**本科毕业论文（设计、创作）**

**题　　目：　多边缘设备协同推理的时延优化研究**

**多边缘设备协同推理的时延优化研究**

**摘要**

随着深度神经网络研究热度的不断增涨，越来越多的深度神经网络技术被应用到我们的生活领域的各个方面，如人脸识别、车牌识别、智能交通等。然而这些神经网络模型的推理往往会占用很大的计算资源和存储资源，因而无法直接部署在很多边缘设备上并有效执行，例如手机、树莓派等，通常需要较长的时间才能够完成一次推理，这在一些实时应用上无法达到预期要求，如无人驾驶、智慧安防等。因此，本设计的目的是寻找一种协同推理方式，将需要较长推理的模型计算任务分配到多个设备上，进行协同处理加速。本设计的主要工作在于根据卷积层的特性将卷积神经网络的卷积层推理任务进行拆分，分配到多个设备上进行并发处理，从而加速模型的推理。

**关键词：神经网络；VGG16；协同推理；gRPC**

**Research on delay optimization of multi edge device cooperative reasoning**

**abstract**

With the continuous increase in the popularity of deep neural network research, more and more deep neural network technology has been applied to all aspects of our lives, such as face recognition, license plate recognition, intelligent transportation, etc. However, the inference of these neural network models often occupies a lot of computing resources and storage resources, so it cannot be directly deployed on many edge devices and effectively executed, such as mobile phones, Raspberry Pi, etc., usually takes a long time to complete a reasoning, which can not meet the expected requirements in some real-time applications, such as unmanned driving, smart security, etc. Therefore, the purpose of this design is to find a collaborative inference method that distributes model computing tasks that require longer inference to multiple devices for collaborative processing acceleration. The main work of this design is to split the convolutional layer inference task of the convolutional neural network according to the characteristics of the convolutional layer, and assign it to multiple devices for concurrent processing, thereby accelerating the inference of the model.

**Keywords: convolutional neural network;pythonVGG16;Edge equipment; gRPC**

**目 录**

[第一章 绪论 1](#_Toc27274)

[1.2 问题的提出与研究目的 1](#_Toc29490)

[1.3 本文工作 1](#_Toc10418)

[1.4 研究方法和手段 1](#_Toc21817)

[1.4.1文献调查法 2](#_Toc30270)

[1.4.2实验法 2](#_Toc13907)

[1.5相关开发技术 2](#_Toc2006)

[第二章 卷积神经网络算法基础 3](#_Toc11385)

[2.1卷积神经网络-VGG16 3](#_Toc18362)

[2.2 VGG16卷积神经网络推理拆分 4](#_Toc7280)

[第三章 系统设计 7](#_Toc9490)

[3.1系统实现框架 7](#_Toc18506)

[3.1.1主设备 8](#_Toc29430)

[3.1.2分设备 8](#_Toc4843)

[3.2 模型的搭建和训练 9](#_Toc31825)

[3.2.1模型搭建 9](#_Toc10429)

[3.2.2模型训练 9](#_Toc23635)

[3.3 rpc模块的搭建 10](#_Toc32500)

[3.3.1设计数据格式 10](#_Toc26389)

[3.3.2编写服务端 11](#_Toc26135)

[3.3.3编写客户端 11](#_Toc3212)

[3.4拆分算法的实现 11](#_Toc4522)

[3.4.1张量拆分/合并模块 11](#_Toc12302)

[3.4.2并发请求模块 12](#_Toc17949)

[3.4.3主函数模块 12](#_Toc234)

[第四章 系统评估 13](#_Toc10114)

[4.1 单元测试 13](#_Toc1259)

[4.1.1模型测试 13](#_Toc13967)

[4.1.2 rpc模块测试 14](#_Toc19384)

[4.2综合测试 16](#_Toc14061)

[4.2.1 准确率测试 16](#_Toc14960)

[4.2.2 延时优化测试 17](#_Toc20723)

[第五章 结束语 18](#_Toc6391)

[主要参考文献 19](#_Toc4661)

[致谢 20](#_Toc5833)

**第一章 绪论**

* 1. **课题的研究背景及意义**

深度神经网络（DNN）已成为支持各种计算机视觉和自然语言处理应用的最重要技术。但是随着需求的不断增长，DNN 模型的大小和复杂性明显呈上升趋势。这会产生繁重的计算工作量和大量的资源消耗。对于边缘设备（例如嵌入式主板和智能手机）来说，它们处理能力和内存空间有限，这样庞大资源需求有些遥不可及。如果设备上的 DNN 模型需要很长时间才能获得一个图像的分类结果 ，这在实际应用中通常是无法忍受的。因此，为了解决上述挑战，本文尝试通过在边缘设备集群上以本地分布式方式执行未修改的DNN模型的推理来解决边缘设备模型处理延时较高的情况。

* 1. **问题的提出与研究目的**

近年来，研究人员提出了许多方法来缩小DNN模型的资源需求与边缘设备的资源约束之间的差距。主要可以分为两类，即模型压缩和云计算。DNN压缩技术是通过权重修剪，知识提炼或紧凑的架构设计来减少于运算的量级和模型大小。然而，这需要复杂的预处理来消除模型冗余，并且不可避免地会导致精度损失，准确度的下降。云计算方案则是将 DNN 模型划分，并将计算密集型部分转移到云服务器，从而减少边缘设备的计算量。然而，推理性能高度依赖于不可预测的网络条件。同时可能会造成隐私数据的泄露，因此，在本地设备资源有限的情况下，研究多边缘设备协同推理的时延优化具有重要的研究价值和实用价值。

* 1. **本文工作**

本项目的研究目标是尝试将一个完整未删减的模型推理任务分配到多个设备上进行协同推理，这样的好处在于可以充分应用边缘资源，同时将本应该由一个设备执行的任务划分为多个可以快速并发执行的小任务，并在执行完成之后合并返回到主设备，从而减少推理所需要的总时间。以上过程主要包括模型拆分算法的研究、多设备之间的网络并发交互，模型的训练以及推理。

* 1. **研究方法和手段**

**1.4.1文献调查法**

查找相关文献，了解现阶段该领域的先进算法，仔细研究论文中高准确率的算法并分析该算法的理论依据。

**1.4.2实验法**

在对课题的研究内容进行了详细的了解之后，使用相关的机器学习算法，训练一个模型，并尝试将模型分割。

**1.5相关开发技术**

本设计是要实现将一个卷积神经网络拆分，将其分配到多个从设备上进行计算，再将结果集中返回到主设备进行合并和最终处理。基于这样的要求，本文就可以选择一些在开发中会使用到的工具：主编程语言使用Python，卷积神经网络模型选择使用结构较为简单但不失准确度的VGG16，主设备和从设备之间的交互使用gRPC框架进行交互。

（1）Python

Python是一种广泛应用的高级语言，具有可移植，面向对象，可拓展，简洁易读，程序结构清晰明了等特点，同时，python自身集成了许多机器学习库，例如tensorflow，pytorch等，可以较为灵活的实现当前应用广泛的神经网络模型。本设计主要使用python语言实现

（2）Pytorch

Pytorch是一个开源的Python机器学习库，不仅能够实现强大的GPU加速，同时还支持动态神经网络。本设计使用的卷积神经网络由Pytorch搭建

（3）VGG16

VGG是由Simonyan 和Zisserman提出的卷积神经网络模型，在图像识别上有着很高的准确率，同时，所有卷积层只使用了的卷积核，使得它的结构非常简单但又有着足够的深度，本设计使用的VGG16模型是其中具有13层卷积层的模型结构。

（4）gRPC

gRPC是一个高性能、开源和通用的 RPC 框架，基于ProtoBuf(Protocol Buffers) 序列化协议开发，且支持众多开发语言。面向服务端和移动端，基于 HTTP/2 设计，带来诸如双向流、流控、头部压缩、单TCP连接上的多复用请求等特。这些特性使得其在移动设备上表现更好，更省电和节省空间占用，本设计主要通过gRPC框架来实现主设备和从设备之间的信息交互。

1. **卷积神经网络算法基础**

卷积神经网络是一类包含卷积计算且具有深度结构的前馈神经网络，是深度学习的代表算法之一，本设计需要将该模型实现并发拆分，势必需要对卷积神经网络的模型做深度的了解和分析。

**2.1卷积神经网络-VGG16**

一个卷积神经网络主要由卷积层，池化层，全连接层组成，其对应的概念如下：

（1）卷积：对图像元素的矩阵变换，本质是矩阵相乘，目的是提取对应图像的图像特征，可以使用多种卷积核提取多种特征。一个卷积核覆盖的原始图像的范围叫做感受野（权值共享）。然而一次卷积运算往往只能够提取局部的特征，难以提取出比较全局的特征，因此需要在一层卷积基础上继续做卷积计算 ，这也就是多层卷积。

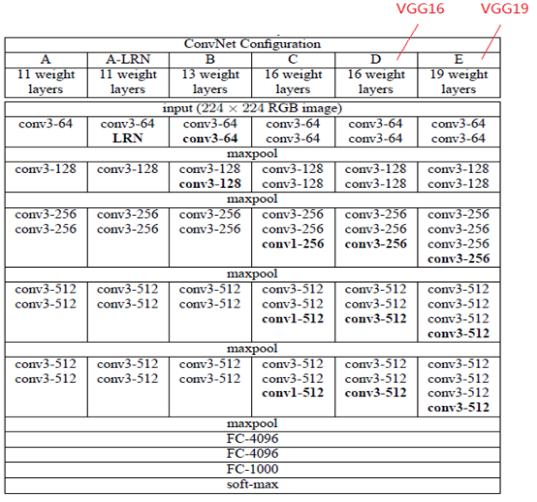


图 1: VGG模型结构

（2）池化：是卷积神经网络用于降维的方法，卷积计算往往会得出过大的特征向量维度，这不仅会对后续的工作产生非常大的计算量，而且容易出现过拟合的情况，为了解决这种情况，人们提出了池化的概念，其核心就是把图像中的局部区域特征做一个平滑压缩处理（极值、均值），让模型尽量“泛化”（“模糊”）。

（3）全连接：该层也可称为softmax层（softmax layer），全连接层中的每个神经元与其前一层的所有神经元全连接，用于整合卷积层或者池化层中具有类别区分性的局部信息，同时一般采用ReLU（线性整流）函数来提升性能，最后采用softmax逻辑回归进行分类。

VGG16卷积神经网络主要由上述三种类型组成，其官方给出的结构如图1所示。

**2.2 VGG16卷积神经网络推理拆分**

在上述模型介绍中，不难看出VGG16神经网络模型由13层卷积层，5层池化层，3层全连接层组成。其中卷积层占用了推理过程中最大的时间（接近90%），因此，一个较为直观的想法就是将卷积层的运算分配到同一网络的多个设备上，进行协同处理加速。基于卷积运算的特性，本文可以将原始卷积层划分为多个可独立执行的部分，分配到多个设备上进行执行，再在下一层执行前将结果合并，如图2所示。

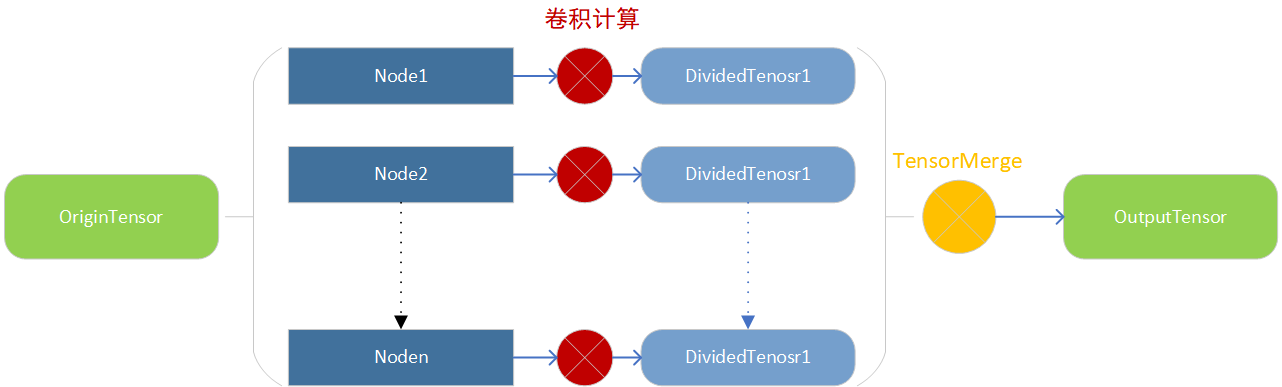


图 2：卷积层并发处理流程

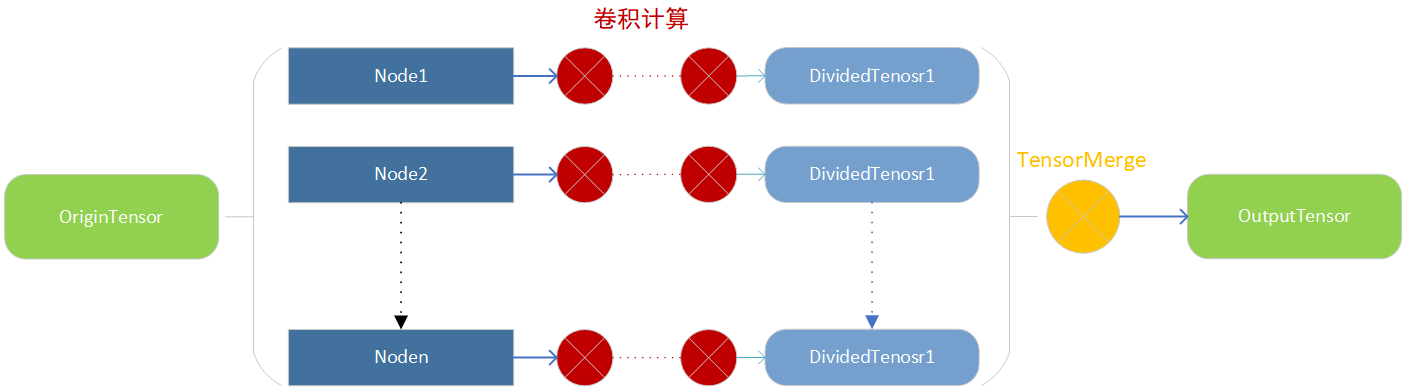


图 3：卷积块并发处理流程

再进一步，由于在分配任务时，需要将原始特征输入划分到多个设备上，再收集汇总到主设备上，这代表一层卷积运算会经过两次网络I/O，这个成本相对来说就会偏高，因此，更好的方案是将连续的卷积层作为计算单位，将其转移到分设备上进行运算，再将整体的计算结果进行返回拼装，如图3所示

但是在这种情况下会产生一个边缘问题，即处于划分边缘的特征向量在卷积计算时，会无法获取当相邻的元素用于计算下一层的特征值，即图4所示情况。

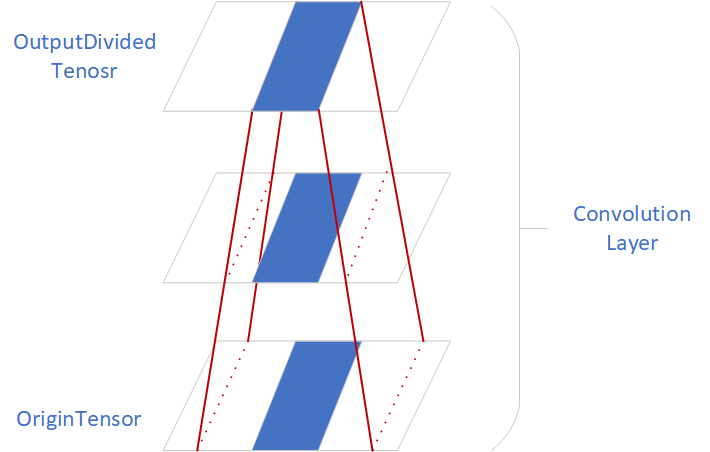


图 4：卷积层拆分的边缘效应

为了解决这个问题，本文需要在划分特征向量时，需要将本块所需要涉及到的其它维度信息合并传输至分设备，虽然这一部分会产生额外的计算开销，但是相比与提升的计算性能来说，这些成本时可以忽略不计。同时，由于在本设计中使用的VGG16模型所采用的卷积核都是的矩阵，这意味这对于当前卷积层对应元素，它的特征值是仅由上一层的相邻元素决定的，公式如下：

|  |  |  |
| --- | --- | --- |
|  |  |  |

由此得知，对于VGG16模型，假设这次计算包括了k层卷积层，那么本文只需要将计算所需的特征向量边界外k层一同放入本次任务，即可保证最后计算结果的局部正确性，再将所有分设备的计算结果合并，便得到正确的计算结果。如图5所示。

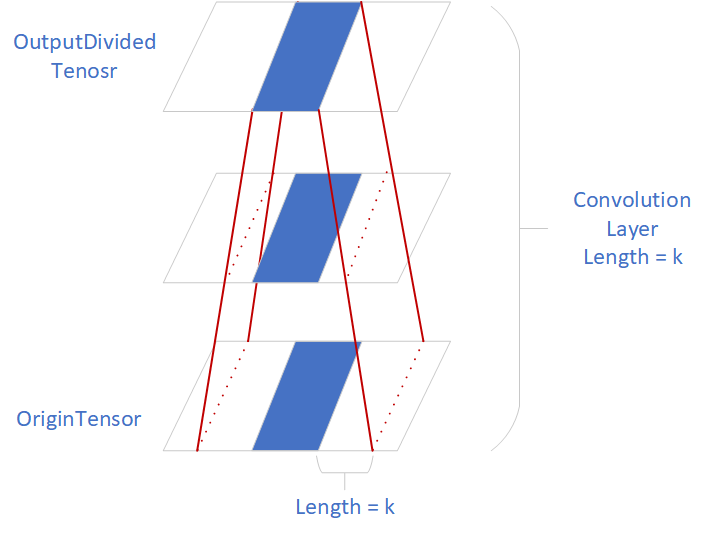


图 5：VGG16卷积层拆分的边缘效应和卷积层数关系

在经过上述推理后，本文可以得出如下的卷积层拆分伪代码如下。

origin\_tensor = input()

node\_num = k

for i in VGG16Layers:

if i is convolution layer:

cross\_layer = current convolution layer length

avg\_length = origin\_tensor.length()/node\_num

divided\_tensor = []

start = 0

end = 0

for j in node\_num:

end = start + avg\_length

divided\_tensor = origin\_tensor[start - cross\_layer : end + cross\_layer]

start = end

divided\_output\_tensor = remote\_cal(divided\_tensor, i , i + cross\_layer)

origin\_tensor = merge\_tensor(divided\_output\_tensor)

i += cross\_layer

else:

origin\_tensor = local\_cal(origin\_tensor, i , i)

由上文提到的VGG16模型可知，该模型可以按照顺序划分为6个块：

Block1：conv3-64，conv3-64（两层连续的卷积层），maxpool（池化层）

Block2：conv3-128，conv3-128（两层连续的卷积层），maxpool（池化层）

Block3：conv3-256，conv3-256，conv3-256（三层连续的卷积层），maxpool（池化层）

Block4：conv3-512，conv3-512，conv3-512（三层连续的卷积层），maxpool（池化层）

Block5：conv3-512，conv3-512，conv3-512（三层连续的卷积层），maxpool（池化层）

Block6：fc-4096，fc-4096，fc-1000（三层全连接层）

不难看出，VGG16模型block1-5中的连续卷积层都可以按照上文提到的卷积层拆分算法进行特征向量图的拆分，再分配到多个设备上进行并发计算。当所有分设备计算完毕，将结果在主设备进行每个块最后的池化层计算之前返回，主设备将这些结果拼装后，进行后续的池化层计算。并在运行到下一个卷积块时，再将计算分配给所有分设备，依次类推，直到最后的全连接层，完成全连接层计算后的结果即为最终的分类结果。

再上述模型和拆分算算法中，一次分类会经过多次网络I/O，取决于卷积层的数量和聚散程度，在网络环境较差的情况下，这种交互会产生较高的时间成本，但是在模型较大导致计算量过大或者时网络条件远优于计算性能的场景下有着较高的应用价值，同时还能够提高设备应用率，增加单个设备的负载能力。在边缘设备上有着很大的应用场景。

1. **系统设计**

在有了上述的理论基础后，就可以进行实际的开发和结果测定。这一部分包括VGG16模型的搭建、训练，主分设备之间的rpc模块的搭建，以及最终拆分算法的实现。

**3.1系统实现框架**

在进行编写程序的具体工作之前，需要建立一个完整的的框架，这样才能对整体设计有一个清晰地认识。系统流程图7所示。

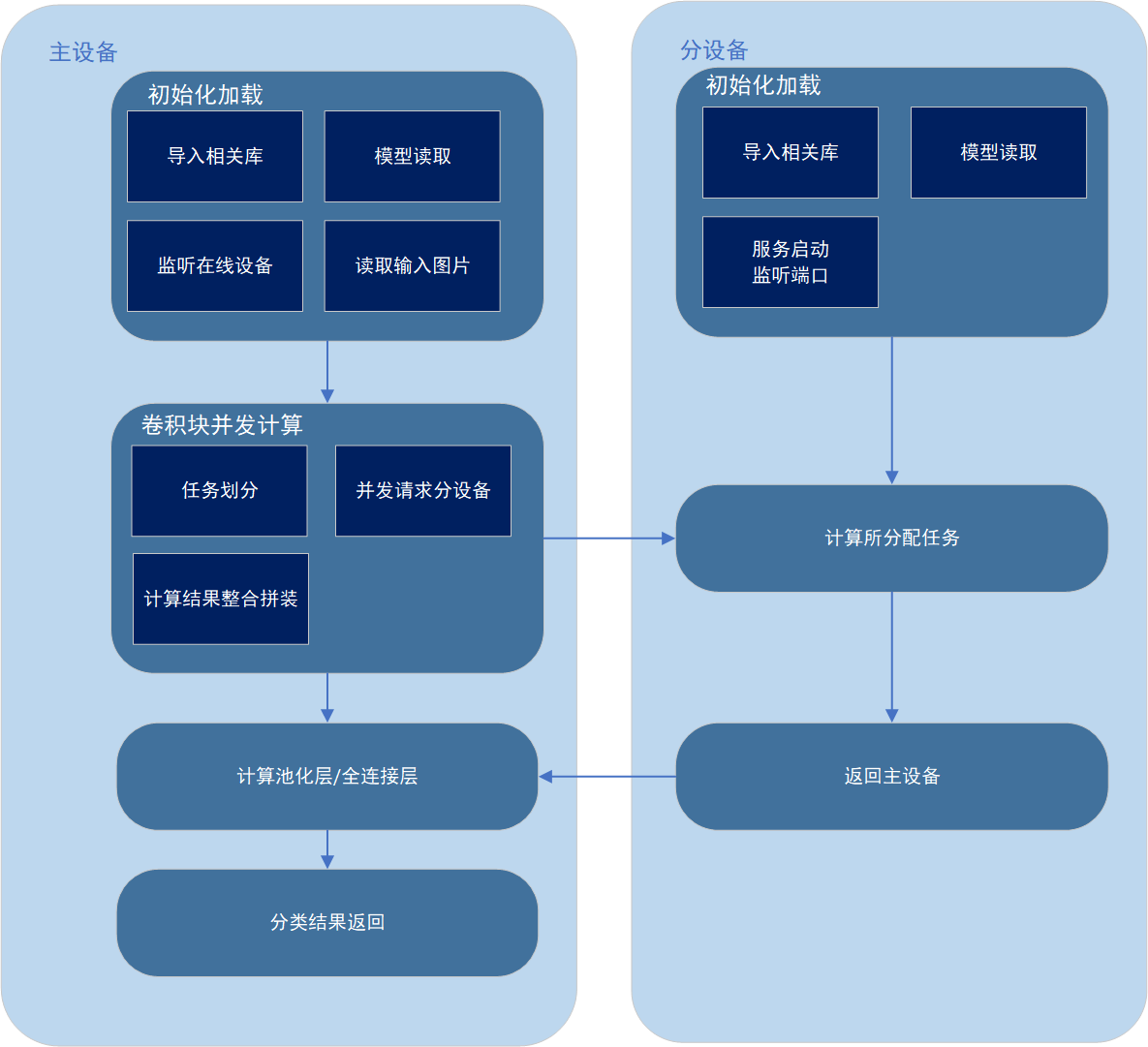


图 6：模型并发处理流程图

**3.1.1主设备**

最开始，主设备需要加载一些基本数据：导入需要的库函数，将已经训练的模型加载进内存，以防止后续多次任务重复加载。其次，需要加载当前在线的所有分设备socket表，以便于后续分配任务，目前是类似与配置项的形式读取。主设备在接收到分类任务时，首先将图片格式转换成VGG16模型可接受的格式，再开始进行模型的计算。当主进程计算时，如果遇到连续卷积层（即上文提到的卷积块），根据当前在线的分设备进行任务划分，并发向分设备发起rpc请求。在所有分设备都返回了计算结果之后，主设备整合所有计算结果，计算当前分块的池化层。最终，当所有分块计算完毕，主设备计算上文提到的VGG16模型第六块，即全连接层，并将最终计算结果返回。

**3.1.2分设备**

最开始，分设备和主设备相同，加载一些基本数据：导入需要的库函数，加载已训练的模型。在这之后，分设备启动rpc服务，监听接口信息。当分设备收到主设备计算任务请求时，开启一个线程用于计算当前子任务。在子任务计算完毕之后，将计算结果返回到主设备。

**3.2 模型的搭建和训练**

**3.2.1模型搭建**

上文中已经提到，本设计使用的卷积神经网络模型为VGG16，具体的层级划分和顺序也在前文中给出。因此只要使用Pytorch构建对应的卷积层、池化层、全连接层，并将其按顺序添加都模型中即可，Pytorch提供了非常便捷的函数用于添加对应层级，只需要指定对应的参数，并重写计算函数即可，模型搭建如下

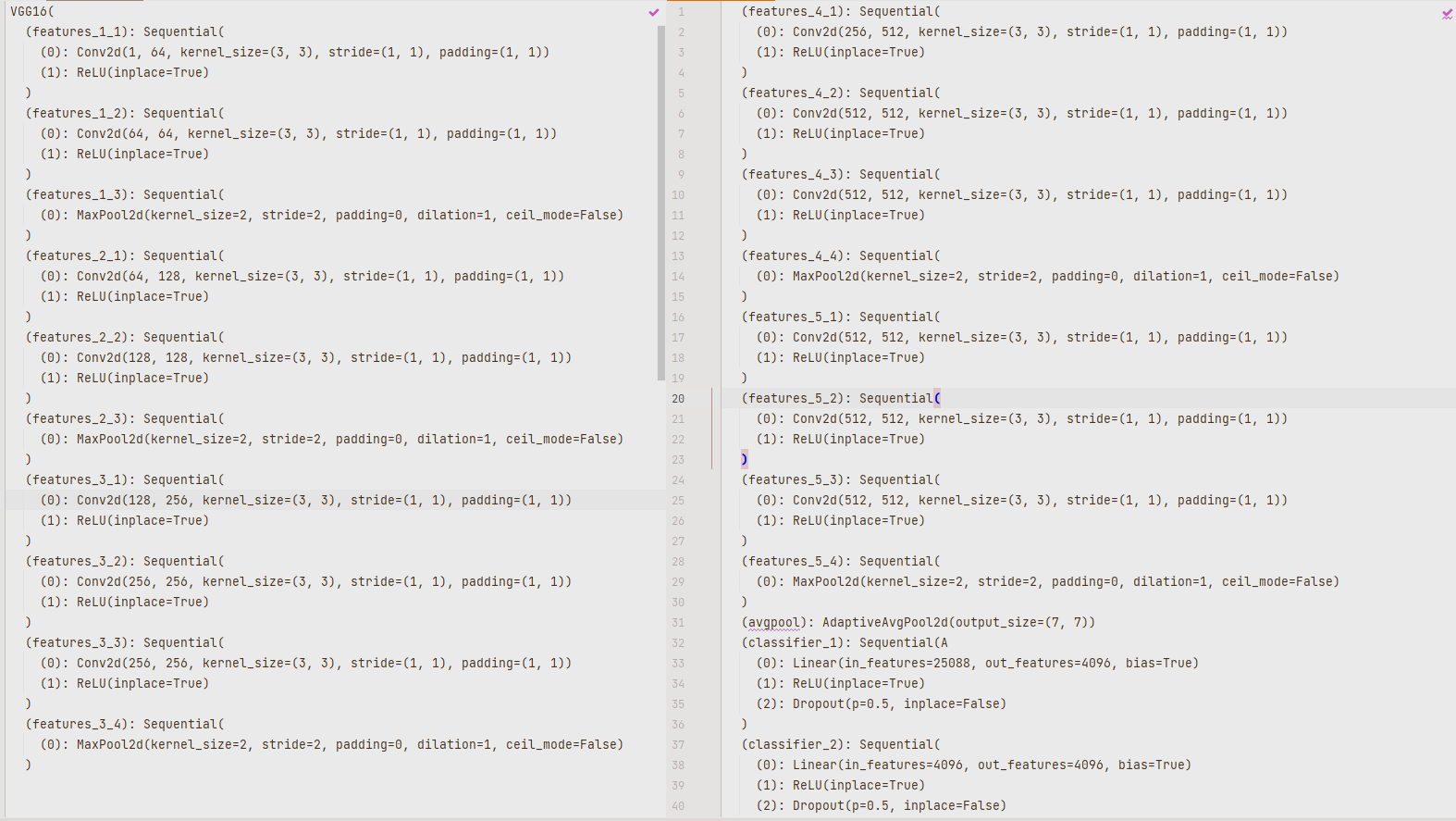


图 7：本设计使用的VGG16模型

**3.2.2模型训练**

模型训练中的数据来源采用的是MNIST数据集，MNIST数据库是非常经典的一个数据集，其内容是由0〜9手写数字图片和数字标签所组成的，由60000个训练样本和10000个测试样本组成，每个样本都是一张2828像素的灰度手写数字图片。并且利用torchVision包可以很方便的获取它的数据集。由于当前的 VGG16模型只接受6464的输入，因此在训练和测试时需要先将图片格式转换成为6464，同时由于这些图片是灰度图片（单通道），故需要修改模型的入口卷积层为单通道。

训练时，将该数据集的前5000张图片作为测试数据，后1000张图片作为训练数据，损失函数采用Pytorch自带的CrossEntropyLoss（交叉熵损失函数）函数，优化方式采用Adam优化器。在实际训练时，因当前gpu核心版本不适配而采用cpu训练。

**3.3 rpc模块的搭建**

在第二章技术准备中有提到，本设计是用的网络交互模块采用的是gRPC框架，它的优势在于生态好、跨语言、性能高、强类型、流式处理特点，有自身适配的数据格式（protobuf），相比于使用json数据格式交互快六倍。

**3.3.1设计数据格式**

使用gRPC框架时，首先需要编写protobuf文件，是一种类似于json、xml的结构。用于定义本文的服务类型、rpc交互的数据类型和格式，它的解释器可以用pip非常快捷的安装。本设计设计的交互数据格式如下：

请求：

message PartTaskRequest{

string txtMessage = 1; //交互文本信息

int32 fromLevel = 2; //计算的开始层级

int32 toLevel = 3; //计算的结束层级

bytes tensorInput = 4; //张量输入

string tensorInputSize = 5; //输入的张量格式

int64 startTime = 6; //主设备发出请求时间戳

}

返回：

message PartTaskResponse{

string txtRes = 1; //交互文本信息

bytes tensorOutput = 2; //计算后的张量

string tensorOutputSize = 5; //计算后的张量格式

int64 endTime = 6; //请求返回时间

int64 calTime = 7; //张量计算时间

}

其中，由于张量是一个维度不确定的高维数组，因此将其转换为字节流进行传输，同时将这个张量正确的格式同步传递到主设备/分设备，在接收后对其进行重新构建。在编写完成之后，使用如下命令完成源代码的生成

python -m grpc\_tools.protoc -I. --python\_out=. --grpc\_python\_out=. divider\_rpc.proto

**3.3.2编写服务端**

gRPC框架提供了一些基本的函数可以很方便的实现服务端的生成。Server（）函数可以用于设置服务的最大并发请求数量以及消息体大小，由于本设计应用于边缘多设备网络，可能会出现同时多设备请求的情况，因此设置最大并发请求为10，增强设备的负载灵活性。同时，由于本设备交互的张量有时较大，超过了gRPC的默认消息体大小，因此需要重新设置消息体大小为256M，减少因为消息体过小而产生的请求拒绝情况。

在编写完成服务端主体后，还需完成请求函数的重写，由于本文在模型创建时已经重写了模型的推理函数，并且完成了对于本设计的适配（支持只运行指定的层级），因此这一步变得非常简单，只需要将输入的张量字节流按照张量格式转换为高维数组，再转换为模型可以接受的张量格式即可。最后再将计算完成后的输出张量转换为字节流，并和张量格式一并返回主设备，完成服务端的编写。

**3.3.3编写客户端**

相比于服务端的编写，客户端的编写就尤为简单，只需要指定请求的socket以及可接受的最大消息体大小（原因如服务端），在请求和接收时完成数据的转换（特别是张量的转换）即可。

**3.4拆分算法的实现**

**3.4.1张量拆分/合并模块**

在本文第三章算法基础中提到，本设计实现卷积层任务并发执行的方式是通过将输入张量进行拆分，将拆分后的张量转移到多个分设备上进行运算，最后再将所有分设备的计算结果进行合并从而实现计算任务的分配。在这个模块，由于本设计使用的VGG16的卷积层交互使用的张量都是四维，故本文将这个划分和合并的位置设置在第四维。

对于张量拆分模块，首先是对于张量和节点数进行任务的平均分配，例如，本文的输入是一个大小为[a,b,c,d]的张量，同时分设备数量为n，那么，分配到n个分设备上的计算任务就会是大小为[a,b,c,d/n]的张量。更进一步，考虑到在第三章提到的多层级聚合卷积块计算时的边缘情况，假设当前卷积块一共包括了k个层级，那么对于VGG16模型，划分到每个分设备的实际计算任务大小应该为[a,b,c,d/n + 2\*k]的张量。基于以上条件，可以推导出如下的公式，对于第i个设备，它的计算范围R应该为：

有了上述公式，利用python的切片特性，便可以很方便的实现计算任务的划分。同理，在拼装计算后的张量时，对于每一个张量，应该取其有效计算的分区进行拼装，即长度为（d/n）的中央分区，对于第i个分设备返回的张量有效分区AC的计算公式如下：

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

|  |  |  |
| --- | --- | --- |
|  |  |  |

根据上述公式，利用torch自带的cat函数便可以非常便捷的实现最终结果的合并，利用合并后的结果完成后续的计算。

**3.4.2并发请求模块**

在本设计中，需要将VGG16中的卷积层拆分到多个分设备上进行计算，在这种情况下，串行请求分设备显然是不合理的，这反会增加了额外的计算量和网络I/O的时间，即使它也能在一定程度上提高整个边缘设备网络内的设备利用率。因此，本文需要一个并发控制模块来实现分设备的并发请求，并保证最后返回结果的有序性用于张量的合并。这里，本文采用python的concurrent.futures库来实现线程的并发控制，本文只需要指定最大的并发线程数和对应的输入即可，本设计设置的并发进程数为10，因为由上述划分公式可知，更多的划分也会产生更多的计算资源的浪费，特别是在卷积块较大的情况下。因此本设计先不设置过多的并发节点，同时也和服务端对应的最大并发请求数为10匹配。

**3.4.3主函数模块**

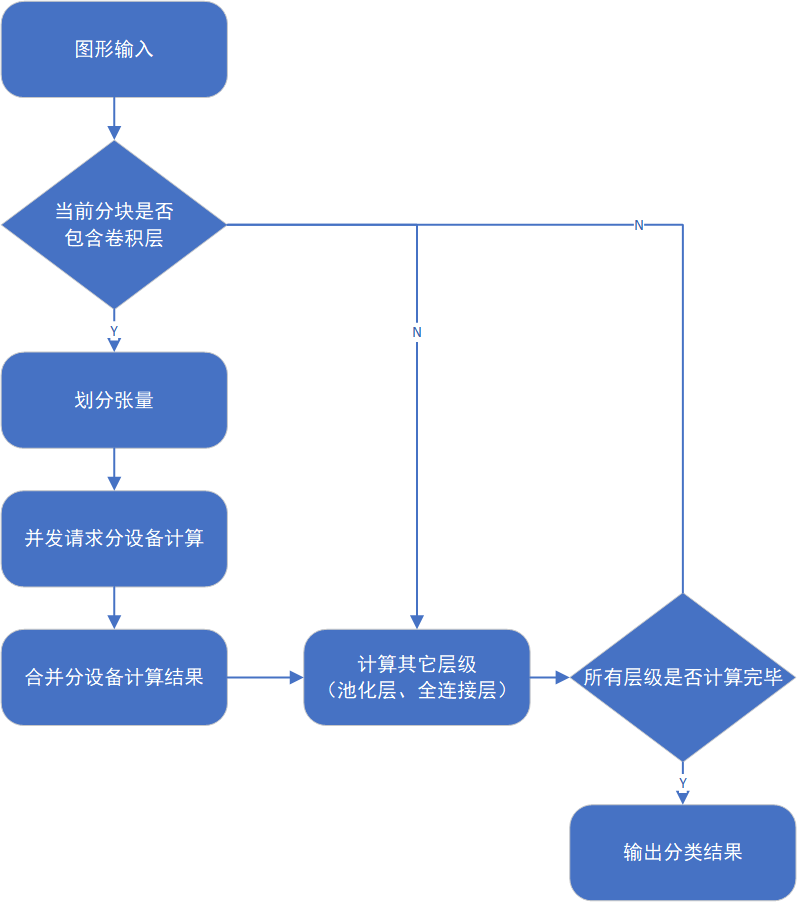


图 8：拆分主函数流程图

由第三章给出的伪代码可知，主函数模块的主要工作是实现卷积块的任务分配，以及除了卷积层之外的层级的计算，因此，本文不难给出主函数模块的流程图，具体流程图如图9所示。根据上述流程图，以及前文中提到的张量拆分/合并模块和并发请求模块，在结合python的分支语句，本文可以很方便的完成主函数代码的编写。

1. **系统评估**

在第四章中，本文完成了本项目所涉及的所有代码模块的编写，为了保证各个模块以及整体的可用性，本章将对设计进行更加详细的测试。

**4.1 单元测试**

这一部分先对本文第五章的各个模块拆分进行单元测试，保证系统的可用。

**4.1.1模型测试**

由于本设计的重点不在于模型训练和其准确率测定，因此只做了简单的单轮

训练，训练结果大致如下：

模型训练结果

图 9：模型训练结果

4**.1.2 rpc模块测试**

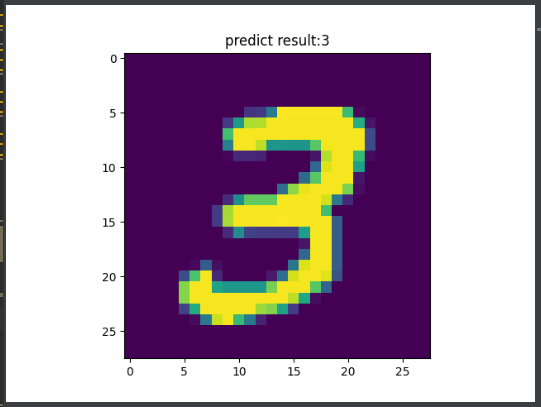


图 10：模型测试

在Rpc测试的中，本文选择在本地启动服务，同时再起另一个进程请求服务

端，服务端和客户端的交互结果如图11、12所示。

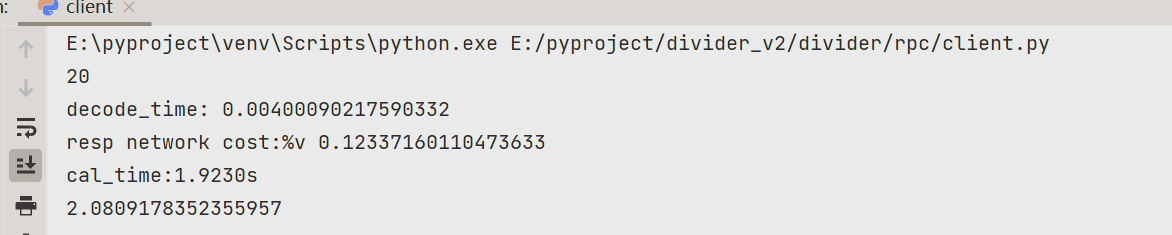


图 11：客户端测试结果



图 12：服务端测试结果

**4.1.3并发模块测试**

在并发模块测试中，本文同样采用本地服务的形式，最终并发请求的结果如下：

客户端：

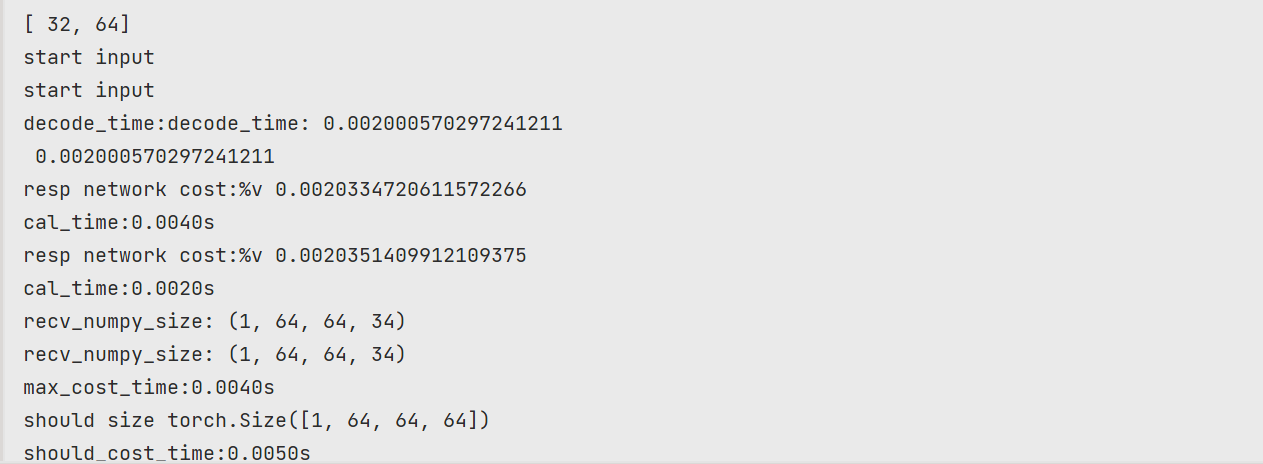


图 13：并发请求客户端测试

服务端：

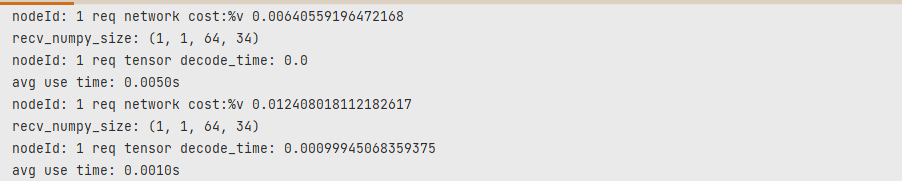


图 14：并发请求服务端测试

**4.1.3 主函数测试**

本测试将上述所有模块进行整合测试，同样采用本地服务形式，最终的运行结果如下：

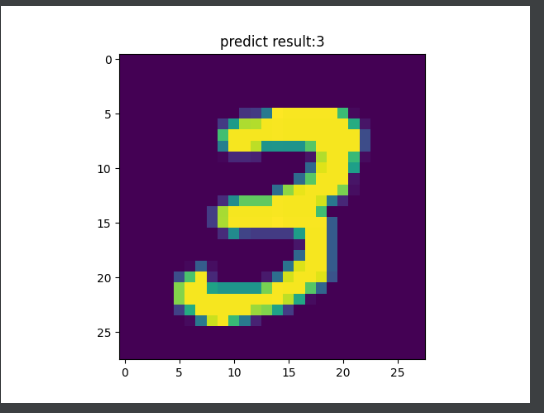


图 15：拆分测试1

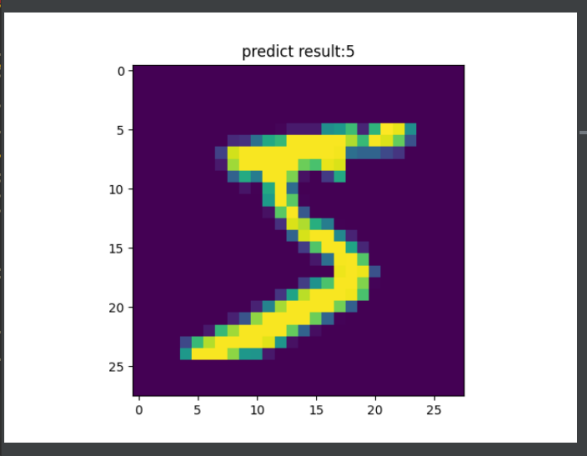


图 16：拆分测试2

**4.2综合测试**

在上述环节中，我们完成了各个模块的测试，保证了模块的可用性，在本环节，我们将对整体系统做综合测试以及性能的评估。

4**.2.1 准确率测试**

本设备执行准确率

图 17：本设备执行准确率

在准确率测试上，本文构造了两个函数用于测试，一个是不采用分布式执行方式的准确率评定，而第二个是采用了分布式执行方式的准确率评定，使用MNIST未经训练的部分数据进行测试，测试结果如下：

可见分布式执行的测试结果与本地执行的测试结果非常接近，甚至还略高于本地执行。这代表着本设计在模型的分布式执行上是成功的。

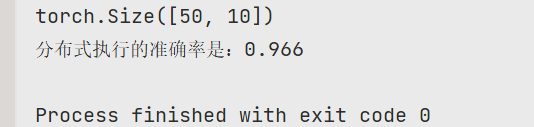


图 18：多设备执行准率

4**.2.2 延时优化测试**

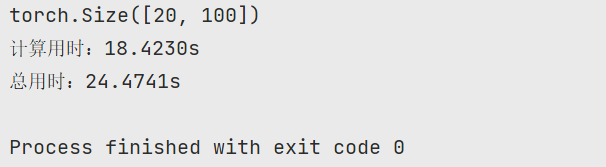


图 19：单机执行vgg16模型

在时间测试上，由于机器资源有限，我们使用了三台机器，两台性能较好，一台性能较差，其中一台机器同时用于充当服务端和客户端，其余两台机器充当服务端，一台机器执行结果如图20所示。

接下来是选择两台性能较好的机器并发执行计算任务，结果如图21所示。

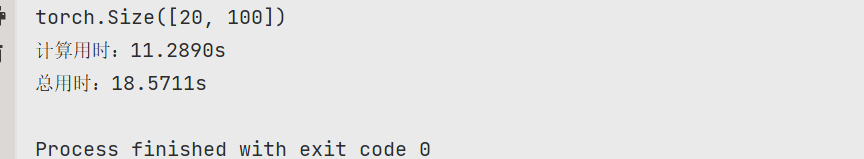


图 20：双机并发执行

由测试结果可以看出计算时间有明显的减少，但是算入网络I/O后相差不多。造成这一问题的原因是网络I/O的占比过多，选择计算更复杂的模型可能会有更好的结果。接下来是三机执行的测试，由于第三台机器的性能较差，因此这一部分测试更改了测试方法，选择将第三台机器的任务转移到第二台机器，相当于第二台机器执行两份任务，但这回导致第二台机器的网络I/O时间增加，测试结果如图21所示

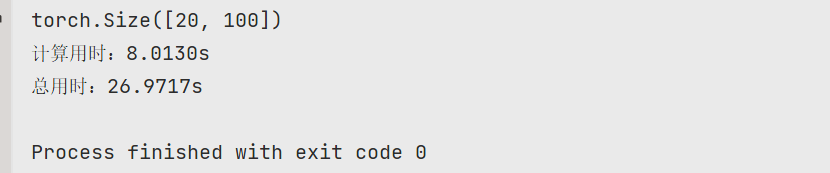


图 21：三机并发执行

可以看出计算时间被进一步压缩了，但是由于第二台机器的I/O压力增大，网络耗时也同样增加。接下来是将性能较差的第三台机器加入测试，结果如图22所示

可以看出，在任务平均分配的情况下，一台性能较差的设备会导致计算耗时增加，因为主机需要在池化层和全连接层之前收集所有结果，最后整理的表格如下：

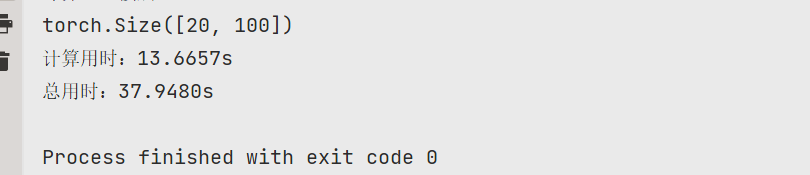


图 22：三机中存在一台机器性能较差

|  |  |  |  |
| --- | --- | --- | --- |
|  | 计算耗时 | 网络I/O耗时 | 总耗时 |
| 单机执行 | 18.423s | 6.0511s | 24.4741s |
| 双机并发执行 | 11.289s | 7.2821s | 18.5711s |
| 三机并发执行(第二台机器参与两份任务) | 8.013s | 18.9587s | 26.9717s |
| 三机并发执行(一台设备性能较差) | 13.6657s | 24.2823s | 37.9480s |

表 1：推理时间汇总

由测试结果不难看出，卷积神经网络的并发拆分在卷积层的计算上起到了一定的作用，计算耗时有着显著的降低，但是网络交互占据了大量的时间，后续网络的升级可能能够减少这些开销。同时，平均分配任务的方式有一些不妥，后续可以根据性能状况对任务进行划分。

1. **结束语**

本设计实现了将一个VGG16卷积神经网络拆分，将占据其大量运算资源的卷积层拆分，分配到多个分设备上进行运行，从而实现多设备加速。加快了主设备的响应速度，增加了边缘设备集群的单点负载能力，同时提高了单点高压力时的边缘设备的设备利用率。

同时，本设计也存在一定的缺陷，由于本设计将计算任务转移到了多个设备上进行执行，这会产生新的网络IO成本，将会占据大量的网络带宽，不利于设备之间的交互。并且若对应设备的网络状况远差于其计算性能，本设计反而会产生更长的推理时间以及不必要的网络开销和计算资源的浪费。另外，目前的设计是将计算任务平均分配到多个分设备上，然而，同一集群内的设备也会有着很大的差异，例如某设备有着更好的计算性能或者和主设备之间有着非常好的网络条件，这些影响因素都应该体现在任务分配上。同时，本文在4.2中提到，过多的分设备分配可能会导致更多的计算资源浪费，甚至相对于较少的分设备有着更长的推理时间，这同样也是一个影响因素，它由卷积块的大小，分设备的数量，以及当前整个边缘设备集群的负载情况共同决定。后续优化应该着重考虑这些因素，能够更好的平衡多个设备的计算时间，从而进一步减少主设备推理时间。

**主要参考文献**

1. Xue F, Fang W, Xu W, et al. EdgeLD: Locally Distributed Deep Learning Inference on Edge Device Clusters[C]//2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2020: 613-619.
2. Zhou Z, Chen X, Li E, et al. Edge intelligence: Paving the last mile of artificial intelligence with edge computing[J]. Proceedings of the IEEE, 2019, 107(8): 1738-1762.
3. Mao J, Chen X, Nixon K W, et al. Modnn: Local distributed mobile computing system for deep neural network[C]//Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. IEEE, 2017: 1396-1401.
4. Mao J, Yang Z, Wen W, et al. Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns[C]//2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, 2017: 751-756.
5. Kang Y, Hauswald J, Gao C, et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge[J]. ACM SIGARCH Computer Architecture News, 2017, 45(1): 615-629.
6. Zhou L, Wen H, Teodorescu R, et al. Distributing deep neural networks with containerized partitions at the edge[C]//2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19). 2019.
7. Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network[J]. arXiv preprint arXiv:1503.02531, 2015, 2(7).
8. Hardy C, Le Merrer E, Sericola B. Distributed deep learning on edge-devices: feasibility via adaptive compression[C]//2017 IEEE 16th International Symposium on Network Computing and Applications (NCA). IEEE, 2017: 1-8.
9. Sergeev A, Del Balso M H. fast and easy distributed deep learning in TensorFlow[J]. arXiv preprint arXiv:1802.05799, 2018.
10. Chollet F. Xception: Deep learning with depthwise separable convolutions[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 1251-1258.
11. Zhao S, Liu B, Wang F, et al. Bridging the Gap Between Memory and Communication Efficiency on Distributed Deep Learning Systems[J]. IEEE Access, 2021, 9: 57075-57088.

**致谢**

大学四年的学习时光已经接近尾声，在此我想对我的母校、我的父母、我的老师和同学们表达我由衷的协议。