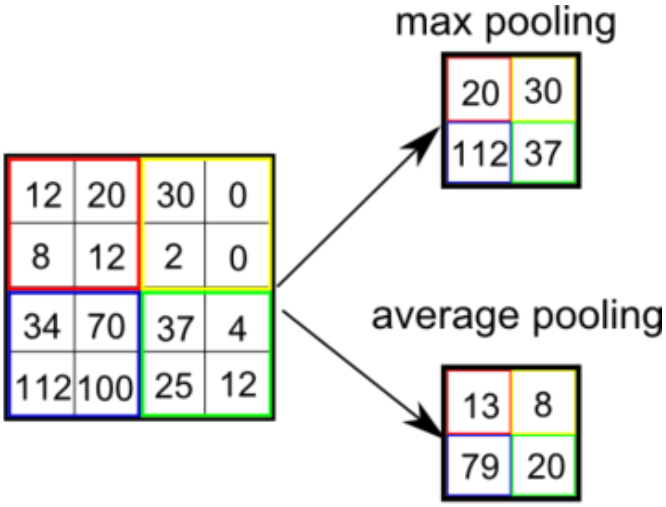

图 2-7 池化操作

图中将 4×4 的矩阵通过池化操作变成了 2×2 的矩阵，减少了权重参数。

特征提取的误差主要来自于两个方面：

（1） 领域大小受限造成估计值方差增大；

（2） 卷积层参数误差造成估计均值的偏移。

最大池化函数能减小第二种误差，也就是说能保留更多纹理特征，而平均池化函数能减小第一种误差，可以保留更多图片的背景信息。

2.3.5 局部响应归一化（LRN）

局部响应归一化（Local Response Normalization，简称 LRN）技术是深度学习训练时提高准确度的一种常见方法，一般跟在池化操作之后。在神经生物学中有一个概念叫侧抑制，指的是被激活的神经元抑制相邻的神经元的情况。归一化的目的就是去抑制部分神经元，而局部响应归一化正是借鉴了侧抑制的思想来实现局部抑制[26]。使用局部响应归一化可以一定程度提高模型的泛化能力。局部响应归一化的公式为：

$$b_{x,y}^{i}=a_{x,y}^{i}\Big/\left(k+\alpha\sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)}(a_{x,y}^{j})^{2}\right)^{\beta} \qquad (2\text{-}10)$$

在 TensorFlow API 中的实现形式如下：

sqr_sum[a, b, c, d] = sum(input[a,b, c, d - depth_radius : d + depth_radius + 1] ** 2)

output = input / (bias +alpha * sqr_sum) ** beta

公式中 a 表示卷积和池化操作后的输出结果，这是一个四维数组[batch_size, height, width, channel]，batch_size 表示批次大小，height 表示图片高度，width 表示图片宽度，channel 表示通道数，$a_{x,y}^{i}$表示输出中的一个具体位置，i 为通道，可以理解为某一张图在某一通道下某个高度某个宽度的值，a，n/2，k，α，β 分别表示 TensorFlow 函数中的 input，depth_radius，bias，alpha，beta，其中 n，k，α，β 都是自定义的。通俗点讲，就是 n 表示临近的通道数，N 表示总通道数，其他参数都是超参数，由验证机决定，举个例子，i=6，n=4，N=100，一次局部响应归一化过程就是用$a_{x,y}^{6}$除以 4、5、6、7、8 层在(x, y)位置提取出的特征之和。

通过公式可以看出，LRN 层可以在局部神经元间创建竞争机制，使得响应比较大的值变得更大，并抑制其他反馈较小的神经元，增强了模型的泛化能力。

2.3.6 基本的卷积神经网络模型

一个基本的卷积神经网络模型一般分为输入层、积层、池化层、全连接层和 Softmax 层：

(1) 输入层

用于神经网络数据的输入，在图像分类中，输入为图像的像素矩阵，对于黑白图片，只有一个通道，深度为 1；对于 RGB 图片，有三个颜色通道，深度为 3，每一层通道都表示一种颜色信息。

(2) 卷积层

卷积层就是执行卷积操作的层，通过使用卷积核提取图片特征，同时为了能提取到更多特征，卷积核一般会有多个，也就是说输出的深度可能会比输入更深。每个卷积核有长宽深三个维度，其中长宽需要指定，但是深度是由输入决定的，一般长宽设置为 3×3 或 5×5，深度对于 RGB 图像来说就是 3。

由于卷积操作实际上也是一种线性变换，所以在卷积操作后也是需要进行去线性化操作的。

(3) 池化层

池化层的作用是执行池化操作，池化操作能缩小来自池化层的矩阵尺寸，但并不会改变深度，其主要作用是减少整个神经网络的参数。池化层一般跟在卷积层之后，可以设置多个卷积层与多个池化层来提高识别准确度。

(4) 全连层

在所有卷积和池化操作完成后，需要经过几个全连层。经过了几轮池化和卷积操作后，图片得到了抽象的表达，但为了分类这些信息，依然需要全连层的帮助。

(5) Softmax 层

Softmax 层的作用就是将得到的结果映射成每种类别的比例，得到每种类别的概率分布情况。

2.4 图像处理

2.4.1 图像预处理

神经网络训练和验证的前提是每张输入的图片大小、分辨率等参数相同，所以需要在训练和验证开始前将图片处理成相同的参数。在 TensorFlow 中提供了简单的图片处理函数，能够对图片进行缩放、剪裁、解码、翻转等操作。

通过图像的预处理，能够尽可能的避免无关因素的影响以及达到数据增强的目的，从而提高了模型的准确度。

2.4.2 图像的标准化

对图像进行标准化的作用是将数据进行去均值实现中心化，数据中心化符合数据的分布规律，能提高泛化效果。

图像的标准化是整个图像处理过程的最后一步，标准化处理的公式如下：

$$s\tan dardization = \frac{x-\mu}{stddev} \qquad (2\text{-}10)$$

其中：

$$stddev = \max\left(\sigma, \frac{1.0}{\sqrt{N}}\right) \quad (2\text{-}11)$$

式中 μ 表示图像的均值，x 表示图像矩阵，σ 表示标准方差，N 表示图像 x 的像素数量。

在 TensorFlow 中提供了 tf.image.per_image_standardization(image)函数来对图像进行标准化，参数 image 表示一个三维 tensor，分别为高、宽、通道数。

2.5 TensorFlow 介绍

2.5.1 基本情况

TensorFlow 是由 Google 公司推出的一个用于机器学习的开源软件库，使用 TensorFlow 可以方便地搭建一个机器学习系统。目前 TensorFlow 被广泛用于各种人工智能领域，并且是使用人数最多的开源深度学习框架[27]。

2.5.2 TensorFlow 的优点

（1） 高灵活性

在 TensorFlow 中，神经网络被看成一个数据流图，每一个节点都可以用 Tensor 来表示，开发者可以使用计算图来建立整个复杂的计算网络，同时也可以方便地对网络进行操作。开发者可以使用 Python 语言来编写上层结构和库，如果没有合适的 API，也可以自己编写底层的 C++代码，并将功能加入 TensorFlow 中。

（2） 可移植性

关于硬件支持，TensorFlow 可以在 CPU、GPU、TPU 上运行。对于使用平台，TensorFlow 可以在台式机、笔记本、甚至移动设备上运行。对于操作系统，TensorFlow 可以在 Windows、Linux、MacOS、Android 等系统上运行。得益于如此多的支持，TensowFlow 实现了真正的可移植性，在任意平台使用 TensorFlow 训练好的模型可以在任意其他平台上使用，且不需要修改代码。

（3） 多语言支持

TensorFlow 支持多种语言，如 Python、Java、C++、Go 语言等，可以使用一

种语言来使用 TensorFlow，也可以使用多种语言的组合来实现系统。

（4） 完善的文档与支持

TensorFlow 的官网提供了非常详细的使用教程，从安装到入门再到精通，TensorFlow 都可以帮助开发者实现。同时 TensorFlow 提供了所有 API 的详细说明，在开发过程中遇到的需要不熟悉的函数时，都可以得到详细解答。同时由于 TensorFlow 是目前使用最为广泛的深度学习框架，交流社区也非常活跃，开发中遇到的问题也能被其他开发者解答。

2.5.3 TensorBoard

TensorBoard 是一款 TensorFlow 的可视化工具，通过网页的形式展示 TensorFlow 程序的运行状态。可以在编程过程中将需要跟踪的变量以各种形式在 TensorBoard 中展示出来。TensorBoard 能展示以下内容：SCALARS、IMAGES、GRAPHS、DISTRIBUTIONS、HISTOGRAMS、EMBEDDINGS、TEXT。

2.6 使用 GPU 加速训练

2.6.1 Nvdia CUDA 介绍

CUDA（Compute Unified Device Architecture，统一计算架构）是由 NVIDIA 公司所推出的一种集成技术，使用该技术可以利用 GPU 加速一些计算。

GPU 核心中一般具有远多于 CPU 核心数量的内处理器，如 GTX1070 拥有 1920 个内处理器，使用 CUDA 技术可以将这些内处理器联合起来，成为线程处理器去解决部分数据密集型的计算。由于 GPU 的各个内处理器能够快速同步数据与共享内存，使得联合运算具有较高效率[28]。

与 CPU 相比，得益于大量的内处理器，使用 CUDA 的 GPU 可以处理大量并行化的问题，在实际运算中，GPU 中可以同时执行上千个线程，虽然每个内处理器的性能较低，但在执行多线程高密度运算时能快速解决问题。

神经网络中存在大量节点和大量参数，每个节点的计算需求不高但总节点数较高，非常适合使用 CUDA 来进行计算，事实证明，使用 GPU 进行神经网络训

练时，效率远远高于使用同价位 CPU。

2.6.2 Tensorflow-gpu 与 CuDNN

TensorFlow 提供了 GPU 版本，可以使用 NVIDIA 的 GPU 来加速模型的训练。要使用 tensorflow-gpu 需要安装 CUDA 和 CuDNN。CuDNN 是由 NVIDIA 公司专门为神经网络打造的加速库。使用 CuDNN 可以大幅加速神经网络在 GPU 上的运算[30]。

第三章 图片分类系统的设计与实现

3.1 整体框架设计

3.1.1 实验环境

（1） 操作系统：Window 10

（2） 开发语言：Python

（3） 开发框架：TensorFlow

（4） 开发集成环境：Pycharm

（5） 硬件环境：

显卡：NVIDIA GTX 1070 8GB

CPU: 酷睿 i7 6400t

内存：12GB

3.1.2 框架设计

本次实验使用卷积神经网络，共分为输入层、卷积层、池化层、全连接层、Softmax 层，其中除了输入层和 Softmax 层均为一层外，其他层都不止一层。具体框架设计如下：

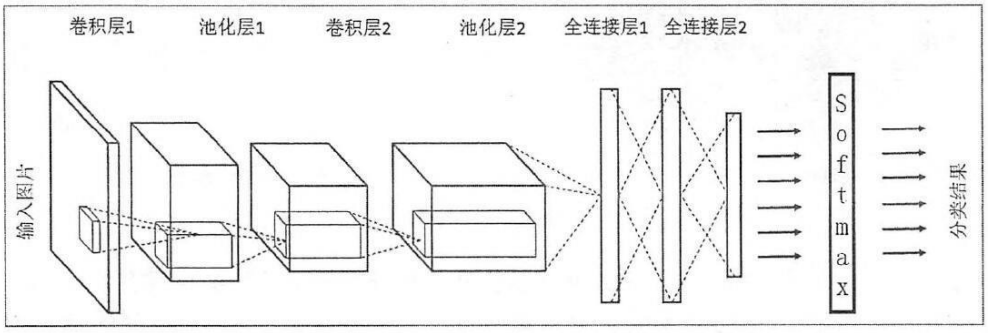

图 3-1 卷积神经网络框架设计

(1) 输入层：

输入层用于输入图片或数据集进行训练或验证，需要将图片或数据集转制成

多维向量或 TensorFlow 专用的数据格式。同时需要对输入的图片进行剪裁变形处理，从而提高图片噪声，以应对图片变形的情况。在本项目中使用 cifar100_input.py 模块实现。输出的结果为图片图片矩阵数组和标签向量，两者都为 TensorFlow 可识别的张量。

(2)卷积层：

卷积层用于将输入的数据进行卷积操作，卷积操作可以提取图片特征且显著减少所需的权重参数，通过使用多个卷积核可以提高特征提取数量。在卷积层中需要进行卷积核生成、卷积运算、去线性化操作。

(3) 池化层：

池化层用于将来自卷积层的输出进行压缩，在本次实验中使用最大池化，从而在减少数据量的情况下更多的保留纹理信息，在池化完成后还要进行归一化操作，也就是将所有值全部映射到一定范围，提高模型泛化能力。

(4) 全连接层：

在完成所有卷积和池化操作后，需要进行全连接层的处理，来更好的分类数据。全连接层要做的就是把上一层输出的多维数组拉直成一维向量，以便于后面的分类。

(5) Softmax 层：

Softmax 层是将全连接层所输出的向量转换成 label 的数量，以便于之后计算各种情况的概率。

3.2 图片数据集获取与处理

3.2.1 数据集的获取

数据集的获取是整个搭建整个神经网络前首先要做的事，目前主流的数据集有 MNIST，MS-COCO，cifar，ImageNet，OpenImage，SVHN 等，每种数据集都有专门注重的领域，如 MNIST 注重手写体识别，而 cifar-10 注重常见单个物体的识别。本次实验的目的是实现生活中常见物体的识别，所以选择了 cifar-100 数据集。

cifar-100 数据集是从由 Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton 收集的 8000 万图像中筛选出的子集，其包含了 20 个大类，100 个小类，每个小类有 600 个图像，其中 500 个被划分为训练集，100 个被划分为测试集。Cifar-100 中的每张图片拥有一个“大类”标签和一个“小类”标签，二进制结构为<大类标签><小类标签><3072 像素>，这里图像并不是由矩阵构成的而是矩阵拉直后的向量，总计有 60000 行。

使用 cifar-100 数据集的方法有很多，官方网站提供了该数据集的 python、Matlab 和二进制版本，可以将数据处理后导入到神经网络中，也可以将数据转换成 TensorFlow 专用 TFRecord 格式以便导入。从 TensorFlow 1.3 开始，官方提供了 dataset API 来实现数据的输入处理，经过多个版本的不断更新，dataset 的功能也越来越强大，现在可以直接使用 tfds.load()函数来导入主流的数据集，cifar-100 就是其中之一。

在本实验中使用_get_images_labels()函数来实现数据导入功能。首先通过 dataset = tfds.load(name='cifar100', split=split)来导入 cifar-100 数据集，之后进行划分操作，将数据集划分成一个个 batch，并创建迭代器，使用迭代器的好处是不用一次性将所用数据到加入神经网络中，从而减少内存占用和提高效率。最终返回的是图像迭代器和标签迭代器。

3.2.2 数据集的处理

导入数据集之后就需要对数据进行处理，图像处理的主要作用是数据增强和较少无关因素的影响，从而提高模型的准确度。TensorFlow 提供了一些简单的图像处理工具，借助这些处理函数能简便地处理导入的图像。

在本次实验中，使用 DataPreprocessor 类来处理所有图像，根据导入类型的不同，处理的方式也会有所不同。

对于训练集，首先要对图像进行随机剪裁，使用 tf.random_crop(img, [IMAGE_SIZE, IMAGE_SIZE, 3])函数可以将图像剪裁成指定的大小。剪裁图像可以使所有图像有相同的大小，更利于神经网络的训练。之后将会进行图像的随机

化操作， 使用 tf.image.random_flip_left_right()函数进行随机左右翻转，使用 tf.image.random_brightness()来进行随机亮度调整，使用 tf.image.random_contrast()来进行随机对比度调整，通过随机化的处理，提高了图片的噪声，加强了训练后模型的泛化能力和抗干扰能力，使得分类结果更为准确。

如果导入的数据为验证集则不需要做和训练集一样的操作，只需进行剪裁来使图形的大小能够满足输入的标准，图像的大小应该要与训练时相同才能被模型所接纳。

最后要进行图像的标准化操作，处理完后输出一个包含图像和标签的字典。

3.3 搭建神经网络模型与训练

3.3.1 搭建神经网络模型

在本章第一节已经完成了模型的设计，本节将介绍模型的搭建部分，主要是隐藏层如卷积层、池化层的搭建过程。

首先需要生成参数，生成参数由 _variable_with_weight_decay() 和 _variable_on_cpu()函数实现，_variable_on_cpu()的目的是在 CPU 上完成参数设置工作，因为 GPU 虽然在神经网络的运算上表现很好，但参数生成是在内存中实现的，内存与 CPU 的通讯效率比 GPU 要高得多。使用 with tf.device('/cpu:0'):可以使 TensorFlow 强制使用 CPU 进行操作。

隐藏层中第一层是卷积层，在本实验中使用 inference()实现整个隐藏层的创建，conv1 表示第一个卷积层，tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')函数表示使用卷积核 kernel 对图像进行卷积处理，SAME 方式为进行填充。之后需要加入偏置值 biases 并使用 tf.nn.relu()函数去线性化。此时生成的卷积核并不需要加入正则化损失。

接下来是池化操作，pool1 为第一个池化层，在 TensorFlow 中可以使用 tf.nn.max_pool()来方便地实现最大池化操作，只需指定核大小核步长即刻实现池化操作。之后再池化层还会完成局部响应归一化操作，提高模型泛化能力。

之后会是卷积层和池化层的重复，这就涉及到优化了，在未优化之前，只使

用一层卷积层和一层池化层，用于构建最基本的模型。

接下来是两层全连接层。在进行全连接操作之前首先是要将矩阵拉直成向量，因为全连接层的输入只能是向量，这里会用到拉直的函数，根据 TensorFlow 的官方推荐，我选用了 tf.keras.layers.Flatten()()函数，这里的 keras 原本是另一个较为流行的深度学习框架，不过 TensorFlow 已经开始将一些 API 替换成 keras 的 API。使用 keras 的 API 进行操作可以大大简化模型的搭建过程，但本项目是基于 TensorFlow 实现的，所以在非必要情况下不会使用 keras API。在全连接层中，需要将正则化损失加入到总损失中，来减少模型的过拟合现象。

最后是 Softmax 层，将全连接层输出的结果映射到 label 数量上，在 Softmax 层中本应进行归一化操作来输出概率值，方便计算交叉熵，不过这里并未进行归一化，因为 TensorFlow 提供的 tf.nn.sparse_softmax_cross_entropy_with_logits()函数可以对未归一化的输入进行计算交叉熵操作，所以只需在损失计算时使用该函数就能免去 Softmax 层中的归一化操作。

在最后的输出中，数值最高的 label 就是概率最高的标签，也就是需要的结果。

3.3.2 神经网络的优化

神经网络模型搭建完成后，效率可能不怎么高，此时需要根据输出的结果对结构进行调优，本节将主要讲述神经网络结构的优化和搭建过程中问题的解决。

首先是卷积核的选择，一般来讲，卷积核的选择不宜过大，因为卷积核的大小直接决定了该层参数的大小，本次数据集使用的图片大小是 32×32，经过剪裁后是 27×27，所以本次实验选择了 5×5 大小的卷积核，在减少权重参数的情况下尽可能减少特征的损失。对于卷积核个数，第一层卷积层使用了 64 个卷积核，卷积步长长宽皆为 1。

接下来是池化层的优化，将池化核设置为 3×3，步长长宽都为 2。

然后是卷积层和池化层的层数，在使用默认的一层构建模型时，发现最后的损失值下降缓慢，而且难以收敛，首先猜测与卷积层的层数不够有关，于是，又增加了一个卷积层 conv2，输入的深度为上一层输出的深度 64，输出深度也为 64。

之后再次进行测试，发现损失值的下降速度明显加快，说明增加卷积层层数能够在一定程度上提高模型准确度，当然也有可能是因为权重参数的增加提高了模型的稳定度。但是层数的增多代表权重参数的爆涨，会影响神经网络的训练性能，所以平衡两者之间的关系就是优化的目的。

然后是激活函数的选择，在第二章介绍了激活函数的作用，对于不同的模型使用不同的激活函数也会对最终的结果造成影响，对于图像分类，TensorFlow 官方推荐的激活函数是 Relu 函数，通过实践，Relu 函数确实能高效地解决过拟合问题。

3.3.3 神经网络的训练

构建完整个神经网络，就要进行训练过程。首先是计算损失函数，本实验中使用 loss()函数来进行计算损失操作。首先要计算的是交叉熵，TensorFlow 提供了计算交叉熵的函数 tf.nn.sparse_softmax_cross_entropy_with_logits()，通过导入参数 labels 和 logits 就可以输出交叉熵损失，此时的交叉熵指的是结果向量中的每一位与标签的差距，需要通过求和平均得到总损失。

除了交叉熵损失，还要加入正则化损失，也就是对权重参数的惩罚，该损失并没有在 loss()中实现，因为计算正则化损失的操作在权重参数生成时就进行过了，在权重参数生成时，使用 tf.nn.l2_loss()直接对将计算损失的操作加入到 losses 集合中，当需要计算损失时会自动调用实现损失计算。最后计算 losses 合集的总和，就是总损失。

之后是学习率的定义，在第二章中介绍了学习率的设置对于训练速度的影响，所以合理设置学习率对模型的执行效率的提高有很大作用。本实验使用了指数衰减法来更新学习率，通过使用 TensorFlow 提供的 tf.train.exponential_decay()函数，输入对应的参数，就能学习率随着训练轮数的增加自动衰减。

然后是影子变量，需要为损失值和所有变量创建影子变量，这是为了使得模型更为健壮，在后续的验证中，将直接使用影子变量而不是最后的权重参数。使用 TensorFlow 提供的 tf.train.ExponentialMovingAverage()函数可以对指定的变量

创建一个影子变量，但需要手动使用 apply()函数来更新。

最后是使用梯度下降来最小化损失值，本实验使用的是 GradientDescentOptimizer()优化器来实现梯度下降，参数为学习率，使用 compute_gradients()方法来计算梯度，使用 apply_gradients()方法来应用梯度。通过不断进行梯度下降，也就是改变权重参数的值，loss 的值也会不断收敛，这就是整个训练过程。

3.3.4 神经网络的可视化

使用程序构建的神经网络是以代码的形式存在的，比较抽象，对于不同参数的变化情况、整个神经网络的结构难以直观地显现出来，这时候就需要将神经网络进行可视化，本实验使用 TensorBoard 来可视化整个神经网络中重要的参数以及结构。

TensorFlow 提供了 tf.summary.histogram()来为一个 tensor 构造一个直方图，所以在编程过程中，需要将生成的重要变量都使用该函数生成一个直方图，如 conv1、losses 等。tf.summary.image()可以将一张图片加入可视化列表中，所以每导入一张图片都要调用这个函数加入可视化列表。tf.summary.scalar()函数可以显示标量信息，通过该函数可以绘制变量的变化曲线，一般用于 loss 值和权重参数。

训练过程中可以实时监测相关变量的值和变化情况，在训练结束后也能查看整个训练流程。通过 tensorboard --logdir=<checkpoint 路径>控制台命令能够打开 TensorBoard，在浏览器中输入 http://localhost:6006 就能查看图像化界面了。

3.4 训练结果验证与分类系统实现

3.4.1 训练结果正确率验证

实现了图片的验证的模块为 cifar100_eval.py，主要功能是将划分出的验证集导入神经网络计算正确率。

对于正确率的验证，就是对于神经网络输出结果与正确的标签进行对比，对于训练集，得到的是损失值；而对于验证集得到的就是正确率。正确率验证是判

断整个神经网络模型的准确度和是否产生过拟合现象的。一般来讲正确率越高，神经网络模型就越准确，而且没有产生过拟合现象。

eval_once()函数实现了验证功能，也就是执行验证流程的函数，首先是导入模型。在该函数中使用了 TensorFlow 提供的 tf.train.get_checkpoint_state()函数导入模型，不过使用这个函数需要配置相关参数，也就是 config 对象。如果使用默认参数，将会占用所有内存，如果内存不足将会报错，所以需要用 config.gpu_options.allow_growth = True 指定其按需申请显存，使用 config.gpu_options.per_process_gpu_memory_fraction = 0.5 指定其内存占用不超过 50%，这样就能避免因内存不做而报错的情况。

加载模型成功后需要将验证集图片通过模型，得到结果也就是 logits，这部分由 evaluate()函数实现，使用首先调用之前写完的 inputs 模块导入验证集，通过 inference()函数计算 logits，logits 是一个 tensor，输出结果为所以图片的结果标签，是一个多维数组，使用 tf.nn.in_top_k(logits, labels, 1)验证 logits 元素中前 1 个最大值是否包含 labels 的正确结果，简单来说就是结果标签是否是正确值。使用 variable_averages.variables_to_restore()导入滑动平均变量，之前提到过滑动平均变量会使得整个神经网络更加健壮，所以导入模型的参数时也应该导入滑动平均变量而不是训练完成后的变量。

接下来回到 eval_once()函数，此时已经导入模型并获得了图片的正确值，将统计完成后正确的图片数除以总样本图片数，即可得到正确率。由于神经网络的不确定性，每次验证的结果可能不尽相同，所以本程序会执行多次统计正确率的操作。

3.4.2 单张图片标签获取

对于单张图片的标签获取，定义了 cifar100_eval_single()模块来实现。具体功能是对于一张输入的图片，确定其对应的标签并输出。大部分操作与训练操作结果正确性验证相似，接下来只介绍增加的部分。

首先定义文件保存的路径，可以通过程序运行时指定参数实现，为了演示，

将会定义默认值，使用 tf.app.flags 定义变量名称 test_file，这是测试用图片放置的位置，可以使用 jpg 格式文件。由于正确性验证使用的二进制文件，而单张图片标签获取使用的是 jpg 格式图片，所以图片读取与预处理函数并不相同，这里定义了 img_read() 函数来实现图片的读取与预处理，首先使用 tf.image.convert_image_dtype(tf.image.decode_jpeg(tf.read_file(filename),channels=3), dtype=tf.float32)来读取、解码图片，并转换成 tf.float32 格式，这是训练集使用的数据类型，接下来使用 tf.image.resize_images() 来重定义大小，使用 tf.image.per_image_standardization()来标准化图像并使用 tf.reshape(image, (1, 27, 27, 3))将图像转换成训练集数据格式，只有这样才能被神经网络接受。

接下来的操作和上一节大致相同，取出各情况的概率值后与正确情况比对，得到最大概率值，使用 PrettyTable 模块将概率最大的 4 种情况打印在屏幕上。

3.4.3 图片批量分类系统

图片分类模块是真正的综合性模块，实现了分类程序的具体应用，具体功能是将文件夹下零散的图片归类到对应标签的文件夹内。流程图如下所示：

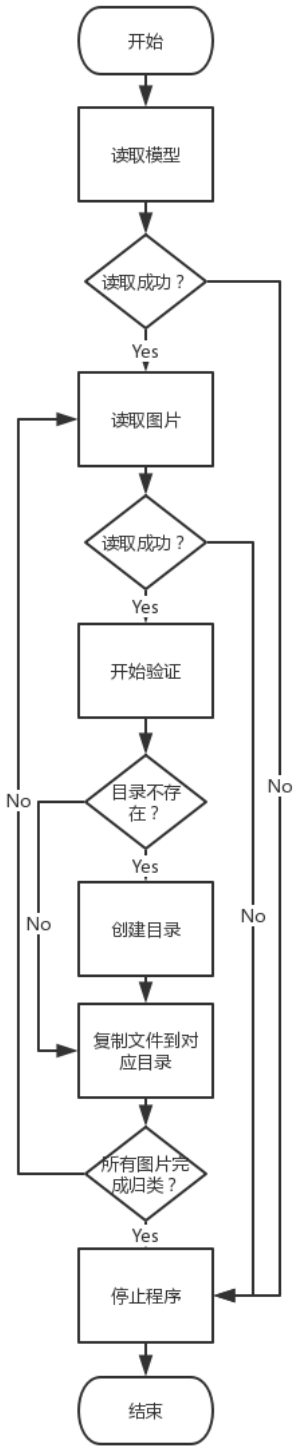

图 3-2 图片批量分类系统流程图

首先，导入图片与预处理的过程与上一节基本相同，增加的部分是通过定义的 img_reader()函数将文件夹下所用的图片做成了一个集合，再将集合转化成 Dataset，这是一种 TensorFlow 的数据集格式，也就是说现在的情况已经和使用 cifar100 验证集的情况相似了，使用 make_one_shot_iterator()创建一个迭代器，随后的操作都使用迭代器单个操作。

处理完图片后需要进行图片的标签获取，这里使用和上一节相同的操作，只不过由于迭代器的加入，整个标签获取函数会自动从管道中读取下一张图片进行标签获取操作，直到集合中的所有图片都完成操作。

标签获取完毕后需要进行归类操作，就是将根据标签将图片放入对应文件夹中，这里定义了 mkdir_and_copy(pathname, filename)函数，参数为目标文件夹和源文件名。首先检测目标目录是否存在，如存在就需要创建文件夹，使用 python 的好处是有现成的模块可以使用，这里使用 os.makedirs(path)就可以直接创建目标目录了。随后使用 shutil.copyfile(os.path.join(FLAGS.test_dir, filename), os.path.join(path, filename))将相关目录拼接后进行复制操作。

运行程序后，图片会依照文件名顺序依次归类到对应文件夹内。

3.5 在 Orange Pi 3 上实现图片分类

将程序移植到其他设备上也是本次实验的目的之一，本次实验选择了将程序移植到 CPU 架构为 ARM64 且搭载 Linux 系统的设备上，通过移植来测试本程序的可移植性和在不同架构 CPU 下的运行效率。

Orange Pi 3 是由迅龙公司推出的一款基于全志 H6 芯片的 ARM 开发板，其搭载了 1GB 的 LPDDR3 内存、千兆以太网、WIFI、4 个 USB3.0 接口，无论是在开发还是使用方面都有不错的表现。

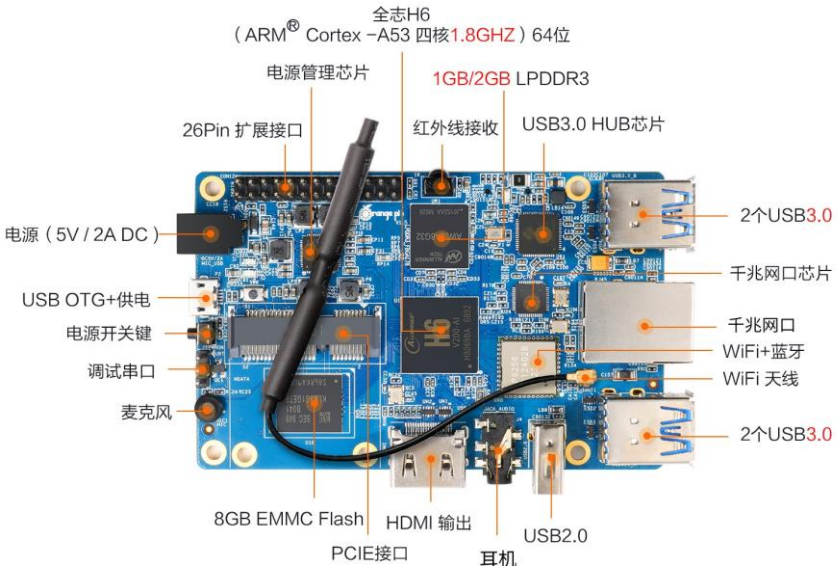

图 3-3 Orange Pi 3 硬件参数图

迅龙官方提供了内核版本为 4.9 的 ubuntu xenial 系统，但是很久没有更新了，所以本次实验使用了 Armbian 系统，这是一个由来自全球各地的开发板爱好者共同维护的专为 ARM 开发板重新编译的 Debian 系列系统。安装完系统后进行相关配置，就可以连接上互联网以及连接 SSH。为了方便起见，整个工作将由 PC 机使用 SSH 连接开发板进行开发。

图 3-4 实验用 Orange Pi 3 运行图

首先是安装 TensorFlow，由于官方并未提供 aarch64 版本的 TensorFlow，所以需要重新编译，好在有开发者已经针对 arm 芯片编译好了最新版 TensorFlow，在 https://github.com/lhelontra/tensorflow-on-arm/releases 下载 aarch64 版本就能直接使

用 pip 管理器安装了。

接下来是代码修改，尽管 Python 是跨平台的语言，但是 Linux 和 Windows 在很多方面都或多或少有一定区别，比如路径格式等。由于在一开始的开发中就考虑到了跨平台问题，所以只需修改路径参数就能实现移植操作。将所有 Windows 格式的路径修改为 Linux 的路径格式，如将'C:\\cifar100 '修改为'/var/cifar100 '，同时还要将 checkpoint 文件内的路径修改为 Linux 系统下模型的保存目录。接下来将 cifar100 目录移动通过 xftp 管理器移动到 Linux 的对应目录下。使用 program 目录下的 Python 程序就能运行了。

第四章 实验结果与分析

4.1 训练结果损失与分析

训练中的损失可以通过控制台打印的方式展示，也可以通过 TensorBoard 以图形化的方式显现出来，下图表示训练过程中损失的变化情况：

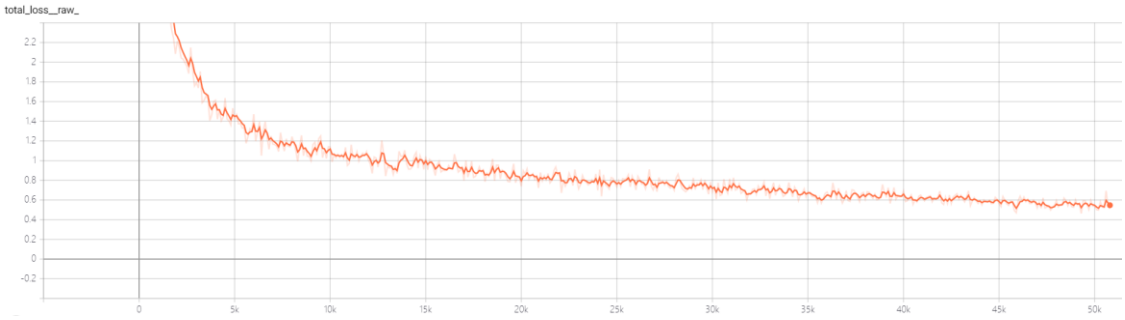

图 4-1 loss 值变化图

从图中可以看出损失值随着训练步数的增加越来越小，前期下降得较快，后期下降得较慢，最后 10000 步几乎没有下降。影响训练损失下降最重要的因素是学习率，越大的学习率意味着更快的训练速度，但是学习率过大会造成损失的无法收敛，所以在训练过程中使用了动态学习率，后期损失的下降速度慢一方面已经接近损失最低点，另一方面是因为学习率的动态下降导致训练速度的放缓。

影响学习率最低点的是隐藏层中参数的总量和结构，当总量越多，整个神经网络就越稳定，越能接近生物的神经网络构造，同样优秀的结构也是必要的。

当损失的下降几乎不产生变化时，后续的训练意义不大，这时候应该考虑如何通过神经网络的优化来降低损失的极限最小值。所以训练过程在 50000 步时停下，此时模型的准确度已经很高了。

4.2 验证集正确率与分析

运行 cifar100_eval.py 的结果展示：

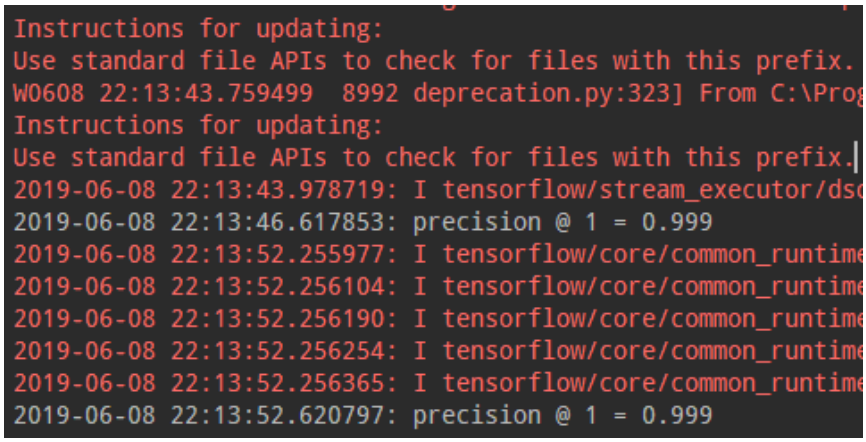

图 4-2 验证集正确率验证运行结果

此处的 0.999 就表示正确率，经过不断优化神经网络后，最终在训练到 50000 步时达到了 99.9%的验证集正确率。该结果说明神经网络模型的准确度非常高，而且也没有出现过拟合现象。

表 4-1 不同训练阶段时的损失值和验证正确率

序号	训练步数	损失值	正确率
1	10000	1.1380	0.900
2	20000	0.7396	0.976
3	30000	0.6824	0.990
4	40000	0.6900	0.993
5	50000	0.5394	0.999

表 4-1 表示不同训练阶段时的损失值和验证正确率，通过对比不同训练阶段时的损失和正确率情况，可以看出随着训练步数的增加损失值呈下降状态，而正确率呈上升状态，30000 步的损失值大于 40000 步是因为正常的波动，整体依然呈下降趋势。同时正确率和损失值呈反比关系，也表明了没有产生过拟合现象。

4.3 单张图片标签获取结果与分析

对于单张图片的标签获取，首先输出的是一张处理后的图片，这是一张自行车在进行预处理后的样子：

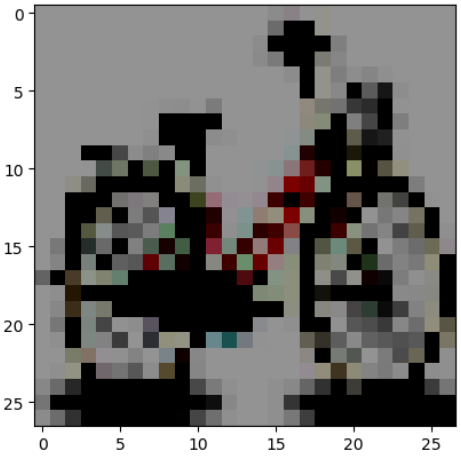

图 4-3 图像预处理后的输出

这样的图片就是输入到神经网络的图片，通过图像处理，图片的特征很好地展示了出来，而且图片的大小也统一了，这对后续的各种操作带来了好处。最终的结果如下：

```
load success
2019-06-09 19:15:39.282052: I tensorflow/stream_executor/dso_loade
+-------+------------------------------+-------------+
| index |            class             | probability |
+-------+------------------------------+-------------+
| 18    |          vehicles_1          |   24.19905  |
| 15    |           reptiles           |   8.303555  |
| 3     |       food_containers        |  4.5713615  |
| 5     | household_electrical_devices |   3.786367  |
+-------+------------------------------+-------------+

进程已结束，退出代码 0
```

图 4-4 单张图片标签获取结果

结果显示这张图片标签的最大可能性是 vehicles_1 也就是中小型车辆标签，表示识别正确。由于训练用数据集是 32×32 低分辨率图片，所以在使用高分辨率图片进行验证时可能效果并没有特别好，最终的概率也仅仅 24.19，这也是需要提高的地方。

4.4 图片批量分类结果与分析

为了进行图片批量分类操作，本次实验查找了搜索引擎根据标签随机查找了一些网络图片，这些图片来自不同的国家、不同的摄影师、不同的文化。

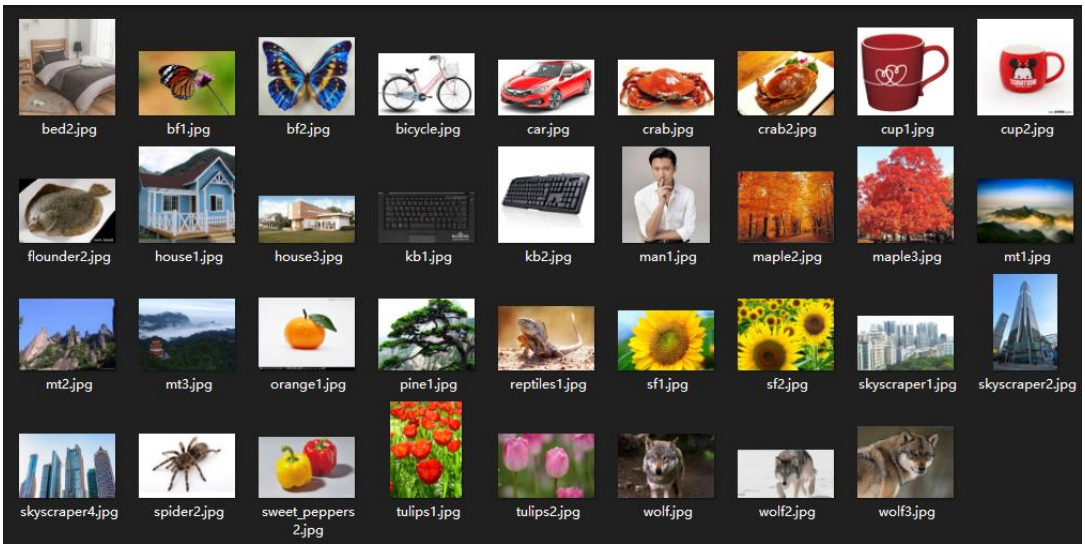

图 4-5 作为样本的图片

本次实验为每张图片预先加上了小类标签，以便整理和之后的结果分析。通过执行分类程序 cifar100_sort.py，将图片归类到对应 labels 的文件夹来测试。

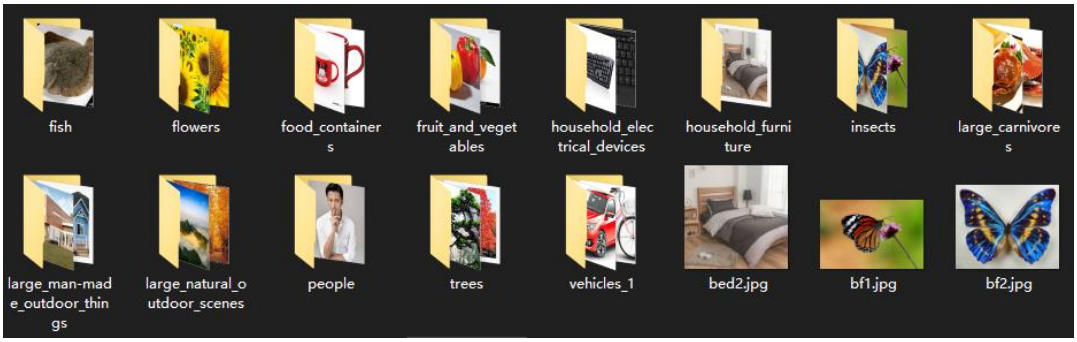

图 4-6 批量分类结果

程序执行完毕后，每种类别生成了一个文件夹，图片归类到了不同文件夹中，结果显示大部分图片分类正确，部分图片分类错误，其中 4 张图片完全错误，1 张图片部分错误，螃蟹和蜥蜴图片被归类到了大型食肉动物文件夹，高山被识别为摩天大楼，不完全错误为枫树本应为树，但枫树林被识别为了大型自然景观。

通过使用不同的样本进行测试，所得结果如下：

表 4-2 多次运行分类的结果

序号	样本数量	分类错误数量	正确率
1	35	4	0.886
2	40	5	0.875
3	32	5	0.844
4	45	6	0.867

序号	样本数量	分类错误数量	正确率
5	32	3	0.906

根据表 4-2 可得本程序在分类大型网络图片时效果并没有使用验证集那么好，但依然保持了一个合理的水平。根据分类结果分析可以得出本程序在花卉、昆虫、交通工具上的分类成功率最高，初步分析为这些图片的“对比度”较大、结构较为清晰、特征较为明显。

4.5 在 Orange Pi 3 上实现图片分类

在 Orange Pi 3 上运行了图片批量分类程序，结果如下图所示：

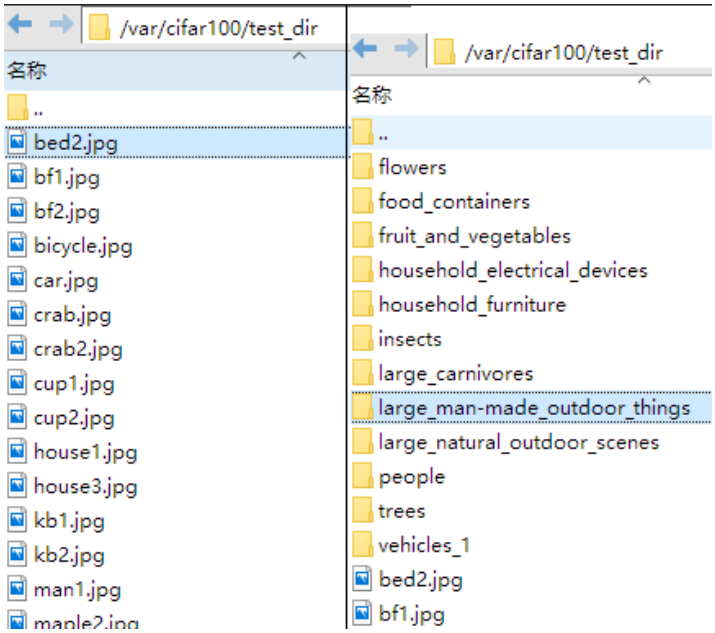

图 4-7 Orange Pi 3 上的批量分类结果

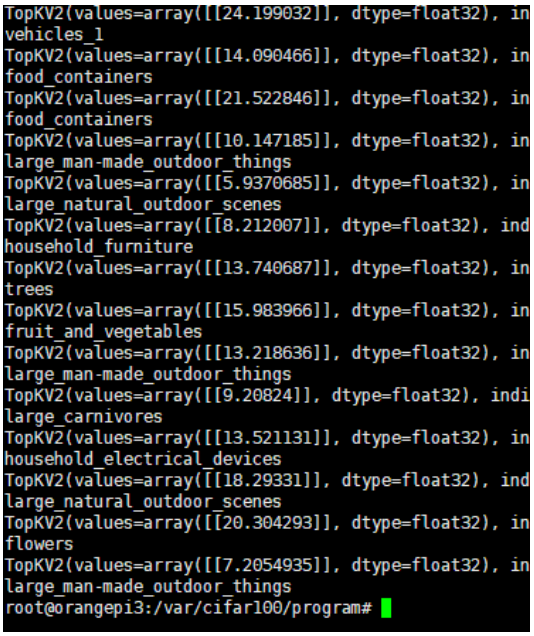

图 4-8 Orange Pi 3 上的程序运行截图

从图中可以看出本程序在 ARM 开发板上的执行结果和 PC 上的基本相同，尽管 Orange Pi 3 的 CPU 性能远低于 PC 机，但整个执行时间仅略微慢于 PC。该结果说明本程序的可移植性较高而且对性能的消耗较少。

第五章 总结

5.1 完成的工作

本论文主要完成的工作是使用 TensorFlow 框架设计并实现了一个通过卷积神经网络进进行图片分类的系统。具体工作如下：

第一，设计了一个卷积神经网络结构，在本论文中共设计了 5 层卷积神经网络，分别是输入层、卷积层、池化层、全连接层、Softmax 层。每一层都有独特的功能。

第二，使用 TensorFlow 提供的函数与模块搭建设计完的整个神经网络，通过查阅和学习官方文档以及查看相关博客的方式，理解各个函数与模块具体的用法与背后的原理，通过测试和实践来确定合适的用法。

第三，针对数据集和训练过程中遇到的问题，对整个神经网络进行参数调优，并使用合适的函数处理数据，实现了在不降低神经网络训练性能的情况下提高了训练性能。

第四，对于神经网络的输出结果进行分析，并实现一个通过分析结果分类图片的系统，用于展示本论文的具体成果。本系统实现自动通过结果对应的图片标签将一组图片分类到对应标签的文件夹内。

第五，将整个项目移植到了 Orange Pi 3 上，实现了跨平台应用。

5.2 存在的问题及下一步工作

通过本次实验，提高了自己的编程水平，拓宽了视野，对整个卷积神经网络框架和TensorFlow框架有了初步的认识。但由于时间仓促，还存在很多不足和需要改进的地方，主要有以下几方面：

（1） 本次使用的训练样本数量为50000，这对于大规模的、应用领域的环境来说是远远不够的，对此需要在之后的项目中使用规模更大的数据集，提高训

练模型的准确度和泛化能力。

（2） 本次只将模型移植到arm开发板上，还有很多平台没有实现，比如安卓、iOS等，人工智能技术已经越来越普及，未来各式各样的设备将都能运行人工智能应用，TensorFlow官方最近发布了TensorFlow-lite，旨在为所有低性能设备提供人工智能应用，所以之后的项目应该要通过TensorFlow-lite移植到更多其他设备中。

（3） 本次实验使用了CPU和GPU进行训练和验证，虽然性能足够，但毕竟不是专门进行机器学习的硬件，如今越来越多的硬件开始为人工智能做优化，比如Google公司最近推出的TPU就是专门为机器学习而生的，在Google云上有大量的TPU，而且使用价格并不高，所以下一步的目标就是将本项目部署到Google云的TPU服务器上。

第六章 参考文献

[1] 吉树军,聂章龙.计算机通信技术与电子信息在人工智能领域的实践应用[J].电子测试,2019(10):121-122+120.

[2] 曹誉栊.从 AlphaGO 战胜李世石窥探人工智能发展方向[J].电脑迷,2018(12):180.

[3] 加日拉·买买提热衣木,常富蓉,刘晨,要秀宏.主流深度学习框架对比[J].电子技术与软件工程,2018(07):74.

[4] 田泱. 基于深度学习的自动分类相册系统的设计与实现[D].中山大学,2015.

[5] 季思文,闫胜业,王蒙.基于双向神经网络的图像分类算法[J].计算机工程与设计,2018,39(10):3113-3117.

[6] 王建霞,王晓东,张行.基于神经网络的图像识别研究[J].计算机产品与流通,2017(10):46-47.

[7] 黄友文,万超伦.基于深度学习的人体行为识别算法[J].电子技术应用,2018,44(10):1-5+10.

[8] 丁鑫. 基于深度卷积网络特征优化的图像分类[D].西安电子科技大学,2018.

[9] 中央人民政府驻香港特别行政区联络办公室副主任 中国科学院院士 谭铁牛. 人工智能的发展趋势及对策[N]. 中华工商时报,2019-02-25(003).

[10] 侯宇青阳,全吉成,王宏伟.深度学习发展综述[J].舰船电子工程,2017,37(04):5-9+111.

[11] 周晟颐.深度学习技术综述[J].科技传播,2018,10(20):116-118.

[12] 程思雨,林锋.计算机围棋 AlphaGo 算法对人类围棋算法的影响[J].中国科技信息,2019(02):40-41.

[13] 屈艺多,胡琳.智能驾驶技术的现状与未来发展趋势[J/OL].电子技术与软件工程,2019(11):243[2019-06-10].http://kns.cnki.net/kcms/detail/10.1108.TP.20190603.1214.352.html.

[14] 孙永杰.百度:打造 AI 系统级开放平台 专注自动驾驶与智能设备[J].通信世界,2018(13):30.

[15] 韩松伯. 基于深度神经网络的英文文本蕴含识别研究[D].北京邮电大学,2018.

[16] Jiafu Li,Wenyan Tang,Jun Wang,Xiaolin Zhang. A multilevel color image thresholding scheme based on minimum cross entropy and alternating direction method of multipliers[J]. Optik,2019,183.

[17] Yaqiong Yao, HaiYing Wang. Optimal subsampling for softmax regression[J]. Statistical Papers,2019,235-249.

[18] Emmanuel Moulay,Vincent Léchappé,Franck Plestan. Properties of the sign gradient descent algorithms[J]. Information Sciences,2019,492.

[19] 周谧. 非均衡随机梯度下降 SVM 在线算法[D].河北大学,2017.

[20] A G Pertiwi,A P Wibawa,U Pujianto. Partus referral classification using backpropagation neural network[J]. Journal of Physics: Conference Series,2019,1193(1).

[21] Boling Guo,Binqiang Xie,Lan Zeng. Exponential decay of Bénard convection problem with surface tension[J]. Journal of Differential Equations,2019,267(4).

[22] M. Ignacia Vicuña,Wilfredo Palma,Ricardo Olea. Minimum distance estimation of locally stationary moving average processes[J]. Computational Statistics and Data Analysis,2019.

[23] 高震宇. 基于深度卷积神经网络的图像分类方法研究及应用[D].中国科学技术大学,2018.

[24] Lu Xiaofeng,Zhang Shengfei,Yi Shengwei. Continuous authentication by free-text keystroke based on CNN plus RNN[J]. Procedia Computer Science,2019,147.

[25] 司琴,李菲菲,陈虬.基于深度学习与特征融合的人脸识别算法[J/OL].电子科技,2020(04):1-6[2019-06-10].http://kns.cnki.net/kcms/detail/61.1291.TN.20190527.0848.006.html.

[26] 孟丹. 基于深度学习的图像分类方法研究[D].华东师范大学,2017.

[27] Mingliang Liu,Dario Grana. Accelerating geostatistical seismic inversion using TensorFlow: A heterogeneous distributed deep learning framework[J]. Computers and Geosciences,2019,124.

[28] Craig Warren,Antonios Giannopoulos,Alan Gray,Iraklis Giannakis,Alan Patterson,Laura Wetter,Andre Hamrah. A CUDA-based GPU engine for gprMax: Open source FDTD electromagnetic simulation software[J]. Computer Physics Communications,2019,237.

[29] Florian Eiler, Simon Graf, and Wolfgang Dorner. 2018. Artificial intelligence and the automatic classification of historical photographs[R]. In Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'18), Francisco José García-Peñalvo (Ed.). ACM, New York, NY, USA, 852-856. DOI: https://doi.org/10.1145/3284179.3284324

[30] 韩菲,李炜.CPU 与 GPU 的计算性能对比[J].电子技术与软件工程,2019(01):125-126.

第七章 致谢

在本科学习过程中，得到了老师和同学们的无私帮助，即使是遇到难题，也最终得到解决。在此，深表感激。

在毕业设计阶段，导师黄洪教授从选题开始，就一直陪伴我，指导我一步一步完成了整个毕业设计。在此，深表感谢。

最后还要感谢我的家人、朋友，以及所有给予过我帮助的人，感谢他们为我付出的一切。

第八章 附录

8.1.1 附件 1　毕业设计文献综述

8.1.2 附件 2　毕业设计开题报告

8.1.3 附件 3　毕业设计外文翻译（中文译文与外文原文）