

# 南 开 大 学

## 本 科 生 毕 业 论 文（ 设 计）

中文题目： 基于 CUDA 的 FaceNet 访存优化

英文题目： Optimization of FaceNet Memory Access Based on  
CUDA

学 号： 1611283

姓 名： 杜乾刚

年 级： 2016级

学 院： 网络空间安全学院

系 别： 物联网工程

专 业： 物联网工程

完成日期： 2020年5月

指导教师： 宫晓利

---


## 南开大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下进行研究工作所取得的科研成果。除文中已经注明引用的内容外，本学位论文的科研成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：\_\_\_\_\_年 月 日

-----

本人声明：该学位论文是本人指导学生完成的科研成果，已经审阅过论文的全部内容，并能够保证题目、关键词、摘要部分中英文内容的一致性和准确性。

学位论文指导教师签名：\_\_\_\_\_  年 月 日

## 摘 要

图像的学习和识别在各种领域中应用越来越多,而且对于识别的速度要求越来越高,例如在交通部门的检票系统中,传统的人工检票一直限制检票速度、加剧检票口人流拥堵,人脸识别智能检测系统的加入可以加快检票速度,节约大量的人工成本和时间成本;在个人设备中,人脸识别应用到设备解锁;人脸识别代替密码登录。

FaceNet 是谷歌基于深度神经网络开发的人脸识别系统,在多人的人脸识别场景中进行人脸验证、人脸识别和人脸的聚类学习具有良好的性能。但是识别过程中需要频繁地进行不必要的 GPU 与 CPU 之间的数据替换,造成 GPU 资源的浪费和数据冗余,加剧训练时间消耗。在单人人脸识别场景中,人脸识别的时间将超出期望,严重影响系统的整体性能。

因此,本文通过改进 FaceNet 存在的上述缺陷,开发一套基于 CUDA 的 GPU 加速,适合单人人脸识别与训练的单人人脸识别系统(Single Face Recognition and Clustering, SFRC),通过 GPU 对传入数据的保存,减少 CPU 与 GPU 之间的非必要数据交换,来提高数据的利用率和降低时间消耗和资源占有。

**关键词:** CUDA; 图计算; 卷积神经网络; 并行计算; 人脸识别

## Abstract

Image learning and recognition are more and more used in various fields, and the speed of recognition is getting higher and higher. For example, in the ticket inspection system of the transportation department, the traditional manual ticket inspection has always restricted the ticket inspection speed and exacerbated the traffic jam at the ticket gate. The addition of the face recognition intelligent detection system can speed up the ticket inspection and save a lot of labor and time costs; in personal devices, face recognition is applied to unlocking the settings; face recognition instead of password login.

FaceNet is a face recognition system developed by Google based on deep neural networks. It has good performance in face recognition, face recognition and clustering learning of faces in a multi-person face recognition scenario. However, during the recognition process, unnecessary data replacement between the GPU and the CPU needs to be frequently performed, which causes waste of GPU resources and data redundancy, and aggravates training time consumption. In a single-person face recognition scenario, the face recognition time will exceed expectations, seriously affecting the overall performance of the system.

Therefore, in this paper, by improving the above defects in FaceNet, a set of CUDA-based GPU acceleration is developed, which is suitable for single face recognition and training (Single Face Recognition and Clustering, SFRC). The saving of data reduces the unnecessary data exchange between CPU and GPU to improve data utilization and reduce time consumption and resource occupation.

**Key Words:** CUDA; Graph computing; convolutional neural network; parallel computing; Face recognition

## 目 录

摘 要 .....	I
Abstract .....	II
目 录 .....	III
第一章 绪论.....	1
第一节 以 CUDA 为代表的 GPU 并行计算发展.....	1
第二节 研究的目的与意义.....	2
第二章 背景知识.....	5
第一节 前馈神经网络与卷积神经网络.....	5
2.1.1 前 馈 神 经 网 络 .....	5
2.1.1 卷 积 神 经 网 络 .....	7
第二节 CUDA 简介.....	7
第三节 FaceNet 原理.....	11
第三章 SFRC 系统设计.....	13
第一节 SFRC 优化原理.....	14
3.1.1 GPU 内部访存优化 .....	15
3.1.2 图像训练与 CPU-GPU 访存优化.....	19
第二节 SFRC 模型训练流程 .....	20

---

第四章 实验与结果分析.....	23
第一节 实验环境与参数.....	23
第二节 实验结果.....	24
第五章 结论与展望.....	30
参考文献 .....	31
致 谢 .....	33

---

## 第一章 绪论

### 第一节 以 CUDA 为代表的 GPU 并行计算发展

随着人类进入信息时代后，计算机在各行各业的广泛应用，产生的数据规模也越来越大，计算机承担的计算量，并行计算是如今大规模数据计算下能够有效提高计算速度、计算精度的计算方法之一。通过并发方式，同时使用多种计算机资源来解决计算问题，提高计算效率。从整体上来说，无论哪种并行计算方法，都是建立在计算机硬件资源基础之上来构造各种算法，而目前计算机中具有计算能力的资源是 CPU 和 GPU。

在计算机的发展历程中，为解决计算机系统模块间不兼容问题，会不断在计算机系统中增加新的硬件资源。GPU 是相对于 CPU 来讲的硬件结构，由于现代图形处理越来越复杂，CPU 无法提供图像的处理需求，因此生产出一个专门负责图形处理的多核心处理——GPU。与 CPU 不同，GPU 中的所有计算均为浮点计算。从结构上看，GPU 具有足够多的小的核心（费米架构就有着 512 核），具有规则的数据结构，因此可以批处理多个尤其是一组相似的计算任务。CPU 虽然单个核心的计算能力远远强于 GPU，但整个 CPU 中核心数很少，且 CPU 需要具有通用性来支持处理多种不同类型的数据结构，同时必须支持处理系统判断导致的分支跳转和中断处理，因此 CPU 的核心数不能太多（目前不能超过 16 核）。CPU 主要体现在逻辑处理能力强大。综合比较 CPU 与 GPU 的结构特点，GPU 相对 CPU 具有浮点计算能力强、能耗低、带宽高、快速访存等优点，但不具有中断处理分支等能力。

基于 CPU 串行能力强，GPU 并行能力强，CPU+GPU 异构协同并行计算模式能最大程度的提高效率，在 CPU 上运行串行计算部分以及负责逻辑控制，并行部分则在具有更小、核心数量更多、更节能的 GPU 上运行，CPU 同时控制 GPU 中的数据流向，为 GPU 传入数据并负责传回数据。

与 CPU 上编程不同，GPU 编程有着许多限制，通用数据结构仍然是 GPU 编程的最大困难之一，而与 CPU 相同之处是在 GPU 中仍然存在着存储墙问题。在 GPU 中通过使用硬件和多线程技术来降低访存延迟，而 CPU 中则通过多级缓存降低访存延迟，并且与 CPU 中不同，在 GPU 中还存在着共享内存的结构，其访

问速度要优于直接访问 GPU 内存速度。GPU 模型如图 1.1 所示。

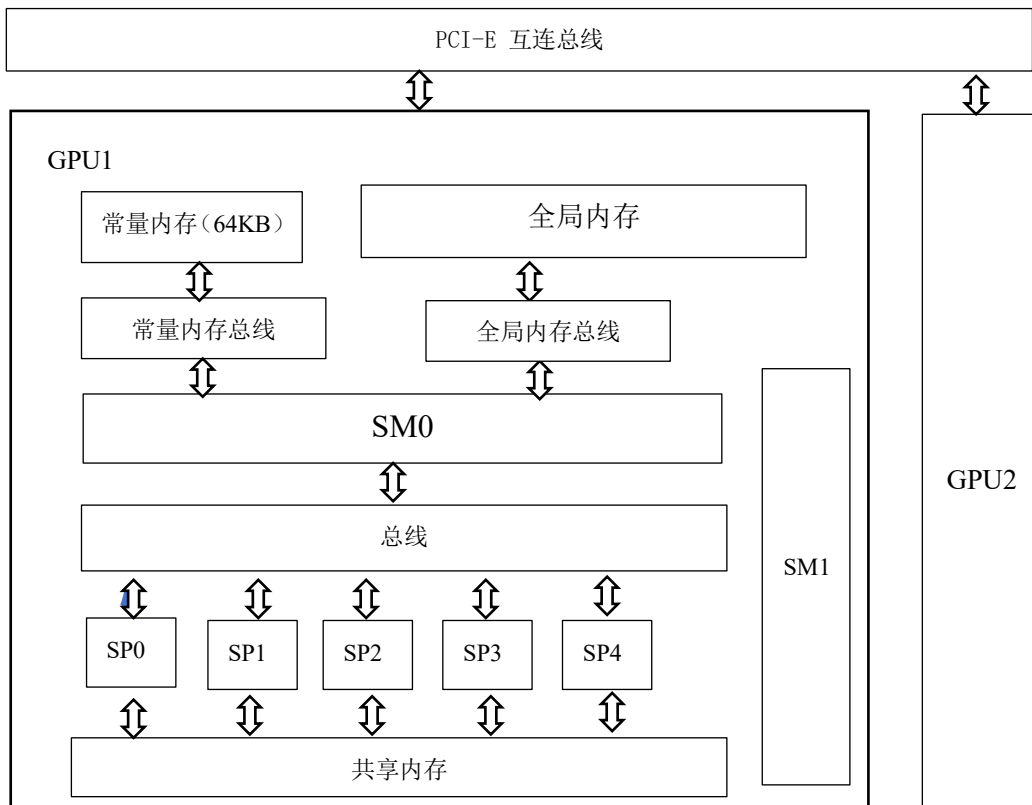


图 1.1 GPU 模型

目前主流的 GPU 编程接口有 CUDA、Open CL、Direct Compute，CUDA 是由 NVIDIA 公司开发的一套适用于 NVIDIA 旗下 GPU 的计算集成技术，以 C-编译器作为开发环境。Open CL 是是开源且适用于几乎所有显卡的一个编程接口，Direct Compute 是由微软开发的但只能适用于 Windows 系统，在高端的服务器中不适用 UNIX 系统。

在计算机的发展历程中，不断会有为解决计算机系统模块间不兼容问题而增加的硬件资源，CPU+GPU 异构计算将在很长一段时间作为并行计算的主流模式。

## 第二节 研究的目的与意义

GPU 最初设计出来是作为图形显示过程中绘图运算的微处理器，但是在如今电子技术日益进步的时代下，随着获取信息的方便也产生了大量的数据信息，尤其是需要大量图像计算的人工智能发展，依赖 CPU 的单核计算系统已经不能承担如此庞大的计算量。以 GPU 为计算核心的 CPU/GPU 异构系统对涉及机器学习、并行计算等各领域的作用日益显著，GPU 已经成为人工智能芯片市



场的重要部分，在未来的发展中计算任务的核心将越来越向着 GPU 偏移，GPU 将承担大部分的计算任务，尤其是不相关的可并行计算任务，对 GPU 加速计算的研究也将成为一个热门。

机器学习是依赖于大量数据处理的代表，机器学习深入物理、互联网行业、AI 等多领域之内。对图像处理的优越性使卷积神经网络成为机器学习领域的一个热点。图像计算中一般是进行浮点数的运算，而且卷积神经网络涉及大量矩阵运算，GPU 虽然没有 CPU 的主频高，但是擅长浮点线性代数运算，研究 CPU 和 GPU 的协调工作能够尽最大程度的提高计算加速能力。

图像识别是人工智能领域的核心部分，包括物的图像识别和人的图像识别，是计算机认识的基础，整个过程包含训练与图像再认，智能设备要认识物体必须经过图像识别。例如文字识别、物体识别，在车站等公共场所进行单人识别、多人识别；各种场所的检票过程，随着客流量的急速增加，人脸识别代替人工检票可以提高效率、节约成本；个人设备的解锁和密码服务使得系统更加人性化。FaceNet 是一个很好的人脸识别通用系统，将三元组训练技术加入到人脸识别的训练过程中，取得了很好的效果。但是 FaceNet 由于是基于 TensorFlow 开发的人脸识别系统，而 TensorFlow 是封闭的数据流系统，FaceNet 在 GPU 加速计算时，每次调用 TensorFlow 接口进行加速计算都需要一个 CPU 向 GPU 传送数据以及计算完成时 GPU 向 CPU 回传数据的过程，而且在一个单次训练结束后 GPU 对这块数据内存就会销毁却并不会保存到下次训练，但是这部分数据恰恰是下次训练的所需要的数据，这样对于单人的训练与识别会产生不必要的时间消耗，而且会造成 GPU 内存资源的浪费，例如在个人设备解锁情景下、检票系统中，会延长系统的识别响应时间。本文就 FaceNet 这个缺点，基于 FaceNet 的算法原理和 CUDA 技术，开发在单人图像训练和识别领域中能够尽可能加速训练和识别过程的单人脸识别系统(Single Face Recognition and Clustering, SFRC)。因为针对的是单个人的图像训练，不存在数据集的变化，所以在进行 GPU 加速的时候，CPU 将整个数据集传送至 GPU，并在 CPU 中记忆这个数据集的位置，在之后涉及 GPU 加速并需要用到该数据集的时候直接引用，并在训练的过程中有保护地更新数据和参数（即在后续训练过程中只有在不需要该数据的前提下才对其在 GPU 中的内存进行释放，并更新该相关参数），在整个训练结束，并且成功训练

不发生训练奔溃的前提下，CPU 释放数据集在 GPU 中的内存并读回训练结果。这样在整个训练中，只进行一次必要的 CPU 与 GPU 之间数据传入和传出，可以减少时间消耗，节省对系统资源的占有，加快在单人脸识别的场景中的系统响应时间。

## 第二章 背景知识

### 第一节 前馈神经网络与卷积神经网络

#### 2.1.1 前馈神经网络

##### 神经元

神经元是神经网络的基本计算单位，模拟生物神经元的特性，一个神经元包含输入输出以及计算功能三部分，神经元的基本模型如图 2.1 所示：

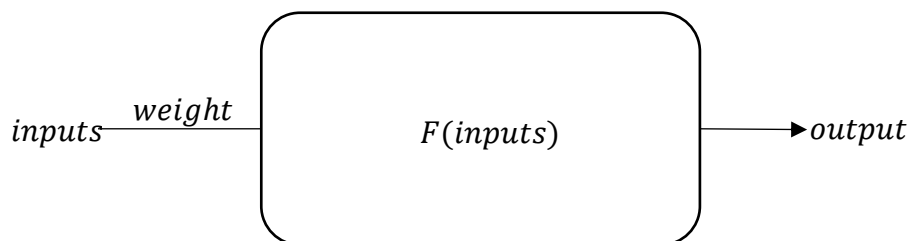


图 2.1 神经元模型

神经元通过一个激活函数 $f$ （或非线性激活函数）接受一组输入 $x$ 经过加权 $w$ 计算产生一个状态 $z$ ，然后输出神经元的活性值 $a$ ，具体定义公式为：

$$z = w^T x + b \quad (2.1)$$

$$a = f(z) \quad (2.2)$$

其中 $w$ 表示权重， $b$ 为偏置量。

##### 前馈神经网络

前馈神经网络是最简单也是最早的人工神经网络，神经网络由多个神经元组合构成，相邻神经元之间有具有权重的连接，一个神经元的输出作为另一个神经元的输入，图 2.2 就是一个简单的前馈神经网络：

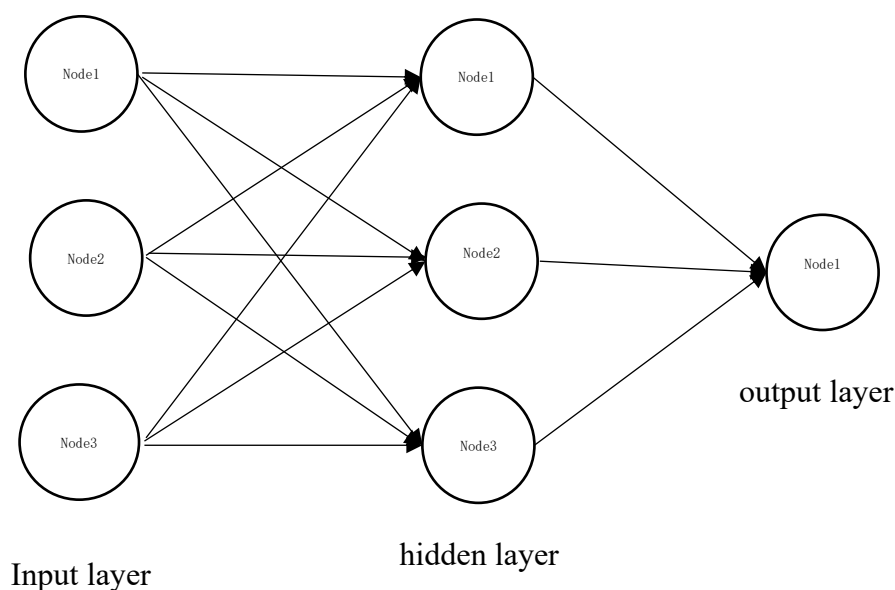


图 2.2 前馈神经网络

一个前馈神经网络有三类节点：输入节点、隐藏节点、输出节点。输入节点组成输入层，负责将外部数据传入神经网络，输入层的所有节点不进行计算，只负责将数据传入隐藏节点；隐藏节点对由输入层传入的信息进行计算并将信息在传入输出层，隐藏节点不与外部世界有直接连接，隐藏节点的集合构成隐藏层；输出节点的集合构成输出层，负责计算以及向外部世界输出信息；一个前馈神经网络只有一个输入层和一个输出层，但是可以有多个隐藏层，并且信息的流动方向只能是由输入层经隐藏层流入输出层，整个网络中并不存在循环。

单层感知器和多层感知器是前馈神经网络的两个基础模型，单层感知器不包含隐藏层，输出层接受输入层的信息经过计算直接输出；一个多层感知器（MLP）包含一个或多个隐藏层，两者最显著的区别就是单层感知器只能做线性计算，而多层感知器则可以学习非线性函数。

### 反向传播（BP）算法

多层感知器的学习过程被称作反向传播算法，模型预测的结果与实际标签的距离方程称为代价函数（cost function）。反向传播算法是一种监督学习的方法，初始时相邻层节点间的连接权重是随机的，算法对每一个输入通过激活函

数来观察输出并与标签作比较，误差反向传入前一层，算法根据误差校正权重，重复过程直到误差在设定的阈值之内。

### 2.1.2 卷积神经网络

卷积神经网络（Convolutional Neural Networks,CNN）是前馈网络的一种，在图像处理方面具有很大优势，本文选取的 FaceNet 中的图像处理就是基于 CNN 的。CNN 与传统神经网络不同的是其采用了局部连接和权值共享的方式，有效解决了传统神经网络参数过多造成效率低训练困难、大量参数导致过拟合以及图像展开造成丢失空间信息的缺点。

#### 卷积神经网络结构

卷积神经网络主要由输入层、卷积层、ReLU 层、池化层和全连接层构成，通过这些结构的叠加构成一个完整的卷积神经网络。通常将卷积层和 ReLU 层共同称为卷积层，所以卷积层卷积操作需要经过激活函数。卷积层提取图像的各项特征，池化层则对信息进行抽象（即通过求取一个视窗内数据的特征值）以减少训练参数。

#### 卷积层

卷积层是卷积神经网络构建的核心，它产生整个网络中的大部分计算量。卷积层的参数是由一些可学习的滤波器集合组成，滤波器是高度和宽度上相对图像而言都很小，但是深度和输入数据一致。滤波器也被称作卷积核，卷积核中的参数相当于整个网络中的共享权值参数，对应的局部像素点做卷积核与像素值的乘积（通常还要加上一个偏置参数），得到卷积层的一个输出点值。

#### 采样层

经过卷积层处理的数据，相对原数据已经缩小了很多，但是仍然要面对因巨大的计算量而导致的过拟合问题。为解决这个问题需要对卷积层的结果进一步做池化处理，通常有两种池化方式，最大值 max-pooling（用池化窗口的最大值作为采样值），均值 mean-pooling（用池化窗口中所有值加权平均作为采样值）。

## 第二节 CUDA 简介

CUDA[3]是 NVIDIA 利用 GPU 平台进行通用并行计算的一种架构，包含

CUDA 指令集架构和 GPU 内部并行计算引擎。开发者可以利用 C 语言、OpenCL、Fortran、C++ 等编写程序，CUDA 提供 C 编译器使得 CUDA 程序可以直接在 GPU 上运行。

在 CUDA 架构中程序可以分为两部分，主机（host）端和设备（device）端。Host 端是在 CPU 上运行的部分，device 端始在 GPU 上运行的部分。其中 device 端程序又被称为核函数（kernel），host 端将数据传入 device 端，由显示芯片执行 kernel 函数来完成计算并传回 CPU，整个 CUDA 程序由一系列 kernel 设备端函数以及 host 端函数组成，设备端函数并行执行，主机端串行执行。CPU 与 GPU 协调工作，CPU 同时负责调节 GPU 的工作和进行逻辑性强、串行化的计算，GPU 则执行高度并行化的计算任务。由于 CPU 与显示设备之间是通过计算机总线传送数据，带宽有限，因此在数据传输时间代价小于 GPU 加速时间才会提高计算效率。

在 CUDA 架构中最小单位是 thread，一组 thread 组成一个 block，一个 grid 中包含一组 block，不同 block 中的 thread 不能存取同一个共享内存，因此这些不同 block 中的 thread 无法直接互通或进行同步。

Kernel 是在 GPU 上执行的核心程序，这个 kernel 函数是运行在某个 Grid 上的。一个核函数 kernel 只能运行在一个上 Grid。每个 block 和每个 thread 都有自己的 ID，通过相应的索引可以找到对应的线程 ID（threadIdx）和线程块 ID（blockIdx）。Block ID 为一维或二维，Thread ID 为一维、二维或三维。

#### CUDA 中两个处理器：流处理器 SP 和 SM

SP（streaming processor）：CUDA 最基本的处理单元，也称为 CUDA core。最后具体的指令和任务都是在 SP 上处理的。GPU 进行并行计算就是很多个 SP 同时并发地运行。

SM（streaming multiprocessor）：多个 SP 加上其他资源如线程调度表、寄存器、共享内存等，组成一个 SM，也叫 GPU 大核。SM 是 GPU 的核心，其中寄存器和共享内存是 SM 的稀缺资源。CUDA 将这些资源分配给所有驻留在 SM 中的线程。有限的资源就限制了 SM 中线程，因此也限制了 GPU 的并行能力。GPU 的线程结构如图 2.3。

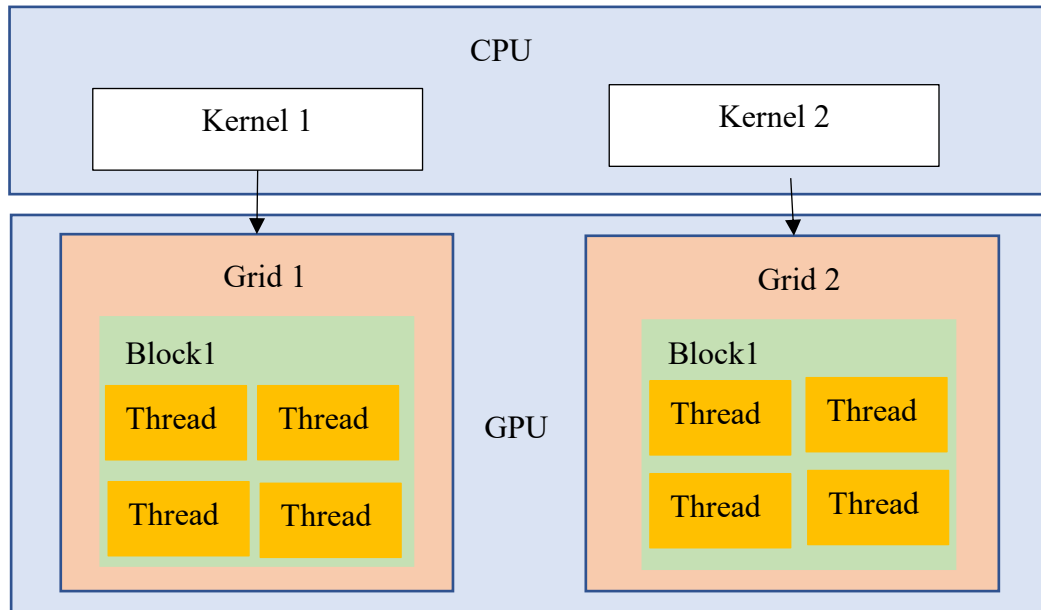


图 2.3 CUDA 线程结构图

### CUDA 中的内存结构

每个 thread 都有一组 register（寄存器）和 local memory（局部内存），在同一个线程块 block 中的线程可以共享一块 shared memory（共享内存），所有的在一个网格（grid）中的线程共享一组全局内存（global memory）、常量内存（constant memory）和纹理内存（texture memory），每个网格中都有一组各自的 global memory、constant memory 和 texture memory，不同的线程都可以使用这些内存。GPU 中所有内存的访问速度从快到慢依次是 register、local memory、shared memory、global memory[3]。GPU 的内存结构如图 2.4 所示。

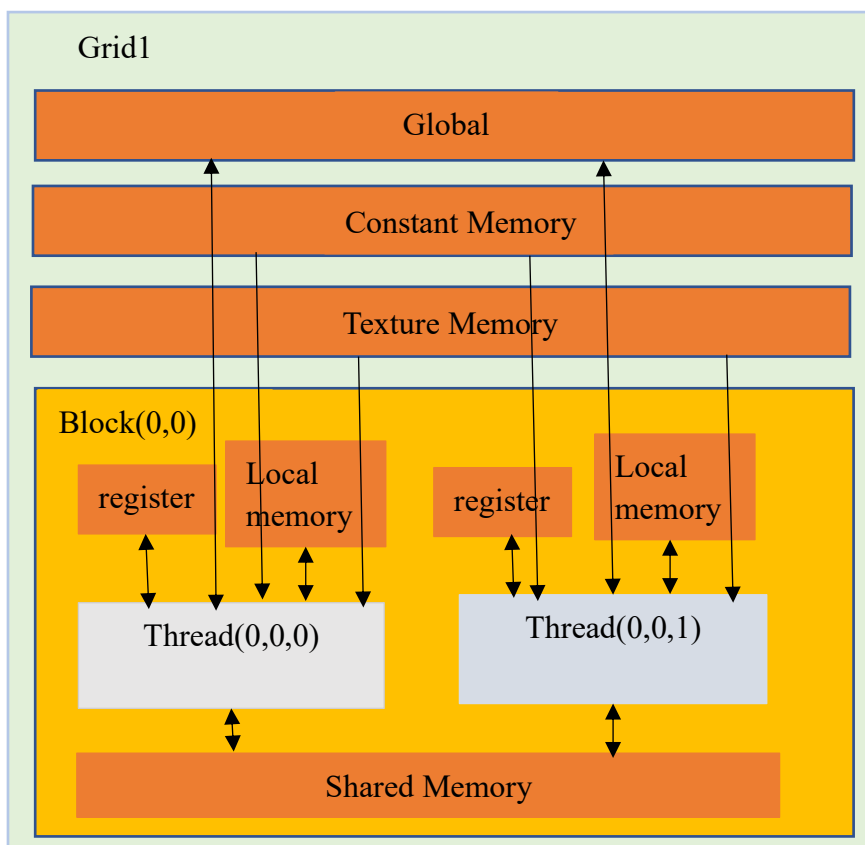


图 2.4 CUDA 内存结构图

不同于 CPU，在 GPU 中没有多级缓存的结构，GPU 中通过线程的调度来隐藏读取内存时的等待时间（latency），即当第一个线程进入读取内存等待阶段，通过线程调度开始执行第二个线程。

#### CUDA 编程模型及 python 环境中 CUDA

CUDA 中通过关键字来标识函数在设备还是在主机上运行，`__device__` 定义的函数只能在设备上运行和被调用；`__global__` 定义的函数在 CPU 上调用，在设备上被执行，且返回值类型必须是 `void`；`__host__` 定义的函数只能在主机 CPU 上来调用和执行。

CUDA 通过 `cudaMalloc()` 为设备端分配全局内存，并通过 `cudaFree()` 释放内存，通过 `cudaMemcpy` 函数进行设备端和主机端数据传输。

本文中是在 python 语言环境下进行编程，在 python 中为 cuda 程序编写专门提供了一个 CUDA 的 python 版本 PYCUDA API，开发者可以植入自己编写的 kernel 函数，内核函数采用 C/C++ 编写的。PyCUDA 中分配设备内存以及主机端与设备端的数据传输都是通过 GPU 驱动 `pycuda.driver` 来实现。



### 第三节 FaceNet 原理

FaceNet 是由 Google 开发的人脸识别系统架构，不同于传统的 softmax 方式分类学习，FaceNet 抽取一层作为特征然后直接做端对端的图像欧式距离编码，基于这个编码进行人脸识别、人脸验证以及人脸聚类。FaceNet 直接使用基于 triplets（三元组）的最大边界近邻分类的损失函数来训练神经网络，网络直接输出 128 维的空间向量。FaceNet 的模型如下图 2.5[1]：



图 2.5 FaceNet 模型图

#### 人脸嵌入与 triplet（三元组）loss

嵌入(embedding)：用  $f(x) \in R^d$  表示，它将图像  $x$  嵌入到  $d$  维的欧几里德空间中。另外，我们将这种嵌入限制在  $d$  维超球面上，即  $\|f(x)\|_2 = 1$ 。由于两幅人脸图像在高维空间中不易发现两者之间的明显特征，而在低维的嵌入空间他们之间的特征差异就会比较明显，因此 FaceNet 是直接将人脸图像输出为紧凑的 128 维嵌入向量，再使用基于 LMNN(Large margin nearest neighbor)的三元组 loss 训练参数。

因此两张图像的处理方式也就化为：首先仅对两个嵌入之间的距离进行阈值处理；然后人脸识别便转化为 k-NN 分类问题，人脸聚类也通过聚类算法如 k 均值或 agglomerative 聚类来实现。

三元组损失(triplet loss)：首先得到计算 embedding 的映射  $f(x)$ ，使相同或相近的图像之间的欧式平方距离尽可能小，不同或来自不同人脸部图像之间的欧式平方距离尽可能大，而且反例也应比正例的距离小于阈值，通过对两者之间的欧式距离的判断也就反映出这组图像中哪些照片出自同一个人。

三元组是由两个匹配的人脸缩略图（thumbnail）和一个不匹配的人脸缩略图组成。缩略图是人脸区域的紧凑裁剪，除了缩放和平移之外，不进行二维或三维空间上对齐。

期望的三元组约束：

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2 \quad \forall (f(x_i^a), f(x_i^p), f(x_i^n)) \in \tau \quad (2.3)$$

$\alpha$  是正负图像对之间区分界限的阈值， $\tau$  是训练集中所有可能的三元组的集合

具有基数  $N$  的三元组损失函数：这是基于 LMNN 的三元组损失函数，并且计算损失值的目的是将正对与负对分开一个距离阈值，损失函数计算如下。

$$L = \sum_i^N [\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha] \quad (2.4)$$

三元组分为三类，简单三元组（easy triplets）、半困难三元组（semi-hard triplets）、困难三元组（hard triplets），简单三元组是损失为零的三元组，半困难三元组是负例比正例远离目标点但是仍然有大于零的损失，困难三元组是负例比正例靠近目标点。很少的数据就会产生许多三元组，因此正确构建三元组影响效率，简单三元组不需要经过训练就很容易识别，而困难三元组又会在训练开始阶段产生局部值太小的问题，导致训练模型崩溃。传统方法是从所有样本中找出最远正例和最近负例，但是耗时大且坏点影响大，模型误差过大。有两种方法解决这种问题，一种是每  $n$  步生成一个三元组，使用最新检查点并找出数据子集上的最值，第二种是 FaceNet 中所采用的在线生成三元组，即在训练的开始，在每个 mini-batch 中筛选所有符合公式 2.3 的正负例样本，初始选择部分图像为正例，然后随机用其他图像作为负例样本，如果出现不正确的负例，将会导致训练过早进入局部最小。

## 第三章 SFRC 系统设计

### 第一节 SFRC 优化原理

本文中开发的基于 CUDA 和 FaceNet 的单人脸识别系统(SFRC)结构上分为三个部分, 分别是基本矩阵运算部分, 卷积神经网络部分和图像训练部分。

第一部分是基本算数计算和矩阵运算 operators 部分, 主要做图像存储矩阵的加、减、乘、求均值, 地板除, 降维求和, 取模运算。所有的矩阵处理是图像数据的处理, 该部分优化主要采用与线程块等大共享内存的访问方式进行内存访问优化, 具体优化如 3.1.1 小节所述。

第二部分是卷积神经网络部分, 卷积神经网络部分负责进行卷积层计算、池化层均值 mean-pooling 池化和最大值 max-pooling 池化运算以及 ReLU 整流处理, 这部分优化 FaceNet 中从三维图像到一维向量的映射过程。该部分优化方式主要采用线程块加卷积窗口大小的共享内存访问的方式做内存优化, 具体优化如 3.2.2 小节所述。

第三部分是图像训练, 该部分则采用构建 CPU/GPU 内存地址映射表来进行优化。训练开始以三通道方式读取图像以及图像的标签, 构造映射函数, 将三维图像映射到欧几里得空间向量, 然后整个训练过程以所得的空间向量作为数据进行训练, 训练过程在 GPU 中完成计算, CPU 通过一个内存指针来控制图像数据在整个训练中的流向。最后在训练完成时通过 CPU 中保存的结果地址指针从 GPU 中读取训练得到的所有参数, 并完成 GPU 中地址表所对应内存的参数。

整个系统对原 FaceNet 的改进模式包括两个方面:

1. 第一种是 CPU-GPU 模式。具体步骤为在 CPU 构建一个独有的 GPU 内存管理映射表 M, 在 CPU 逻辑端需要 GPU 加速时, 从一开始 CPU 向 GPU 中传入图像数据, GPU 端就保存这部分数据, 并且 CPU 端通过逻辑控制保持对这部分数据管理, 只有当 CPU 发出释放内存信号, GPU 才对该块内存释放, 整个过程中 GPU 对数据内存的访问都是直接访问 GPU 中已保存的内存块。
2. 第二种是在 GPU 内部内存访问的优化。基于第二章第二节中所述 GPU

的内存结构，在以 CUDA 编写的 GPU 加速程序中，采用共享内存进行对模块内部数据管理，采用列优先的方式读取全局内存，从而达到尽可能减少 GPU 内部的内存读取时间消耗。GPU 中共享内存的读取有两种模式，一种是窗口（也叫感受野）大于 1 的方式，另一种是窗口等于 1 的方式，根据这两种不同的访问模式，设计出了两种共享内存读取模型。具体两种模式的优化细节如节 3.1.1 阐述。

### 3.1.1 GPU 内部访存优化

#### 矩阵运算内存访问优化

在整个架构中，GPU 线程块大小都设置为  $32 * 32$  大小。SFRC 中涉及的矩阵运算统一采用线程块和网格都是二维的结构，这样线程和线程块的空间索引将与图像像素空间坐标一一映射，方便程序的内存访问与图像计算。

如第二章中 GPU 内存访问速度所述，在 GPU 中全局内存的访问速度要远大于共享内存的访问速度，且涉及的矩阵基本运算中数据量大，在一个线程块 block 中无法读取全部数据，而且这些数据在同一个 block 的每个线程中将会多次被访问到，例如在本文中进行的三维图片数据到 128 维向量的映射中涉及的矩阵乘法、卷积计算中卷积核的读取均存在线程间的数据共享。传统方式采用的是读取全局内存来获取数据，而且因为算法结构问题，需要多次重复的读取，SFRC 内存管理结构通过在 GPU 中申请一块共享内存来保存卷积核等这些需要进行重复读取和写入操作的数据，避免直接从 GPU 全局内存中读取，由此提高 GPU 内部内存访问速度。本文中提出下列算法 3.1 来解决上述提到的共享内存问题。

表 3.1 GPU 内部数据加载优化算法

算法 3.1: GPU 内部数据加载优化算法 1

Input:

数据 data, data 的尺寸 width、height, 读取窗口尺寸 filter(x, y),  
线程号 threadIdx(x,y), 线程块大小 blockDim(x, y)

程序主体:

Initialize shared\_memory size(x, y) ← (0, 0)

size(x, y) ← (blockDim.x + filter.x, blockDim.y + filter.y)

申请 size 大小的 shared\_memory

if threadIdx.x > 0 | threadIdx.y > 0 | threadIdx.y < height | threadIdx.x < width do  
    shared\_memory[i, j] ← data[index] 输出:

输出池化结果 out

以在训练中，比较两张图片之间距离时所用到的求矩阵行或列的最大值 `reduce_max` 为例分析，如图 3.1 所示

```

shared_Max[tid]=d_a[tid];
__syncthreads();
for(int stride = blockDim.x / 2; stride > 0; stride >>= 1)
{
    __syncthreads();
    if(tid < stride)
        if(shared_Max[tid] < shared_Max[tid+stride])
        {
            int temp = shared_Max[tid];
            shared_Max[tid] = shared_Max[tid+stride];
            shared_Max[tid+stride] = temp;
        }
}

```

图 3.1 视窗为 1 的共享内存访存示例

基于前文介绍的 CUDA 对内存的管理模式，以列优先的方式读取可以加快读取速度，所以这里以列优先的方式并采用二分法读取来减少内存读取开销，在图 3.1 中，求取最大值的时候，申请一块共享内存 `shared_Max` 保存行或列中最大值，以二分法比较求取最值并更新 `shared_Max` 中对应值，到求得最终的最大值，再通过共享内存 `shared_Max` 将数据赋值写入到输出值对应的内存 `out` 中。

### 卷积内存访问优化

SFRC 对 FaceNet 卷积过程的优化包含如下两点：

1. SFRC 对传统 FaceNet 优化首先在于做卷积的这部分数据在进行卷积之前的数据准备部分已经传入 GPU 中，不需要再次传入，只是在 CPU 端维护的 GPU 内存地址映射表 `M` 中保存指向这部分数据的 GPU 内存指针，让 GPU 知道从哪里读取数据。
2. SFRC 在做卷积时与传统的 FaceNet 卷积一样提供 SAME 和 VALID 两种图像填充模式，但是不同之处在于，在传统 FaceNet 中通过 TensorFlow 做卷积，每次会先对原始图像数据进行填充（被填充部分数据值为 0），填充之后再再将这部分数据传入 GPU，这样就会产生一个问题，填充的这部分数据不具有计算意义却占有大量 GPU 和 CPU 内存，同时浪费 CPU 向 GPU 数据传输带宽。因此，在 SFRC 模型中，GPU 端和 CPU 端实际上都不对图片数据进行真实的填充，而是通过前文设计的共享内存管理算法，在 GPU 中申请一块共享内存 `shared_data`，在这块内存中保存当前线程块中需要用到的图像数

据，而且如果该部分数据是填充数据，只需将 `shared_data` 赋值为 0，通过这种虚拟的方式进行数据填充。但这样又会产生一个问题，即在当前线程块中需要的数据尺寸要比线程块尺寸大。共享内存的大小不知道，而且无法知道当前所需的图像数据是否包含被填充数据，因此，为解决上列问题，SFRC 在进行卷积时 CPU 需要告诉 GPU 实际上原始的数据大小  $S_0$  和填充后的逻辑上数据块的大小  $S_1$ 、卷积核大小  $s$  和线程块大小  $s_1$ ，在 GPU 内部卷积操作时首先判断当前线程号是否超出  $S_0$  且小于  $S_1$ ，如果不是说明该部分对应数据是原始真实的数据值，通过读取全局内存为 `shared_data` 对应地址数值赋值为真实数值；如果是，说明这部分数据是被填充数据，则不进行读取操作，直接为 `shared_data` 对应地址数值赋值为 0；共享内存大小则设置为  $s$  与  $s_1$  之和。当前线程在为共享内存赋值时首先需要判断当前线程在线程块中是否属于边缘线程（即线程 ID 处于线程块的边位置），如果是则该线程负责读取在 `shared_data` 中本身线程号对应位置的数据以及超出自身位置的数据（即在该线程对应位置之前或之后的数据），而如果当前线程对应位置属于内部线程，则只读取自身位置的数据。共享内存的读写方式如图 3.2 所示。

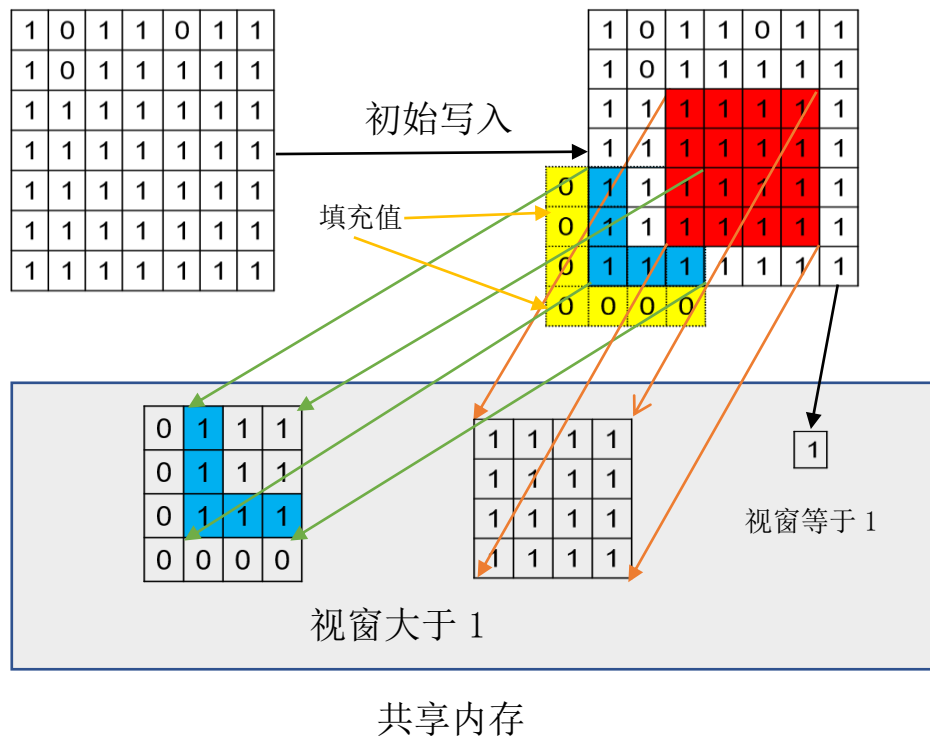


图 3.2 GPU 内部共享内存读写的两种模型

共享内存的读写具体算法如表 3.2

### SAME 和 VALID 两种填充方式下数据尺寸的计算

1、SAME 方式：在输入图像边缘填充若干圈 0，使得滤波器能够对图像边缘的数据进行卷积，从初始位置能以步长为单位刚好滑到图像末尾位置，也就是填充后的图像尺寸能够被步长整除。卷积后图像尺寸计算如公式 3.3：

$$H = \lceil \frac{h}{s} \rceil \quad W = \lceil \frac{w}{s} \rceil \quad (3.1)$$

$H$ 、 $W$ 表示卷积后输出图片的高度和宽度， $h$ 、 $w$ 为原图像高度和宽度， $s$ 表示步长， $\lceil \cdot \rceil$ 表示向下取整。

填充大小计算如下：

$$pad_h = (H - 1) * s + f - h \quad (3.2)$$

$$pad_w = (W - 1) * s + f - w \quad (3.3)$$

$$l = \left\lfloor \frac{pad_w}{2} \right\rfloor \quad r = pad_w - l \quad (3.4)$$

$$t = \left\lfloor \frac{pad_h}{2} \right\rfloor \quad b = pad_h - t \quad (3.5)$$

其中 $pad$ 为需要填充的尺寸， $H$ 为输出图像的高度， $f$ 为滤波器大小， $h$ 为原始图像高度， $s$ 为步长， $pad_h$ 为高度上需要填充的大小， $pad_w$ 为宽度上需要填充尺寸， $l$ 为左边添加像素数， $r$ 为右边添加像素数， $t$ 为上方填充像素数， $b$ 为下方填充像素数， $\lfloor \cdot \rfloor$ 表示向下取整

2、VALID 方式：不对图像边缘进行任何填充，当滤波器不能以步长为单位滑到末尾时，舍弃末尾数据，卷积后图像尺寸计算如下：

$$H = \left\lfloor \frac{h-f+1}{s} \right\rfloor \quad (3.6)$$

$$W = \left\lfloor \frac{w-f+1}{s} \right\rfloor \quad (3.7)$$

$H$ 、 $W$ 表示卷积后图片高度和宽度， $h$ 、 $w$ 为原图像高度和宽度， $f$ 表示滤波器大小， $s$ 表示步长， $\lfloor \cdot \rfloor$ 表示向下取整

表 3.2 卷积操作共享内存读取算法

---

#### 算法 3.2：GPU 内部数据加载优化算法 2

---

输入：

图像数据  $X$ ，步长  $s$ ，滤波器  $filter$

程序：

续表 3.2

1: Initialize  $H$ ,  $W$ ,  $l$ ,  $r$ ,  $t$ ,  $b$ , 线程号  $(x, y)$ ，block 中线程 ID 为  $tx$ ，初始化空的共享内存块  $sX$ ,  $pad_h$ ,  $pad_w$

---

---

```

2: if y<h && x<w then
3:   sXtx=Xx,y
4: else
5:   sXtx=0
6:   end if
7: if x mod s is not 0 | y mod s is not 0
8:   Return
9: if y<h+ padh || x<w+ padw do
10:   窗口内加权平均和 sum
11:   i=y/s j=x/s
12:   if i<H,j<W do
13:     outi,j=sum
14:   end if
15: end if
输出:
卷积结果 out

```

---

在文中涉及到需要卷积核的计算，如卷积和池化中，GPU 内部的内存管理均采用算法 3.2 进行优化，原始数据均采用已保存在 GPU 中的数据，在计算过程中需要多次访问与写入则采用共享内存管理。

### 3.1.2 图像训练与 CPU-GPU 访存优化

图像训练部分算法采用 FaceNet 提出的三元组进行梯度训练，SFRC 针传统 FaceNet 优化方式主要是在 CPU 端维护一个数据地址集合  $M$  保存 CPU 端数据地址（也是从 CPU 来看 GPU 中的逻辑地址），同时在 GPU 中也维护一个相同的内存地址集合  $M_0$ （保存真实的 GPU 中数据的地址）， $M$  与  $M_0$  中的地址一一对应。在整个系统中通过维护  $M$  与  $M_0$  来管理 GPU 中数据的存放、阶段训练中产生的数据和训练产生的参数。映射表模型图 3.3 所示。



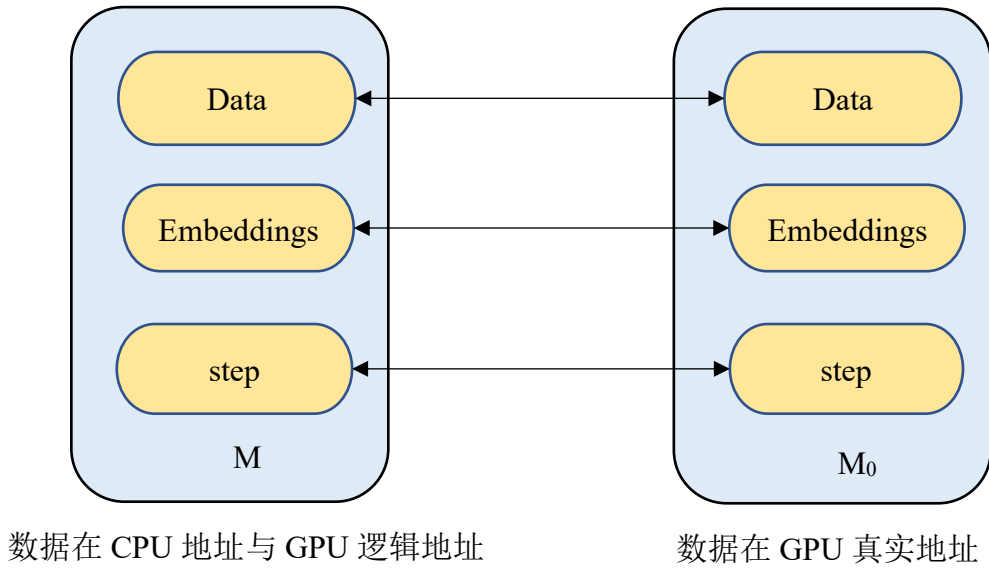


图 3.3 地址映射表模型图

训练需要的数据只进行一次必要的初始化传输（从 CPU 传入 GPU），之后通过该地址集合  $M$  与  $M_0$  的映射来管理训练过程产生的参数以及对原始数据的更新。在所有训练完成后才释放 GPU 中  $M_0$  中地址占有的内存。

具体优化过程分为两部分：

1. 首先是在训练的开始通过 `opencv` 从文件系统读入本次训练需要的所有图片作为数据集，并对图片进行裁剪、随机翻转和灰度化等预处理，然后 CPU 端通过 CUDA 提供的接口在 GPU 中申请一块和 CPU 中同等大小的内存块，并在 CPU 端维护的 GPU 内存地址集合  $M$  中加入该内存块地址，CPU 启动向 GPU 传输数据进程将 CPU 中图像数据传送到 GPU 中对应的内存中，再初始化学习率、训练梯度 `step` 等参数。在 GPU 程序模块中构建所有训练所需要的 `kernel` 函数，包括三维图像到 128 维向量的卷积映射操作  $f_1$ ，映射的参数训练操作  $T$ 。

传统 FaceNet 在每个训练轮次  $T$  中都需要进行一次  $f_1$  操作，而且训练开始需要从 CPU 向 GPU 传输训练需要的图像数据，而在 SFRC 中进行  $f_1$  操作时不再需要向 GPU 中传入数据，直接在 CPU 端通过 GPU 地址映射集合  $M$  控制 GPU 内存读取图像并通过  $f_1$  将所有图像的数据映射为 128 维向量，且只进行一次  $f_1$  操作， $f_1$  映射结束后，用图片向量 `embeddings` 的地址更新  $M$  中图像数据对应地址，将该地址作为下文所有训练的图像数

据, GPU 中凡是需要读取 `embeddings` 的操作都避开 CPU 直接从 GPU 地址集合  $M_0$  获取。训练结束后 CPU 从 GPU 中读取训练得到的 `step` 和映射过程参数等数据, 并保存在 `model` 文件中。

2. SFRC 相对传统 FaceNet 的第二个优化在于, 当进行第二个人的图像训练的时候, 首先在 CPU 端生成一个人脸图像在数据集 `dataset` 中的地址列表, 然后与上一个人的地址列表作比较, 如果两个人的图像有重复, 剔除新数据集中的重复部分, 只保留不同的图像数据构成 `new_data`, CPU 将这部分数据传入 GPU 并在 GPU 中进行欧几里得空间向量的映射计算  $f_1$ , 替换 GPU 中原有的 `embeddings` 中不相同的向量保留相同的向量, 同时 CPU 向 GPU 传入上次训练得到的 `step` 等参数进行新的训练。当训练结束后, CPU 重新从 GPU 中传回 `step` 等参数, 并更新 CPU 中保存的 `step` 等参数, 保存在 `model` 文件中。

当所有人的训练结束, 将训练得到的参数写入文件后, 释放 GPU 中在  $M$  中保存的所有内存。

## 第二节 SFRC 模型训练流程

在 CPU 中构建所有的逻辑函数, 包括向量映射、三元组选择和三元组训练, CPU 中所有的计算操作都是调用 GPU 中的计算函数来实现。在 GPU 中构建所有的实际的计算操作, 包括向量映射和训练时需要用到的卷积、池化、损失计算、正则化等函数, GPU 中的程序为 CPU 中函数通过 CUDA 提供了一个运行接口。整个 SFRC 系统训练流程如图 3.4 所示, 图中蓝色线条表示输入, 灰色线条表示输出, 黑色线条表示 CPU-GPU 内存数据传输, 红色线条表示函数的调用。



重复上述步骤 2 与步骤 3 直到训练完整个 **batch**，完成训练。当完成整个的训练之后，CPU 从 GPU 读出模型参数，保存入模型文件中，通过地址表 **M** 释放 GPU 中的所有内存，退出训练。

## 第四章 实验与结果分析

本节实现了第三章所提出来的单人人脸识别 SFRC 系统，并且根据第三章提出的模型，选取合适大小的数据集，通过运行传统的 FaceNet 模型和本文中的 SFRC 模型对同一数据集训练，并在 LFW 数据集上进行验证正确率等指标来评测优化效果。

### 第一节 实验环境与参数

本文中的实验运行硬件平台参数如下表 4.1.

表 4.1 实验硬件参数

项目	属性
操作系统	Linux version 3.10.0-1062.4.1.el7.x86_64
CUDA 版本	10.1
Pycuda 版本	2019.2.1
Python 版本	3.6.8
CPU 型号	Intel(R) Xeon(R) Silver 4210
CPU 核心数	10
GPU 型号	NVIDIA Corporation GM107GL
GPU 数量	4

本文中 CPU 中全部代码使用 python，GPU 中运行代码采用 python 中为 CUDA 提供的 pycuda 模块使用 CUDA 编写，并在表 4.1 硬件环境下运行。

本文中进行训练所使用的数据集以及训练情况所使用的训练集数据为 LFW 人脸数据集中前 2016 个人的人脸图像，并经过 MTCNN 算法人脸检测并对其，图像大小为 $160 * 160$ 。所有图像循环训练两次。每次训练数据集为随机的 1800 张图像数据。每一个批 batch 中为 90 张图片，整个数据集划分为 20 个 batch。实验中分别运行 SFRC 模型 FaceNet 模型对该数据集训练，程序在训练过程记录每个 batch 训练的时间数据并保存在日志文件，之后通过将这些时间信息绘制成表来比较两种模型训练消耗时间的关系，并在 LFW 数据集上进行验证准确率等参数，比较两种模型的性能。

## 第二节 实验结果

首先在第一节所述硬件配置和参数下对数据集中图片进行图像训练, 在训练过程中分别追踪在每个 batch 训练中符合三元组条件的图像训练所消耗的时间(train\_time)、三元组数量(triplets number)、三维图像到 128 维向量的映射时间和三元组选择的时间(select triplets time)三个指标, 三元组训练花费时间情况结果如图 4.1 所示。

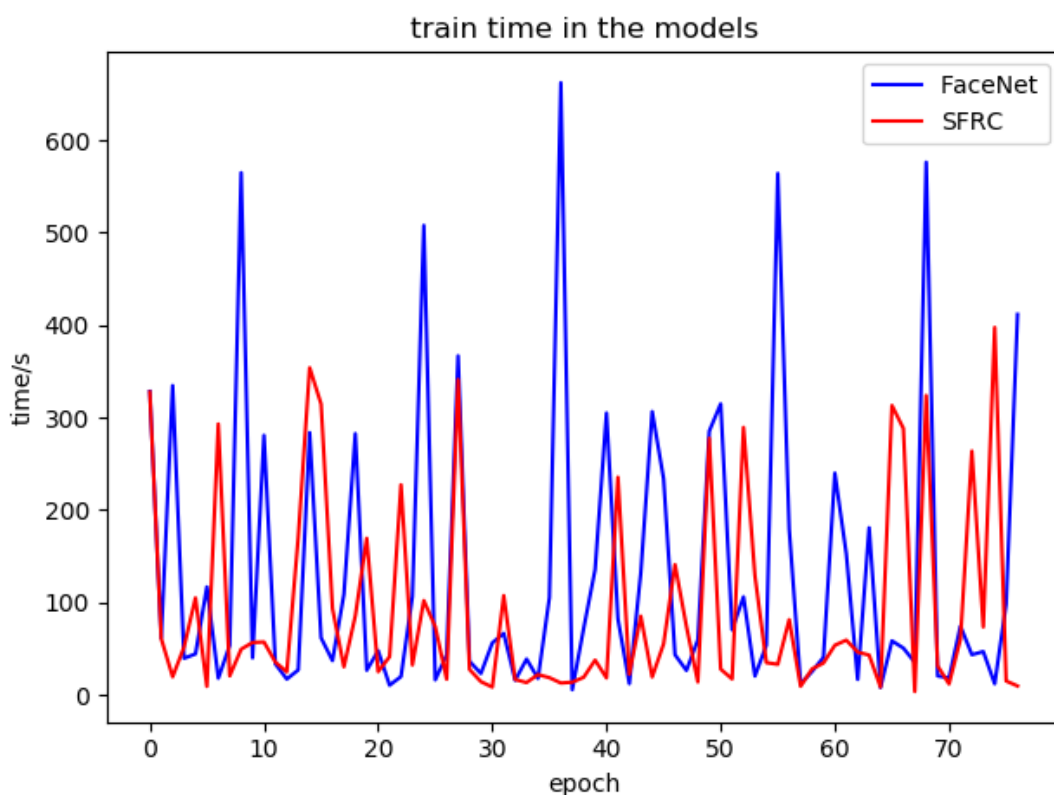


图 4.1 每个 batch 训练消耗时间图

从图 4.1 可以看出, 图像上整体 SFRC 对应曲线(红色曲线)要比 FaceNet(蓝色曲线)对应曲线低, 即 SFRC 训练消耗的时间要小

将两种模型在向量映射以及进行三元组选择消耗的时间绘制成图, 结果如下图 4.2 所示,

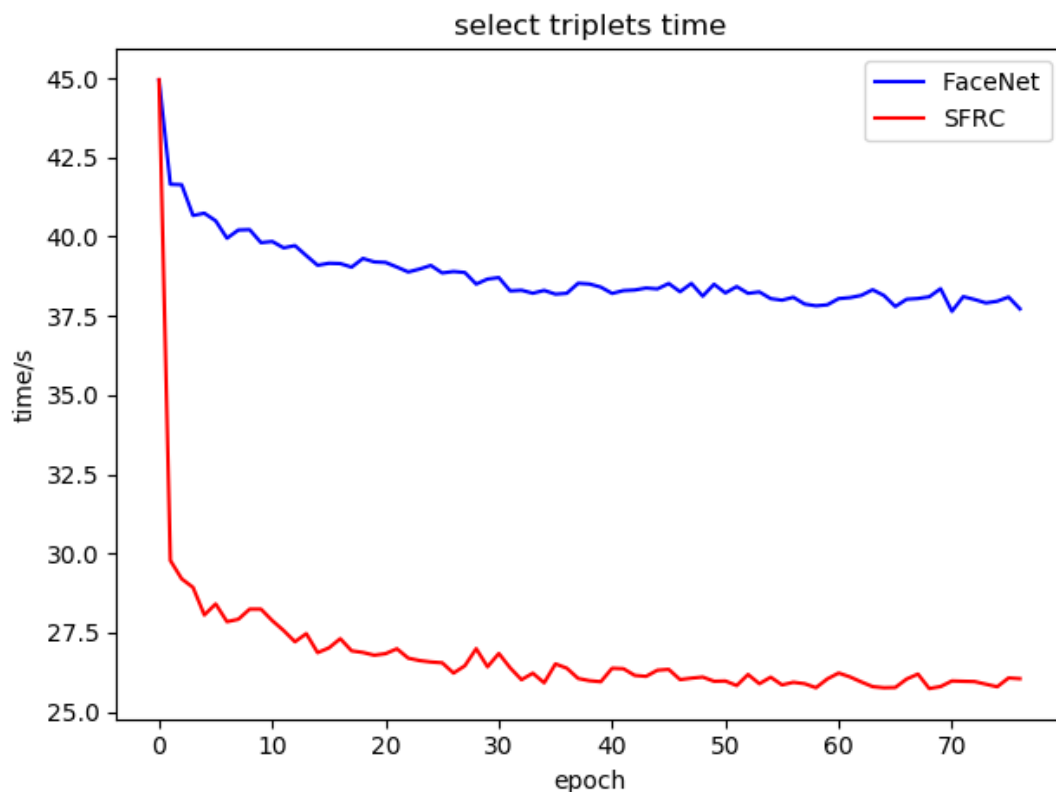


图 4.2 三元组选择和向量映射时间图

从图 4.2 中可以看到 SFRC 的曲线（红色曲线）要低于 FaceNet 对应曲线（蓝色曲线），但是曲线的开始是几乎重合的，分析 SFRC 和 FaceNet 对数据的读取可以分析得到，SFRC 只在训练的开始从 CPU 向 GPU 中传入数据，之后都是经过地址映射表 M 管理数据内存，而 FaceNet 是每个轮次训练都要通过 tensorflow 来从 CPU 向 GPU 传入数据，并且这部分数据是表示图像的  $160 \times 160$  的三维数组，数据量大，因此两者只在训练的开始时间几乎重合，之后的训练中由于 SFRC 不进行 CPU 与 GPU 数据传输，因此时间消耗要远低于 Facenet，两个曲线分离并且相差较大。

每次训练的三元组数量比较如下图 4.3

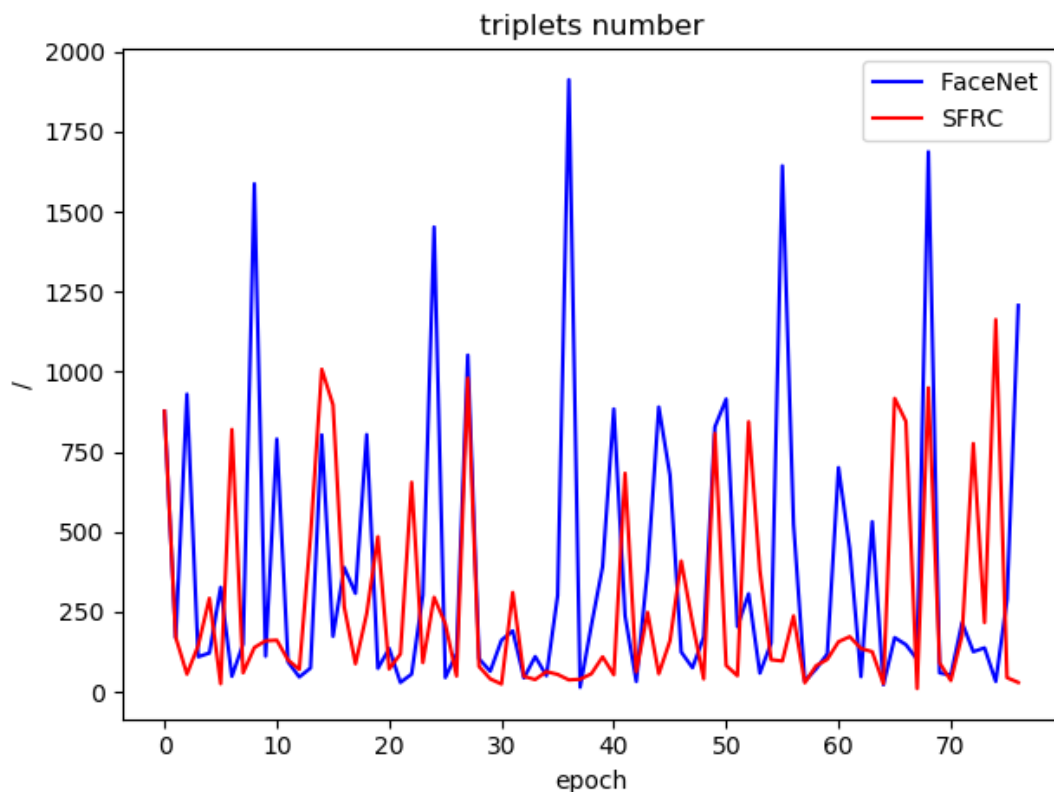


图 4.3 每个 batch 的三元组数量

从图 4.1 和图 4.3 来看，整体上 SFRC 的训练速度要比 FaceNet 快，但是在图中却也出现 Facenet 模型比 SFRC 所花费更少时间的情况，分析算法结构，因为在进行图片的选择时采用的是随机的生成一个 batch 来进行训练，因此在进行梯度训练的时候会出现符合条件三元组的数量差异，而三元组数量是梯度训练的关键参数，直接影响训练时常，为更加直观的观察到两者之间的差异，对两种模型下求取平均每个三元组训练花费的时间，结果如图 4.3 所示。



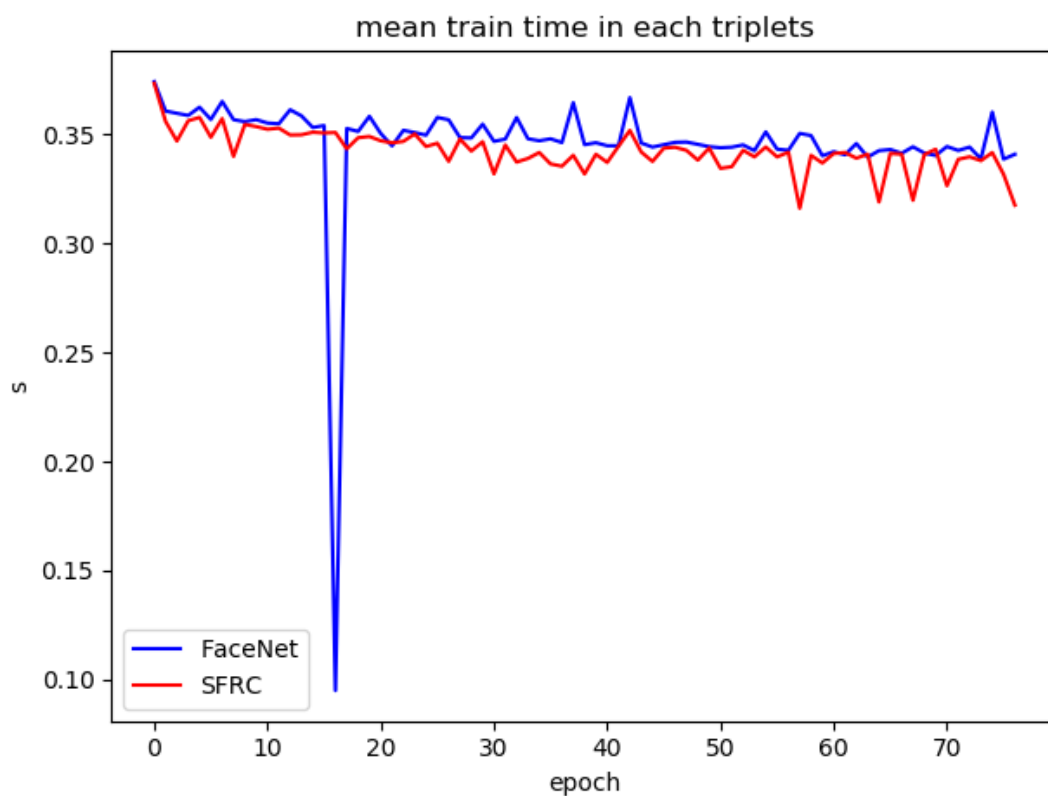


图 4.4 平均每个三元组训练消耗时间

从图 4.4 中可以观察到出现一个反常点，分析算法，该点的产生是由于在本次训练中出现训练奔溃提前终止本轮训练造成的，属于坏点，剔除坏点之后平均每个 triplet 训练花费时间如图 4.5 所示。

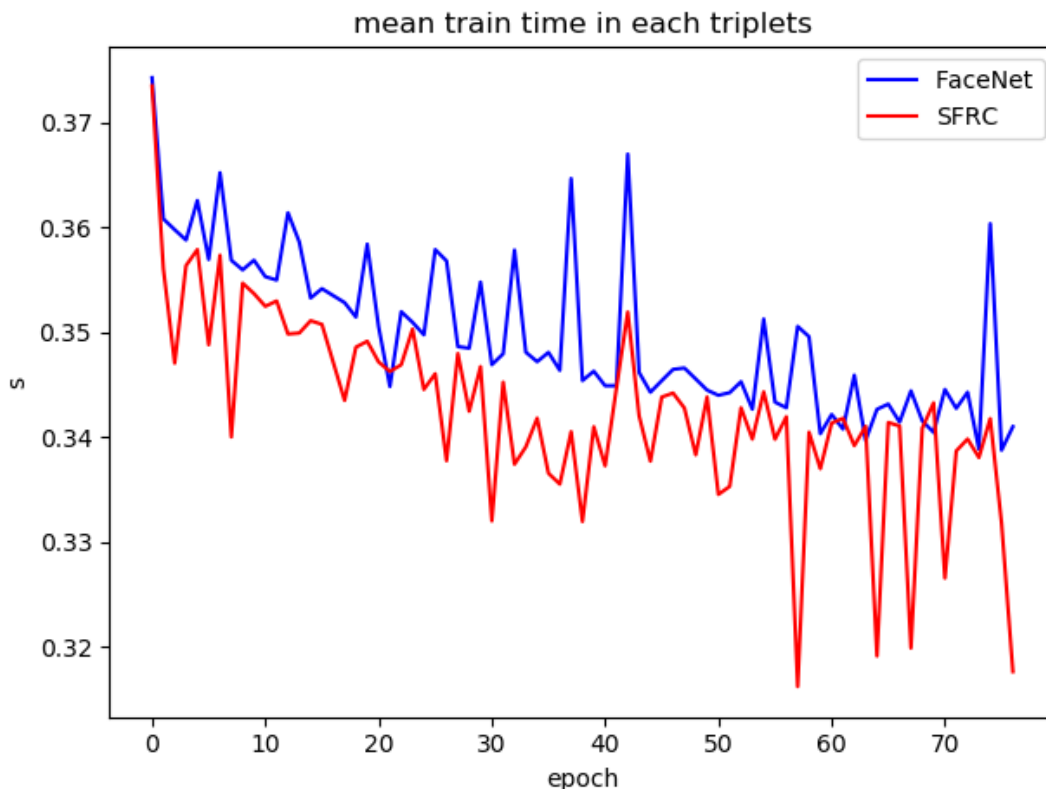


图 4.5 剔除坏点后的 triplet 平均训练时间图

忽略坏点之后可以发现 SFRC 消耗的时间要略低于 FaceNet 模型消耗的时间。从整体上来看, SFRC 模型对 FaceNet 模型优化在时间上产生了一定的效果。

进一步分析比较在训练和向量映射这两部分对 FaceNet 的优化效果, 因此求取两种模型优化的比率, 结果如表 4.2.

表 4.2 SFRC 平均优化率

项目	平均优化比率
训练时间	2.21%
三元组选择	30.83%

可以看到 SFRC 在向量映射部分对 FaceNet 的优化效果要比在训练部分做的优化效果更佳。分析两者这两种情况下的数据形式可以得到, 向量映射中 CPU 与 GPU 之间传输的数据是  $160 \times 160 \times 3 \times \text{int}32$  格式的真实图像数据, 而在训练部分, CPU 与 GPU 之间传输的数据为  $128 \times \text{float}32$  格式的数据, 两者比较可以得到向量映射时的数据集所占内存要更大, 系统花费在 CPU 与 GPU 数据交换上的时间也就占比更大, 因此相对训练部分的优化效果要更佳。

从时间消耗上可以得出 SFRC 模型对 FaceNet 的优化效果良好。

为判断优化后是否对结果的正确率产生影响,再在 LFW 数据集上验证 SFRC 模型和 FaceNet 模型训练得到模型,求验证的正确率,结果如表 4.3.

表 4.3 在数据集上测试训练模型

项目	FaceNet	SFRC
Accuracy	0.76967+-0.01980	0.75550+-0.01416
Area Under Curve	0.848	0.837
Equal Error Rate	0.233	0.242
Validation rate	0.06233+-0.02432	0.06467+-0.02007

从表 4.2 中可以得到两种模型下的识别正确率相差不大。

综上实验结果进行分析, SFRC 对 FaceNet 的优化起到了一定的成果,尤其在向量映射中,传统 FaceNet 由于多次重复在 CPU 与 GPU 之间进行原始图片 160\*160 的数据传输,对整体时间消耗较大,因此优化能够取得显著成效;而在训练过程中因为传输的是 128 维向量,对整个系统时间消耗相对来说较小,所以优化效果不是特别明显(仅占 2.21%)

## 第五章 结论与展望

随着计算机的发展以及信息规模的逐渐增加，计算机并行计算越来越发挥重要作用，人工智能的逐渐发展使机器学习成为热门，GPU/CPU 异构加速在人工智能，尤其是需要进行大量图像处理以及矩阵计算的领域，逐渐成为主要计算模式。

本文通过 CPU 与 GPU 之间以及 GPU 内部内存管理两部分来对 FaceNet 进行优化，第一部分是在 CPU 中维护一个 GPU 数据内存地址集合映射表，在 CPU 中通过这个映射表来进行 GPU 端逻辑计算，避免每个 batch 训练时数据集都重复进行 CPU 与 GPU 的数据交换；第二部分是将在 GPU 内部涉及内存读取时需要多次读取写入的内存块取入一个共享内存中，通过维护这个共享内存来减少 GPU 端访问全局内存的次数，从而降低时间消耗。通过与传统 FaceNet 模型对比得出优化效果较为明显(平均优化 10%)

但是本文中的 SFRC 模型仍然存在需要改进的地方，主要在下列几个方面需要改进：

- 1、在 SFRC 模型中每张图片的整个训练过程中只训练一次，而且数据集的只采用大小为 2016 个人为整个训练数据集，在以后可以通过增加每张图片的重复训练次数，同时增加数据集大小来提高准确率。
- 2、在 SFRC 中没有进行 CPU 端的并行计算，而且 GPU 加速只在一个 GPU 中进行加速计算，以后可以通过实现 CPU 端的并行计算来进一步对训练时间进行优化。

## 参考文献

- [1] Florian Schroff. FaceNet: A Unified Embedding for Face Recognition and Clustering. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition .2015
- [2] 张佳康, 陈庆奎. 基于 CUDA 技术的卷积神经网络识别算法 [J]. 计算机工程, 2010, 15:179-181.
- [3] Nvidiadeveloper home [EB/OL]. <https://developer.nvidia.com/>, 2016-03.
- [4] 董丽丽, 董玮. 利用 CUDA 提高内存数据聚类效能的研究. 西安建筑科技大学.西安 710055
- [5] 肖军.基于 GPU 的图计算研究.湖南大学.2015
- [6] CUDA C PROGRAMMING GUIDE.<https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>
- [7] Pycuda programming guide.<https://document.tician.de/pycuda/>
- [8] Song K, Liu Y, Wang R, et al. Restricted boltzmann machines and deep belief networks on Sunway cluster//Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications. Sydney, Australia, 2016: 245-252
- [9] Looks M, Herreshoff M, et al. Deep learning with dynamic computation graphs//Proceedings of the International Conference on Learning Representations. Palais des Congrès Neptune, Toulon, France. arXiv: 1702.02181v2, 2017
- [10] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In Proc. of ICML, New York, NY, USA, 2009. 2
- [11] D. Chen, X. Cao, L. Wang, F. Wen, and J. Sun. Bayesian face revisited: A joint formulation. In Proc. ECCV, 2012. 2
- [12] Z. Zhu, P. Luo, X. Wang, and X. Tang. Recover canonicalview faces in the wild with deep neural networks. CoRR, abs/1404.3543, 2014. 2
- [13] Y. Sun, X. Wang, and X. Tang. Deep learning face representation by joint

- identification-verification. CoRR, abs/1406.4773, 2014. 1, 2, 3
- [14] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In IEEE Conf. on CVPR, 2014. 1, 2, 5, 7, 8, 9
- [15] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007. 5
- [16] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In S. Thrun, L. Saul, and B. Schölkopf, editors, NIPS, pages 41–48. MIT Press, 2004. 2
- [17] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. CoRR, abs/1311.2901, 2013. 2, 3, 4, 6

## 致 谢

感谢我的导师宫晓利老师，感谢老师的信任和支持，在我学习期间耐心指导，时常为我鼓励加油，但出现问题也会严厉地训正我，及时纠正我的错误。

感谢唐瑞琦学长在我课程学习期间无尽的支持和鼓励，解惑答疑，耐心为我解决不断出现的各种问题。

但我知道论文完稿并不意味着学习的结束，学海无涯，学习的过程中出现的各种问题意味着我需要更加努力，经历各种问题尤其是程序调试和环境搭建中的各种复杂而奇怪的问题常常让人心生挫败。在跟着老师学习的整个过程宫老师对待科研的态度也一直影响着我，在以后的学习工作生活中，也应该保持这种一丝不苟的严谨态度和对知识的渴望，成为自己人生的态度。

再次感谢老师和学长们的不断支持，祝愿老师和学长们在工作学习中一切顺利！将铭记这份感动。