DOI: 10.19650/j.cnki.cjsi.J2312181

资源受限的机械振动 WSN 层次分解 CNN 边缘计算方法*

付 豪,邓 蕾,汤宝平,李子昊,吴艳灵
(重庆大学机械传动国家重点实验室 重庆 400030)

摘 要:用于机械振动监测的无线传感器网络节点的微控制器需要进行复杂的边缘计算,然而硬件资源受到限制。卷积神经网络作为一种性能优越的深度学习算法,若将其运行在 MCU 上可增强边缘 WSN 节点的计算能力。本文提出了一种不修改 CNN 模型的层次分解方法,解决了难以在资源受限的 MCU 上运行不轻量化 CNN 的问题,实现了机械振动 WSN 节点的计算能力增强。首先通过设计文件结构用于分解并存储 CNN 模型参数,然后提出内存管理方法并推导随机存取存储器的消耗过程,最后提出参数定位方法准确高效地读取模型参数。实验表明仅使用 1.76 KB RAM 与 2.14 KB Flash,在 3.15 ms 内便可实现高准确率的边缘计算识别任务。

关键词: CNN;边缘计算;MCU;资源受限;机械振动
中图分类号: TP393.1 TH17 **文献标识码**: A **国家标准学科分类代码**: 460.40

Hierarchical decomposition of CNN for resource-constrained mechanical vibration WSN edge computing

Fu Hao, Deng Lei, Tang Baoping, Li Zihao, Wu Yanling
(*State Key Laboratory of Mechanical Transmission, Chongqing University, Chongqing 400030, China*)

Abstract: The microcontroller of wireless sensor network (WSN) nodes used for mechanical vibration monitoring requires intricate edge computing, yet face limitations in hardware resources. Convolutional neural network (CNN), as a high-performance and commonly used deep learning algorithm, can enhance the computational capabilities of edge WSN nodes when run on microcontroller units (MCUs). This paper proposes a hierarchical decomposition method for CNN models without modification, addressing the challenge of running non-lightweight CNN on resource-constrained MCU and enhancing the computational capabilities of mechanical vibration WSN nodes. First, a file structure is designed to decompose and store CNN model parameters. Subsequently, a memory management method is proposed, and the consumption process of random-access memory is derived. Finally, a parameter localization method is introduced to accurately and efficiently retrieve model parameters. Experiments demonstrated that with only 1.76 KB of RAM and 2.14 KB of Flash, high-precision edge computing recognition tasks can be accomplished within 3.15 ms.

Keywords: CNN; edge computing; MCU; resource constrain; mechanical vibration

0 引 言

由于机械振动无线传感器网络(wireless sensor network, WSN)数据量大且宽较窄[1-2],难以实时传输导致数据阻塞,降低数据处理实时性。因此,为提高诊断效率需要尽可能减少数据传输,可以将数据处理过程直接在靠近数据源的 WSN 节点上进行。近年来,边缘计算的迅速发展为 WSN 提升计算性能提供了新的发展方向。边缘计算主要在数据源侧进行数据处理,仅需上传少量结果至云平台[3-4]。一般来说,WSN 节点控制与运算核心是微控制器(microcontroller units, MCU),其资源非常有限,计算能力较弱。一方面 MCU 的存储器大小与主频受到限制,另一方面深度学习的模型参数量庞大、运算过程复杂[5]。因此亟需解决将深度学习模型运行在资源受限 MCU 上的问题。在众多的深度学习模型中,CNN 是

收稿日期:2023-11-22 Received Date: 2023-11-22
*基金项目:国家自然科学基金(52375082,52275087)项目资助

一种应用广泛、效果优异的基础模型,被广泛应用于故障诊断[6]。文献[7]提出了一种基于卷积神经网络(convolutional neural network,CNN)与长短期记忆网络(long short-term memory,LSTM)结合的故障诊断方法,其CNN层不依赖先验知识进行进行特征学习。文献[8]通过多输入振动与声音的CNN模型来进行故障诊断。然而上述方法是直接将CNN模型运行在硬件资源丰富的计算机上,未考虑将其应用在边缘MCU中。

将CNN模型运行在MCU上需要着重考虑MCU的RAM与Flash的大小、运行主频以及CNN结构。目前使CNN运行在MCU上最常用的方法是将模型轻量化,轻量化常用方法有网络剪枝、参数量化、知识蒸馏、深度可分离卷积等[9-15]。在文献[16]中提出了一种网络修剪方法,该方法通过识别CNN的结构冗余并修建最大冗余选定层的过滤器来减轻网络复杂度。文献[17]提出了一种用于在通道级别修剪CNN的变分贝叶斯方案以删除冗余通道。文献[18]在参数量化方面将float 32位数据转换成int 8数据以减少数据量,并使用CMSIS函数自动生成推理代码。文献[19]为减少存储和计算的参数量,用单独卷积运算代替深度可分离卷积运算,实现了不同大小的深度可分离卷积运算。上述轻量化方法在一定程度上解决了在资源受限的硬件上运行CNN网络的局限性,但每种方法都需要对CNN网络模型构或者权重参数进行修改,CNN模的完整性进行了破坏,在一定程度上对精度造成了损失。

除了考虑CNN模型本身,在WSN中进行边缘计算还需注意MCU硬件资源。一般来说。MCU的随机存取存储器(random access memory, RAM)内存仅为几KB至几百KB,难以一次性加载所有的模型参数;MCU的主频一般在数十MHz到数百MHz,远小于计算机CPU的主频。文献[20]以Raspberry Pi 4B为载体,在主频为1.5 GHz,RAM内存为4 GB的CPU上运行深度学习算法以进行风电齿轮箱故障诊断。文献[21]在基于RAM处理器的STM32MP157芯片上进行采用全局复合压缩方法进行智能故障诊断,其处理器频率为800 MHz,运行内存为8 G。文献[20-21]虽然在边缘测进行故障诊断,但所采用的处理器硬件资源丰富,难以在资源受限的条件下使用。文献[22]通过能量熵的蒸馏量化压缩方法压缩CNN实现了在192 KB的Cortex-M4上进行滚动轴承的故障特征提取,该方法实现了在低RAM上的CNN算法运行,但改变了模型本身,存在一定的精度损失。

上述文献从模型本身轻量化与采用高性能硬件方面实现了CNN用于边缘数据处理,但采用模型修改与高性能处理器的方法会带来精度降低、功耗与成本增加的问题。因此,在本研究中提出了一种层次分解的CNN方法,该方法既不改变CNN模型本身结构也无需高性能的处理器,即可实现MCU的边缘计算。

1　层次分解CNN过程

如图1所示,层次分解(hierarchical decomposition, HD)包括横向与纵向,横向按照每个滤波器的通道方向进行分解,纵向按照多个滤波器方向进行分解。通过横向与纵向的层次分解可避免一次性加载所有参数,节省RAM空间。

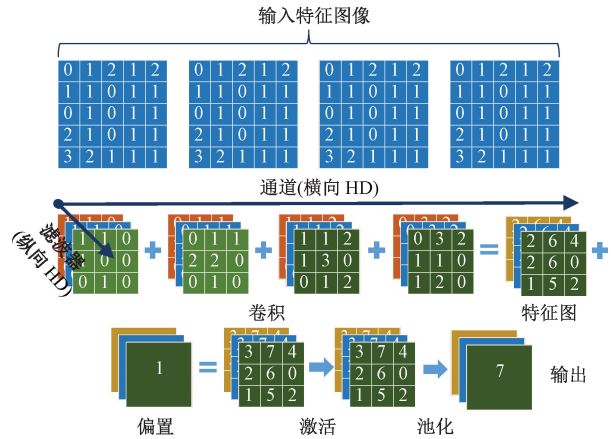


图1　HD-CNN层次分解核心流程

Fig. 1　Hierarchical decomposition core process of the HD-CNN

1.1　文件结构设计

文件结构主要用于模型参数保存与读取,文件结构组成包括文件头长度h_l、CNN卷积层数l_n、总参数量p_s、输入图像大小i_s、CNN结构中的网络参数、全连接输入长度f_l、输出长度o_l、每层CNN的权重参数w_i与偏置参数b_i以及全连接权重参数w_f,每个变量占用4字节。其中CNN网络参数包括每层卷积的输入输出维度、卷积与池化核大小,具体定义如表1,在进行参数保存与读取时需按照表1定义的顺序读取。

表1　HD-CNN文件结构

Table 1　The file structure of HD-CNN

类型	文件头参数											
项目	h_l	l_n	p_s	i_s	i_{di}	o_{di}	k_i	p_i	f_l	o_l	w_i	b_i	w_f
字节	4	4	4	4	–	4	4	–	–	–			

其中,i_{di}与o_{di}分别为每层卷积的输入输出维度,k_i与p_i分别为每层卷积的卷积与池化核大小。w_i与b_i的具体字节数由实际模型结构确定。

1.2　RAM与Flash内存管理

内存管理用于判断MCU中的Flash大小是否存储CNN全部参数,以及判断在HD-CNN运行过程中RAM

内存是否溢出。根据表 1 可以看出,部署到 Flash 中的数据量 t_f 为非固定字节 t_{fu} 与固定字节 t_{fc} 总和,计算公式为:

$$t_f = t_{fc} + t_{fu} \tag{1}$$

其中,t_{fc} 计算公式为:

$$t_{fc} = h_l + l_n + p_s + i_s + f_l + o_l = 4 \times 6 = 24 \tag{2}$$

非固定字节 t_{fu} 由每层网络结构参数、权重与偏置参数组成,具体计算为:

$$t_{fu} = t_{fuc} + t_{fub} + t_{fuw} + t_{fwf} \tag{3}$$

其中,t_{fuc} 为所有层网络结构参数总和,t_{fub} 为所有层 CNN 偏置参数之和,t_{fuw} 为所有层 CNN 权重参数总和,t_{fwf} 为全连接权重参数。

t_{fuc} 计算公式为式(4),其中 t_{fuci} 为每层 CNN 网络参数总和。

$$t_{fuc} = 4 \times \sum_{i=1}^{l_n} t_{fuci} = 4 \times \sum_{i=1}^{l_n} (i_{di} + o_{di} + k_i + p_i) \tag{4}$$

t_{fub} 计算公式为:

$$t_{fub} = 4 \times \sum_{i=1}^{l_n} t_{fubi} = 4 \times \sum_{i=1}^{l_n} (o_{di}) \tag{5}$$

t_{fuw} 计算公式为:

$$t_{fuw} = 4 \times \sum_{i=1}^{n} t_{fuwi} = 4 \times \sum_{i=1}^{n} (o_{di} \times i_{di} \times k_i^2) \tag{6}$$

t_{fwf} 计算公式为:

$$t_{fwf} = 4 \times f_l \times o_l \tag{7}$$

按照传统方式,在运行 CNN 时一次性将权重参数与偏置参数加载到 MCU 的 RAM 中,此时需要的 RAM 内存约等于存放在 Flash 中的参数量。而实际 MCU 的 RAM 空间远小于 Flash 空间,因此将导致 RAM 内存溢出,无法直接运行 CNN。采用 HD-CNN 方法,在合适的时机动态申请与释放 RAM 内存可大大降低 RAM 使用量,此时 RAM 占用量 t_r 计算如下:

$$t_r = t_{ri} + t_{rh} + t_{rc} + t_{rl} + t_{re} \tag{8}$$

其中,t_{ri} 为输入大小,t_{rh} 为文件头大小,t_{rc} 为卷积占用大小,t_{re} 为程序运行时局部变量、函数形参、循环变量与内存指针大小等,t_{re} 在 CNN 实际运行中占比远小于 t_r,可以忽略不计,因此有:

$$t_r = t_{ri} + t_{rh} + t_{rc} + t_{rl} + t_{re} \approx t_{ri} + t_{rh} + t_{rc} \tag{9}$$

当 CNN 网络结构与输入图像大小确定时,即确定了 t_{ri}、t_{rh},此时只需要计算 t_{rc} 与 t_{rl}。

如图 2 所示一层 CNN 卷积计算过程,可以看出,需要申请 RAM 内存的阶段有 3 处。首先是在卷积计算前为每个 filter 的输出特征图申请 RAM,申请大小 t_{rf} 为所有 filters 的输出特征图大小之和,其计算公式为:

$$t_{rc} = t_{rf} = 4 \times n \times w_{out} \times h_{out} \tag{10}$$

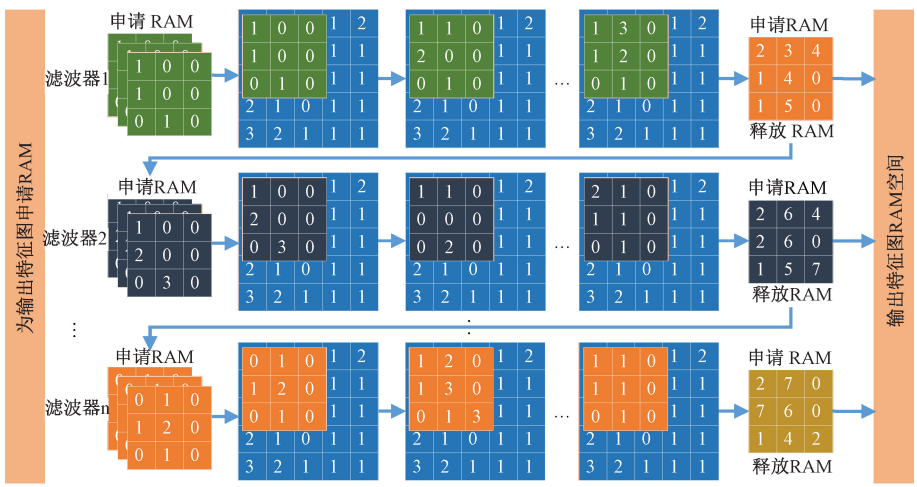


图 2 在 MCU 上进行 HD-CNN 卷积计算

Fig. 2 The convolution computing of HD-CNN on MCU

其中,n 为 filter 个数,w_{out} 与 h_{out} 分别为每个 filter 输出特征图的宽与高,它们的计算公式如下:

$$w_{out} = \left\lfloor \frac{w_{in} + 2 \times p - d \times (k-1) - 1}{s} + 1 \right\rfloor \tag{11}$$

$$h_{out} = \left\lfloor \frac{h_{in} + 2 \times p - d \times (k-1) - 1}{s} + 1 \right\rfloor \tag{12}$$

其中,w_{in} 与 h_{in} 为每个 filter 输入特征图的宽与高,p 为边缘填充,d 为空洞大小,k 为卷积核大小,s 为步长。在本研究中卷积核长宽相等,填充与空洞均为 0。

接着需要为每个 filter 的卷积核申请 RAM,申请大小 t_{rk} 为每个 filter 的通道大小之和,其计算公式为:

$$t_{rk} = 4 \times i_d \times k^2 \tag{13}$$

其中,i_d 为输入特征维度(每个 filter 的通道数)。

最后为每个 filter 的输出特征图大小申请 RAM,申请大小 t_{rs} 为每个 filter 的输出特征图大小,其计算公式为:

$$t_{rs} = 4 \times w_{out} \times h_{out} \tag{14}$$

在每个 filter 卷积计算完成后需要将特征图赋给输出特征图 RAM,然后释放掉先前 filter 的输出特征图与卷积核 RAM。此时 t_{rc} 为:

$$t_{rc} = t_{rf} + t_{rk} + t_{rs} \tag{15}$$

当所有 filters 的卷积计算完成后便可释放掉 t_{rk} 与 t_{rs},此时 t_{rc} 为:

$$t_{rc} = t_{rf} \tag{16}$$

卷积计算后需要在特征图上加上偏置,同样需要为偏置申请 RAM,申请大小为:

$$t_{rb} = 4 \times i_d \tag{17}$$

此时 t_{rc} 为:

$$t_{rc} = t_{rf} + t_{rb} \tag{18}$$

计算完偏置后便可释放掉 t_{rb},此时 t_{rc} 为:

$$t_{rc} = t_{rf} \tag{19}$$

采用 ReLu 激活过程不需要申请内存,因此 t_r 值不改变。

图 3 描述了池化过程,同样在池化初始阶段需要为输出特征图申请内存,申请 RAM 大小为 t_{rp}:

$$t_{rp} = 4 \times o_d \times w_{out} \times h_{out} \tag{20}$$

其中,o_d 为输出池化特征图数量,w_{out} 与 h_{out} 为输出特征图长宽尺寸,根据式(10)与(11),此时的 k 为池化核大小,s 与 k 相等。此时 t_{rc} 为:

$$t_{rc} = t_{rf} + t_{rp} \tag{21}$$

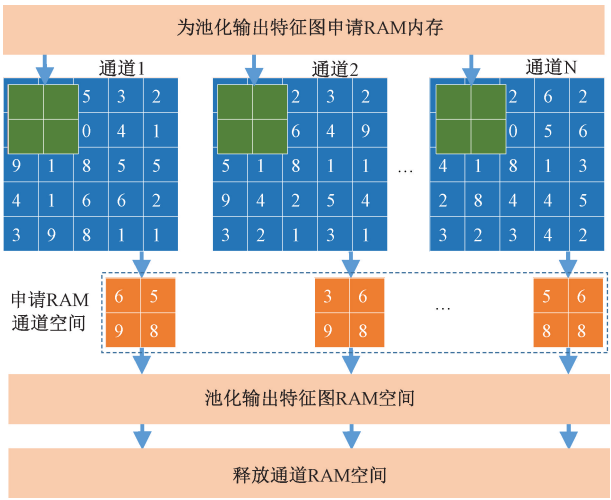


图 3　在 MCU 上进行 HD-CNN 池化计算

Fig. 3　The pooling computing of HD-CNN on MCU

在进行池化计算时,还需要为每个输出特征图单独申请 RAM,申请大小 t_{rl} 为:

$$t_{rl} = 4 \times w_{out} \times h_{out} \tag{22}$$

此时 t_r 为:

$$t_{rc} = t_{rf} + t_{rp} + t_{rl} \tag{23}$$

当池化操作结束后便可释放 t_{rf} 与 t_{rl},此时 t_{rc} 为:

$$t_{rc} = t_{rp} \tag{24}$$

式(10)~(24)描述了进行一层 CNN 运算时采用 HD 方法 RAM 占用大小 t_r 的变化过程,每层卷积运算都需要经历该过程,但由于每层卷积参数不同,因此 t_{rc} 最大值为任意层卷积 t_{rci} 层占用的 RAM,即:

$$t_{rc} = \max t_{rci} \tag{25}$$

同理全连接层初始阶段需要为权重参数 t_{ro} 与输出结果 t_{ru} 申请 RAM:

$$t_{ro} = 4 \times f_l \times o_l \tag{26}$$

$$t_{ru} = 4 \times o_l \tag{27}$$

此时,t_{rc} 为式(28)。

$$t_{rc} = \max t_{rci} + t_{ro} + t_{ru} \tag{28}$$

当模型运行完后只保留模型输出结果,此时可释放 t_{rci},t_{ro},t_{ri} 与 t_{rh},此时:

$$t_{rc} = t_{ru} \tag{29}$$

根据式(8)~(29)可以得出在 CNN 模型运行阶段 RAM 变化趋势,如图 4 所示展示了 3 层卷积与 1 层全连接的 RAM 变化趋势,可以看出 RAM 内存最高占用出现在进行卷积计算过程中。

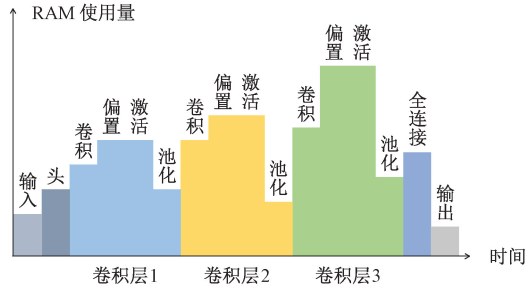


图 4　HD-CNN 运行过程中 RAM 占用变化趋势

Fig. 4　The trend of RAM usage during the HD-CNN running

1.3　权重与偏置参数搜寻

对每张特征图进行卷积运算时,特征图与卷积核的运算按照传统方式一般是以多维向量的方式存储在与读取。然而在资源受限的 MCU 上参数数据始终以一维形式保存在 Flash 中,若将其转换成多维数据进行计算,将大幅增加 RAM 使用,因此在进行 HD-CNN 运算时需要定位参数在 Flash 中位置并将参数读取出来,然后在特征图运算时按照一维数据定位相关参数。

假设参数文件存放在 Flash 中的初始位置为 a_b,第 i 层的卷积权重参数大小为 p_{si},偏置参数大小为 b_{si},则按照表 1 描述,文件头的起始位置也为 a_b,第 1 层卷积的权重参数 a_{p1} 与偏置参数起始位置分别为式(30)与(31),从第 2 层开始,第 i 层的卷积权重 a_{pi} 与偏置参数 a_{bi} 起始位置分为式(32)与(33)。其中,p_{sj} 与 b_{sj} 分别为第 j 层卷积的权重与偏置参数大小。

$$a_h = t_{fc} + t_{fuc} + t_{fub} \tag{30}$$

$$a_{b1} = a_h + p_{s1} \tag{31}$$

$$a_{pi} = a_h + \sum_{j=1}^{j=i-1} p_{sj} + \sum_{j=1}^{j=i-1} b_{sj}, i \geqslant 2 \tag{32}$$

$$a_{bi} = a_h + \sum_{j=1}^{j=i} p_{sj} + \sum_{j=1}^{j=i} b_{sj} - b_{s(i-1)}, i \geqslant 2 \tag{33}$$

由于池化与激活运算没有权重与偏置参数,所以不需要参数定位,CNN 最后一层为全连接,全连接的权重参数起始位置 a_f 为:

$$a_f = a_h + \sum_{i=1}^{i=l_n} p_{si} + \sum_{i=1}^{i=l_n} b_{si} \tag{34}$$

2 性能分析

2.1 RAM 与 Flash 分析

Flash 的占用主要包括文件头和所有卷积参数,即式(1)所述,t_f 必须满足:

$$t_f \leqslant F_s \times (1 - R_f), 0 < R_f < 1 \tag{35}$$

其中,F_s 为 MCU 内部 Flash 大小,具体取决于 MCU 型号。R_f 为保留因子,表示 MCU 程序指令所占 Flash 的百分比,具体取决于程序代码量。

通过图 4 可知,运行过程中最大的 RAM 占用取决于卷积层占用最大的 RAM,同时根据式(28)的推导过程结合(9)可得,最大的 RAM 使用量为:

$$t_{rc} = t_{rh} + \max t_{rci} \tag{36}$$

2.2 HD 方法运行效率

HD-CNN 运行时间主要包括数据采集时间、数据预处理时间与模型处理时间。由于数据采集与数据处理可以同时进行,因此时间效率主要是指在数据进行预处理到模型输出结果的时间内所处理的数据量,由于输入数据的大小一般可调节,因此运行效率 t_e 可以定义为单位时间内处理数据数据的速度。

$$t_e = d_s/(t_{STFT} + t_{HD-CNN}) \tag{37}$$

其中,t_{STFT} 为预处理时间,t_{HD-CNN} 为运行 HD-CNN 的时间。数据采集时间计算公式为式(38)。

$$t_{sample} = f_l/f_s \tag{38}$$

其中,f_l 为采样长度,f_s 为采样频率。

由于在模型训练与验证是在计算机的 CPU 或者 GPU 上,浮点数据类型一般为 64 位,计算精度较高。然而在 MCU 上浮点数据一般为 16 或者 32 位,计算精度降低,在 CNN 中间计算过程可能存在精度损失。因此需要验证单个输入数据模型输出结果是否与计算机一致。此外,还需要将在计算机上使用的测试集迁移至 MCU 上,进行总体识别准确率 a_r 验证。其中,N_t 为总的测试数据个数,N_c 为测试正确的个数。

$$a_r = N_c/N_t \times 100\% \tag{39}$$

2.3 MCU 时钟频率影响

不同的 MCU 拥有不同的主频率,不同的主频率将影响时间运行效率。一般来说,MCU 的主频主要分布在 20~550 MHz,主频越高运行 HD-CNN 速度越快,但 MCU 功耗也越高,因此需要根据运行时间、运行效率与 MCU 供电方式选择合适的 MCU 主频。

3 机械振动 WSN 边缘计算 HD-CNN 流程

如图 5 所示描述了采用所提 HD 方法实施的简要流程,主要包括 3 个阶段,详细描述如下。

1)数据预处理。首先利用 WSN 节点的加速度振动传感器采集原始数据,然后对原始数据归一化处理并进行短时傅里叶变换(short-time fourier transform, STFT),将一维振动数据转换成二维图像数据,接着将图像数据制作成训练数据集与测试数据集。在数据采集时,每个 WSN 节点负责采集一种轴承故障或传感器故障的样本,并在制作数据集时标记标签。

2)模型参数训练与部署。在进行模型训练前,需要构建 CNN 结构。考虑到 CNN 参数大且 MCU 硬件资源的限制,CNN 的输入输出特征通道数、卷积池化核大小以及网络深度需要着重设计。其中网络深度可以通过先设置较深网络,然后在保证识别精度的情况下逐层减少网络深度,以寻找到合适的网络深度。其余变量则可根据实验得出合适值。CNN 模型结构确立后,通过输入训练数据集便可训练权重参数与偏置参数,接着将训练好的满足测试精度需求的参数部署到 MCU 的 Flash 上。部署规则按照提前设定好的 Flash 地址进行参数写入,参数量不得超过 MCU 的 Flash 大小。此外,在一些应用场景中,可以通过外接内存拓展 Flash 以增加可存储的参数量。需要注意的是,Flash 还应当保留存储程序代码的空间,实际上这一部分空间非常小。在实际应用中,WSN 节点数量庞大,需要考虑算法部署。算法部署主要有两种,一种是将算法在训练完成时,将 MCU 程序与算法形成一个二进制文件通过烧录工具下载至 MCU 的 Flash 中,算法存放在 Flash 中的起始地址由烧录工具确定;另一种方法是在线部署,部署时网关节点根据目标采集节点的唯一编号与 Flash 地址向个目标节点发送算法文件。第一种方法的优点在于算法与应用程序在初始时统一部署在 MCU 上,节省部署时间;第二种方法可以在线更新模型,灵活度大。在实际应用中,可结合两种方法,动态更新模型。

3)模型运行。模型运行主要是指 HD-CNN 在 MCU 上运行以进行故障边缘识别。由于 CNN 每层网络的卷积操作、偏置计算以及激活运算在单核的 MCU 上只能串行运行,因此为尽可能地减少 RAM 内存使用,应当在需要的时候读取权重与偏置参数,并为之动态地申请与释放 RAM 内存。根据这一原则,将 CNN 模型进行层次分解,并按层次分解原则进行计算。

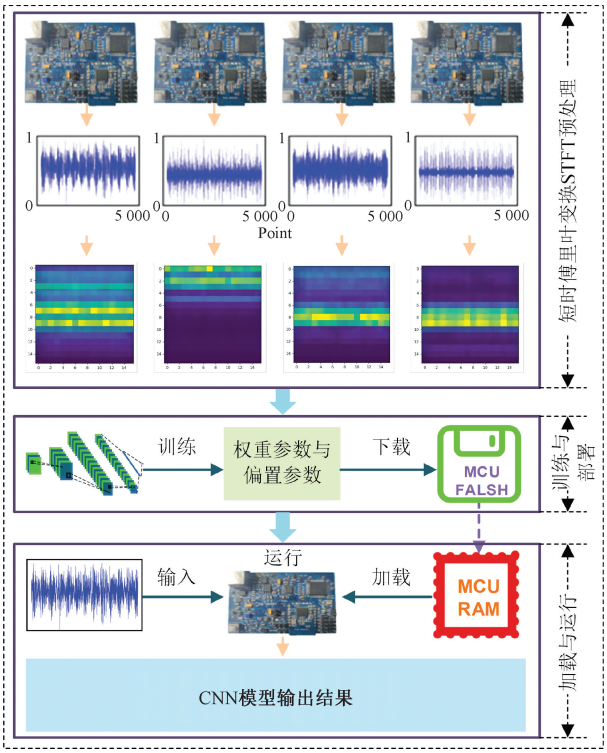


图 5　边缘计算 HD-CNN 流程

Fig. 5　The workflow of the edge computing HD-CNN

4　实验性能验证

4.1　实验环境设置

为验证本文所提 HD-CNN 方法在 MCU 上的边缘计算能力,设计以下实验,数据采集平台如图 6 所示,由齿轮箱实验台产生振动信号,采样率为 12 800 Hz,训练数据也由该实验台产生并由 WSN 节点采集,在实际应用中对实时采集的数据进行边缘计算处理。为增加训练数据的多样化与实验的可靠性,本文实验数据采用 2 种故障数据如图 7、8,现场采集的平行齿轮箱轴承故障数据与行星齿轮箱轴承故障数据。

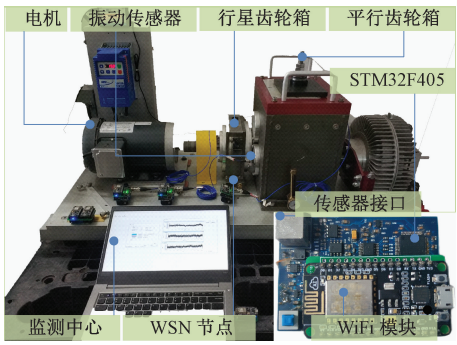


图 6　数据采集实验平台

Fig. 6　Experimental platform of collecting data

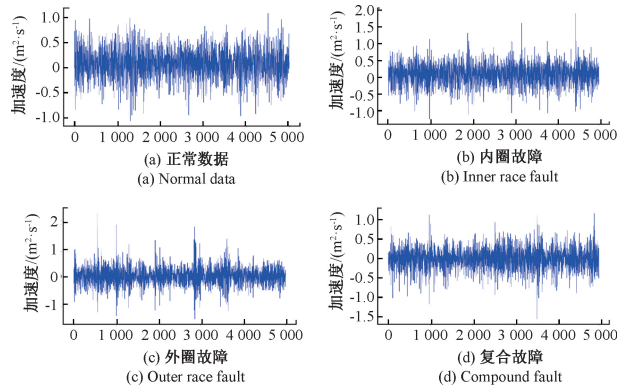


图 7　行星齿轮箱轴承故障数据

Fig. 7　Planetary gearbox bearing fault data

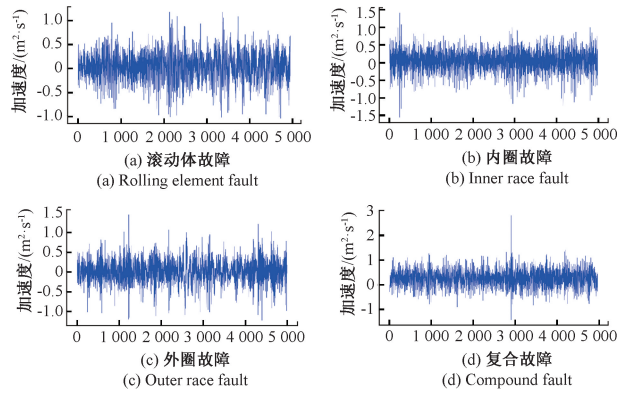


图 8　平行齿轮箱轴承故障数据

Fig. 8　Parallel gearbox bearing fault data

每种原始故障数据先划分成多段,每段数据在 STFT 后形成图像数据,在输入图像大小为 32×32 下,总共训练集为 9 894 个,测试集 2 474 个;在输入图像大小为 16×16 下,总共训练集为 9 971 个,测试集 2 493 个。训练 Epoch 为 100 次,Batch size 为 16。

其中 WSN 采集节点为自研,该节点采用的 MCU 型号为 STM32F405,并用 WiFi 作为数据传输。STM32F405 用于控制节点采集并进行 HD-CNN 运算,该 MCU 拥有 192 KB 的 RAM 内存与 1 MB 的 Flash 内存,其中 64 KB RAM 用于程序自动管理,128 KB RAM 用于运行 HD-CNN,本文 CNN 结构为 3 层卷积与 1 层全连接。探索在 MCU 上运行 HD-CNN 的性能主要从内存使用、MCU 主频与运行效率 3 个方面进行。

4.2　RAM 与 Flash 使用量

内存分为 Flash 内存与 RAM 内存,不同 MCU 具有不同的 Flash 与 RAM 大小,在部署 CNN 前需要通过式(35)评估 Flash 参数量,且在运行过程中最大 RAM 占用不得超过可用 RAM。设置不同的 HD-CNN 结构参数可实现不同的参数量。具体参数量设置如表 2,在实际

测试中,不合理的参数将会被丢弃,如每层输入特征图像小于核大小的情况。

表 2 HD-CNN 结构参数

Table 2 The parameters structure of the HD-CNN

参数	i_s	i_{d1}	o_{d1}	k_1	p_1	i_{d2}	o_{d2}	k_2	p_2	i_{d3}	o_{d3}	k_3
起始	16	1	4	2	2	4	4	2	2	1	2	2
结束	32	1	20	4	4	20	20	4	4	20	4	4
步长	16	0	8	1	1	8	8	1	1	8	1	1

为观察最大 RAM 与 Flash 使用之间的关系,分别以 Flash 与 RAM 大小为横纵坐标,将数据结果分别绘制成如下图 9 与 10。具体网络结构参数如表 3,其中 No. 1 结构简单,No. 2 与 No. 3 结构复杂。

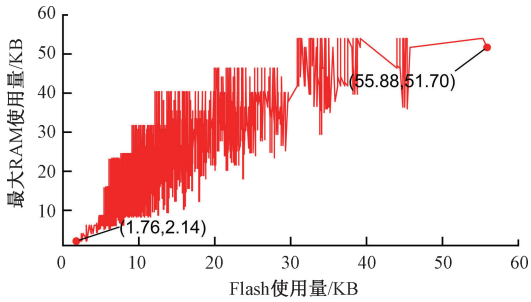


图 9 最大 RAM 占用量与 Flash 占用量

Fig. 9 The maximum RAM and Flash occupancy

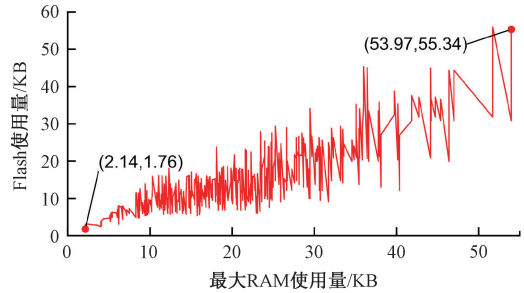


图 10 Flash 占用量随最大 RAM 占用的变化趋势

Fig. 10 The trend of Flash usage with the maximum RAM usage

从图 9 中可以看出,在 Flash 使用量从 1.76 KB 增加至 55.88 KB 过程中,最大的 RAM 使用量从 2.14 KB 增加到 51.70 KB。在 Flash 增加的过程中,由于参数的变化会导致最大 RAM 使用在局部出现波动,但整体趋势与 Flash 使用呈正相关。同样在图 10 中,当最大 RAM 使用为 2.14 KB 时,Flash 占用为 1.76 KB,RAM 使用为 53.97 KB 时,Flash 占用为 55.34 KB。Flash 与最大 RAM 整体呈正相关。

4.3 MCU 时钟频率分析

由于数据集的类型对 HD-CNN 运行时间与效率没有影响,因此本节以实验室自采集行星齿轮箱轴承故障数据为对象进行分析。MCU 的主频主要影响 CNN 的运行速度,部分 MCU 的主频仅有几 MHz,少部分可达 550 MHz,因此为探索 HD-CNN 在不同主频下的运行时间,将 MCU 从 STM32F405 替换成 STM32H723 以扩大最大主频。通过调整主频观察 HD-CNN 的运行时间。将表 3 中的 3 组作为 HD-CNN 结构参数进行实验,主频范围为 50~550 MHz,步长为 50 MHz。实验结果如图 11~12 所示。

表 3 图 9 与 10 的 HD-CNN 结构参数

Table 3 HD-CNN structures of Figs. 9 and 10

No.	i_s	i_{d1}	o_{d1}	k_1	p_1	i_{d2}	o_{d2}	k_2	p_2	i_{d3}	o_{d3}	k_3	p_3	*RAM*	*Flash*
1	16	1	4	2	2	4	4	2	2	4	4	2	2	1.76	2 014
2	32	1	20	4	2	20	20	4	2	20	20	4	2	51.70	55.88
3	32	1	20	3	2	20	20	4	2	20	20	4	3	53.97	55.34

如图 11 所示,从整体上看运行时间随着主频增加而减小。在主频小于 300 MHz 时,主频对运行时间影响较大,主频大于 300 MHz 时,增加主频对降低运行时间效果不明显。此外还可以看出,复杂的模型 No. 2 与 No. 3 相比于简单的模型 No. 1 具有更长的运行时间。如图 11(c)与图 12(c),当输入图像大小 16×16 时,采用较小 CNN 结构 No. 1,在主频为 50 MHz 时运行一次 STFT 与 HD-CNN 时间为 34.70 ms,运行一次整个流程为 117.2 ms;主频在 550 MHz 时,运行时间仅为 3.15 ms 与 85.65 ms。

如图 11(b)与 12(b),当输入图像大小 32×32 时,采用较复杂 CNN 结构,在主频为 50 MHz 时运行一次 STFT 与 HD-CNN 时间为 603.92 ms,运行一次整个流程为 686.42 ms;主频在 550 MHz 时,运行时间仅为 54.1 ms、54.9 ms 与 137.4 ms;在主频为较大 550 MHz 时,运行时间为 54.1 ms、54.9 ms 与 137.4 ms。当输入图像为 16×16 与 32×32,分别需要采样 272 与 1 056 个数据点,所需要的采样时间分别为 21.25 ms 与 82.5 ms。如图 11 所示,在主频为 550 MHz 下,运行简单与复杂 HD-CNN 模型时,STFT 与 HD-CNN 的所用总时间均小于采样时间 21.25 ms 与 82.5 ms,证明了在 550 MHz 下可以对数据进行实时处理。

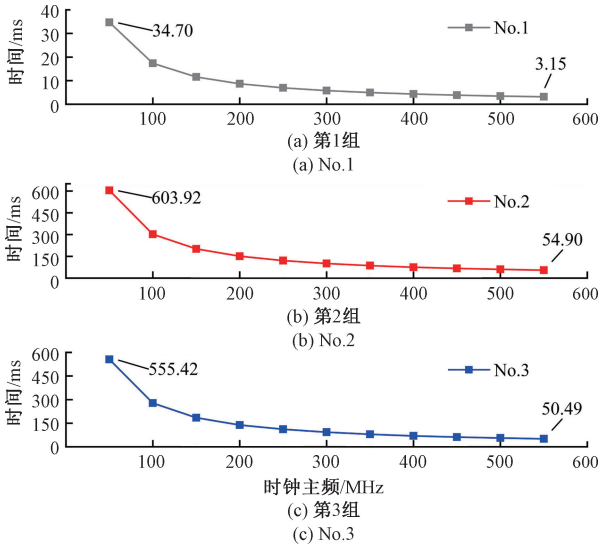


图 11　不同时钟频率运行 HD-CNN 与 STFT 总时间

Fig. 11　The HD-CNN and STFT runtime at different clock frequencies

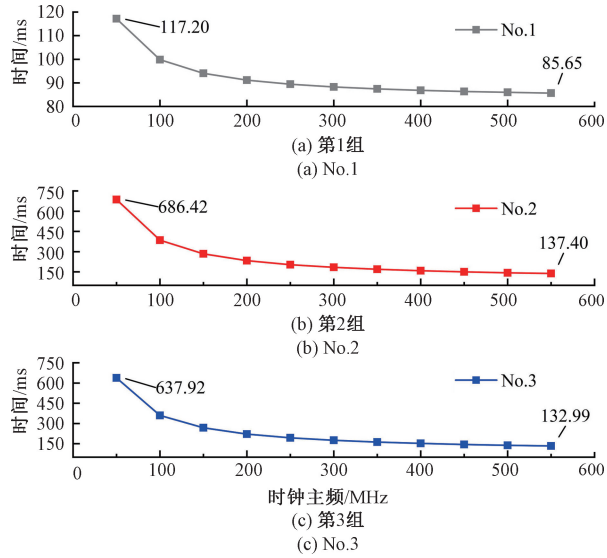


图 12　不同主频采样时间、运行 HD-CNN 与 STFT 的总时间

Fig. 12　The sample timeand running time of HD-CNN and STFT at different clock frequencies

4.4　运行效率分析

HD-CNN 结构参数如表 4,其中 No. 1 与 No. 3 HD-CNN 结构简单,No. 2 与 No. 4 结构复杂。

不同 HD-CNN 结构参数、输入长度与主频会影响运行效率。图 13 列出了 4 种参数下的运行效率,其中输入长度为 16×16 与 32×32,主频为 50～550 MHz,步长为 50 MHz。

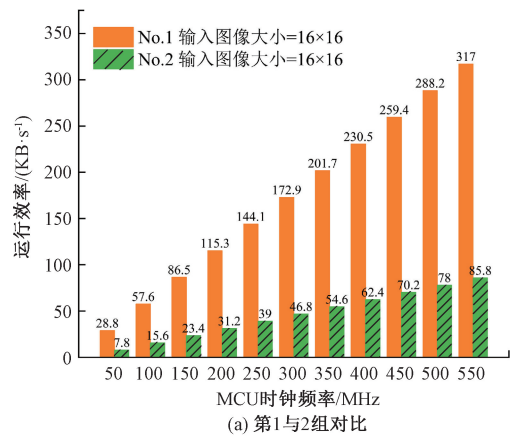
表 4　不同复杂程度的 HD-CNN 结构参数

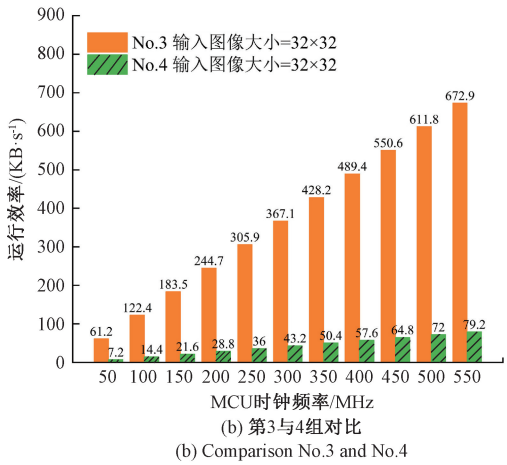
Table 4　HD-CNN structures with different complexity levels

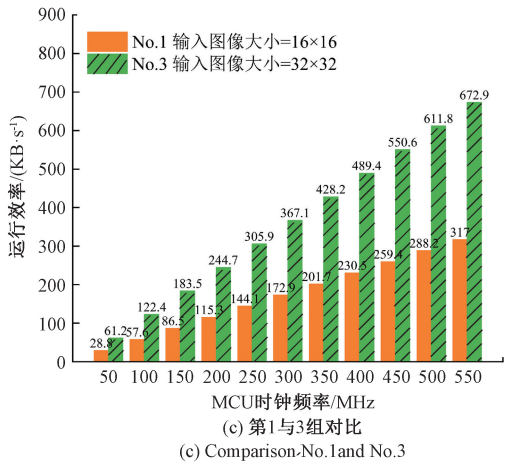
No	i_s	i_{d1}	o_{d1}	k_1	p_1	i_{d2}	o_{d2}	k_2	p_2	i_{d3}	o_{d3}	k_3	p_3	*RAM*	*Flash*
1	16	1	4	2	2	4	4	2	2	4	4	2	2	1.76	2.14
2	16	1	20	3	2	20	20	2	2	20	20	2	2	13.51	14.84
3	32	1	4	2	2	4	4	2	2	4	4	2	2	8.57	5.26
4	32	1	20	4	2	20	20	4	2	20	20	4	2	51.70	55.88

从图 13(a)与(b)都可以看出,在相同输入图像大小下,HD-CNN 模型越简单,数据处理越快,运行效率也越高,简单模型处理相同输入图像的运行效率是复杂模型的至少 3 倍,这是因为简单模型参数量较少,相应的运算量也较少。从图 13(c)中可以看出,对于相同小模型,输入图像越大,单位时间内处理数据量越多,运行效率越高,最高的运行效率可达 672.9 KB/s,这是因为原始数据进行 STFT 预处理所花费时间较多,而运行模型时间相差不大。从图 13(d)中可以看出,复杂模型即使在输入图像大小较大的情况下,运行效率也相差不大,这是因为运行复杂模型所消耗的时间占比较多。上述模型测试精度均达到 100%。

内存方面,模型在达到 100% 测试精度的条件下,输入图像大小为 16×16 时,最大 RAM 占用仅为 1.76 KB,Flash 占用仅为 2.14 KB;输入图像大小为 32×32 时,最大 RAM 使用仅为 8.57 KB,Flash 使用仅为 5.26 KB。在







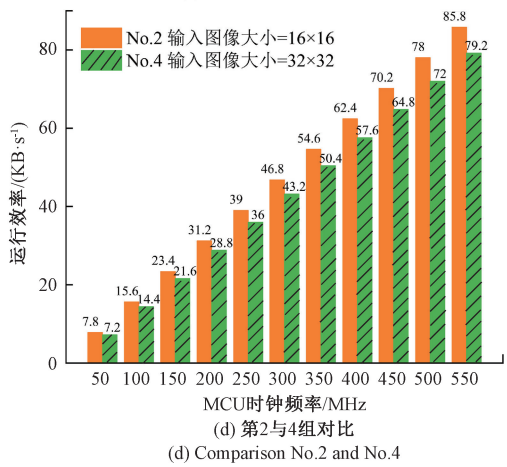


图 13 不同 HD-CNN 参数与输入图像大小下的运行效率

Fig. 13 The running efficiency under various HD-CNN parameters and input images size

采用复杂模型时,最大 RAM 使用最多 51.70 KB,最大 Flash 使用最多为 55.88 KB。大部分 MCU 的 RAM 内存在 64 KB 以上,Flash 在 128 KB 以上,因此不论 HD-CNN 模型简单与复杂,均可在大部分资源受限的 MCU 上运行。

4.5 模型性能比较

为比较不同深度学习模型在故障边缘识别中的性能,将本文提出的 HD-CNN 与 HD-DNN 网络模型、最新文献[20-21]以及其它先进网络模型进行比较,如表 5 所示。在 DNN 模型中,将 DNN 采用 HD 方法进行实现。在文献[20]中,采用的主频可达 800 MHz,RAM 可达 8 GB 的 STM32MP157 处理器;在文献[20]中的 BCM2711B0 主频可达 1.5 GHz,运行内存可达 4 GB 处理器;本文采用的 MCU 主频最高仅为 550 MHz,运行内存仅为 320 KB,运行内存远远小于文献[20]与[21]。文献[20]同时将其它模型运行在 BCM2711B0 上并观察了效果。

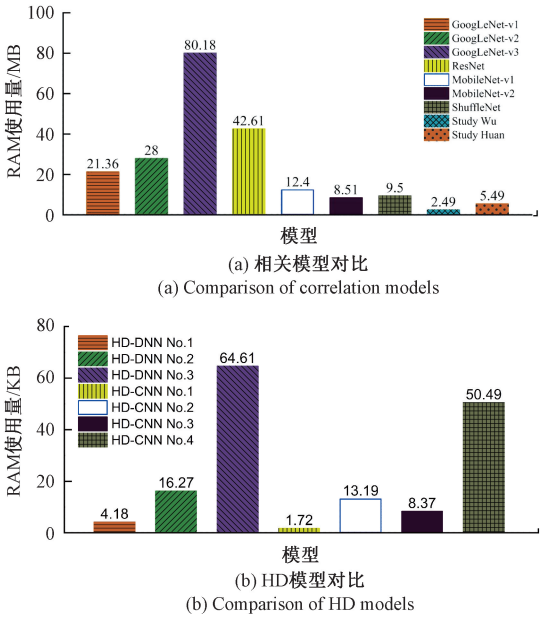
表 5 不同模型的性能比较

Table 5 The performance comparison of different models

模型	F_r /GHz	R_s /KB	R_u /MB	F_u /MB	I_s /KB	A_c on MCU/%
Study[Wu]	1.5	8 192	0.35	0.35	16	98.32
GoogLeNet-v1	1.5	8 192	5.60	5.60	16	96.08
GoogLeNet-v2	1.5	8 192	7.34	7.34	16	88.75
GoogLeNet-v3	1.5	8 192	21.02	21.02	16	92.62
ResNet	1.5	8 192	11.17	11.17	16	97.45
MobileNet-v1	1.5	8 192	3.25	3.25	16	90.80
MobileNet-v2	1.5	8 192	2.23	2.23	16	95.49
ShuffleNet	1.5	8 192	2.49	2.49	16	94.74
Study[Huan]	0.8	4 096	5.49	5.49	160	98.92
HD-DNN No. 1	0.147	192	0.004 2	0.002 9	0.125	99.62
HD-DNN No. 2	0.147	192	0.016 3	0.040 3	0.5	100
HD-DNN No. 3	0.147	192	0.064 6	0.190 3	2	100
HD-CNN (Max)	0.550	320	0.052 7	0.054 0	4	100

图 14、15 展示了不同模型的内存使用情况,从图 14(a)中可以看出,在完成相同故障分类任务下,文献[21]的 RAM 使用最少,仅为 0.35 MB,其余模型均在 2.23 MB 以上,考虑到较少的 MCU 拥有 1 MB 以上的内置 RAM,因此,其他模型难以运行在 MCU 上。从图 14(b)中可以看出采用 HD 的 DNN 最小 RAM 使用量为 4.18 KB。最大为 64.61 KB。更进一步地,在完成相同故障分类任务下,采用 HD 方法的 CNN 运行时 RAM 使用量最少仅为 1.72 KB,最大为 50.49 KB。综上,无论 DNN 还是 CNN,采用 HD 方法完全可以在绝大多数 MCU 上运行。

运行效率是衡量模型处理输入数据快慢的指标,在部分工况中需要及时获取输出状态,以便系统做相关响应,其计算公式如式(37)所示。考虑到除 HD-DNN 与 HD-CNN 以及 Study[Wu]的其它模型不能运行在 MCU 上,因此参考文献[20]中将其他模型运行在 BCM2711B0 上,并将每个模型的相关参数绘制成表 4。从图 16 与表 5 中可以得知,在处理器频率为 MCU 频率的 10 倍 150 MHz 左右时,HD-CNN No. 3 中其运行效率也达到了

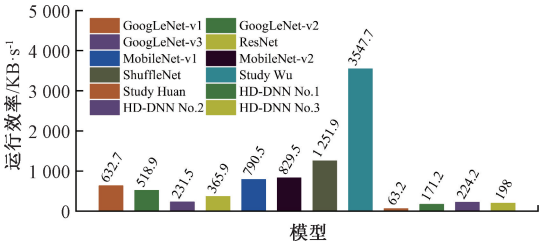


(a) 相关模型对比
(a) Comparison of correlation models

(b) HD模型对比
(b) Comparison of HD models

图 14　不同模型的 RAM 内存使用

Fig. 14　The RAM memory usage of different models



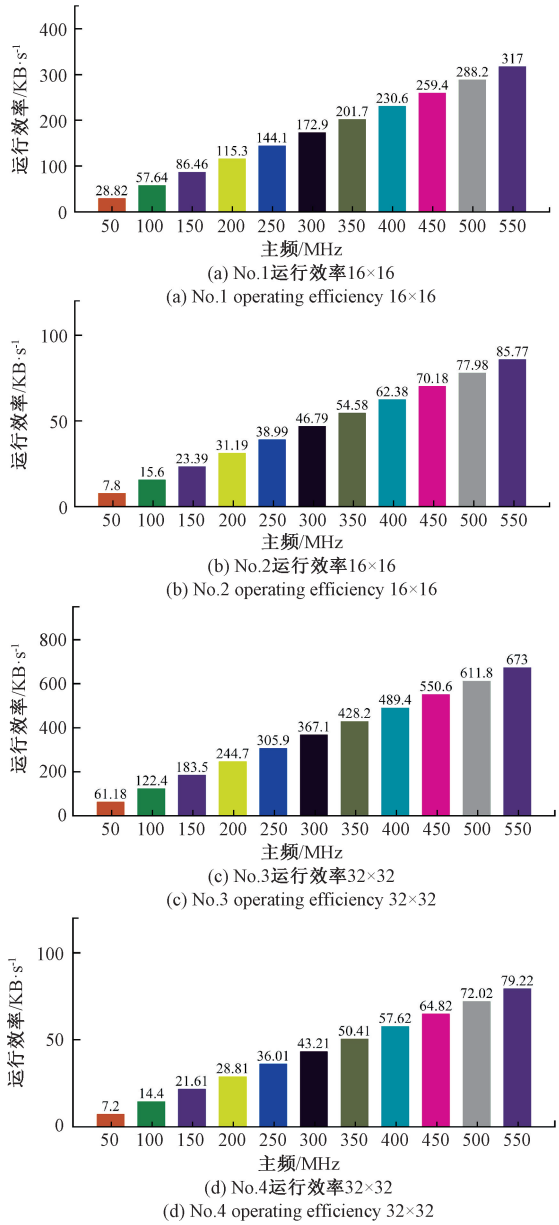
注:图中从左至右展示的模型依次为:GooLeNet-v1,GooLeNet-v2,GooLeNet-v3,ResNet,MobileNet-v1,MobileNet-v2,Shuttle Net,Study Wu,Study Huan,HD-DNN No. 1,HD-DNN No. 2,HD-DNN No. 3

图 15　不同模型运行性效率

Fig. 15　The running efficiency of different models

183. 5 KB/s;在主频为 550 MHz 下,运行效率也达到了 673 KB/s,超越了绝大部分模型的运行效率。若 HD-CNN 采用相同的时钟频率频率,将大幅提高其运行效率。

故障分类识别的准确率也是评判模型性能的重要指标,由于本研究所采用的 HD-CNN 未对模型本身结构进行任何修改,只是针对运行过程在算法层面进行了设计,因此在 MCU 上运行的算法除浮点数据位数不一致以外,其它无任何区别。在计算机上训练时模型以 64 位浮点型表示,在 MCU 上以 32 位浮点型表示,中间计算过程中可能会带来一定误差,然而根据实验,在进行故障分类任务时,与计算机 CPU 或 GPU 上测试精度保持一致。图 17 展示了不同模型进行分类任务的识别精度。



(a) No.1运行效率16×16
(a) No.1 operating efficiency 16×16

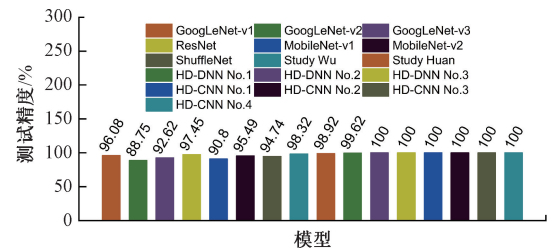
(b) No.2运行效率16×16
(b) No.2 operating efficiency 16×16

(c) No.3运行效率32×32
(c) No.3 operating efficiency 32×32

(d) No.4运行效率32×32
(d) No.4 operating efficiency 32×32

图 16　HD-CNN 运行性效率

Fig. 16　The running efficiency of HD-CNN



注:图中从左至右展示的模型依次为:GooLeNet-v1,GooLeNet-v2,GooLeNet-v3,ResNet,MobileNet-v1,MobileNet-v2,ShuttleNet,Study Wu,Study Huan,HD-DNN No. 1,HD-DNN No. 2,HD-DNN No. 3,HD-CNN No. 1,HD-CNN No. 2,HD-CNN No. 3,HD-CNN No. 4

图 17　不同模型在边缘设备上的测试精度

Fig. 17　The testing accuracy of different models on edge devices

5 结 论

本文通过所提 HD 方法对 MCU 的计算能力进行了增强,该方法可应用在资源受限但需要进行计算能力增强的边缘设备上。在针对 HD-CNN 相关性能实验中,讨论了 RAM 与 Flash 的使用量,主频对 HD-CNN 运行时间的影响以及不同 HD-CNN 模型对运行效率的影响。在仅使用 1.76 KB RAM 与 2.14 KB Flash 的情况下,便可实现边缘计算故障识别任务;在主频为 550 MHz 下,仅用 3.15 ms 下便可运行完一次 STFT 预处理与 HD-CNN,即使主频降低到 50 MHz 时,也仅需 34.7 ms。在 550 MHz 下运行效率最高可达 673 KB/s,远大于自研采集节点最高 51.2 KB/s 的采样率,可实现数据边缘实时处理。本文设计的 HD-CNN 在读取权重参数时是将整个滤波器的权重参数读取出来,这将导致 RAM 使用增加,在这一环节中还可每次只读取每个滤波器的权重参数来进一步节省 RAM 内存。此外,还可将其它如参数量化、蒸馏与剪枝等方法与 HD 方法结合,进一步降低 RAM 使用。

参考文献

[1] 朱亮亮,汤宝平,黄艺,等. 机械振动无线传感器网络大量数据传输功率优化控制方法[J]. 振动与冲击,2020,39(17):275-280.
ZHU L L, TANG B P, HUANG Y, et al. Transmit power optimization control method for a mechanical vibration wireless sensor network with a lot of transmission data[J]. Vibration and Shock,2020,39(17):275-280.

[2] 肖鑫,汤宝平,邓蕾,等. 机械振动无线传感器网络跨层同步采集进度控制方法[J]. 机械工程学报,2019,55(15):202-207.
XIAO X, TANG B P, DENG L, et al. Accumulated synchronous acquisition error control method based on cross-layer design for mechanical vibration wireless sensor networks[J]. Chinese Journal of Mechanical Engineering,2019,55(15):202-207.

[3] 孙春霞,杨丽,王小鹏,等. 结合深度强化学习的边缘计算网络服务功能链时延优化部署方法[J/OL]. 电子与信息学报,1-10[2024-03-26]. http://kns.cnki.net/kcms/detail/11.4494.TN.20231113.1721.004.html.
SUN CH X, YANG L, WANG X P, et al. Optimized Deployment method of edge computing network service function chain delay combined with deep reinforcement learning[J/OL]. Journal of Electronics and Information, 1-10[2024-03-26]. http://kns.cnki.net/kcms/detail/11.4494.TN.20231113.1721.004.html.

[4] 薛建强,史彦军,李波. 面向无人集群的边缘计算技术综述[J]. 兵工学报,2023,44(9):2546-2555.
XUE J Q, SHI Y J, LI B. Overview of edge computing technology for unmanned cluster[J]. Acta Ordnance Engineering,2023,44(9):2546-2555.

[5] 孙兵,刘艳,王田,等. 移动边缘网络中联邦学习效率优化综述[J]. 计算机研究与发,2022,59(7):1439-1469.
SUN B, LIU Y, WANG T, et al. Review on optimization of federated learning efficiency in mobile edge networks[J]. Journal of Computer Research and Development,2022,59(7):1439-1469.

[6] 刘培开,李博强,秦亮,等. 深度学习目标检测算法在架空输出电路绝缘子缺陷检测中的应用研究综述[J]. 高电压技术,2023,49(9):3584-3595.
LIU P K, LI B Q, QIN L, et al. Application of deep learning object detection algorithm in insulator defect detection of overhead output circuits[J]. High Voltage Technology, 2019,49(9):3584-3595.

[7] HUANG T, ZHANG Q, TANG X, et al. A novel fault diagnosis method based on CNN and LSTM and its application in fault diagnosis for complex systems[J]. Artificial Intelligence Review, 2022,55:1289-1315.

[8] CHOUDHARY A, MISHRA R K, FATIMA S, et al. Multi-input CNN based vibro-acoustic fusion for accurate fault diagnosis of induction motor[J]. Engineering Applications of Artificial Intelligence, 2023, 120: 105872.

[9] SUN H, ZHAO Z, FU X, et al. Limited feedback double directional massive MIMO channel estimation: From low-rank modeling to deep learning[C]. 2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), IEEE, 2018: 1-5.

[10] CAI G, LI J, LIU X, et al. Learning and compressing: low-rank matrix factorization for deep neural network compression[J]. Applied Sciences, 2023, 13(4): 2704.

[11] KULKARNI U, HOSAMANI A S, MASUR A S, et al. A survey on quantization methods for optimization of deep neural networks[C]. 2022 International Conference on Automation, Computing and Renewable Systems (ICACRS), IEEE, 2022: 827-834.

[12] GONG C, LI T, LU Y, et al. An ultra-low loss quantization method for DNN compression[C]. 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, 2019: 1-8.

[13] JIN H, WU D, ZHANG S, et al. Design of a quantization-based DNN delta compression framework for

model snapshots and federated learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(3): 923-937.

[14] WANG Y, QIN Y, LIU L, et al. HPPU: An energy-efficient sparse DNN training processor with hybrid weight pruning[C]. 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), IEEE, 2021: 1-4.

[15] KIM S, LEE J, KANG S, et al. A 146.52 TOPS/W deep-neural-network learning processor with stochastic coarse-fine pruning and adaptive input/output/weight skipping[C]. 2020 IEEE Symposium on VLSI Circuits, IEEE, 2020: 1-2.

[16] WANG Z, LI C, WANG X. Convolutional neural network pruning with structural redundancy reduction[C]. Proceedings of the IEEE/CVF Conference On Computer Vision and Pattern Recognition, 2021: 14913-14922.

[17] FAN H, MU J, ZHANG W. Bayesian optimization with clustering and rollback for cnn auto pruning[C]. European Conference on Computer Vision. Cham: Springer Nature Switzerland, 2022: 494-511.

[18] SAILESH M, SELVAKUMAR K, PRASANTH N. A novel framework for deployment of CNN models using post-training quantization on microcontroller[J]. Microprocessors and Microsystems, 2022, 94: 104634.

[19] LAI Y K, HWANG Y H. FPGA-based depth separable convolution neural network[C]. 2020 IEEE International Conference on Consumer Electronics (ICCE), IEEE, 2020: 1-2.

[20] WU Y, TANG B, DENG L, et al. Hardware-resource-constrained neural architecture search for edge-side fault diagnosis of wind-turbine gearboxes[J]. IEEE Transactions on Industrial Electronics, 2023:1-11.

[21] HUAN L, TANG B, ZHAO C. Global composite compression of deep neural network in wireless sensor networks for edge intelligent fault diagnosis[J]. IEEE Sensors Journal, 2023, 23(16):19968-19978.

[22] FU L, YAN K, ZHANG Y, et al. EdgeCog: A real-time bearing fault diagnosis system based on lightweight edge computing[J]. IEEE Transactions on Instrumentation and Measurement, 2023, 27, DOI: 10.1109/TIM.2023.3298403.

作者简介

付豪,2020 年重庆大学获得硕士学位。2020 年 7 月至 2021 年 7 月期间担任芯片验证工程师。目前,他正在重庆大学攻读博士学位。主要研究方向为边缘计算、无线传感器网络、物联网与深度学习。

E-mail: fuhowe@qq.com

Fu Hao obtained his M. Sc. degree from Chongqing University in 2020. From July 2020 to July 2021, he worked as an engineer in the field of chip verify. Currently, he is pursuing a Ph. D. at Chongqing University and Nanyang Technological University. His primary research interests include edge computing, wireless sensor networks, the internet of things, and deep learning.

邓蕾(通信作者),2010 年获得重庆大学博士学位,2014 年至 2015 年在宾夕法尼亚州立大学访问学者,重庆大学智能制造与工业工程系副教授。主要研究方向智能运维、调度优化决策与无线传感器网络。

E-mail: denglei@cqu.edu.cn

Deng Lei (Corresponding author) obtained her Ph. D. degree from Chongqing University in 2010. From 2014 to 2015, she served as a visiting scholar at Pennsylvania State University. She is currently an associate professor in the Department of Intelligent Manufacturing and Industrial Engineering at Chongqing University. Her primary research focuses on intelligent maintenance, scheduling optimization decisions, and wireless sensor networks.