MASTER THESIS XIDIAN UNIVERSITY

西安電子科技大學

硕士学位论文

基于神经网络的语音关键词识别嵌入式系统实现

作者姓名 蒲洋廷

学校导师姓名、职称 相 征 教授

企业导师姓名、职称 王卫斌 高工

申请学位类别 工程硕士

学校代码 10701　　学　号 18011210481
分 类 号 TN91　　密　级 公开

西安电子科技大学

硕士学位论文

基于神经网络的语音关键词识别嵌入式系统实现

作者姓名：蒲洋廷

领　　域：电子与通信工程

学位类别：工程硕士

学校导师姓名、职称：相征 教授

企业导师姓名、职称：王卫斌 高工

学　　院：通信工程学院

提交日期：2021 年 3 月

Implementation of Embedded System for Speech Keyword Spotting Based on Neural Network

A thesis submitted to

XIDIAN UNIVERSITY

in partial fulfillment of the requirements

for the degree of Master

in Engineering

By

Pu Yangting

Supervisor: Xiang Zheng Title: Professor

Supervisor: Wang Weibin Title: Senior Engineer

March 2021

摘要

语音交互操作是用户与机器之间沟通的重要方式。随着家庭物联网以及移动设备等低功耗物联网机器的广泛应用，如何利用语音关键词识别技术（Keyword Spotting，KWS），从而保障人机交互可靠性成为了研究热点。近年来，随着深度学习的不断发展，基于卷积神经网络（Convolutional Neural Network，CNN）的相关技术在 KWS 领域显现了其优异性能。得益于 CNN 对局部特征的有效提取及良好的空间不变性，语音关键词可以被高效、准确识别。但由于 CNN 网络的参数数量众多且卷积操作计算量巨大，对在内存及计算资源有限的嵌入式设备上的应用带来挑战。

针对卷积神经网络在嵌入式设备上部署带来的计算资源及存储资源过高的问题，本文在分析 CNN 的参数储存结构与卷积计算模式的基础上，研究了针对卷积神经网络模型的压缩和加速算法。网络模型压缩算法方面，使用网络量化技术，根据网络参数对数据位宽的敏感程度，使用不同数据位宽的量化策略。此外，针对量化带来的模型精度下降问题，使用渐进式的量化策略来优化量化过程，通过多次重训练弥补了量化过程带来的模型精度损失。网络模型加速算法方面，使用网络剪枝技术，改进了剪枝中的通道选择策略，综合考虑前后卷积核的基础上，采用最小化特征误差指标的方法，并使用贪婪算法求解卷积核的冗余通道。最后在低成本、低功耗嵌入式模组 ESP32 上部署了应用本文算法压缩加速后的关键词识别系统，并进行了实机测试。

实验结果表明，渐进式量化策略将量化带来的精度下降控制在 5%以内，且所提算法的模型精度与同类算法相比提高了约 10%。网络模型加速方面，所提算法比传统经典滤波器网络减枝算法提升约 20%。通过对网络的压缩与加速，同结构 CNN 网络在内存体积上压缩约 54 倍，在计算量上压缩约 3.3 倍，且整体模型精度损失不超过 10%。在实际测试环节中，语音关键词识别系统可在平均 400ms 的时间内进行语音关键词识别操作。

关 键 词：关键词识别， 嵌入式， 卷积神经网络， 网络量化， 网络剪枝

ABSTRACT

Voice interaction is an important way of communication between the human user and the machine. With the wide application of home IoT and mobile devices and other low-power IoT machines, how to use keyword spotting (KWS) to ensure the reliability of human-computer interaction has become a research hotspot. In recent years, with the continuous development of deep learning, the relevant technology based on Convolutional Neural Network (CNN) has shown its excellent performance in the field of KWS. Thanks to CNN's effective extraction of local features and good spatial invariance, speech keywords can be recognized efficiently and accurately. However, due to the large number of parameters of CNN network and the huge computational amount of convolution operation, the application of CNN network in embedded devices with limited memory and computing resources is challenged.

In order to solve the problem of excessive computing and storage resources caused by the deployment of convolutional neural network on embedded devices, this thesis studies the compression and acceleration algorithms for the convolutional neural network model based on the analysis of CNN's parameter storage structure and the convolutional computing model. In the aspect of network model compression algorithm, the neural network quantization technology is used. According to the sensitivity of network parameters to the data bit width, the quantization strategy of different data bit width is used. In addition, a progressive quantization strategy is used to optimize the quantization process and compensate for the loss of model precision caused by the quantization process through repeated retraining. In the aspect of network model acceleration algorithm, the network pruning technology is used to improve the channel selection strategy in pruning. Based on comprehensive consideration of the convolution kernel before and after the convolution kernel, the method of minimizing the characteristic error index is adopted, and the greedy algorithm is used to solve the redundant channels of the convolution kernel. Finally, a keyword spotting system based on compression and acceleration of the proposed algorithm is deployed on ESP32, an embedded module with low cost and low power consumption, and an experiment on the real machine is carried out.

Experimental results show that the progressive quantization strategy can control the

reduction of precision within 5%, and the model precision of the proposed algorithm is improved by about 10% compared with similar algorithms. In the aspect of network model acceleration, the proposed algorithm is about 20% better than the traditional pruning algorithm of classical filter network. Through the network compression and acceleration, the CNN network with the same structure can compress about 54 times the memory volume and 3.3 times the computation amount, and the overall model accuracy loss is not more than 10%. In the experiment on ESP32, the speech keyword spotting system can perform speech keyword recognition operation in an average of 400ms.

Keywords: Keyword recognition, Embedded system, CNN, Network quantification, Network pruning

插图索引

表格索引

符号对照表

符号	符号名称
$\odot$	卷积层执行的卷积操作
$\oplus$	按位异或操作
$\otimes$	按位相乘操作
α	学习率
C	量化比例
$floor(\cdot)$	向下取整
r	剪枝率

缩略语对照表

缩略语	英文全称	中文对照
ASR	Automatic Speech Recognition	自动语音识别
KWS	Keyword Spotting	关键词识别
HMM	Hidden Markov Model	隐马尔科夫模型
GMM	Gaussian Mixture Model	高斯混合模型
DNN	Deep Neural Networks	深度神经网络
LSTM	Long Short-Term Memory	长短期记忆网络
CNN	Convolutional Neural Network	卷积神经网络
CONV	Convolutional Layer	卷积层
BN	Batch Normalization	批标准化
FC	Full Connected Layer	全连接层
MFCC	Mel-Frequency Cepstral Coefficients	梅尔频率倒谱系数
TC	Toolchain	工具链

目录

第一章　绪论

1.1 研究背景与意义

随着计算机技术的普及，该技术在现代生活中极大地增加了人类的生产力，让现代社会的学习、工作效率远高于从前。最开始的计算机操作形式是以命令行代码的形式进行交互的，程序员通过将想让计算机执行的操作，以命令行或者代码的形式输入至电脑终端让计算机执行。早期计算机虽然可以增加效率，但是终究只是专业人员的辅助工具，普罗大众很难使用。自从图形化技术出现之后，计算机才正式走入人类生活，越来越多的行业开始使用计算机代替从前落后的设备，人类才真正走进了信息时代。图形化界面的人机交互是通过鼠标点击加键盘输入的形式进行的，但如果人类和计算机可以像人与人之间通过自然语言直接沟通的话，人类社会的效率将会有更大的突破。目前，相关技术仍在演进中。

随着语音关键词识别技术的出现，计算机与人类的交流过程中就有了“耳朵”，用户可以通过该技术“唤醒”计算机。语音关键词识别技术是机器可以将设定的一个或多个语音关键词在不间断的自然语音流中识别出来的一种技术，设备唤醒、语音检索、人机交互、语音监听等领域广泛应用了该技术。

语音关键词识别技术作为语音识别技术的一个重要研究分支，在 20 世纪 90 年代逐渐被重视起来。近些年来，该技术也被实际应用在了国内外众多厂商的产品之上，比如苹果公司在该公司的手机产品上使用的 Siri 语音交互系统，便使用“Hey，Siri”进行语音关键词识别，改进其语音交互体验，相似的 Google 也是用“OK，Google”短语进行语音交互唤醒，国内的互联网厂商比如小米也有类似的“小爱同学”的交互系统。

语音关键词识别系统广泛存在于人们的生活里，比如在智能汽车驾驶领域，在车辆行驶过程中，如果出现需要回复信息或者联系外界的情况，驾驶员将注意力转移至手机上是非常危险的行为，此时高精度的语音交互系统就可以帮助驾驶员在视线不偏离驾驶视野的情况下进行操作，大大降低了因为外界打扰而给驾驶过程带来的危险性。同时，语音关键词识别技术可以帮助驾驶员在不直接触碰通讯设备的情况下，唤起语音交互系统，这对于驾驶安全是非常重要的。比如蔚来汽车中内置的语音助手 NOMI，可以使用“NOMI”关键词进行唤醒，不仅可以进行拨打与收听电话，还可以通过语音交互控制车内诸如空调、座椅等组件，实现驾驶员双手不离开方向盘即可控制车内的其余部件。同时在智能家居领域，智能音箱作为众多的物联网设备的操作入口产品，同样使用语音关键词识别技术来唤醒交互系统，使得用户对其的操作范围可以扩展至

更大的范围，极大地扩展了产品使用场景。

在该技术早期，人们使用高斯混合模型等方法来实现关键词识别，但技术效果并不理想。之后，采用了隐马尔科夫模型方法大大提高了语音关键词识别的准确性。但是该方法因为算法复杂度问题并未在实际生产环境中有过多的应用。近些年来，深度学习计算在各个领域大放异彩，各种各样的深度神经网络结构展现了该技术在各种领域里的潜力。与古早的建模技术相比其建模方法相比相对简单，所以当前的关键词识别系统大多采用深度学习模型进行实现。

随着物联网行业的迅速崛起，关键词识别系统的硬件载体的运算能力在逐渐变弱，以往的深度学习模型不再适合在低功耗的物联网芯片上运行。这类物联网芯片普遍具有低功耗、低计算量、长待机的特点，所以以往的深度学习模型需要对目标芯片进行内存体积和计算量上的改进才能匹配物联网芯片的运算能力。

本文使用的嵌入式芯片乐鑫 ESP32 被广泛应用在各类的物联网设备上，其作为一款优秀的 Wi-Fi 芯片被诸如小米、百度、阿里、亚马逊等众多国内外厂商应用在各类物联网设备上。目前使用到语音交互的物联网设备大多采用独立语音芯片的方式实现语音关键词唤醒，而如果可以在被广泛使用的 Wi-Fi 芯片 ESP32 上实现语音关键词唤醒，不仅可以进一步的压缩成本，还可以在设备内部集成度上进一步精简，可以有效地提升设备的市场竞争力。

1.2 国内外研究现状

本论文主要聚焦于，将用于语音关键词识别的深度神经网络模型部署到低计算资源的嵌入式设备上，而现有的深度神经网络模型普遍具有参数量冗余、计算量庞大的特点，所以要实现本文的目标，必须对全精度的网络模型进行压缩和加速，使其模型的内存占用和推理计算所需要的计算资源与嵌入式设备相匹配，所以在本小节，首先介绍了关键词识别技术在国内外的发展与研究现状，之后介绍对于深度神经网络的压缩、加速方法的发展及现状。

1.2.1 关键词识别技术的研究现状

当前自动语音识别（Automatic Speech Recognition，ASR）[1]技术已在人们的日常生活中得到广泛应用。关键字检测（Keyword Detection，KWD），也被称为关键词识别（Keyword Spotting，KWS）或口语检测（Spoken Term Detection，STD），是自动语音识别技术的重要分支，并且关键词检测也是连续自动语音识别中的特殊分支[2]。关键字检测的过程是检测相对少量的被指定为关键字的单词。关键字检测提供了一种有效的手段，在一段连续的音频流中找到特定的关键词[3]。目前关键词识别技术已经成

为各种设备上不可或缺的一项功能。

经过长期的研究，目前语音关键词识别领域中已经存在多种解决方案。其中主流的关键词检测技术方案有（1）基于样例的关键词检测技术[4]（Query-by-Example，QBE）；（2）基于隐马尔可夫模型的关键词检测技术[5]（Hidden Markov Model，HMM）；（3）大词汇量连续语音识别技术[6]（LVCSR）；（4）基于深度学习技术的关键词识别技术[7]。

其中，QBE 技术是最早被提出的 KWS 技术，该技术中将关键词当作基本样本（Example），将接收到的语音与样本进行对比比较，进而判断是否触发关键词唤醒操作，是一种典型的基于模板匹配的方法。该方法首先将关键词进行建模，以特征向量或其他方式表示，之后将待检测语音也进行建模，并比较二者差异从而实现关键词检测的目的。常用的对比算法采用动态时间规整算法[8]（dynamic time warping，DTW），其采用了动态规划的思想。基于隐马尔可夫模型的 KWS 技术，使用 HMM 模型对关键词进行建模，在语音关键词领域一直有着相当重要的地位。在该语音关键词识别方案中，使用 HMM 模型的关键是生成两个关键词识别部分，分别是关键词分量（keyword）和非关键词（filler）分量，二者分别对语音的不同部位进行建模，前者在词级及音素级等上进行建模，后者则是在关键词外的其余部分，如噪音进行建模。之后在实际系统中，使用动态规划算法来解码模型。因为系统被分成未两个结构不同的分量，要进行关键词识别需要花费大量的时间[8-10]。LVCSR 技术是将一段连续的语音转化为文字，在关键词识别问题中即是将接收到的语音转化为一个个的单词或者字，之后对其进行自然语言处理，进行关键词检索工作[11]。该方法可以一劳永逸的解决关键词识别问题，但是其缺点也较为明显，一是该技术会面临计算量过大的问题，且在前期训练阶段要求充足的训练样本，在后续语音转化过程中也需要极大的系统资源；二是该技术会存在脱离字符集问题[12]（out-of-vocabulary，OOV），即如果关键词不在词典中，将无法进行识别。

目前最为流行的 KWS 技术则是基于深度学习技术的端到端关键词唤醒技术[13-15]。深度学习网络（Deep Neural Network，DNN）的网络结构是由一层一层的神经元相互连接组成的，其中在网络最前与最后的神经元被称为输入层和输出层，在中间不与输入输出数据直接接触的神经元被称为隐层。深度神经网络与传统人工神经网络（Artificial Neural Network，ANN）相比，DNN 的隐藏层更多，这使得 DNN 的搜索空间相较于 ANN 被极大的扩展，提高了训练后模型的表征能力，具有更好的泛化表现。图 1.1 是单层神经网络结构。

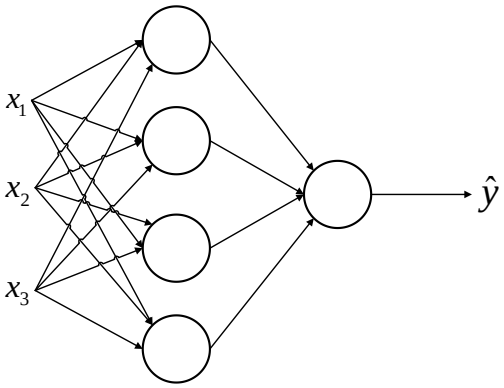

图1.1 单层神经网络

因为 DNN 的良好的性能，在关键词识别领域，越来越多的 DNN 结构被用来解决语音关键词识别问题。其中长短期记忆网络（Long Short-Term Memory，LSTM）是一种 DNN 的变体结构，该结构是一种时间循环结构，可以通过先前的网络特征来修正后续的网络表现，这使得该结构在关键词识别问题上，展现出了优异的效果，但是也正是因为其时间循环特质，导致其系统响应时间长，实时性低[16]。卷积神经网络（Convolutional Neural Network，CNN）是一种将卷积操作引入隐藏层的 DNN 结构，通过卷积层的特征提取过程，使得该结构在对抗语音噪声方面表现优异，在关键词识别问题中也有着良好的应用。但是正因其结构特性所致，CNN 网络普遍存在模型巨大，计算复杂度高等问题[17-18]。

本小节通过对关键词识别技术的研究现状的对比，发现在主流的四种关键词识别技术中，深度神经网络技术在该问题上性能优异，并且在资源需求上有较大的改进空间，并且其计算模式在嵌入式设备上较为实现，其浮点运算在一些嵌入式平台上甚至有专门的浮点运算芯片进行辅助运算，所以在本文嵌入式设备实现的需求下，本文选择使用深度神经技术来进行关键词识别。

1.2.2 深度神经网络模型压缩技术的发展现状

深度神经网络自出现以来，在图像识别、目标检测、语音识别等等众多可以改善人类社会的领域里大放异彩，不断突破这些领域中固有方法的天花板，展现了极佳的性能[19-25]。但是因为 DNN 模型要到达这样优异的性能所依赖的结构复杂，在完成训练之后，模型的内存需求和计算需求也是巨大的[26-28]。

深度神经网络压缩技术在智能物联网领域作用巨大。因为智能物联网领域的终端设备普遍计算性能不强，储存空间较少，能承受的计算复杂度较低，所以要让单个物联网设备执行深度学习模型任务是很困难的。与之相对的，物联网终端设备更依赖于边缘计算的计算形式，在多个弱计算平台上，共同解决一个复杂问题。要想在这些边缘计算终端设备上部署深度学习网络模型，势必就需要将网络模型修改、优化至可以

被部署到该平台的状态。

所以深度神经网络压缩加速技术可以使得网络模型在经历压缩优化之后，仍然可以保持 DNN 的高性能，同时还可以使得模型的空间复杂度和计算时间复杂度大大下降。目前对于 DNN 模型的压缩加速算法，可以分为主要的五类。其一是试图将 DNN 网络中的权重参数和特征图分解为更低维数据的低秩分解方法。其二是将网络中的参数的储存数据位宽降低的参数量化方法。其三是试图找到在不增加更多的网络隐层的情况下保持模型精度的致密结构方法。其四是通过更高维的模型训练低维模型的知识蒸馏方法。其五是将训练完成的网络模型裁剪之后，恢复原本精度的模型裁剪方法[29-30]。上述方法各有优劣，并在各种领域均有应用。

1.3 论文的研究思路和结构安排

本论文的研究思路是结合卷积神经网络的卷积层与全连接层的参数储存特性与计算特性，分析适用于嵌入式设备的模型优化策略，通过对全精度神经网络模型进行模型压缩和加速，在嵌入式设备 ESP32 上实现了关键词识别系统。

全文总共分为五章，各章节的安排内容如下。

第一章：绪论。首先，介绍了关键词识别技术诞生的背景，以及该项技术的意义，并阐述了将关键词识别技术应用于嵌入式设备的重要性；其次，介绍了目前国内外对关键词识别技术发展的趋势及研究现状，并深入介绍了将深度学习技术应用于关键词识别的研究历史和现状；最后，介绍本论文各章节的研究内容和结构安排；

第二章：对卷积神经网络进行详细介绍。首先，主要介绍卷积神将网络的构成，包括各层的作用与计算原理；之后介绍了对神经网络进行压缩和加速的现有方法；

第三章：设计了适用于嵌入式设备的神经网络关键词识别系统。首先，介绍了卷积神经网络中的卷积层与全连接层，分析了这两种结构的参数特性；其次，介绍了该系统中对全精度的神经网络进行模型压缩的方法，设计了对于不同的权值而采用的不同的渐进式的量化方法；之后，介绍了该系统中对网络模型进行模型加速的方法，设计了应对在模型加速过程中复杂度过高问题的算法；最后，给出了 KWS 系统的总体模型，包含系统的输入输出模块以及实现关键词识别的神经网络模块；

第四章：在嵌入式芯片 ESP32 上实现了第三章设计的关键词识别系统。首先完成了嵌入式系统的开发，实现了该嵌入式设备的正常运作；之后完成了第三章所设计的关键词识别卷积网络的训练，并对该网络模型实现了网络压缩和加速，并将该模型部署到了嵌入式设备上；最后测试了实现的关键词识别系统的性能。

第五章：总结本文所做所有工作以及贡献，并对所研究的方向做进一步展望。

第二章　面向关键词识别的卷积神经网络理论

由 1.2 小节的分析可知，深度神经网络技术在实现语音关键词唤醒的问题上，具有优异的性能。本文选取深度神经网络中的卷积神经网络模型来实现 KWS 系统。同时因为本文的目标是在嵌入式芯片模组上实现该系统，而若不对训练完成的神经网络模型进行压缩和加速，嵌入式平台的内存和算力资源是远不能胜任该任务的。所以针对本文设计的卷积神经网络模型，对其进行网络压缩与网络加速，减小其对储存、运行、算力的资源需求。本章将在 2.1 小节中介绍卷积神经网路的网络结构，并详细介绍其计算算法。之后在 2.2 小节中，详细介绍了目前常见的网络压缩与加速算法。

2.1 卷积神经网络结构

在深度学习中，卷积神经网络是一类深度神经网络。平移空间不变人工神经网络（SIANN）是 CNN 的另一名称[32]，其原因在于 CNN 的滤波器具有权重共享性质，平移空间不变性特征正是根据这样的性质而衍生的[33]。该网络结构最初的设想是来自于动物的神经元连接。在特征提取方面，CNN 主要使用其卷积层以及其他的一些隐层来达到提取输入的高级特征的作用，之后将这些高级特征输出至全连接层，进而得到属于各类别的概率[34]。CNN 抽象模型如图 2.1 所示：

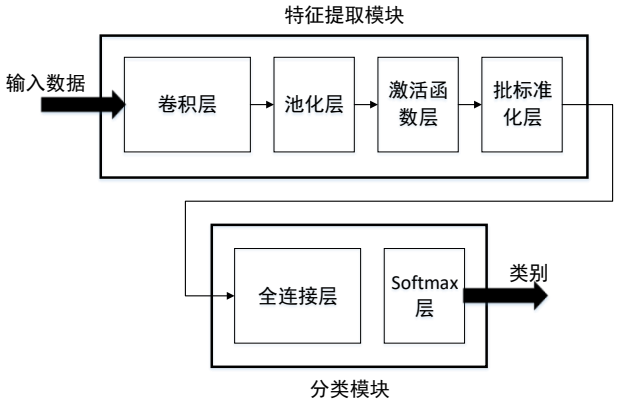

图2.1 卷积神经网络抽象模型

如图 2.1 所示，特征提取模块中起主要提取高级特征作用的是卷积层。该层使用 CNN 中特有的卷积操作进行高级特征的提取。卷积层中的卷积核负责绝大多数的特征提取操作，并且一般来说 CNN 的卷积层中会布置数量巨大的卷积核用于提取不同维度的特征，这样提取的特征越多，在后续的分类模块进行输入类别判定的时候，就越有利于网络做出正确的判断。特征模块里还有一些其他模块，例如池化层、激活函

数以及批标准层，这些层的作用会在下文详述。在 CNN 的分类模块中，全连接层主要负责拟合上一模块输出的特征，并输出至 Softmax 层，并由该层输出最终分类的概率。

1）卷积层（Convolutional layer，CONV）

CNN 当中，卷积操作是在卷积层当中使用互相关（cross-correlation）运算来进行的：

$$[f \odot g](i) = \sum_{a} f(a)\, g(i+a) \tag{2-1}$$

其中以符号$\odot$代替符号$*$，用于区分数学意义上的卷积操作。

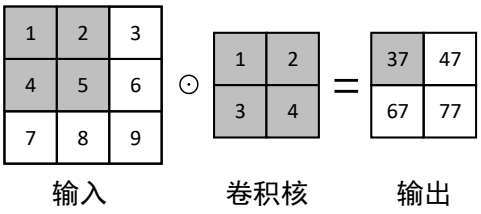

图2.2 二维卷积运算过程

图 2.2 演示了一次二维卷积的运算过程，其输入为一个3×3的矩阵，卷积核大小为2×2，且卷积核当中的参数在卷积神经网络中都是可训练的。我们可以将输入矩阵当作上述的函数g，卷积核当作函数f，上述卷积过程即是使用如下公式在进行$f \odot g$过程：

$$[f \odot g](i,j) = \sum_{a=1}^{h}\sum_{b=1}^{w} f(a,b)\, g(i+a, j+a) \tag{2-2}$$

对于多层的卷积输入，原卷积公式升级为如下的三元卷积公式：

$$[f \odot g](i,j) = \sum_{t=1}^{c}\sum_{a=1}^{h}\sum_{b=1}^{w} f(t,a,b)\, g(t, i+a, j+b) \tag{2-3}$$

其中参数c表示多层卷积输入的层数，这个维度被称之为通道（channel）。

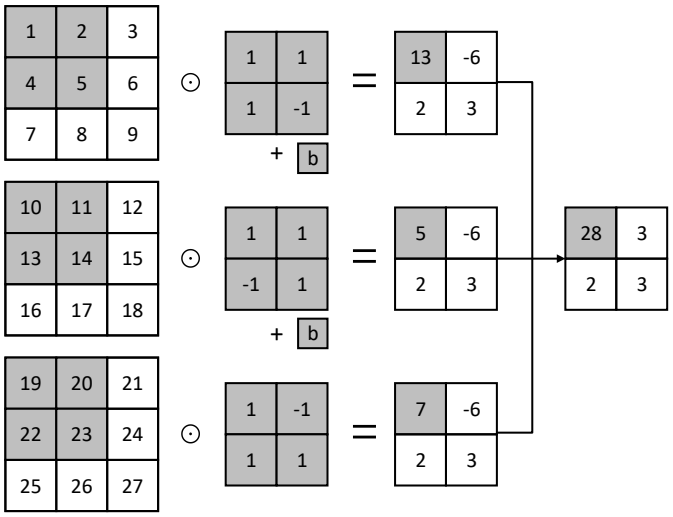

图2.3 具有三通道的二维卷积运算过程

图 2.3 演示了多通道时的卷积操作。在上述例子之中会得到一个具有三通道的输出特征图，并且经过后续层的处理变为单通道的输出层。上述三维卷积过程中对某一子区域的卷积计算公式如下：

$$Y_{i,j}=\sum_{c=1}^{C}\sum_{m=1}^{H}\sum_{n=1}^{L}X_{i\times s+m,j\times s+n,c}\times X_{i,m,n,c}+b \tag{2-4}$$

其中，参数C为输入的通道数，H和L分别表示卷积核的长和宽，b为偏置参数，X为卷积窗口内的特征数据。

2）池化层

池化层一般处于卷积层的后面，主要对模型参数起到降维的作用。一般来说，池化操作有两种，平均池化（Average pooling）和最大池化（Max pooling），其中当前使用最多的是最大池化。下图 2.4 演示一个针对尺寸为4×4的特征图的最大池化过程，其中池化窗口大小为2×2，滑动步长为 2：

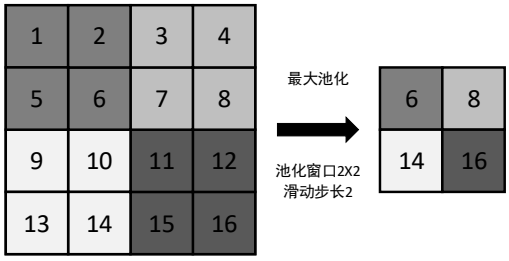

图2.4 最大池化操作过程

3）激活函数

激活函数可以为特征图增添多样性的特征，若没有激活函数，那么卷积神经网络只能表征特征的线性组合，所以激活函数通过各类的非线性运算帮助神经网络扩展搜索空间。当前最为常用的激活函数是修正线性函数 Relu 函数，该函数因为其特性，可以避免梯度消失问题，非常有利于神经网络的训练。Relu 函数的公式如下：

$$f\left(x\right)=\max\left(0,x\right) \tag{2-5}$$

4）批标准化层（Batch Normalization，BN）

批标准化层可以使得网络的数据分布具有更小程度的变化，这使得训练过程被大大加速了。但如果直接对数据做标准化，模型地表征能力将会被大大削弱，因为标准化使得学习到的特征信息被丢失了，因此通过可学习的参数做标准化处理。假设原始数据为 X_i，输入数据的均值为 μ_β，输入数据的均方差为 σ_β^2，标准化过程中的缩放因子与平移因子为 γ 和 β，这两个参数都是可学习的，样本个数为 m，原始数据 X_i 经过标准化之后的输出值为 Y_i，则批标准化的公式如下：

$$\begin{cases}\mu_\beta=\dfrac{1}{m}\sum\limits_{i=1}^{m}X_i\\ \sigma_\beta^2=\dfrac{1}{m}\sum\limits_{i=1}^{m}\left(X_i-\mu_\beta\right)^2\\ \hat{X}_i=\dfrac{X_i}{\sqrt{\sigma_\beta^2+\varepsilon}}\\ Y_i=\gamma\hat{X}_i+\beta\end{cases} \tag{2-6}$$

5）全连接层（Full Connected Layer，FC）

全连接层作为卷积神经网络模型的分类部分的开始，全连接层可以整合卷积层或者池化层中具有类别区分性的局部信息，和后面的 Softmax 层结合组成了 CNN 的分类层。全连接层的计算公式如式所示：

$$Y=b+W^T\times X \tag{2-7}$$

6）Softmax 层

Softmax 层作为 CNN 模型的输出层，主要作用在于对输入进行非线性变换，使得在经过 Softmax 层之后数值处于 $\left(0,1\right)$ 区间，即表示当前输入的置信区间，通过概率的大小判定所属的类别。Softmax 函数公式如下：

$$Y\left(x_i\right)=\frac{e^{x_i}}{\sum_{i=j}^{M}e^{x_j}} \tag{2-8}$$

2.2 卷积神经网络常见压缩加速方法

2.2.1 网络量化

神经网络量化（Neural Network Quantization），简称为网络量化技术，该技术通过减少表示每个权重所需的比特数来压缩原始网络。深度神经网络模型中的权值及激活值参数一般是以 32bit 浮点数形式储存的，但随着研究的深入，越来越多的研究表明，以降低精度的方式来储存网络中的参数的方法是可行的，并不会引起网络精度的明显下降。权值量化方法将原参数降低为小于 32 位的数据类型，并且替换为定点数据类型进行计算，一般会计算速度更快。

网络量化方法作用于网络模型时可以作用在三种参数上：（1）权重参数；（2）激活值；（3）梯度值。文献[47]对这三种参数均进行了量化测试，结果显示这三种参数对于数据位宽的敏感度是不同的。其中梯度值对数据位宽要求最高，少量的位宽下降都将极大地影响网络的性能。而权重参数对于数据位宽的敏感度最低，可以在很大程度上对其进行量化也不会影响网络性能。激活值参数对于数据位宽的敏感度处于二者中间，可以在一定程度上进行量化操作。

网络量化方法在作用形式上也存在两种形式，分别是在模型训练后进行和在模型训练时进行。但无论是哪种作用形式，在基本的量化方法上都是相同的，即都需要确定四个参数，分别为原参数最大值 m^{float} 、原参数最小值 v^{float} 、量化参数最大值 $m^{\text{int}8}$ 和量化参数最小值 $v^{\text{int}8}$ 。

之后可以通过如下(2-9)的方程组求解出缩放系数 s 与偏移量 t ：

$$\begin{cases} m^{\text{int}8}=\dfrac{m^{float}}{s}+t \\ v^{\text{int}8}=\dfrac{v^{float}}{s}+t \end{cases} \tag{2-9}$$

因此，根据上式计算对应的缩放系数 s 和偏移量 t 。保存量化处理后的参数及 s 和 t ，可以实现对网络存储体积的压缩。

论文[48]中比较了网络量化的两种作用形式，发现在模型训练完成后进行量化的网络量化方法无法在量化之后恢复原本模型的性能，而在模型训练的过程中进行量化，

可以在保留原始精度的情况下，同时进行量化，通过分步骤优化，达成模型精度不下降的目的。

网络量化方法在参数数据位宽量化的程度上，最大可以将参数从 32bit 的全精度浮点值量化为只能是+1 和-1 两个值的 1bit 定点值[35]。式(2-10)、(2-11)展示了文献[35]中将参数二值化的过程。

$$w_b = \begin{cases} +1, & \text{根据概率} p = \sigma(w) \\ -1, & \text{根据概率} 1-p \end{cases} \tag{2-10}$$

$$\sigma(w) = clip\left(\frac{w+1}{2}, 0, 1\right) = \max\left(0, \min\left(1, \frac{w+1}{2}\right)\right) \tag{2-11}$$

文献[37]提出了一种将权重参数和激活值参数均进行二值化的网络结构 XNOR-Net，该结构的计算方法如图 2.5 所示。

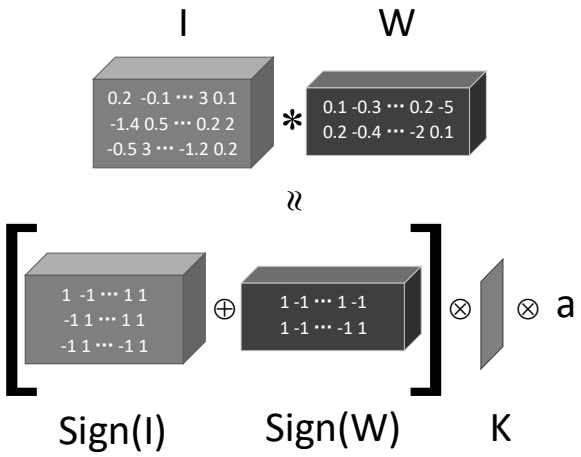

图2.5 XNOR-Net 计算过程

其中 I 表示输入特征图，W 表示卷积层卷积核参数，K 是 I 上每一个卷积窗口内元素绝对值的均值，a 是 W 内元素的绝对值的均值，⊕表示按位异或的卷积操作，⊗表示按位相乘，其中二值化公式计算如式(2-12)：

$$W_b = sign(W) = \begin{cases} +1, W \geq 0 \\ -1, W < 0 \end{cases} \tag{2-12}$$

其中W_b表示二值化后的权重值。

2.2.2 网络剪枝

DNN 网络模型是由神经元组成的，其中的网络参数就隐藏在每个神经元的连接之中。网络剪枝（pruning）方法就是将神经元之间的连接参数去除，从而在结构上裁剪神经元的连接，从而优化网络模型的计算过程的加速方法。该方法从作用结构上可以分为两类，非结构化裁剪与结构化裁剪。

（1）非结构化剪枝

非结构化裁剪可以将模型中任何神经元之间的连接进行移除。剪枝操作旨在去除那些当前不重要的参数，而恢复操作旨在恢复被错误剪掉的连接。正是因为该种剪枝方法可以在任何位置剪枝，因此可以在无损或者精度损失很小的条件下，大幅度移除网络中的权值，但是，在这样的优异压缩性能的背后是无限制的剪枝带来的网络稀疏化而导致的模型性能下降。

（2）结构化剪枝

与之相对的，结构化剪枝方法规定了不是网络中的所有位置都是可以被去除的。根据结构化粒度的不同，又可以分为向量级剪枝（vector-level pruning），核级剪枝（kernel-level pruning），组级剪枝（group-levelpruning）和通道级剪枝（filter-level or channel-level pruning）四种。

第一种向量级剪枝指的是，滤波器中的某一行被删去。文献[40]对多种剪枝进行比较，发现该种剪枝方法比非结构化剪枝压缩能力更强，因为向量级剪枝需要较少的索引来指示剪枝后的参数。

核级剪枝是指每次把 3-D 卷积核中某一个 2-D 卷积核全部去掉。

通道级剪枝则是把某个滤波器，包含其所有通道，全部去除。而选择去除哪一些卷积核就是该方法的核心。文献[38]将权重参数绝对值之和大小作为选择滤波器的指标，该指标作为朴素的滤波器选择方法，在网络剪枝技术早期确有其作用，但是因为不够有效如今也已经很少使用。文献[39]通过评估激活值的大小以及稀疏度作为选择滤波器的指标，该指标未有严格的理论支持，并且效果在如今也不再理想。文献[40]选择将卷积层中多余的卷积核去除达到网络剪枝的目的。文献[41]扩展了文献[40]的剪枝方法，将上一层卷积层的输出子集的重要程度作为筛选上一层卷积核重要程度的指标，将子集可以复现原输出精度时，该子集的补集对应的卷积核去除。

2.2.3 低秩分解

CNN 中任何一个参数都可以表示为$W \in R^{w \times h \times c \times n}$的形式，其中的子维度分别表示卷积核宽度、高度、通道数、卷积核数。而低秩分解的方法就是将这个张量转化为$t\left(t=1,...,4\right)$维张量。当被分解的张量的维度变小，那么在整个卷积神经网络的尺度上，就会有相当大的压缩空间可以利用，因为一般来说，卷积层中的卷积核的数量是非常

巨大的，对每个卷积核都进行低秩分解将会有效地压缩模型体积，降低模型对储存地要求，同时还可能加速模型地推理过程，加速模型运行速度。

关于低秩分解，目前研究领域主要集中在（1）如何对这个四维张量进行分解；（2）在哪些维度上进行约束。根据卷积核被分解成多少个分量，粗略地将基于低秩分解的方法分成三类：二分量分解，三分量分解和四分量分解。

对于二分量分解，卷积层的权值张量被分解成两部分，并且该卷积层可以被两个连续的卷积层替换。在文献[42]中提到运用极大无关组中所含向量的个数为d的奇异值分解（SVD）。一个卷积层滤波器可以分裂为两个，尺寸分别为$w\times h\times c\times n$和$1\times 1\times d\times n$。并且论文[42]还提到一种非正交分解的方法，该方法可以在进行分解之后，提到被处理模型的性能。

在对两分量分解方法进行分析的基础上，一种很直观的三分量分解方法是使用两个连续的两分量分解。低秩分解之后得到两个变量：尺寸为$w\times h\times c\times n$的特征图和尺寸为$d\times n$的矩阵。可以发现的是，分解之后前者的尺寸依旧没有变化，所以可以对其进行二次分解，这就是三分量的分解方法。之后，文献[43]中提到四分量的分解方法，该方法被称为基于 CP 分解的模型加速策略。该策略将参数分解为1×1、$w\times 1$、$1\times h$、1×1的四个卷积核。

2.2.4 知识迁移

知识迁移技术与上述的方法都不相同，上述方法都是在原有网络基础上进行操作，达到压缩或者加速的目的，而知识迁移不是，该方法是在两个网络的融合下进行模型的优化。两个网络分别是教师网络和学生网络。教师网络是高性能、高可靠、高复杂度网络，其网络规模将远大于后者，同样性能也会是远高于后者。学生网络的作用就是尽可能靠拢教师网络，拟合其性能。

论文[44]提出一种知识蒸馏（knowledge distillation，KD）方法。该方法通过高效网络的最终输出层的输出结果训练紧凑网络。论文[45]提出 FitNets 训练网络，该网络旨在训练单层网络层结构薄但是深的学生网络，这样会提高学生网络的模型性能。论文[46]提出一种注意力图（attention map，AM）机制。该机制旨在让紧凑网络在 AM 上学习高效网络，而非学习其中间层。

第三章　基于网络量化与卷积核剪枝的神经网络系统设计

第二章介绍了面向关键词识别的卷积神经网络的结构以及常见的深度学习模型的压缩加速方法。那么对于本文希望实现的嵌入式关键词识别系统使用的卷积神经网络，应该使用哪些压缩加速方法。要回答这个问题需要分析 CNN 网络中参数的特性，并根据其特性来挑选适合的方法进行模型压缩和加速。

在本章中，3.1 小节首先分析了 CNN 网络中卷积层和全连接层的参数特性，根据 3.1 小节的分析，3.2、3.3 小节详细介绍了本文要使用的模型压缩加速方法，最后 3.4 小节给出了本文用于构建关键词识别系统的模型框架。

3.1 卷积神经网络参数特性分析

本小节主要分析卷积层与全连接层的参数特性，通过分析可以得到 CNN 网络的计算特点和内存占用的特性，这将有利于选择后续合适的模型优化方法。

3.1.1　卷积层参数特性分析

为了对 CNN 的卷积层的参数性质进行分析，本小节将使用一张二维输入特征图，以其和卷积核的卷积操作为例，对此间的计算过程进行分析。该卷积运算的示意图如图 3.1 所示，该示意图中输入特征图中深色矩形区域为当前输入特征图与卷积核进行卷积操作的卷积窗口，而其中黑色箭头表示了卷积核在同一水平线上的移动轨迹，黑色虚线表示了当卷积核在同一层结束卷积操作之后，从下一行的头部重新开始。输入特征图尺寸 6×6，卷积窗口 3×3，滑动步长为 1。

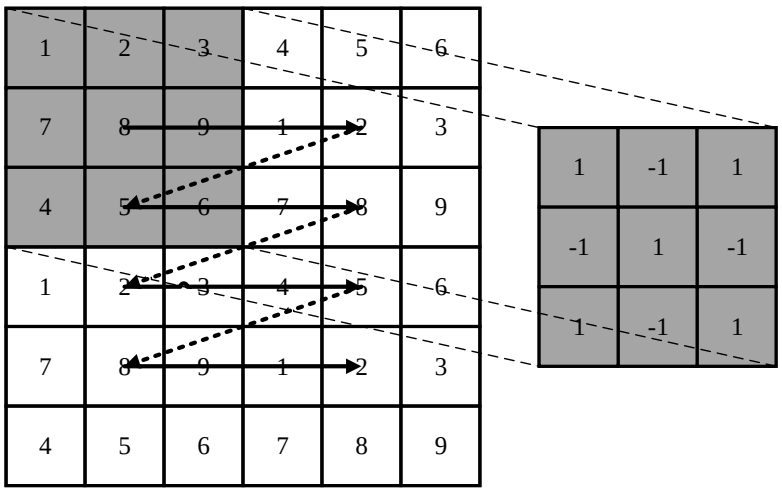

图3.1 卷积运算演示

根据上述演示图可知：

1）卷积层参数量较小。如图 3.1 所示一个 3×3 的滤波器对 6×6 的输入数据进行卷积操作。在几何上，滤波卷积核首先对输入特征图的第 1、2、3 行与第 1、2、3 列的 9 个元素进行卷积操作，之后对第 1、2、3 行与第 2、3、4 列的 9 个元素再进行卷积操作，这样几何上向右的一次移动被称为步长为 1 的横向滑动。这样的横向移动的总次数为(6-3)/1+1=4 次，同理步长为 1 的纵向移动的总次数也是 4 次，所以总共会进行 4×4=16 次卷积操作。但是对于卷积网络来说，这 16 次的卷积操作只用到了一个卷积核的参数，这被称为权值共享，这也使得卷积层的参数量相对较小，所以卷积层并不是网络的压缩重点。

2）卷积层计算量巨大。如图 3.1 所示的卷积运算，是大小为 3×3 的卷积核，那么在一次卷积运算里，就包含 9 次乘法和 8 次加法，即一次卷积计算的计算量为 17 次。同时会进行 16 次卷积运算，那么总共的计算量为 272 次。在实际的卷积层计算中，输入特征图的尺寸是远大于 6×6 的，这将导致卷积层的计算量扩大。

同时在实际的 CNN 的卷积层里，输入数据的通道数并不会只有 1，通常是输入数据是多通道的，相应的滤波器也会是多通道的，此时每一个滤波器的通道会和每一个输入通道进行卷积操作，这样会大大提高计算量。另外，通常一层的卷积层是具有多个滤波器的，其目的是在于用不同的滤波器提取不同的特性，丰富输出特征图，为分类模块提供更多的信息。这样会带来计算量的爆炸式增长。

AlexNet 是 2012 年 ImageNet 项目的大规模视觉识别挑战（ILSVRC）中的胜出者。表 3.1 展示了 AlexNet 网络的参数量和计算量。其中单位 M 表示百万，单位 K 表示千。

表3.1 Alexnet 参数表

网络层	输入特征图大小	卷积核尺寸	参数量	计算量
1	256×256×3	96×11×11	34.8K	105.7M
2	27×27×48	256×5×5	306.6K	226.3M
3	13×13×256	384×3×3	883.5K	152.4M
4	13×13×192	384×3×3	1.3M	119.6M
5	13×13×192	256×3×3	442.8K	74.2M
总计			2.97M	706M

3）卷积计算的可重用度高。如图 3.1，进行一次卷积运算之后，卷积核进行一次步长为 1 的横向移动，此时靠右的两列数据就是可以重用的。如果不参与重用，运行 CNN 的计算平台会首先将原本加载在内存中的数据移除，再将新的数据从外存中读

取到内存中，由上所述，其实原本加载的数据有一部分是可重用的，也即是这样的读取过程是重复且频繁的，这在算力充沛的平台上固然可以接受的，但是在嵌入式平台上，这样频繁地存取数据显然是不合理的。

通过分析可知虽然 CNN 的卷积层参数量少，但却有巨大的计算量，而且数据的重用度很高，因此降低 CNN 的计算量重点在于减少卷积层的计算量。

3.1.2 全连接层参数特性分析

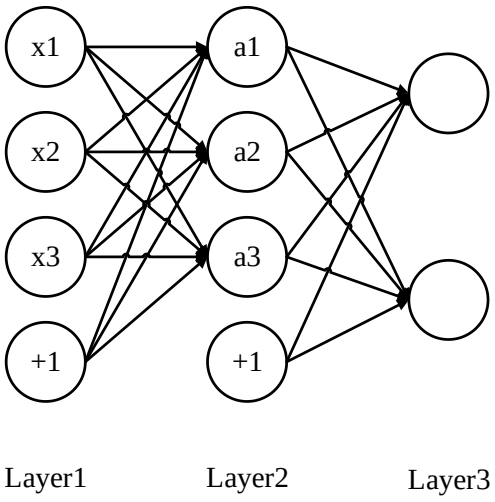

图3.2 全连接层示意图

图 3.2 显示了一个典型的全连接层的结构，其中每一个节点都是上一层每一个节点的输出节点，并且这些输出节点是上一层全部的节点，即它的输入节点，数据的加权求和，这些输入节点所对应的权值，即是图 3.2 中的黑色实线箭头。所以可以说，全连接层中的所有连接关系都代表着一个权值，且均不相同。那么假设输入节点有 N 个，其下一层的输出节点有 M 个，那么这一层的权重参数就有 N×M 个，此外存在 M 个偏置参数，总共一层全连接层就有(N+1)×M 个参数。这还仅仅是只有一层输入、输出节点的情况，在实际情况中，一些输出节点也会作为下一层输出节点的输入节点，这会使得全连接层的参数量极大增加，所以可以说在 CNN 中全连接层的参数量占据了整个网络的绝大部分。所以后续工作中需要针对全连接层进行网络压缩工作。

如图 3.2 所示，全连接层的计算只需做矩阵乘法运算再加上偏置项即可，因此乘法计算次数为 N×M 次，加法次数也为 N×M 次。全连接层的计算量相对卷积层来说并不大，因此在模型加速的工作方面暂不考虑全连接层。

第 3.1.1 节分析可知卷积层的参数量与全连接层相比确实相对较少，但是因为其通道数和卷积核数量巨大，导致发生在卷积层的计算量占据了整个网络的绝大多数，所以如果可以对卷积层采用模型加速的方法，有效压缩整个卷积层的计算量的话，整个模型对于硬件平台的算力要求就可以大为改善；第 3.1.2 节通过分析可知全连接层

参数量巨大，需要通过模型压缩的方法尽可能地压缩全连接层的参数大小，因为其占据了网络中的绝大多数的参数量，所以如果可以有效压缩全连接层的参数量，整个网络对于储存和运行的内存需求就会大大下降。所以只需要对分别对这两层进行模型加速和模型压缩，就可以极大地提高卷积神经网络的硬件适用性，即对于经过加速、压缩后的模型就可以运行在硬件资源有限的嵌入式平台上了。

第二章的 2.2 小节分析了常见的模型加速方法，其中网络量化方法可以有效地压缩网络中的参数量，一方面可以减小网络参数的储存要求，另一方面因为高精度的参数被量化为了低精度的参数，在运行计算过程时也可以达到加速的效果；此外，网络剪枝的方法可以有效地压缩卷积层的计算单元，该层是 CNN 主要的计算量的来源，通过对卷积层进行网络剪枝，可以极大地加速卷积层的计算过程。

所以在本章 3.2 中将介绍卷积神经网络的量化方法，包括对权重和激活值的量化，降低网络的参数存储量和模型大小。在 3.3 小节中将介绍卷积神经网络的剪枝方法，加速网络的计算速度。

3.2 渐进式神经网络参数量化

深度神经网络得益于其结构，在诸多问题里取得了优异的性能，但是同样因为其结构，模型体积也越来越大。该小结将通过对网络参数的数据位宽进行量化，压缩参数位宽、压缩模型大小的同时保持模型精度。本节将介绍本文对卷积神经网络权重和激活值的量化方案。

3.2.1 权重的渐进式量化

网络量化作为模型压缩的工具，可以有效地进行模型储存需求的压缩，其原理是通过缩小参数的数据位宽。但是网络中存在不同作用的参数，常见的包括梯度、激活值以及权重。本文在进行权值量化的过程中也需要考虑模型精度的变化，要求在量化之后网络的精度不应该有很大的下降。所以对不同的参数采取不同的量化方案。

在神经网络中，梯度、激活值和权重对数据位宽的要求逐次降低。那么在本文的量化方案中，为了保证模型精度，不会对对数据位宽最为敏感的梯度进行量化。相应的，对数据位宽最为不敏感的权值参数进行二值化，即将一个权重参数的数据位宽量化至 1bit。

用 $CNN\langle I,W,L,*\rangle$ 来表示一个卷积神经网络， L 表示该网络具有 L 层； I 是一个集合，其中的每一个元素 $I=I_{l(l=1,2,...,L)}$ 是 CNN 第 l 层的输入值； W 是 CNN 所有层权重参数的集合， $W_{lk(l=1,2,...,L,k=1,2,...,K)}$ 是 CNN 网络第 l 层的第 k 个卷积核； $*$ 表示网络为卷积运算。

通过符号函数将权重约束到 $\{-1,+1\}$ ，二值化的函数计算公式如式(3-1)所示：

$$W_b = sign(W) = \begin{cases} +1, W \geq 0 \\ -1, W < 0 \end{cases} \tag{3-1}$$

上式中W_b为二值化后的权重值。为了在量化后留下尽可能多的有效信息，对通过符号函数变换的权重引入缩放分子E，对于每一层来说，该缩放因子的计算公式如式(3-2)所示：

$$E = \frac{1}{c \times w \times h \times K} \times \left\| W_l \right\|_{L1} \tag{3-2}$$

即对于每一层卷积层来说，缩放因子的意义就是对该层的所有卷积核的所有通道内的权重参数进行求平均，在网络的前向传播中权重的量化公式如式(3-3)所示：

$$W_{lq} = E \times sign(W_l) \tag{3-3}$$

本文采用对权值参数的二值化量化，虽然在数据位宽方面极大地进行了压缩，但也会带来离散化权值导致的其导数消失的问题。通过随机梯度下降法更新权重，公式如式(3-5)所示，其中C为网络的代价函数，α为学习率。

$$\frac{\partial C}{\partial W_{lq}} = \frac{\partial C}{\partial W_l} \tag{3-4}$$

$$W_l \leftarrow W_l - \alpha \frac{\partial C}{\partial W_{lq}} \tag{3-5}$$

在渐进式量化过程中是会存在训练过程的，那么在前向的训练过程里，若一个权重参数已经被量化为W_{lq}，那么该量化值将参与到前向传播计算；但是在反向传播的过程中，被更新的权重值W_l仍然是全精度的。

二值量化确实在很大程度上压缩了数据位宽，让模型的储存需求大幅下降。但是在二值化的过程中，大量的有效信息也被丢失了，如果直接对所有的权重参数直接进行二值化，那么将不可避免地导致模型精度明显下降。所以本文在量化过程中使用渐进式，即分批次的量化策略。如下图 3.3 所示，首先选择全部参数的一半参数进行量化操作，之后对剩下的参数中再挑选一半继续进行量化，重复这个步骤，直至整个网络被量化。

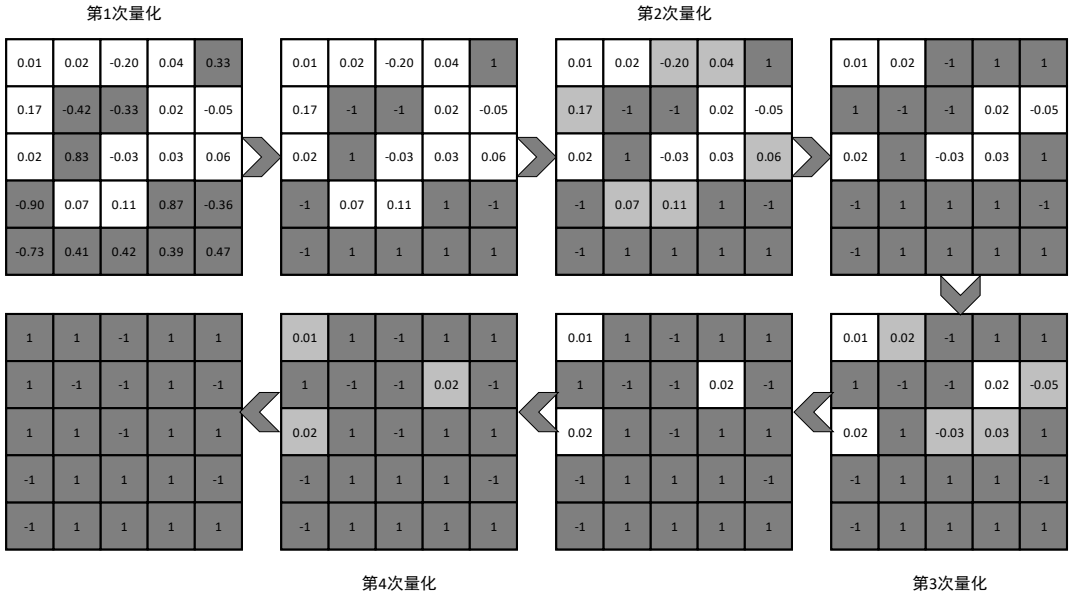

图3.3 权值的渐进式二值量化过程示例

图 3.3 展示了渐进式量化的过程，这个过程中伴随着三个步骤：（1）参数分割；（2）分组量化；（3）重训练。首先是参数分割，网络中权重参数的绝对值大小决定了其在网络里重要性的大小，权重的绝对值越大，那么在网络中的重要性也就越大。本文采用的渐进量化策略中，优先选取最为重要的参数进行量化，也即是在参数分割阶段，优先选择绝对值大的参数加入到量化分组中。为每一次的参数分割设置一个量化比例 C，将还未进行量化的权重参数的前 C 比例的参数加入量化分组，其余 1-C 比例的权重在本次量化过程中仍保持全精度浮点值类型。之后根据分组情况，对量化组实施上文所述的二值量化过程，而未进行量化的权值参数将在后续的重训练过程里进行训练，弥补刚刚的量化过程给模型带来的精度损失。

之后对未量化的权重参数重复上述三种操作，不断迭代直至模型权重全部量化。模型精度在二值量化和重训练的交替过程中维持稳定。

3.2.2 激活值的渐进式量化

如 3.1 小节所述，卷积神经网络中还存在着大量的激活层参数，因为神经网络每层有大量的输出值，要想使网络模型得到有效压缩，仅仅量化权重参数是不够的，激活值的量化对模型的极大压缩有重要作用。并且与普通的权值参数不同，激活值参数对于数据位宽的敏感度是介于对位宽有强需求的梯度值与对位宽不敏感的权值参数之间的。因为激活值对于参数数据位宽有一定的要求，所以就不能将对权值参数的量化方案直接照搬至激活值的量化上，需要针对其特性另行采取策略。

在这一小节中对于激活值采用低比特量化，即将激活值从浮点数量化至低精度值，在压缩激活值占据的模型空间，加速计算的同时保证了激活值的量化不会对模型精度

有过大的影响。激活值量化后的低精度指的是都是 2 的整数次幂（如2^{-1}、2^{-2}）或 0。激活值量化后为 2 的整数次幂，一是可以用移位操作代替乘法，加速在嵌入式平台上的模型推理过程；二是量化后的权值可以通过编码来存储（压缩嵌入式设备上的模型存储大小）。

本文对激活值的量化方法与权值的量化方法相似，均是采用渐进式量化方法，与权值量化相同，激活值的量化也包含三步：（1）参数分割；（2）分组量化；（3）重训练，实际上在后续的实际训练过程中，权重值与激活值是在同时进行量化工作的，每一层的权值与激活值同时经历相同步骤的重训练过程。

（1）参数分割

激活值在量化过程中，参数分割方法与权值分割的方法相同，即也是按照激活值绝对值的大小来决定量化的前后顺序。在网络中，绝对值较大的参数在前向传播以及模型的推理过程中能给传播和推理结果带来较高的改变，而与之相比绝对值较小的权重参数给这两个过程带来的改变就相当有限了。所以，本文在较小渐进式量化的过程中，都是优先选择绝对值较大的参数进行，原因是较大的网络行为改变可以在后续的重训练过程中弥补回来。也即是将绝对值大的量化范围内的权值参数加入量化集中，将其余的参数加入重训练集中。

对于第l^{th}层，激活值分组被定义为：

$$A_l^{(1)} \cup A_l^{(2)} = \{W_l(i,j)\} \tag{3-6}$$

$$A_l^{(1)} \cap A_l^{(2)} = \varnothing \tag{3-7}$$

其中，$A_l^{(1)}$代表需要被量化的那一组权重，$A_l^{(2)}$代表需要被重新训练的那一组权重。

为在后续实际训练中更好地实现这个分组算法，定义了一个二值矩阵T_l来区分两组权重类别：

$$\begin{cases} T_l(i,j) = 0, \text{ if } W_l(i,j) \in A_l^{(1)} \\ T_l(i,j) = 1, \text{ if } W_l(i,j) \in A_l^{(2)} \end{cases} \tag{3-8}$$

（2）分组量化

首先假定一个预训练的全精度（32 位浮点数）CNN 模型可以表示为

$$W_l : 1 \le l \le L \tag{3-9}$$

其中，W_l 为第 l^{th} 层的激活值，L 表示激活层的层数。对于激活值的量化，目的是将 W_l 量化为低精度的 $\tilde{W}_l$，其中 $\tilde{W}_l$ 的每一个元素均来自于：

$$P_l = \left\{ \pm 2^{n_1}, \ldots, \pm 2^{n_2}, 0 \right\} \tag{3-10}$$

其中 n_1 和 n_2 都是整数，且满足 $n_2 \le n_1$。则表示所有的非零激活值被限制在 $\left[-2^{n_1}, -2^{n_2}\right]$ 或 $\left[2^{n_2}, 2^{n_1}\right]$ 之间，那些绝对值小于 2^{n_2} 的激活值会被被量化为 0。

在激活值的量化过程中，需要指定两个超参数，位宽 b 以及 n_1，指定后，n_2 可以通过下式推算出：

$$s = \max\left(abs\left(W_l\right)\right) \tag{3-11}$$

$$n_1 = floor\left(\log_2\left(4s/3\right)\right) \tag{3-12}$$

$$n_2 = n_1 + 1 - \frac{2^{b-1}}{2} \tag{3-13}$$

其中，s 为某一层激活值集合中的绝对值的最大值，$floor(\cdot)$ 表示向下取整。n_1 和 n_2 指定后，P_l 也就被指定了。

那么 $\tilde{W}_l$ 可以通过下式(3-14)求出：

$$\begin{cases} \tilde{W}_l(i,j) = \beta * \mathrm{sgn}\left(W_l(i,j)\right), \ if \ \ (\alpha+\beta)/2 \le abs\left(W_l(i,j)\right) < 3*\beta/2 \\ \tilde{W}_l(i,j) = 0, \ 其他 \end{cases} \tag{3-14}$$

其中，α 和 β 是 P_l 中排序后的两个相邻的元素。即被量化的那个数距离 P_l 中最近的那个数就是最后的量化值。

（3）重训练

重训练的目的就是最大程度上减少由量化造成的精度损失，即：

$$\begin{aligned} &min \ W_l = L\left(W_l\right) + \lambda R\left(W_l\right) \\ &s.t. \ W_l(i,j) \in P_l, \ \text{if } T_l(i,j) = 0, \ 1 \le l \le L \end{aligned} \tag{3-15}$$

在重训练过程中，只更新未量化的那部分参数（即 $T_l = 1$），对应的部分还是保持 32 位全精度浮点数。

在重训练过程中使用下式来更新参数：

$$W_l(i,j) \leftarrow W_l(i,j) - \gamma \frac{dE}{dW_l(i,j)} T_l(i,j) \tag{3-16}$$

也即是只有当该参数是为量化的参数（$T_l = 1$）时，该参数才会被更新，否则就保持原样。

本节对权重采用渐进量化的策略进行了二值化，将 32bit 的权重参数压缩至 1bit，使卷积神经网络的参数存储空间压缩了 32 倍，通过对激活值的量化也使激活值的数据位宽得到了有效压缩，压缩倍数取决于采取的量化数据位宽，本文在部署实际的关键词识别系统时采用的量化数据位宽将在后文中叙述。

因为本文对激活值使用的是低位宽的量化策略，所以在整体上看，当模型量化完成后，整体压缩比例达不到 32 倍，因为其中激活值并非从 32bit 浮点值二值化为 1bit 的定点值，二是数据位宽大于 1 的定点值。华盛顿大学提出的 Binary Weight Networks（BWN）算法以及论文[37]提出的 XNOR-NET 算法，均是二值量化算法中较为经典的算法。二者与本文研究的渐进式量化方法的不同之处在于，前者均为整体量化算法，量化过程不存在迭代过程，而本文使用的算法，为了保证模型准确度，使用了重训练机制，通过动态调整未量化权值，弥补模型精度损失。

以 AlexNet 网络为测试网络，数据集为 Google Speech Command Dataset，使用 BWN、XNOR-Net 以及渐进式量化算法分别进行测试，统计三者的模型压缩倍率和模型准确度。

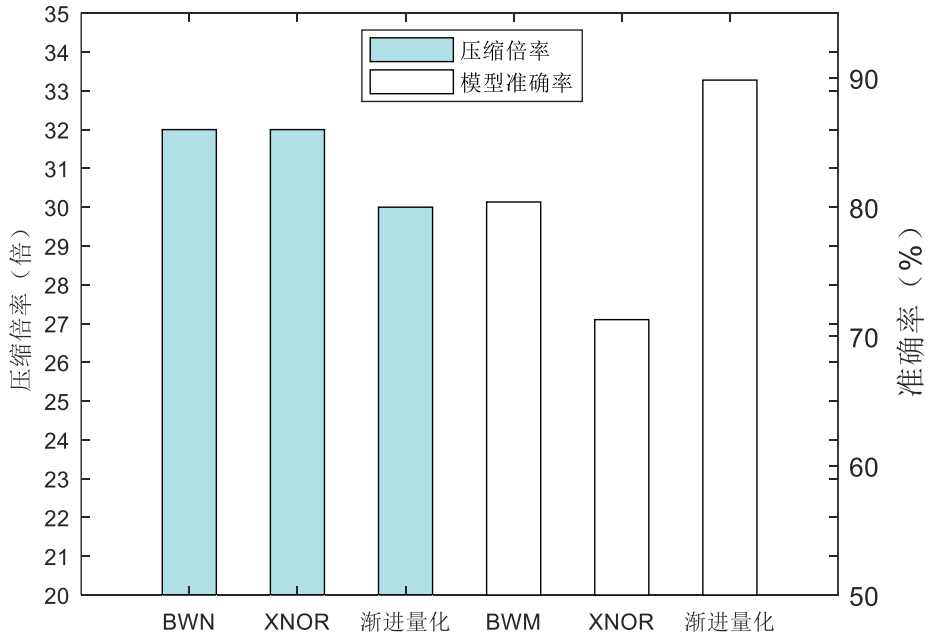

图3.4 量化算法性能对比

仿真结果显示，本文使用的方法在压缩倍率上略逊于 BWN 以及 XNOR 算法，其原因在于这两种算法是将所有的权值参数以及激活值均二值化，而本文为了保障激活值对数据位宽的需求，而降低了压缩比例。在准确度维度上，本文的算法性能高于经典算法，性能优异。渐进式网络量化算法在压缩倍率以及模型准确度上取得了较好的平衡。

3.3 基于最小化特征重建误差策略的卷积核剪枝

3.2 小节通过调整 CNN 网络中的权值参数的储存模式，即从浮点数转化为整型数或者低精度浮点数的形式，降低了模型的储存要求，可以兼容更多小内存的设备。但是 CNN 模型的计算资源的需求依旧迫切，在嵌入式设备上不进行模型加速，模型的推理过程将会极慢。

本小节使用 2.2.2 小节所述的网络剪枝的方法，针对 CNN 中的卷积层中的卷积核进行滤波器剪枝。CNN 中的卷积核数量极多，选择哪些卷积核进行剪枝，能在保证模型精度的情况下有效压缩模型计算量就是实施该策略的重点。

本文在卷积核剪枝的策略上，不再使用只考虑单卷积层中卷积核重要程度的传统剪枝策略，而是考量如果去除当前层的某一些卷积核会对后续的卷积运算结果产生的变化大小的前后多级评价标准。当去除某一些卷积核之后，产生的输出特征图在后续的卷积操作中带来的与原卷积过程不同的结果差异被称为特征误差，当某一些卷积核的去除不会导致特征误差扩大时，我们就可以认为该卷积核是冗余的，是可以被去除的，所以该剪枝策略也被称为最小化特征重建误差的剪枝策略。该策略的实际过程将在后续详述。

当前卷积核剪枝有两种方向，一种是对卷积层中的滤波器中的参数求绝对值之和，根据该值较小的卷积核说明其在网络中的重要性比较低；另一种是考察卷积核输出的特征图，将其稀疏度为考量指标，较大的被剪枝。但是如上文所述，这两种传统的方法都是仅仅只考虑了卷积核在当前层的作用，而忽略了当前卷积核在整个网络中的作用，传统方法在剪枝率高时，会导致明显的模型精度下降。本文使用的联合前后层一同评估一个卷积核的方法，将该层输出的特征图的一部分作为输入传入下一层的卷积核，若能产生特征误差不大的结果，那么证明产生该特征图的这一部分的部分卷积核就是有效的，反之，当前层的其余卷积核就是冗余的，在剪枝率的范围内就可以安全去除，而不会导致模型精度下降。

本文使用的剪枝方案为：（1）对于第 i 层卷积层，其输出特征图的一个子集作为输入特征图输入到第 i+1 层的卷积层中，若产生的新的输出特征图与原本全集输入特征图产生的特征图相近，那么该特征图子集的补集就可以安全去除；（2）因为第 i+1

层的输入特征图是由第 i 层的卷积核产生的，特征图的一部分可以被去除，那么其对应的滤波器，也即是卷积核，也可以同样被去除；（3）对每一层的卷积层重复上述操作。最终当所有卷积层都经过剪枝之后，原本模型的输出结果应该与剪枝后的模型输出结果相近，但是极大的压缩了原本卷积层的计算量。

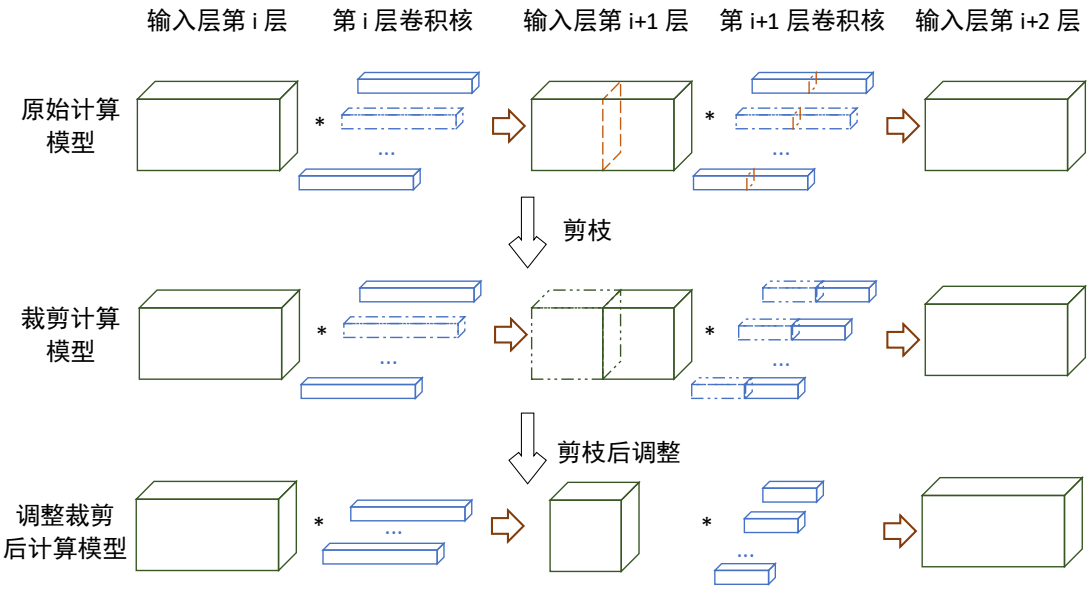

图3.5 卷积核剪枝方案

卷积核剪枝方案如上图 3.5。对于第 i+1 层的卷积操作如下图 3.6 所示：

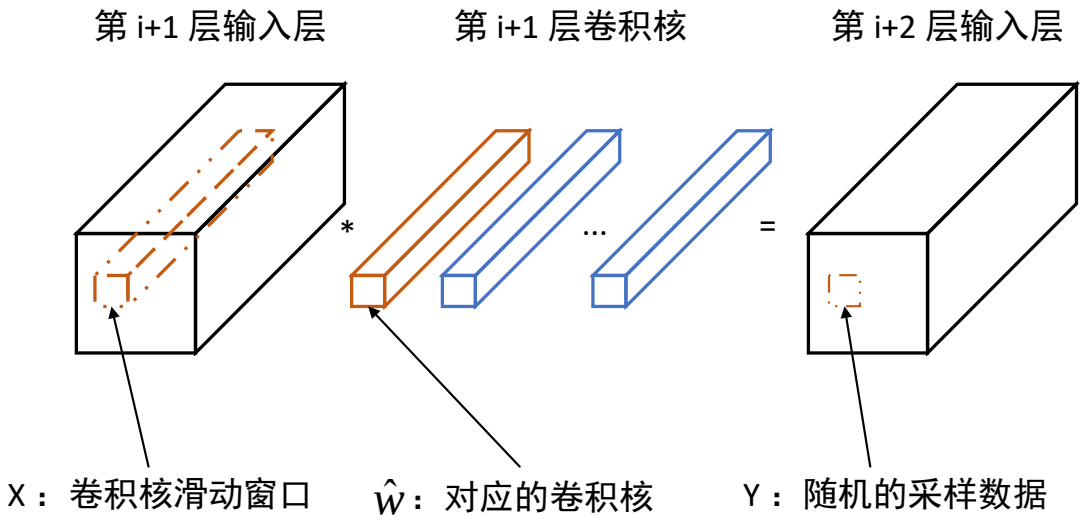

图3.6 第 i+1 层的卷积操作示例

其中， x 为输入数据， w 为卷积核，则带有偏置项 b 的卷积计算如下：

$$y=\sum_{c=1}^{C}\sum_{k_1=1}^{K}\sum_{k_2=1}^{K}\hat{w}_{c,k_1,k_2}\times x_{c,k_1,k_2}+b \tag{3-17}$$

之后，做如下定义：

$$\hat{x}_c=\sum_{k_1=1}^{K}\sum_{k_2=1}^{K}\hat{w}_{c,k_1,k_2}\times x_{c,k_1,k_2} \tag{3-18}$$

则上述公式可以简化为：

$$\hat{y}=\sum_{c=1}^{C}\hat{x}_c \tag{3-19}$$

$$\hat{y}=y-b \tag{3-20}$$

则如果存在一个通道子集S使得下式成立：

$$\hat{y}=\sum_{c\in S}\hat{x}_c \tag{3-21}$$

$$S\subset\{1,2,\ldots,C\} \tag{3-22}$$

则凡是不属于S集合内的通道及对应的滤波器（即卷积核）便能在不改变 CNN 网络模型精度的前提下被安全移除。则子集S表示保留特征图的集合序列，现假设T为表示丢弃的集合序列。则S与T的关系如下：

$$S\bigcup T=\{1,2,\ldots,C\}\ \text{and}\ S\bigcap T=\varnothing \tag{3-23}$$

那么现假设给定m个训练样本$\{(\hat{x}_i,\hat{y}_i)\}$，则原通道选择问题可以转化为如下的优化问题：

$$\begin{aligned}&\arg\min_{S}\sum_{i=1}^{m}\left(\hat{y}_i-\sum_{j\in S}\hat{x}_{i,j}\right)^2\\&s.t.\ |S|=C\times r,\ S\subset\{1,2,\ldots,C\}\end{aligned} \tag{3-24}$$

其中$|S|$表示子集S的数量，r表示预定的压缩率，即保留多少通道。同时上式可以被表示为：

$$\arg\min_{T}\sum_{i=1}^{m}\left(\sum_{j\in T}\hat{x}_{i,j}\right)^2 \quad s.t.\ |T|=C\times(1-r),\ \mathrm{T}\subset\{1,2,\ \ldots,C\} \tag{3-25}$$

$$E=\sum_{i=1}^{m}\left(\sum_{j\in T}\hat{x}_{i,j}\right)^2 \tag{3-26}$$

因为$|T|$通常比$|S|$要小很多，所以式(3-25)的求解速度会更快。

求解式(3-25)是为了最小化特征重建误差，裁剪多余的卷积核。但是需要面对的一个实际问题是，在实际使用的 CNN 模型中，每一层的卷积核数量非常多，每一层卷积层输出的特征图尺寸也会非常大，想要找到最大化冗余特征图的子集，其求解的计算量将会巨大，如果直接使用暴力组合的方法，将会极大地拖慢剪枝工作的速度，并且也会有着不能有效收敛的隐患。所以本文使用贪心算法求解该问题，该方法将每一步的局部最优策略作为当前的选择，将原问题划归为一个个的子问题，通过子问题的最优化来达成原问题的求解。

使用贪心算法求最优集合T的算法如下：

算法 1：贪心算法解决最小化重建误差问题

输入：训练集$\{(\hat{x}_i,\hat{y}_i)\}$以及压缩率$r$

输出：需要被剪枝的卷积核子集

```
T ← ∅; I ← {1,2, ..., C};
while |T| < C×(1−r) do
    min_value ← +∞;
    for each item i ∈ I and i ∉ T do
        tmpT ← T ∪ {i};
        使用 tmpT 计算 value 值
        if value < min_value then
            min_value ← value; min_i ← i;
        end if
    end for
    从集合 I 中将 min_i 移至集合 T 中;
end while
```

如算法 1 所以，使用该算法求解最佳集合T时，首先将T设置为空集，之后选择

该卷积层的众多通道中的一个作为临时变量放入 T 中，表示假设该通道是冗余的，将放入临时通道的集合 T 称为 $tmpT$ ，用 $tmpT$ 来计算式(3-25)，在此过程中记录一个全局变量 min_value ，若 $tmpT$ 计算结果小于了该全局变量，那么选择该临时通道就是正确的，这条通道确实是冗余的的，那么该通道就可以被确定的放在集合 T 中。重复上述操作，直到遍历所有的通道。

本文使用的网络剪枝方法是考虑了前后卷积核之间的联系，通过该联系进行的剪枝策略。论文[38]提出的剪枝方法通过评估卷积核的冗余度进行剪枝，是较为经典的滤波器剪枝方法。

以 AlexNet 网络为测试网络，数据集为 Google Speech Command Dataset，使用上述的滤波器剪枝算法与本文的最小化特征误差剪枝算法分别进行测试，统计二者的模型计算量压缩倍率和模型准确度。本次仿真最小化特征误差的剪枝率设置为 0.5。

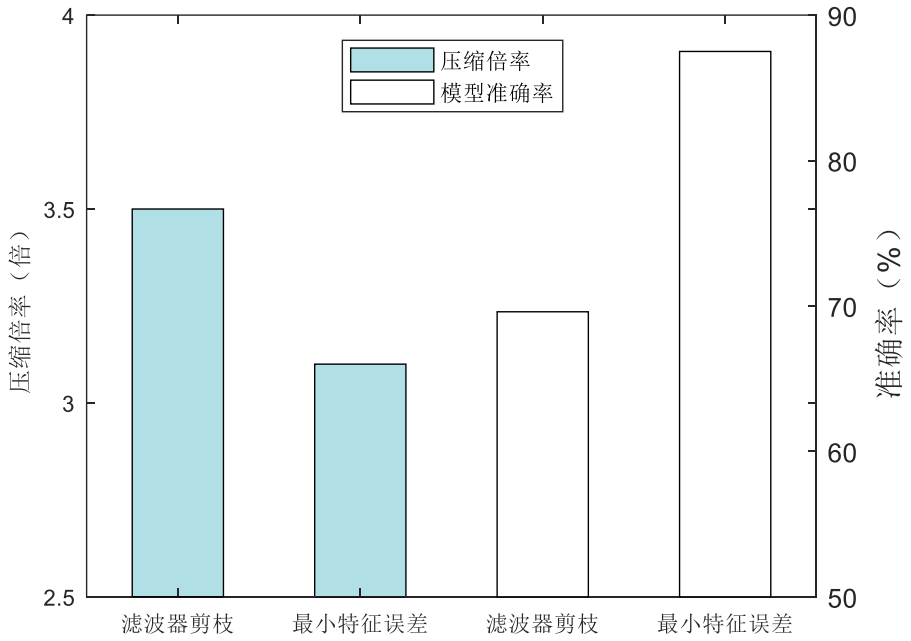

图3.7 剪枝算法性能对比

从仿真图中可以看出二者的计算量压缩差距不大，但是模型准确度相差巨大。本文方法在剪枝后的模型准确度高于前者，在加速以及保证准确度方面达成了比较好的平衡。

3.4 基于量化剪枝卷积神经网络的关键词识别系统设计

图 3.8 展示了本文的关键词识别系统的三个主要组成部分。音频数据将从拾取模块进入系统，作为整个系统的输入接受整个网络模型的识别。本文后续的 KWS 系统

实现中，ADC 模块直接使用现有元器件进行，则 ADC 原理则不再赘述。第二个部分为特征提取，即将第一组件得到的数字信号进行处理，得到比较易于后续模型推导的数据，通过第三部分的模型得到语音对应的关键词。

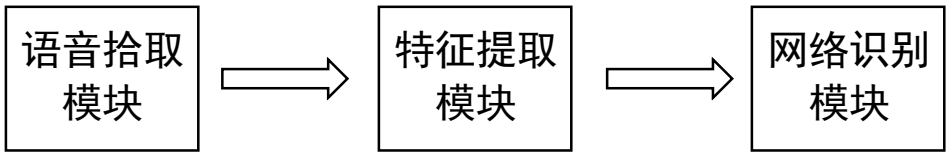

图3.8 系统流程图

3.4.1 特征提取方案

在传统的语音关键词识别系统中，在将音频传入识别模型之前，需要将语音的特征提取出来，再作为输入传入模型得到识别结果。深度神经网络结构本身是具有提取特征的功能的，一些深度神经网络甚至可以直接将原始的音频数据流直接输入，特征提取的过程在网络训练的过程中，就被固化到参数训练的过程里了。但是这样的特征提取过程本身是隐藏的以及不可解释的，并且因为数据流的直接输入，特征提取的工作交给了网络模型本身，将会导致模型结构的复杂化以及导致推理过程缓慢。本文采用 CNN 作为关键词识别系统的网络模型，并且最终会部署到嵌入式的平台上，所以需要尽可能地压缩模型结构和加快推理速度。因此，在语音拾取模块捕获音频数据之后，会首先将音频的特征提取出来，再送入网络模型之中。

最常用的音频特征提取方案是提取音频的梅尔频率倒谱系数（Mel-Frequency Cepstral Coefficients，MFCC）[49]。声道的形状显示在语音短期功率谱的包络中，而 MFCC 是准确描述此包络的一种特征表示。提取 MFCC 主要分为以下几个步骤：（1）预加重；（2）分帧；（3）加窗；（4）快速傅里叶变换（Fast Fourier Transformation，FFT）；（5）Mel 滤波器组；（6）对数运算器；（7）离散余弦变换。

对于输入长度为 L 的语音信号将其划分为重叠长度为 l，步长为 s 的语音信号，共计 T=(L-l)/s+1 帧。对于每一帧，F 为语音特征数量，对于整个长度为 L 的语音生成共计 T*F 个特征量。

在本文中，采用步长为 25ms 的 25ms 重叠窗口提取的 40MFCC 特征，这样对于一秒的音频我们可以获得 1600（即 40*40）维特征。之后将一段口语音频训练信号得到的 MFCC 特征按照从上到下从左到右的顺序组织到一张语音特征图当中，使得原本的语音信号转化为一张二维图像，之后将此特征图送到后续的 CNN 训练模型中。

3.4.2 关键词识别模型方案

如图 3.8 的第三部分，当音频信号的 MFCC 特征被提取出之后，会被送到系统的

关键词识别模型中，最终得到语音关键词的分类信息。本文在语音关键词识别模型部分选用卷积神经网络结构，并为其结构布置 5 层卷积层和 3 层的全连接层。

网络结构示意图如下图 3.9 所示。

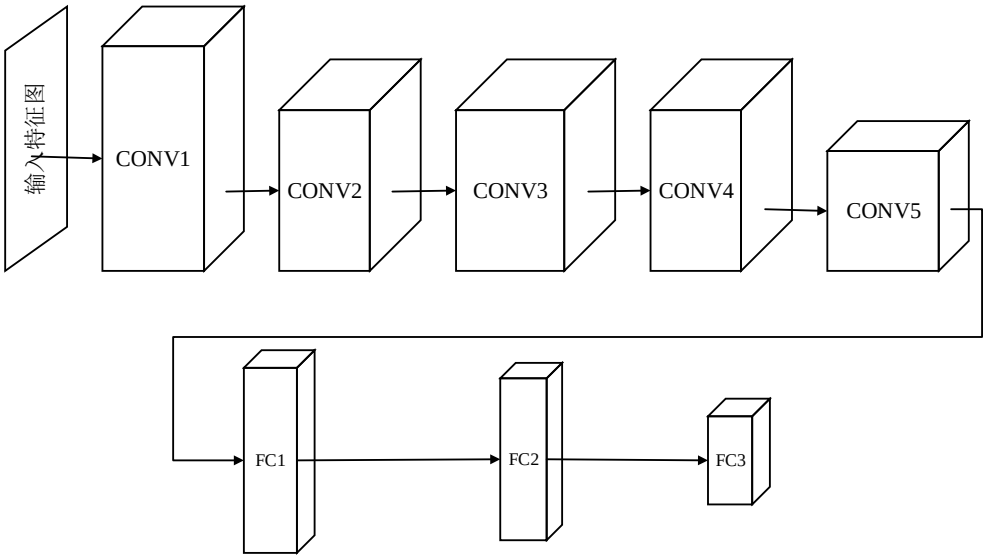

图3.9 关键词识别卷积神经网络结构

图 3.9 中，CONV 表示卷积层，FC 表示全连接层，其中包含五个卷积层和三个全连接层。在卷积层中包含了卷积计算层、批标准化层和激活函数层。由 2.1 小节介绍的，向卷积层中加入批标准化层的目的是为了改变输入参数的分布，这一层会对所有输入的特征图中的数据进行归一化处理。当这些数据归一化了之后，数据本身的分布情况将不再会大幅地影响训练过程，这将有利于模型训练的收敛。

此外第一、第二和第五个卷积层还含有最大池化层卷积层的参数如表 3.2 所示。

表3.2　卷积层参数

卷积层	输入尺寸（w×h×c）	卷积层参数（n×w×h×c）
卷积层 1	40×40×1	64×3×3×1
卷积层 2	20×20×64	64×3×3×64
卷积层 3	10×10×64	128×3×3×64
卷积层 4	10×10×128	128×3×3×128
卷积层 5	10×10×128	64×3×3×128

本文的模型的计算量和参数量如下表 3.3 所示，中 M 表示百万，K 表示千。

表3.3 模型网络各层的参数与计算量

模型网络	参数量（bit）	计算量
卷积层 1	18.4K	0.9M
卷积层 2	1,208.3K	14.7M
卷积层 3	2,416.6K	7.4M
卷积层 4	4,833.3K	14.7M
卷积层 5	2,416.6K	7.4M
全连接层 1	6,758.4K	204K
全连接层 2	524K	16K
全连接层 3	41K	1.3K
模型总体	18,227.2K	45.3M

3.5 本章小结

本章详细地介绍了本文地嵌入式语音关键词识别系统的识别模型的优化策略以及模型结构。本文使用卷积神经网络作为关键词识别的网络模型，并根据 3.1 小节中的对卷积神经网络的特性分析，详细阐述了对该模型的压缩和加速方法，也即是对普通的权值参数进行渐进式的二值量化，对激活函数层的激活值进行渐进式的低精度量化，并且对卷积层内的冗余卷积核进行最小化特征重建误差的剪枝，通过这三步的优化策略，使得原本的模型可以适应本文的目标嵌入式平台的硬件条件。

在本文设计的关键词识别系统中，首先需要一个完整的、全精度的、高可靠性的卷积神经网络模型。在得到这样一个对计算资源要求极高的模型之后，在对其进行量化和剪枝操作。本文使用的量化操作是渐进式的，可以在不会大幅度损失精度的情况下极大地压缩模型参数量，并根据网络中参数的对数据位宽的敏感度不同，分别采取不同的量化策略，使得模型的准确率和模型大小可以兼顾。

在对模型进行了量化压缩之后，还需要对量化后的模型进行剪枝。剪枝是为了加快模型在嵌入式设备上的推理速度，缩小模型所需要的计算量，降低其依赖的推理计算性能。并且还是为了维持原本模型的高精度和高可靠性，剪枝采用的是联合前后卷积层共同评价一个卷积核重要性的基于最小化特征重建误差策略的剪枝方法，通过划分上一层输出特征图子集的方法，找到最可靠、最重要的卷积核，并裁剪之外的卷积核，进而压缩卷积层计算量。

因此对本文的关键词识别网络进行压缩时首先进行量化处理，在量化的基础上对量化模型实施基于最小化特征重建误差策略的卷积核剪枝方案。

第四章　量化剪枝 KWS 系统的嵌入式实现

4.1 嵌入式系统开发

4.1.1　嵌入式系统硬件开发

（1）嵌入式芯片模组性能概况

本文的关键词识别系统选用的嵌入式模组为 Espressif　ESP32-WROOM-32，ESP32 是支持 Wi-Fi 与蓝牙功能的 MCU，被广泛使用来构建可感知和反馈现实世界数据的项目与产品，并且通常部署在嵌入式智能家电中，例如智能灯泡，智能开关，及其他智能家电中，为其提供对于公共网络或其他智能设备的连通性。

该 ESP32 模组内置了双核 32-bit 的 Xtensa LX6 的微处理器，40nm 工艺制成，最高 240MHz 的计算频率，支持 DSP 指令，支持浮点计算单元（FPU）。该模组的片上储存介质类型丰富，其主要片上储存如下表 4.1：

表4.1　ESP32 模组的片上储存

储存介质	大小	作用
ROM	448KB	用于系统 Kernal
$SRAM_1$	520KB	储存数据和指令
$SRAM_2$	8KB	RTC 快速储存器
$SRAM_3$	8KB	RTC 慢速储存器
eFuse	1Kbit	前 256bit 用作系统专用，其余 768bit 保留给用户程序

表4.2 ESP32 模组的外接接口

接口名称	数量	详情	
GPIO	34	类型	状态
		数字、模拟	上拉、下拉、高阻
UART	3	速率	通信方式
		5Mbps	异步通信、IrDA
I2S	2	频率	支持接口
		10kHz-40MHz	PDM、BT-PCM

除内置片上储存外，外部 Flash 最大可支持 16MB，本文使用的 KWS 系统外置

了 4MB 大小的外部 Flash。

此外，ESP32 芯片组支持丰富的外接接口，其中接口包括 GPIO 接口、UART 接口、I2S 接口等。其中本文主要使用的外接接口如表 4.2 所示。

该芯片组的主要功能如下图 4.1：

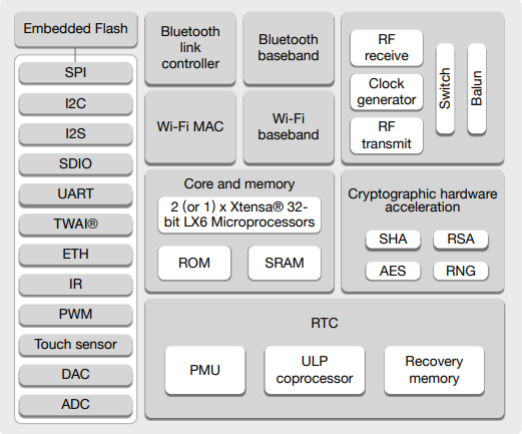

图4.1 ESP32 功能图

（2）嵌入式系统音频接口开发

本文主要设计在该芯片模组上进行语音关键词识别系统的开发，根据图 4.1 的功能流程的描述，该系统需要应对外接音频信号的输入，则需要在该芯片模组的外接接口上应用麦克风音频输入接口。

如上文所述，该芯片模组自带有 I2S 接口，即内置 IC 音频接口，该接口被设计用来在设备之间传输 PCM 信号。ESP32 内置两个 I2S 接口，即 I2S0 和 I2S1。在 ESP32 芯片模组设计之初，两个 I2S 接口的功能有些许不同。在本文的嵌入式关键词识别系统的实现中，并不会使用到 I2S 的全部功能，所以下文将简单介绍在本文中使用到的 I2S 功能，并且阐述选择哪一个 I2S 接口的原因。

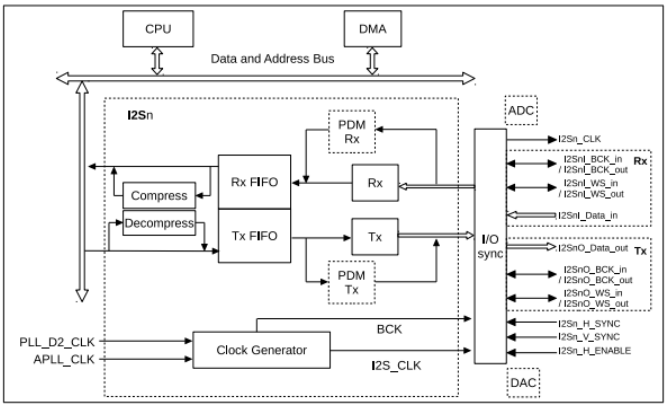

图4.2 ESP32 的 I2S 接口结构图

上图 4.2 是 ESP32 的 I2S 模块的结构框图，图中 n 对应为 0 或 1，即 I2S0 或 I2S1。I2S 模块发送和接收单元各有一块宽 32bit、深 64bit 的 FIFO。两个接口中，只有 I2S0 支持片上 DAC/ADC 模块。由 3.4 小节设计的本文的关键词识别系统的框图，在外部拾音阶段采取的是 ADC 模块进行将外界模拟的音频信号转换为数字信号的方案，该方案可以使用价格廉价的 ADC 元件，这对于实现廉价且高效的嵌入式系统是重要的一环。支持 ADC 输入的 I2S0 接口满足这样的要求，就不必使用价格更加高昂的 I2S 专用麦克风元件。所以选择可以直接支持 ADC 模块的 I2S0 接口作为音频输入接口。

关于 I2S 的各信号总线的具体描述见下表 4.3。

表4.3 ESP32 的 I2S 接口总线

信号总线	数据信号方向	信号方向
I2SnI_BCK_in	表示 I2S 模块接收数据	从机模式下，I2S 模块输入信号
I2SnI_BCK_out	表示 I2S 模块接收数据	主机模式下，I2S 模块输出信号
I2SnI_WS_in	表示 I2S 模块接收数据	从机模式下，I2S 模块输入信号
I2SnI_WS_out	表示 I2S 模块接收数据	主机模式下，I2S 模块输出信号
I2SnI_Data_in	I2S 模式下，I2SnI_Data_in[15]为 I2S 的串行数据总线	I2S 模块输入信号
I2SnO_Data_out	I2S 模式下，I2SnO_Data_out[23]为 I2S 的串行数据总线	I2S 模块输出信号
I2SnO_BCK_in	表示 I2S 模块发送数据	从机模式下，I2S 模块输入信号
I2SnO_BCK_out	表示 I2S 模块发送数据	主机模式下，I2S 模块输出信号
I2SnO_WS_in	表示 I2S 模块发送数据	从机模式下，I2S 模块输入信号
I2SnO_WS_out	表示 I2S 模块发送数据	主机模式下，I2S 模块输出信号
I2Sn_CLK	作为外部芯片的时钟源	I2S 模块输出信号

本文选择 ESP32 的 I2S0 接口进行本文的关键词识别系统的音频输入接口，连接外接的 ADC 模块获取外界的音频流。当 I2S0 模块连接片上 ADC 时，需要将 I2S0 模块配置为主机接收模式。下图 4.3 为 I2S0 模块与 ADC 控制器之间的信号连接。

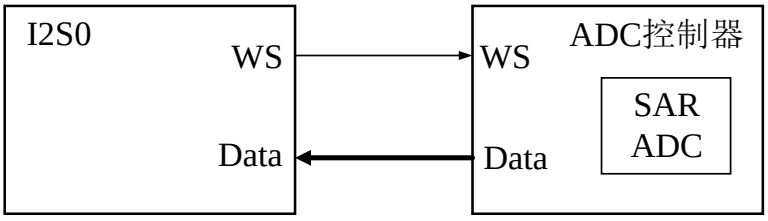

图4.3 I2S0 模块与 ADC 模块的连接

下图 4.4 显示了芯片模组与音频接口模组的连接情况。

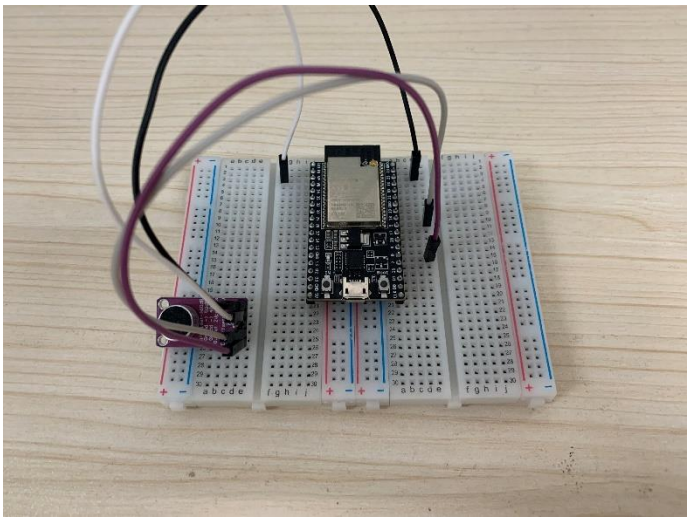

图4.4 芯片模组与麦克风连接

4.1.2 嵌入式系统软件开发

ESP-IDF（Espressif IoT Development Framework）是 Espressif 公司官方推出的物联网开发框架，适用于 ESP32 和 ESP32-S 系列 SoC[50]。该框架是基于 C/C++语言提供的一个自给自足的 SDK，方便用户在这些平台上开发通用应用程序。本文搭建的嵌入式关键词识别系统则是基于该框架开发的，图 4.5 展示了在 ESP-IDF 框架下，对于 ESP 芯片模组的应用开发流程，其中 ESP-IDF 软件开发框架和 Toolchain 工具链组是需要外部部署的，由芯片方 Espressif 和该模组 CPU 架构方 Tensilica 所提供的官方开发工具，二者具体的部署将在下文中详述。图 4.5 中的 make 指的是 Linux 开发环境下的软件编译工具，该工具可以在开发者提供对应编译配置文件的情况下高效的将软件代码编译到对应的运行平台上；除使用 Linux 的编译工具 make 之外，ESP-IDF 还支持将其部署到集成开发环境 Eclipse 中。本文主要的开发环境在基于 Linux 内核的 Ubuntu 系统，所以采用的编译工具为 make，关于 Eclipse 的配置方法不在本文中赘述。在软件环境部署完成之后，通过 make 工具可以交叉编译开发者提供的项目源码与 ESP-IDF 框架提供的 API 库和工具链，从而生成可以被烧录进 ESP 芯片模组内存且可运行的固件文件，通过特定的烧录程序将此固件文件通过芯片模组的串口上载至芯片内形成可运行的应用程序。

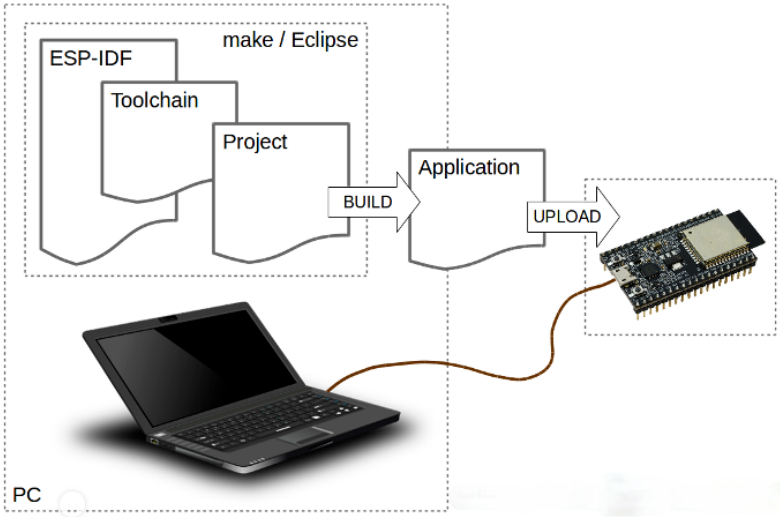

图4.5 ESP 芯片模组软件开发流程

在 ESP 模组上构建 IDF 框架需要设置该框架所需要的工具链（Toolchain，TC），其作用在于编译可用于 ESP32 的应用程序。本文的开发环境基于 Ubuntu 16.04 系统，在该系统下通过如下指令获得所需要的工具链所依赖的软件：

sudo apt-get install gcc git wget make libncurses-dev flex bison gperf python python-serial

在下载完成工具链依赖软件之后，将该 TC 源码文件下载至本地并解压缩：

tar -xzf ~/Downloads/xtensa-esp32-elf-linux64-1.22.0-61-gab8375a-5.2.0.tar.gz

该工具链是 Tensilica 公司为其旗下的 xtensa 架构 CPU 开发的一系列文件系统与软件开发工具与库的集合。因为本文采用的 ESP32 正是 xtensa 架构的 CPU，所以后续的软件编译工作均需要使用到该工具链。通过如下命令将该 TC 的路径信息加入环境变量中，这一步是为让后续的 make 工具进行编译工作时可以通过系统的环境变量找到该开发工具。

export PATH="$PATH:$HOME/esp/xtensa-esp32-elf/bin"

TC 安装完成后，还需要一些由 Espressif 公司提供的芯片接口库才能正常的调用芯片的各种接口与开发应用。这些官方接口库可以在 Espressif 公司的 Github 主页中找到，通过如下命令将其获取到本地：

git clone --recursive https://github.com/espressif/esp-idf.git

使用环境变量 IDF_PATH 来访问 ESP-IDF。可以通过如下指令设置该环境变量：

export IDF_PATH=~/esp/esp-idf

此时 ESP-IDF 系统开发框架已经搭建完成，可以通过调用相应的 API 进行嵌入式软件的开发。

```
pytin@pytin-Ubuntu:~$ . $HOME/Pytin/esp32/esp-idf/export.sh
Adding ESP-IDF tools to PATH...
Checking if Python packages are up to date...
Python requirements from /home/pytin/Pytin/esp32/esp-idf/requirements.txt are sa
tisfied.
Added the following directories to PATH:
  /home/pytin/Pytin/esp32/esp-idf/components/esptool_py/esptool
  /home/pytin/Pytin/esp32/esp-idf/components/espcoredump
  /home/pytin/Pytin/esp32/esp-idf/components/partition_table/
  /home/pytin/.espressif/tools/xtensa-esp32-elf/esp-2019r2-8.2.0/xtensa-esp32-el
f/bin
  /home/pytin/.espressif/tools/xtensa-esp32s2-elf/esp-2019r2-8.2.0/xtensa-esp32s
2-elf/bin
  /home/pytin/.espressif/tools/esp32ulp-elf/2.28.51.20170517/esp32ulp-elf-binuti
ls/bin
  /home/pytin/.espressif/tools/openocd-esp32/v0.10.0-esp32-20190708/openocd-esp3
2/bin
  /home/pytin/.espressif/python_env/idf4.1_py3.7_env/bin
  /home/pytin/Pytin/esp32/esp-idf/tools
Done! You can now compile ESP-IDF projects.
Go to the project directory and run:

  idf.py build
```

图4.6 ESP-IDF 框架搭建成功

ESP-IDF 框架开发环境并未提供可视化的开发界面，如果将其布置到集成开发环境如 Eclipse 中，可以使用可视化的开发、编译功能。本文只使用命令行的方式开发 ESP 应用。如图 4.6 所示，此时执行/esp-idf/目录下的 export.sh 脚本后可以将开发所需要的环境变量添加到当前的软件环境中，此时可以执行下列命令：

idf.py

之后出现如图 4.7 所示的显示，则表示 ESP-IDF 开发环境可以正常运行。

```
pytin@pytin-Ubuntu:~$ idf.py
Usage: /home/pytin/Pytin/esp32/esp-idf/tools/idf.py [OPTIONS] COMMAND1
                                                    [ARGS]... [COMMAND2
                                                    [ARGS]...]...

  ESP-IDF build management

Options:
  --version                       Show IDF version and exit.
  -C, --project-dir PATH          Project directory.
  -B, --build-dir PATH            Build directory.
  -n, --no-warnings               Disable Cmake warnings.
  -v, --verbose                   Verbose build output.
  --ccache / --no-ccache          Use ccache in build. Disabled by default,
                                  unless IDF_CCACHE_ENABLE environment
                                  variable is set to a non-zero value.
  -G, --generator [Ninja|Unix Makefiles]
                                  CMake generator.
  -D, --define-cache-entry TEXT   Create a cmake cache entry. This option can
                                  be used at most once either globally, or for
                                  one subcommand.
  -b, --baud INTEGER              Baud rate. This option can be used at most
                                  once either globally, or for one subcommand.
  -p, --port TEXT                 Serial port. This option can be used at most
                                  once either globally, or for one subcommand.
  --help                          Show this message and exit.

Commands:
  all                   Aliases: build. Build the project.
  app                   Build only the app.
  app-flash             Flash the app only.
  bootloader            Build only bootloader.
  bootloader-flash      Flash bootloader only.
  clean                 Delete build output files from the build directory.
  confserver            Run JSON configuration server.
  efuse_common_table    Genereate C-source for IDF's eFuse fields.
  efuse_custom_table    Genereate C-source for user's eFuse fields.
  encrypted-app-flash   Flash the encrypted app only.
  encrypted-flash       Flash the encrypted project.
  erase_flash           Erase entire flash chip.
  erase_otadata         Erase otadata partition.
  flash                 Flash the project.
```

图4.7 ESP-IDF 框架使用指南

4.2 系统主要功能实现

4.2.1 训练全精度卷积神经网络模型

为了实现嵌入式的关键词检测系统，整个实验平台主要分为两部分，一个是最终量化和加速后的模型运行的嵌入式平台，另一个是全精度模型训练时的 GPU 服务器。前者的平台搭建已在 4.1 节论述，对于后者，本文使用的 GPU 服务器的配置如下表 4.4 所示：

表4.4 GPU 服务器的配置

操作系统	Window10 专业版 19041.804 版本	
硬件配置	CPU	Intel(R) Core (TM)i7-8700 CPU 3.20GHz
	GPU	Nvidia Geforce GTX 1070TI （6GB）
	内存	16GB

Google 公司开发的 TensorFlow 框架是最为流行的深度学习框架之一，本文采取 TensorFlow 框架作为网络训练与准确率测试的平台。训练数据采用 Google 公司的语音数据集 Speech Commands，该数据集包含的语音关键词包含：静音，未知单词，“yes”，“no”，“up”，“down”，“left”，“right”，“on”，“off”，“stop”，“go”，一共 12 个标签。使用第三章所使用的本文的关键词识别卷积神经网络作为训练模型继续训练，使用 Python 3.7 作为调用 TensorFlow 的编程语言，使用 Jupyter Notebook 作为 IDE。

首先设置模型在训练过程中的一些超参数，其中包括语音识别关键字 WANTED_WORDS，即是上述语音数据集 Speech Commands 包含的 10 个关键字，之后设置前 12000 轮的训练的学习率为 0.001，后 3000 轮的学习率为 0.0001。之后数据集按照 8：1：1 的比例划分训练集、验证集与测试集。之后将训练集、验证集、测试集中的所有音频数据都转化为其对应的频谱图，送入后续的卷积神经网络进行训练。

之后对训练集数据与验证集数据进行批处理化，并为二者在代码层面添加 cache 和 prefetch 函数，前者在模型训练过程中为特定的数据集缓存数据，后者在训练过程中允许在处理当前元素时准备以后的元素，添加这两个函数的意义在于减少模型训练时的读取延迟。

训练数据准备就绪，使用 TensorFlow 框架下的接口函数快速创建 3.4.2 小节定义的本文使用的 CNN 网络。之后在 IDE 开发环境下进行网络训练。

```
model = models.Sequential([
    layers.Input(shape=input_shape),
    layers.Conv2D(64, 3, activation='relu'),
```

```
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, activation='relu'),
        layers.MaxPooling2D(),
        layers.Conv2D(128, 3, activation='relu'),
        layers.Conv2D(128, 3, activation='relu'),
        layers.Conv2D(64, 3, activation='relu'),
        layers.MaxPooling2D(),
        layers.Dropout(0.25),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_labels),
    ])
```

模型训练完成后，使用测试集数据进行测试，得到本文的关键词识别卷积神经网络全精度模型的模型精度为 92.26%。同时可以得到在测试集上，该模型的混淆矩阵的分布情况如图 4.8。

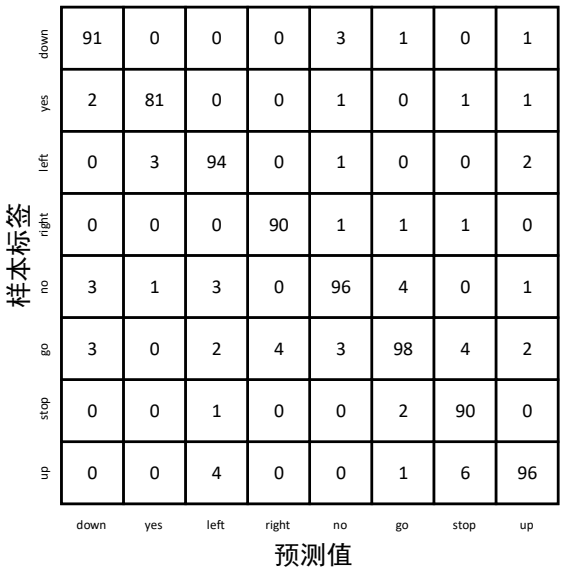

图4.8 混淆矩阵

在统计与机器学习中，混淆矩阵（confusion matrix）是一个相当有效的可视化工具，特别在监督学习问题中，更是非常常见。匹配矩阵是其在无监督学习问题中的名

称。该矩阵中，一个类的实例预测为矩阵的一列，一个实际的类的实例为矩阵的一行。混淆矩阵名称的由来，是因为观察该矩阵就可以很轻松地发现两个类是否被模型混淆了，也即是模型是否会经常性地将一个类误认为另一个类。混淆矩阵是一种特殊的,并且两维度中都有着一样的类别的集合。

混淆矩阵纵坐标为验证集中样本的标签，横坐标为将验证集中的样本通过网络模型之后由网络推理预测出的样本的分类，该分类与样本标签一一对应。所以在混淆矩阵中，对角线上的矩阵元素表示被正确分类的样本标签，而除此之外的矩阵元素则是未被正确分类的样本。一个优异的分类网络模型的混淆矩阵，应该是只有对角线上存在元素，其余均为 0 的对角矩阵。

在上图 4.8 中，绝大多数的元素集中在混淆矩阵的对角线上，说明本文训练的全精度卷积神经网络模型的性能优异。

4.2.2 对全精度网络模型进行渐进式量化

对于本文的关键词识别卷积神经网络模型内部的量化，需要对权重进行渐进量化和对激活值进行低比特量化。根据 3.2 节所述，对于权重和激活值的量化遵循（1）参数分组（2）参数量化（3）重训练三步，对权重进行渐进量化和对激活值进行低比特量化。设定渐进量化的比例因子 D=[0.5，0.75，0.875，1]，即先选取权重的量化比例为 0.5，按照 3.2.1 节的权重量化策略对所需量化的权重进行量化，其他权重保持不变，之后选取剩余未量化的权重的 50%进行量化，量化完成之后总体量化比例达到 0.75，以此类推。在对权重值进行二值量化的过程中，针对每层的激活值进行同时低比特量化，假定量化位宽为 b，测试多种激活值量化位宽带来的影响。

神经网络量化时对第一层的输入数据和激活值一同量化，会对网络准确率造成明显影响，为了保证神经网络的准确率，输入数据一般不进行量化，保持 8bit 或者 l6bit。在本文的关键词识别卷积神经网络模型上进行测试，测试数据集为 Google 公司的 Speech Commands Dataset v0.2。

对于激活值的量化，不同量化位宽带来的量化压缩效果和准确率的下降程度均是不同的。不同的量化位宽的压缩率如下表 4.5 所示：

表4.5 量化位宽对应的参数压缩率

激活值量化位宽	激活值压缩率
2 bit	16 倍
4 bit	8 倍
6 bit	5.3 倍
8 bit	4 倍

本文测试了在不同的位宽值的情况下，量化模型与全精度模型间的相对准确率。测试位宽包括从 0 到 10，测试结果如图 4.9：

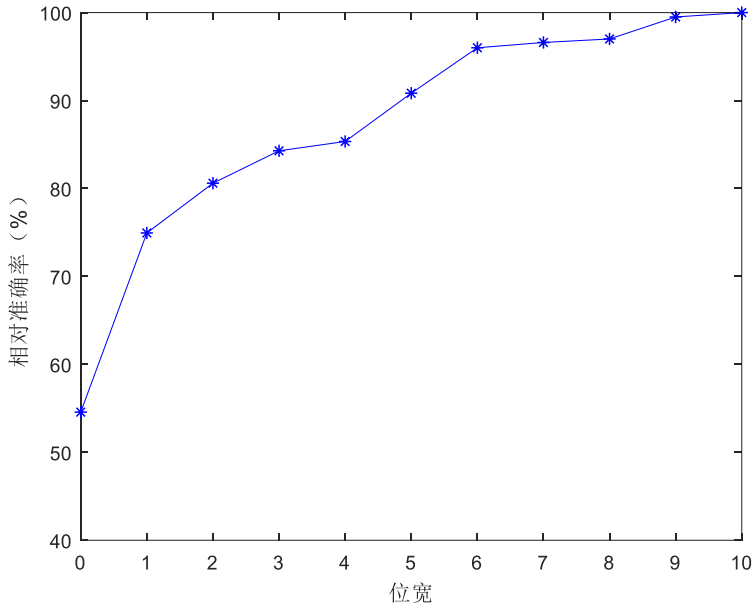

图4.9 不同量化位宽带来的模型准确率变化示意图

从测试结果可以发现，激活值的量化位宽在 5 时开始明显下降，量化位宽大于等于 6 时，模型准确率几乎没有下降。所以在后续的关键词识别模型的实现中，对于激活值的量化采用的量化位宽为 6。

下列代码实现了权值与激活值的分组、量化过程，之后整个模型进行重新训练。

根据公式（3-5）可以得到在对权值参数进行量化的变量更新代码如下。

```
weight_toINQ[abs(weight_toINQ) < lower_bound] = 0
weight_toINQ[weight_toINQ != 0] = np.sign(weight_toINQ[weight_toINQ != 0])
```

根据公式（3-14）可以得到在对激活值参数进行量化的变量更新代码如下。

```
activation_toINQ[abs(activation_toINQ) < lower_bound] = 0
activation_toINQ[activation != 0] = 2 ** \
(np.floor(np.log2(abs(activation_toINQ[activation_toINQ != 0] * 4 / 3)))) * \
np.sign(activation_toINQ[activation_toINQ != 0])
```

本文通过 4 次渐进量化将原网络模型进行压缩，通过将权值参数量化为 1bit，将激活值参数量化为 6bit 低精度定点值，将模型压缩至原模型大小的大约三十分之一。下图 4.10 记录了在量化过程中的模型大小的变化：

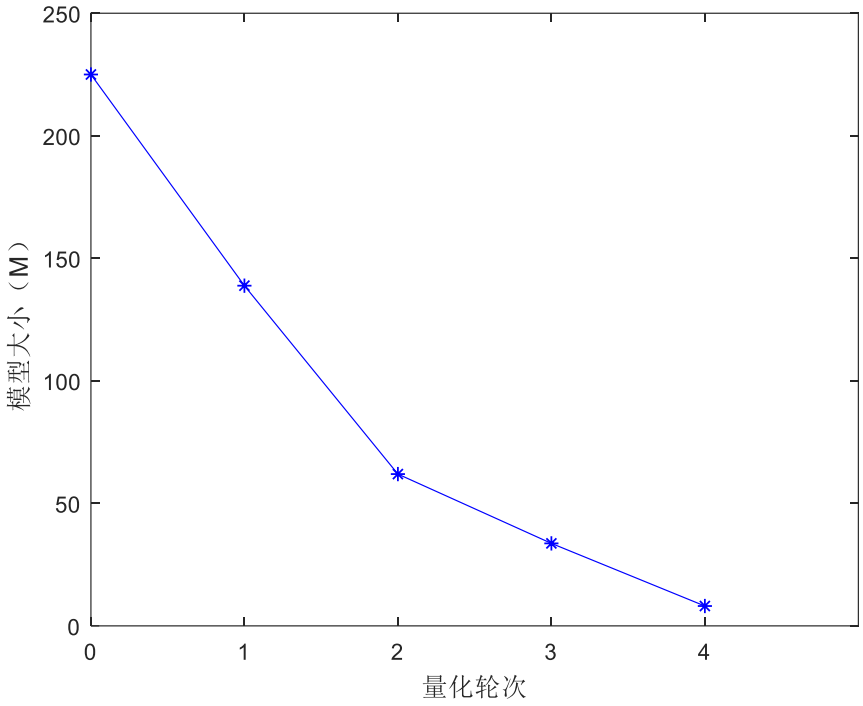

图4.10 各个量化阶段模型大小变化示意图

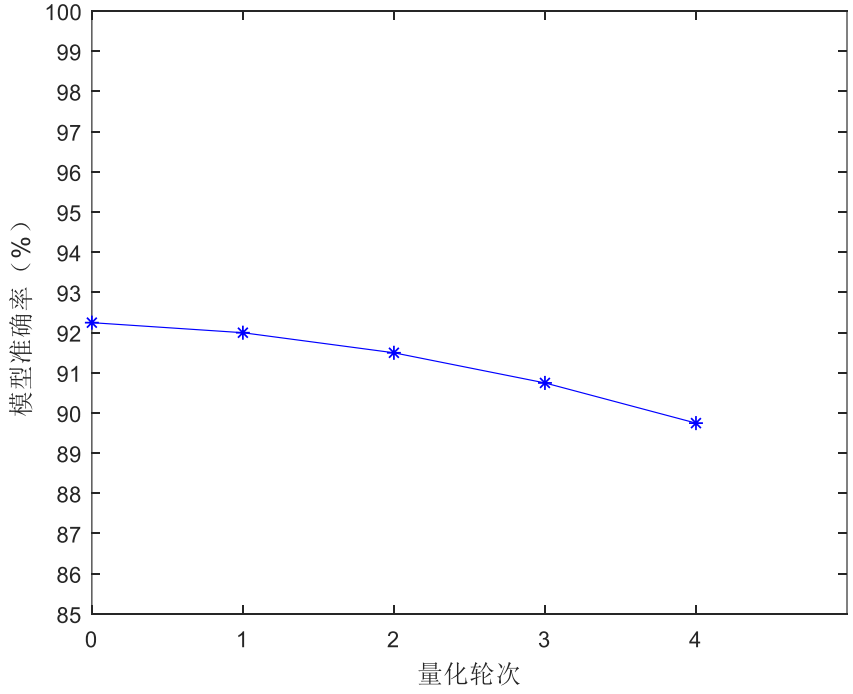

图4.11 各个量化阶段模型准确率变化示意图

上图 4.11，展示了在渐进量化过程中，压缩模型的准确度的变化，可以发现在 4 轮的量化之后，模型准确度并未发生大幅的下降，模型准确度的下降幅度未超过 3%。

4.2.3 对量化后网络模型进行卷积核剪枝

本文使用的卷积核剪枝策略在实施之后，模型的权重参数和激活值会被固定，若在进行训练操作，会导致反向传播无法通过剪枝后的网络进行，所以剪枝操作需要等量化操作完成之后再继续。本文使用的是基于最小化特征重建误差策略的卷积核剪枝方案，在本小节会对该已进行量化后的模型进行剪枝，从第一层卷积层开始，直至达到剪枝率。

本文关于最小化重建误差的剪枝策略设置了超参数剪枝率，表示对一层中卷积单元进行剪枝的比例。下图 4.12 为对各种剪枝率进行的测试，观察不同大小的剪枝率对模型准确率的影响。

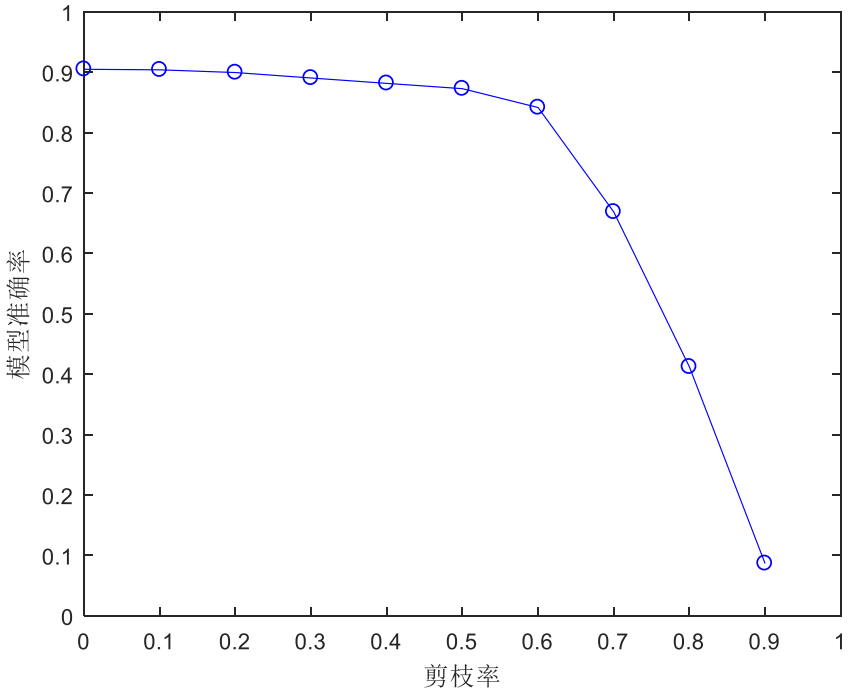

图4.12 不同剪枝率带来的模型准确率的变化示意图

可以发现，当剪枝率超过 0.6 时，经过剪枝加速后的模型准确率开始大幅下降，即当剪枝率超过 0.5 之后，模型中必要的卷积计算单元开始被剪枝，导致后续的重建误差开始失控，精度开始大幅下降。所以在本文中剪枝率设置为 0.5，此时经过剪枝后的网络模型的精度损失不超过 5%。

在量化、剪枝优化过程完成之后，本文的关键词识别卷积神经网络模型基本成型，下表统计了各网络层的变化。由 3.1 小节分析，本文对于全精度的模型的量化、剪枝操作重要集中在卷积层和全连接层，所以在下表中就不再统计其余各层的参数及计算量的变化了。如表 4.6 所示：

表4.6 卷积层剪枝效果

网络层	剪枝前	剪枝后
	卷积核尺寸（n×w×h×c）	
卷积层 1	64×3×3×1	32×3×3×1
卷积层 2	64×3×3×64	32×3×3×32
卷积层 3	128×3×3×64	64×3×3×32
卷积层 4	128×3×3×128	64×3×3×64
卷积层 5	64×3×3×128	64×3×3×64

对所有的卷积层进行剪枝率为 0.5 的剪枝之后，对于第一层卷积层的卷积核数量压缩了一半，对于第二、三、四层卷积层的每个卷积核的通道数压缩了一半，并且还去除了一半的卷积核，对于第五层卷积层，压缩了一半的卷积核通道数。

表4.7 全网络压缩加速效果

网络层	量化剪枝前		量化剪枝后		压缩倍数	
	参数量（bit）	计算量	参数量（bit）	计算量	参数	计算量
卷积层 1	18.4K	0.9M	312	0.46M	约 60	2
卷积层 2	1.18M	14.7M	10.51K	3.7M	约 115	4
卷积层 3	2.36M	7.4M	21.01K	1.8M	约 115	4
卷积层 4	4.72M	14.7M	42.03K	3.7M	约 115	4
卷积层 5	2.36M	7.4M	40.28K	3.7M	约 60	2
全连接层 1	6.6M	204K	204K	204K	32	1
全连接层 2	524K	16K	16K	16K	32	1
全连接层 3	41K	1.3K	1.3K	1.3K	32	1
全网络	17.8M	45.3M	335.43K	13.6M	约 54	3.3

表 4.7 展示了在经历了渐进式量化以及基于最小化特征重建误差剪枝之后的网络前后的参数量以及计算量的对比。可以很明显地观察到，量化方案在整体网络上达成了压缩模型地倍数到了约 54 倍的优异效果，其中卷积层的量化结果因为其中包含了激活函数层中激活值的低精度量化。在剪枝方面，因为本文使用的剪辑策略是属于结构化剪枝技术，所以只有卷积层具有被压缩的机会，所以全连接层的计算量压缩倍率只有 1。其中第二、三、四层卷积层因为卷积核的通道数和卷积核数同时被压缩，所以计算量压缩倍率可以达到 4 倍，网络整体上计算量压缩了 3.3 倍。

4.2.4 将量化剪枝网络模型部署到嵌入式设备

对于已经进行了量化、剪枝后的关键词识别卷积神经网络模型，要将其部署到嵌入式设备，首先需要将已训练好的模型文件进行固化，即将相关的训练结果（张量图、权重等）合并到一个文件中以共平台机器进行推断。此过程被称为模型冻结，该过程生成的模型称为冻结模型，因为在此过程之后无法对其进行进一步的重新训练。通过模型冻结，可以得到冻结完成后的.pb 模型文件。

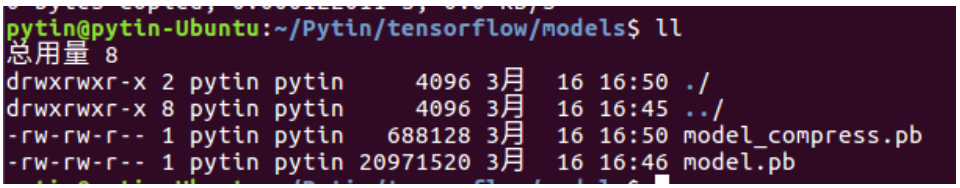

图4.13 模型生成

如图 4.13 所示，其中 model.pb 为全精度模型在进行冻结操作后的模型文件，model_compress.pb 为对全精度模型进行量化剪枝操作后的压缩加速网络模型文件。在图中可以看到，后者占用的空间大小远小于前者。

为了可以将平台级别的神经网络模型移植到嵌入式设备当中，TensorFlow 框架提供了将嵌入式的深度学习框架 TensorFlow Lite，下称 TFL。该框架可以让使用这个框架训练的神经网络模型运行在低性能的平台之上，所以被称之为“Lite”。在一般通用运算平台，诸如个人电脑、服务器上，运行全精度的深度学习模型需要特定的运算软件提供对模型解析服务，但是一般的低性能嵌入式设备不具备运行或者兼容这些运算软件的功能，甚至一些嵌入式设备并不存在操作系统，更加不可能运行全精度模型了。所以 TFL 将模型文件和运行的辅助软件打包生成 TFL 模型，之后将模型转化为 C 语言形式，因为 C 语言的特性，可以被运行在各种各样的嵌入式设备之上，这样就可以达成在嵌入式设备上运行深度学习的目的了。要使用 TFL，首先需要将模型转换为特殊格式，该转换将减小模型文件大小，并提高执行速度。

上述转换之后，原本的冻结模型.pb 文件转化为专用于 TensorFlow Lite 的.tflite 文件，如图 4.14 所示。

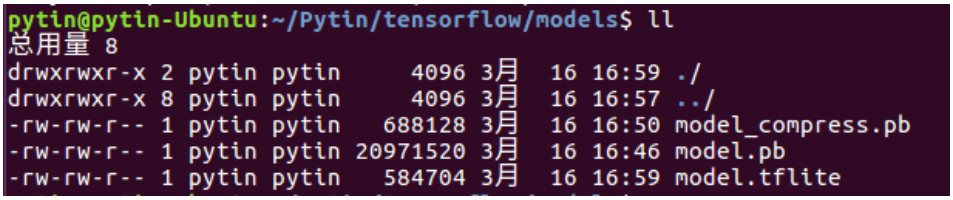

图4.14 嵌入式模型生成

之后将 TensorFlow Lite 模型转换为可由 TensorFlow Lite for Microcontrollers 加载的 C 源文件。TensorFlow Lite for Microcontrollers 是将 TensorFlow 模型运行在移动设备上的底层框架，主要由 C 语言编译执行。转换后的模型为.cc 文件。由于核心神经网络操作是纯算法，不需要任何 I/O 或其他特定于系统的功能，因此代码不必具有很多依赖关系，所以使 TensorFlow Lite Micro 即使在没有操作系统的“裸机”系统上运行也相对容易，这恰好方便于许多嵌入式设备的开发。之后将得到的.cc 文件在目标平台上进行交叉编译即可生成可以在嵌入式设备平台上执行的可执行文件。

本文使用的嵌入式平台为 ESP32，模型在部署到该平台之后需要设置音频信号的输入接口，此处设置为使用 I2S 接口接收来自外接麦克风的音频信号，外接麦克风组件内置 ADC 组件，通过 I2S 接口传入外接音频的数字信号。

```
static void i2s_init(void) {
  i2s_config_t i2s_config = {
      .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX | \
        I2S_MODE_TX),
      .sample_rate = 16000,
      .bits_per_sample = (i2s_bits_per_sample_t)16,
      .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
      .communication_format = I2S_COMM_FORMAT_I2S,
      .intr_alloc_flags = 0,
      .dma_buf_count = 3,
      .dma_buf_len = 300,
      .use_apll = false,
      .tx_desc_auto_clear = false,
      .fixed_mclk = -1,
  };
```

最后通过编写交叉编译所需要的 Makefile 文件将编译过程所依赖的 ESP-IDF 文件与编译工具（如 GCC 等）连接起来，得到可以烧录至嵌入式设备的固件文件。当前固件文件为 micro_speech.bin，如图 4.15。

图4.15 固件生成

将 ESP32 模组开发板连接 UART 接口至电脑，在 Ubuntu 环境下选择 ttyUSB0 串口，此时可以通过 ESP-IDF 框架提供的烧录工具将含有关键词识别功能的固件烧录至模组中：

idf.py --port /dev/ttyUSB0 flash monitor

4.3 系统性能测试

```
I (0) cpu_start: App cpu up.
I (239) heap_init: Initializing. RAM available for dynamic allocation:
I (245) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (251) heap_init: At 3FFB3130 len 0002CED0 (179 KiB): DRAM
I (258) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (264) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (270) heap_init: At 400899BC len 00016644 (89 KiB): IRAM
I (277) cpu_start: Pro cpu start user code
I (295) spi_flash: detected chip: gd
I (295) spi_flash: flash io: dio
W (296) spi_flash: Detected size(8192k) larger than the size in the bina
I (305) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
Hello Welcome!
This is ESP32 chip with 2 CPU cores, WiFi/BT/BLE, 2MB external flash

Keywords Spotting System is runing...
The identification results are as follows:
----------------------------------------
[Waiting...]
[2021-03-17-11:37:04]Yes[429ms]
[2021-03-17-11:38:51]No[352ms]
[2021-03-17-11:39:11]Up[388ms]
[2021-03-17-11:39:35]Down[551ms]
[Waiting...]
[2021-03-17-11:52:23]Left[482ms]
[2021-03-17-11:52:55]Right[610ms]
[2021-03-17-11:53:23]On[299ms]
[2021-03-17-11:53:50]Off[251ms]
[2021-03-17-11:54:09]Stop[628ms]
[2021-03-17-11:55:02]No[222ms]
[2021-03-17-11:55:39]Go[274ms]
[Waiting...]
```

图4.16 嵌入式关键词识别系统实机测试结果

如上图 4.16 所示，当嵌入式芯片模组通过接口上电之后，系统信息会通过 UART 串口打印至与之相连的终端机，本文的测试终端为运行 Ubuntu 16.04 的虚拟机。

当嵌入式关键词识别系统上电之后会首先运行系统自检，检测内容包括 CPU 可用核心数、内置 RAM 内存大小、外置 Flash 大小等。系统自检之后，将启动烧录的关键词识别模型，嵌入式芯片会通过外置的 ADC 传感器将外界音频信号传入，识别结果将会在关键词识别模型识别成功之后打印到终端中。识别结果由三部分组成：（1）日志打印时间；（2）语音关键词识别结果；（3）本次识别所消耗的时间。其中当系统在超过 1 分钟的时间内没有接收到有效音频时，将会在终端打印[Waiting…]标识符。

在实测中，测试人员在安静环境下距离嵌入式系统的 ADC 传感器大约 20 厘米处口述设置的语音关键词。在实机测试里，三名测试员，分别为测试员 A、B、C。每一位测试员重复每一个关键词 10 次，查看、记录系统输出结果。在实机测试中，语音关键词的识别平均耗时为 444.63ms，系统识别精度为 87.5%，接近 90%。测试内容以及测试结果如表 4.8。同时，安排测试人员在嘈杂环境下进行实机测试，同样让每一位测试人员重复每一个关键词 10 次，并记录系统输出结果，该环境下的测试结果如表 4.9 所示。在嘈杂环境下，关键词识别平均耗时与安静环境下相差不大，但是系统识别精度有所下降，为 78.75。

表4.8 安静环境下实机测试结果

关键词	测试人员	识别准确率（%）	平均识别耗时（ms）
Down	测试 A	80	523.45
	测试 B	80	
	测试 C	80	
Yes	测试 A	100	400.12
	测试 B	100	
	测试 C	90	
Left	测试 A	100	432.67
	测试 B	100	
	测试 C	90	
Right	测试 A	90	598.15
	测试 B	100	
	测试 C	80	
No	测试 A	80	230.76
	测试 B	80	
	测试 C	80	
Go	测试 A	70	350.56
	测试 B	80	
	测试 C	80	
Stop	测试 A	90	620.89
	测试 B	100	
	测试 C	80	
Up	测试 A	100	400.45
	测试 B	90	

	测试 C	80	
平均性能		87.5	444.63

表4.9 嘈杂环境下实机测试结果

关键词	测试人员	识别准确率（%）	平均识别耗时（ms）
Down	测试 A	40	510.56
	测试 B	50	
	测试 C	70	
Yes	测试 A	90	440.87
	测试 B	60	
	测试 C	70	
Left	测试 A	90	478.1
	测试 B	80	
	测试 C	90	
Right	测试 A	60	530.45
	测试 B	70	
	测试 C	80	
No	测试 A	90	250.54
	测试 B	100	
	测试 C	80	
Go	测试 A	80	329.56
	测试 B	90	
	测试 C	100	
Stop	测试 A	70	590.87
	测试 B	80	
	测试 C	80	
Up	测试 A	100	410.78
	测试 B	90	
	测试 C	80	
平均性能		78.75	442.72

第五章 总结与展望

5.1 总结

在智能物联网快速发展的当下，关键词识别技术也在这个领域大放异彩。在该技术长时间的发展后，深度学习技术在关键词识别领域越来越得到关注，取得的性能也远超过去的技术，其中卷积神经网络被广泛应用在该领域。但是虽然卷积神经网络性能优异，但是因为其模型运行时的高内存占用率以及过高的运算量，传统的神经网络并不适用与嵌入式设备。本文实现了一个基于嵌入式芯片 ESP32 的语音关键词识别系统，该系统具有低成本、低内存占用、低计算量等的优点。在模型大小方面，本文的关键词识别卷积神经网络模型在进行量化操作之后，模型大小压缩了 50 倍左右；在计算量方面，模型进行卷积核剪枝之后，计算量大概压缩了 4 倍，同时在识别精确度方面与原模型差距不大。

本文使用的模型压缩主要有网络量化与网络剪枝技术。通过渐进式的量化方案，在保证模型性能的情况下，大幅减小了模型参数占用的位宽，使得该模型可以应用在有限储存条件的嵌入式设备上；通过卷积核剪枝技术，去除卷积神经网络模型中冗余的卷积核，同时还压缩了卷积层的参数总量，大幅减少了模型推理时的计算量，使得该模型可以在计算性能有限的嵌入式设备上运行。本文研究工作主要有以下几个部分：

（1）对卷积神经网络的权重参数采取渐进式二值量化方法，将其占用的位宽从 32bit 降低为 1bit；对网络的激活值采取低精度量化，在保证一定精度的情况下，从浮点类型量化为定点类型。

（2）对卷积神经网络的卷积核采用最小化重建误差的剪枝策略，将剪枝对后续层的影响纳入对当前层的剪枝考量中，在降低模型计算量的情况下，保障模型的精度。

（3）在低功耗嵌入式芯片 ESP32 上部署了上述压缩加速后的卷积神经模型，实现了低成本、低内存占用、低计算量的关键词识别系统的搭建。

5.2 展望

本文在嵌入式芯片上实现了基于压缩加速卷积神经网络的语音关键词识别系统，采用了量化、剪枝方法使得原本内存占用高、计算量巨大的模型可以被部署到低低成本的芯片上，并正常运行。之后的工作可以在以下几点上改进与优化：

（1）本文中激活值的量化遵循相同的量化位宽，一些高于量化上限或者低于量化下限的激活值被量化为了上限值或者 0。但对于不同层的激活函数的激活值，所需要的量化位宽不必是相同的，对于激活值普遍较大的层，可以适当提高量化上限，而

激活值普遍较小的层，量化下限也可以降低，以此提高量化后的模型准确度。

（2）在将压缩加速后的模型部署到嵌入式设备上时，可以根据嵌入式设备的特有计算设备，如乘法器、位移计算器等，来加速模型部署后的推理速度。

参考文献

[1] Schalkwyk J, Beeferman D, Beaufays F, et al. Your word is my command Google search by voice: A case study[M]. Boston, MA, 2010: 61-90.

[2] S Bridle, M D Brown. An efficient elastic-template method for detecting given words in running speech[C]. Proc of the Brit, Acoust, Soc, 1974, 1-4.

[3] Miller D R H, Kleber M, Kao C L, et al. Rapid and accurate spoken term detection[C]. Eighth Annual Conference of the intemational speech communication association. IEEE, 2007: 314-317.

[4] Rothacker Leonard, Wolf Fabian, Fink Gernot A. Annotation-Free Word Spotting with Bag-of-Features HMMs[J]. International Journal of Pattern Recognition and Artificial Intelligence, 2021, 35(04)

[5] Ahmed Cheikhrouhou, Yousri Kessentini, Slim Kanoun. Hybrid HMM/BLSTM system for multi-script keyword spotting in printed and handwritten documents with identification stage[J]. Neural Computing and Applications, 2019, 1-15.

[6] Yanhua Long, Yijie Li, Bo Zhang. Offline to online speaker adaptation for real-time deep neural network based LVCSR systems[J]. Multimedia Tools and Applications, 2018, 77(21) : 28101-28119.

[7] Toktam Zoughi, Mohammad M Homayounpour, Mahmood Deypir. Adaptive windows multiple deep residual networks for speech recognition[J]. Expert Systems With Applications, 2020, 139

[8] 叶硕,彭春堂,杜珍珍,贺娟.基于 DTW 的孤立词语音识别系统设计[J].长江大学学报(自科版),2018,15(17):33-37+5.

[9] Liu Hui, Wang Wei, Wang Wenchuang. A Novel Research in Low Altitude Acoustic Target Recognition Based on HMM[J]. International Journal of Multimedia Data Engineering and Management (IJMDEM),2021,12(2).

[10] Liangpan Ye, Tao He. HMM speech recognition study of an Improved Particle Swarm Optimization Based on Self-Adaptive Escape (AEPSO)[J]. IOP Conference Series: Earth and Environmental Science,2021,634(1).

[11] Donghyun Kim, Sanghun Kim. Fast speaker adaptation using extended diagonal linear transformation for deep neural networks[J]. ETRI Journal,2019,41(1).

[12] Modelling Semantic Context of OOV Words in Large Vocabulary Continuous Speech Recognition[J]. IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP),2017,25(3).

[13] Daraee Fatemeh, Mozaffari Saeed, Razavi Seyyed Mohammad. Handwritten keyword spotting

using deep neural networks and certainty prediction[J]. Computers and Electrical Engineering,2021,92.

[14] Peter Mølgaard Sørensen, Bastian Epp, Tobias May. A depthwise separable convolutional neural network for keyword spotting on an embedded system[J]. EURASIP Journal on Audio, Speech, and Music Processing,2020,2020(2).

[15] Yu Zhang, Mingxiang Tuo, Qingyu Yin, et al. Keywords extraction with deep neural network model[J]. Neurocomputing,2020,383.

[16] J. S. P. Giraldo, M. Verhelst. Laika: A 5uW Programmable LSTM Accelerator for Always-on Keyword Spotting in 65nm CMOS[C]. ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC), Dresden, Germany, 2018, pp. 166-169.

[17] Jie G, Wang H, Fan J, et al. Deep Supervised and Contractive Neural Network for SAR Image Classification[J]. IEEE Transactions on Geoscience and Remote Sensing, 2017, 55(4): 2442-2459.

[18] Acharya U R, Shu L O, Hagiwara Y, et al. A deep convolutional neural network model to classify heartbeats[J]. Computers in Biology and Medicine, 2017(89): 389-396.

[19] Phillips P J , Moon H, Rizvi S A, et al. The FERET Evaluation Methodology for Face-Recognition Algorithms[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000, 22(10): 1090-11.

[20] Georghiades A S, Belhumeur P N, Kriegman D J. From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 23(6): 643-660.

[21] Zhong Y, Chen J, Bo H. Toward End-to-End Face Recognition Through Alignment Learning[J]. IEEE Signal Processing Letters, 2017, 24(8): 1213-1217.

[22] Feng Q, Yuan C, Pan J S, et al. Superimposed Sparse Parameter Classifiers for Face Recognition[J]. IEEE Transactions on Cybernetics, 2017, 47(2): 378-390.

[23] Ullah A, Ahmad J, Muhammad K, et al. Action Recognition in Video Sequences using Deep Bi-Directiona1 LSTM With CNN Features[J]. IEEE Access, 2018, 6(99): 1155-1166.

[24] Bo L, He M, Dai Y, et al. 3D skeleton based action recognition by video-domain translation-scale invariant mapping and multi-scale dilated CNN[J]. Multimedia Tools and Applications, 2018, 77(1): 1-21.

[25] Shi Y, Tian Y, Wang Y, et al. Sequential Deep Trajectory Descriptor for Action Recognition With Three-Stream CNN[J]. IEEE Transactions on Multimedia, 2017, 19(7): 1510-1520.

[26] He Y, Kang G, Dong X, et al. Soft filter pnming for accelerating deep convolutional neural networks[C]. //Proceedings of the 27th Intemational Joint Conference on Altificial Intelligence. Stockholms Ian : IJCAI 2018: 2234-2240.

[27] He Y, Liu P, Wang Z, et al. Filter pruning via geometric median for deep convolutional neural networks acceleration[C]. //Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Los Angeles, America : CVPR, 2019: 4340-4349.

[28] Tung F, Mori G. Clip-q: Deep network compression learning by in-parallel pruning-quantization[C]. //Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, America : CVPR. 2018: 7873-7882.

[29] Han S, Mao H, Dally W J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding[C]. In: International Conference on Learning Representations. 2016.

[30] Kim Y, Park E, Yoo S, et al. Compression of deep convolutional neural networks for fast and low power mobile applications[C]. In: International Conference on Learning Representations. 2016.

[31] Denton E L, Zaremba W, Bluna J, et al. Exploiting linear structure within convolutional networks for efficient evaluation[C]. In: Advances in Neural Information Processing Systems. 2014: 1269-1277.

[32] G. Shekar, S. Revathy, E. K. Goud, Malaria Detection using Deep Learning[J]. //2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184), Tirunelveli, India, 2020, pp. 746-750.

[33] M. Lee, J. Lee, J. Kim, et al. The Sparsity and Activation Analysis of Compressed CNN Networks in a HW CNN Accelerator Model[C]. //2019 International SoC Design Conference (ISOCC), Jeju, Korea (South), 2019, pp. 255-256.

[34] 王帅，彭意兵，何顶新．基于深度可分离卷积神经网络的关键词识别系统[J]．微电子学与计算机, 2019, 36(9): 103-108.

[35] Zhou S, Ni Z, Zhou X, et al. Dorefa-net: Training low bitwidth convolutional neural networks with low bit width gradients[J/OL]. http://arxiv.org/abs/1606.0616[2020-03-30].

[36] Courbariaux M, Bengio Y. Binarynet: Training deep neural networks with weights and activations constrainedto+lor-1[J/OL]. http://arxiv.org/abs/1602.02830[2020-03-30].

[37] Rastegani M, Ordonez V, Redmon J, et al. Xnor-net: Imagenet classification using binary convolutional neural networks[J]. In: European Conference on Computer Vision. 2016: 525-542.

[38] Li h, Kadav A, Durdanovic I, et al. Pruning filters for efficient convnets[J]. In: International Conference on Learning Representations. 2017.

[39] Hu H, Peng R, Tai Y, et al. Network trimming: A data-driven neuron pruning approach towards deep architectures[J/OL]. http://arxiv.org/abs/1607.03250[2020-03-30].

[40] Luo J, Wu J, Lin W. Thinet: A filter level pruning method for deep neural network compression.[J] In: IEEE International Conference on Computer Vision. 2017: 5068-5076.

[41] He y, Zhang X, Sun J. Channel pruning for accelerating very deep neural networks[J]. In: IEEE International Conference on Computer Vision. 2017: 1398-1406.

[42] Xiangyu Zhang, Jianhua Zou, Kaiming He, et al. Accelerating very deep convolutional networks for classification and detection[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2015.

[43] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, et al. Speeding up convolutional neural networks using fine-tuned cp-decomposition[J/OL]. arXiv preprint arXiv:1412.6553[2014].

[44] Geoffrey Hinton, Oriol Vinyals, Jeff Dean. Distilling the knowledge in a neural network[J/OL]. arXiv preprint arXiv:1503.02531[2015].

[45] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, et al. Fitnets: Hints for thin deep nets[J/OL]. arXiv preprint arXiv:1412.6550[2014].

[46] Sergey Zagoruyko, Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer[J/OL]. arXiv preprint arXiv:1612.03928[2016].

[47] Choudry F, Fiesler E, Choudry A, et al. A weight discretization paradigm for optical neural networks[J]. In: Proceedings of the International Congress on Optical Science and Engineering. 1990164-173.

[48] Courbariaux M, Bengio Y, David J. Binary connect: Training deep neural networks with binary weights during propagations[J]. In: Advances in Neural inform ation Processing Systems. 2015: 31233131.

[49] C. Paseddula, S. V. Gangashetty, DNN based Acoustic Scene Classification using Score Fusion of MFCC and Inverse MFCC[C]. //2018 IEEE 13th International Conference on Industrial and Information Systems (ICIIS), Rupnagar, India, 2018, pp. 18-21.

[50] Espressif. ESP-IDF 编程指南[EB/OL]. https://docs.espressif.com/projects/esp-idf/[2020].

致谢

一晃两三年，匆匆又夏天。三年精彩生活即将离去，在本论文即将完成之际，谨向所有在我研究生生涯的学习及生活上给予关心、帮助和指导的亲人、老师、同学致以衷心的感谢！

首先感谢的是我的导师相征教授，他刻苦钻研和敬业的态度让我敬佩，研究生生涯这三年中他对我的专业指导，让我受益匪浅。

还要感谢任鹏老师，无论是在学术研究中还是日常生活中，他都能给予我大量的帮助。任鹏老师非常耐心的指导学生的研究工作，在论文研究中任鹏老师提出了许多宝贵建议，避免了我走许多弯路，确保了最终课题及论文的完成在此，对任鹏老师再次致以最深的谢意。

此外，感谢李桥、许宝毅、董川元师兄在最初的工作中帮助我理清思路，确定课题方向。还有李晨晨、殷平泽、张李峰、葛宝尧、何茹梦和实验室的所有师弟师妹，十分感谢他们陪伴我度过这难忘的研究生时光。另外感谢我的室友秦皓楠和他的女朋友陈宜乔，不仅在科研生活中用他们丰富的学术经历给我指导，还在在枯燥的科研生活之外，带我走出自闭，参加丰富的聚会娱乐活动，在聚会里带我认识了有趣的人。感谢我的室友黄文欣，在大雪隆冬又天寒地冻的地方带我运动，寒冷的天气也阻挡不了他雪亮的光头散发的热情，让我在科研生活之外开发了值得爱好一生的兴趣。感谢我的室友策杰尼玛，虽然他不常在我的身边，但是网络情缘一线牵，他以他那敏锐到几乎玄学的直觉，预言了我研究生期间众多关键事件，为我的感情生活添加了一抹亮色，感谢他。另外还想感谢我的朋友孟佳蓉，虽然认识她的时间不长，但是自从认识她开始，我似乎就更想把自己变得更好，有一个人激励自己变得更优秀，也是很开心的一件事情。

最后，我要特别感谢我的父母，感谢他们一直默默的付出并且不求回报，感谢父母对我的关爱、理解和支持，他们的温暖伴随着我不断成长。

西安電子科技大學

地址：西安市太白南路2号
邮编：710071
网址：www.xidian.edu.cn