分类号 TP391 学号 14060041

UDC 密级 公 开

工学硕士学位论文

# 计算机视觉核心算法在移动 GPU 上的高性能实现

硕士生姓名 时洋

学 科 专 业 计算机科学与技术

研 究 方 向 计算机视觉与并行计算

指 导 教 师 张春元 教授

国防科学技术大学研究生院

二〇一六年十一月

# High-performance Implementation of Core Computer Vision Algorithms on Mobile GPU

**Candidate：Shi Yang**

**Advisor：Prof. Zhang Chunyuan**

**A dissertation**

**Submitted in partial fulfillment of the requirements**

**for the degree of Master of Engineering**

**in Computer Science and Technology**

**Graduate School of National University of Defense Technology**

**Changsha，Hunan，P.R.China**

**November 6，2016**

# 独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：计算机视觉核心算法在移动 GPU 上的高性能实现

学位论文作者签名：时洋　　日期：2016 年 11 月 1 日

# 学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅；可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：计算机视觉核心算法在移动 GPU 上的高性能实现

学位论文作者签名：时洋　　日期：2016 年 11 月 1 日

作者指导教师签名：张国元　　日期：2016 年 11 月 1 日

# 目 录

# 表 目 录

# 图 目 录

# 摘　要

图像分类与目标跟踪一直是计算机视觉领域中的核心问题。近些年来卷积神经网络（CNN）与 TLD 算法分别在这两个领域取得了瞩目的成果。与此同时，随着手机等移动设备的计算性能不断提升，研究人员开始尝试将复杂的计算机视觉算法应用在移动设备之上。

对于 CNN，本文首先借助 MXNet 深度学习框架将其前向过程在手机上进行了实现。测试结果显示卷积层的执行时间占到了总时间的 70%以上，因此本文通过 OpenCL 编程框架将这一部分计算转移到了手机的另一个更加适合密集计算的设备——GPU 上。之后又对于 GPU 上的算法实现采取了优化任务划分、利用片上存储等策略进行改进。最终在卷积层计算中取得了 16 倍的加速比，整体分类过程上也取得了 2.1 倍的加速效果。

对于 TLD 算法，它的计算任务主要集中在检测模块的级联分类器上。级联分类器由方差分类器、集成分类器以及最近邻分类器串联组成。本文将这一部分检测计算放在了 GPU 上执行，并且针对任务的负载情况进行了线程的组织以及算法的优化。经过测试，本文提出的针对 GPU 的高性能实现算法对于每一帧视频的目标跟踪任务，可以将用时减少到只有原来的二分之一。

本文通过在移动 GPU 上对于 CNN 与 TLD 这两个计算机视觉核心算法进行高性能实现，探索如何在通用并行计算中高效利用移动 GPU，最终经过优化取得了理想的加速效果。

**主题词：计算机视觉　卷积神经网络　目标跟踪　移动 GPU　并行计算**

# ABSTRACT

Image classification and object tracking have always been the core issue of computer vision. In recent years the convolution neural network (CNN) has achieved better and better results in image classification. Also, in object tracking area, the Tracking-Learning-Detection (TLD) algorithm made the tracking process no longer depends on the pre-trained model. In the same time, mobile devices like phones and tablets are getting so powerful that people start to think how to apply these amazing computer vision algorithms on mobile devices.

For the CNN, we first implemented the forward process on a mobile phone with the deep learning framework MXNet. We analyzed the execution time of different layers in the network and found the convolution layers take up more than 70% of all. We offload the computing work of these layers to the other device in the phone – GPU, which is more suitable for compute-intensive tasks. Based on the structure of mobile GPU and the work load of the task, we have taken several methods to improve the performance, like dividing the task, using on-chip memory, using wider data types and so on. In the final test of our high-performance implementation, we accelerate the matrix multiplication in convolution layer by 16 times, and the time of forward process also gets a speed-up of about 2.1.

With the Consideration of the bottleneck of TLD lies in the detection module, which is consist of three classifiers, the patch variance classifier, the assemble classifier and the nearest neighbor classifier, we offload this part of task to the mobile GPU. We also made some improvements to gain better performance. In the tests, our high-performance implementation of TLD on mobile GPU successfully reduced the time of tracking an object in a single frame to only a half.

This thesis tries to study how to make better performance on mobile GPUs in general parallel computing through implementing the CNN and TLD algorithm on mobile devices, and the experiment results have shown we have made the ideal acceleration effect.

Key Words：Computer Vision　Convolutional Neural Network
Object Tracking　Mobile GPU　Parallel Computing

# 第一章　绪论

## 1.1　课题研究背景

### 1.1.1 移动 GPU

随着手机、平板电脑等移动设备的不断普及以及它们性能的持续提升，有许多以前由于性能不足或者存储限制等原因而不能在移动设备上实现的功能也得以实现。以手机为例：诺基亚（Nokia）在 2005 年推出了一款当时很先进的娱乐手机 N70[1]，这台手机搭载一个 200 万像素的摄像头（前置摄像头 30 万像素），拥有 32MB 内存，以及一颗 220MHz 的 ARM-926CPU，并没有 GPU；2010 年发布的 N8[2]手机的摄像头则提升到了 1200 万像素，内存 256MB 并且携带了一颗 680MHz 的 ARM-11 CPU 以及 BCM2727 GPU；到了 2016 年，最新发布的小米 5 手机[3]，摄像头达到 1600 万像素，所使用的骁龙 820 CPU 主频高达 2.15GHz，内存也达到了桌面级计算机的 4GB，搭载 Adreno 530 GPU。移动设备性能的飞速提升为它们赋予了更多可能。

虽然移动设备上的应用类型随着设备性能的提升而不断丰富，但是对于移动设备的性能利用，还是存在不小的浪费。目前绝大部分的主流移动设备都同时拥有 CPU 与 GPU 两个计算核心，但是除了游戏这一类程序之外，很少有应用会去利用 GPU 的计算性能。造成这一情况的主要原因有两个：首先是对于移动设备 GPU 来说，通用计算的编程支持在之前并不完善。也就是在最近几年，OpenCL[4]以及 CUDA[5]这样的通用计算框架才慢慢在移动 GPU 上得以适配；其次，很多计算负载比较重的应用会选择将这一部分计算任务放在远程服务器上进行，计算完毕之后再将结果发送回移动设备。这种实现方式的缺点显而易见，在没有网络支持的情况下程序就无法正常运行。因此，研究如何利用移动 GPU 进行高性能的通用计算是当前技术发展形势下的研究热点，具有很深的实际意义。

### 1.1.2 图像分类算法

在计算机视觉的众多任务中，图像分类（Image Classification）是根据图像的内容来将图片进行归类，以替代人眼的分类判断工作。给定一张图片，图像分类回答的问题就是这张图片中的内容属于哪一类，飞机还是奶牛？

图像分类不仅在许多实际场景比如交通检测[6]，图像检索[7]中得到了广泛应用，而且还是目标跟踪[8]、图像分割[9]等更复杂视觉任务的基础，因此，图像分类是计算机视觉、机器学习以及模式识别等领域备受关注的研究方向之一，也是计算机视

觉领域的核心问题之一。

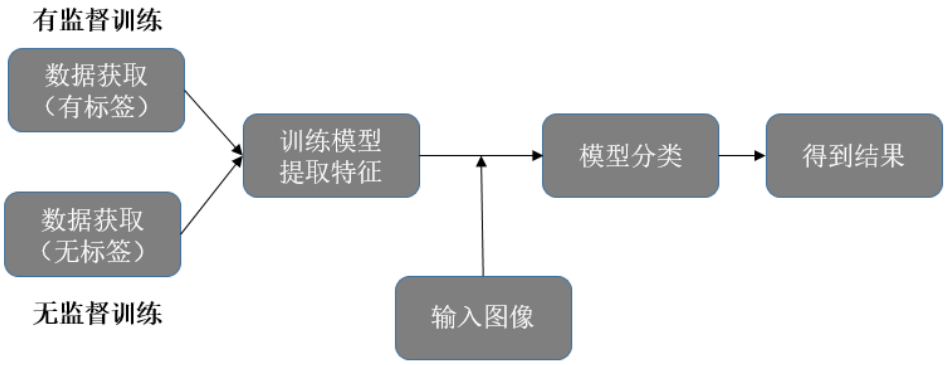


图 1.1　图像分类的基本流程

图像分类中的核心概念是特征，图像分类可以看做一个从图像的像素信息中提取特征，再用这些特征来匹配图像类别的过程。决定图像分类准确性最关键的因素就是特征如何选取以及提取特征方式的选择，这两个问题可以通过对图像分类训练数据集的训练学习来解决。运用计算机进行图像分类的基本流程如图 1.1 所示，从图中可以看到学习过程又分为两种，有监督学习[10]与无监督学习[10]。顾名思义，无监督学习就是让特征提取器自行探索图像中的特征，有监督学习则是给予一个训练集图片到图像所属类别的映射来指导学习过程。一般来说，有监督学习的训练过程速度更快，而且结果往往也更加准确。

### 1.1.3 目标跟踪算法

图 1.2　目标跟踪在现实生活中的应用

目标跟踪（Object Tracking）是计算机视觉领域另一个值得关注的核心问题，它也是在视频监控[11]、信息检索[12]等领域应用最为广泛的计算机视觉应用之一。所谓目标跟踪，指的是对于视频或者是一组图像中某一目标进行检测、提取、识别，

最终确定每一帧图像中目标的位置信息。各式各样的跟踪算法中，依据其指导思想可以分为检测与跟踪两大类。

检测算法起源于图像分类算法，通过对包含跟踪目标的大量训练图片进行学习，使得算法提取到跟踪目标的特征；然后在每一帧图像中对所有的图像片进行检测，来判定图像片是否是要跟踪的目标。从算法原理中可以看出，通过检测来实现跟踪的主要缺点是需要提前学习目标特征；训练数据集对跟踪算法的准确性影响较大；算法不能无限定的跟踪任意目标。

还有一类算法利用跟踪的原理，这类算法以光流法[13]为典型代表。这一类方法的基本原理是对于初始图像中的目标进行特征提取，然后利用运动连续性原理——目标在下一帧图像中的出现位置应当在上一帧目标位置附近——选取下一帧图像中在目标初始位置附近的图像片进行相同的特征提取。通过比较图像片的特征与跟踪目标的特征，来判定目标新的位置。但是这种方法同样存在不小的缺点，即当存在障碍物遮挡、运动模糊或者目标运动速度过快等复杂情况时算法准确性会受到很大影响。因此，一个在各种条件下都能比较准确进行跟踪的算法一直是研究人员追求的目标。

## 1.2 相关研究现状

本节主要从图像分类算法研究、目标跟踪算法研究以及针对移动 GPU 的相关加速研究这三个方面来介绍本课题的研究现状。

### 1.2.1 图像分类算法研究之 CNN

深度学习[14]起源于上世纪提出的神经网络概念。早期的神经网络也称为浅层学习，通过 BP 算法[15]来从原始数据集中学习特征的表达以及分类，并且将这些学习到的信息存储在神经网络的参数之中。这种网络被人们称为多层感知机，但是由于那个时代计算能力的限制，这些网络通常只有一层隐含层。因此，学习表达特征的能力也是极其有限的。2006 年 Hinton 在《Science》上发表的论文[14]首次提出了深度学习的概念，不仅于此，论文中还给出了两个相当重要的结论：首先，含有更多隐含层的神经网络具有更加强大的特征学习与表达能力；其二，多隐含层神经网络的训练可以通过逐层初始化方法来降低训练难度。自此以后，深度学习在许多领域迎来了雨后春笋一般的发展。

在图像分类领域，深度学习也在 2012 年取得了历史性的突破——Hinton 所在的研究团队使用一个深度卷积神经网络[16]（Convolutional Neural Network，CNN）在 ImageNet[17]的图像分类比赛中取得了第一名的成绩，并且领先其它的小组 1%以上，拉开了深度学习方法与传统分类方法的差距。自此以后，在 ImageNet 的图像

分类比赛中，获得最好成绩的无一例外都使用了卷积神经网络的分类方法，在 2016 年微软甚至使用一个拥有 152 层的卷积神经网络[18]在 ImageNet 数据集上将 Top1 分类错误率降至 3.75%，首次低于人眼分类的错误率 5.45%。

表 1.1　2012 年 ImageNet 图像分类比赛结果

| 成绩排名 | 研究人员 | Top5 错误率 | 采用方法 |
|---|---|---|---|
| 1 | 多伦多大学 | 0.15315 | 深度学习 |
| 2 | 东京大学 | 0.26172 | 手工设计特征 |
| 3 | 牛津大学 | 0.26979 | |
| 4 | Zerox/INRIA | 0.2758 | |

卷积神经网络是一种典型的多隐含层深度神经网络。其中的网络层有着不同的类型，完成不同的计算，常见的有卷积层、池化层、全连接层等等。其中，最复杂也是在特征提取中起到关键作用的就是卷积层。卷积层利用多个卷积核在图像上滑动，把图像上对应位置的像素点与卷积核中的参数进行计算，得到这个图像在卷积下的结果。通过这个过程，完成从低层图像到高层特征表达的转换过程。

下图所示的结构就是在 2012 年比赛中取得第一名的多伦多大学所使用的 AlexNet 网络结构，其中有 5 个卷积层以及 3 个全连接层。

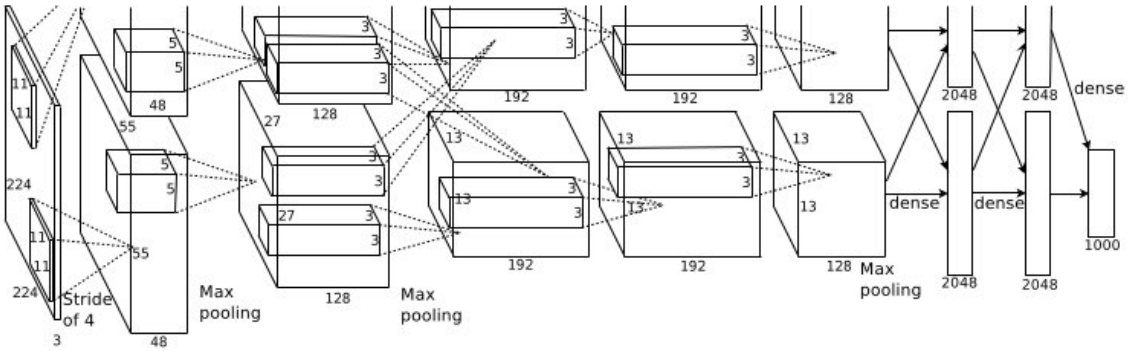


图 1. 3　AlexNet 的基本结构

### 1.2.2 目标跟踪算法研究之 TLD

目标跟踪仍然是一个尚未完美解决跨领域的综合性问题，因此，提升目标跟踪算法的准确性、鲁棒性以及实时性是众多算法研究人员的目标。

在前文研究背景中提到，目标跟踪算法分为检测与跟踪两大类。每一类中都有几个代表性的算法。检测算法可以分为三类：光流法，差分法以及背景差法。

光流法是通过将计算二维图像中的运动场情况来定位运动目标的方法。最经典的算法有 Lucas-Kanade 算法[19]以及 Hom-Schunck 算法[20]。光流法的优点是得到的目标运动情况比较稳定，比较容易提取目标；缺点则是计算量很大，并且受到噪

声信号的影响也比较明显；差分法又被称为帧间差分方法，通过对相邻帧的图像进行做差比较来得到运动目标的位置。这种方法的优点是算法原理简单，并且对于给环境的适应性比较强，缺点是受目标运动速度的影响较大。当运动速度比较小的时候，不能检测出目标的整体，而当速度比较大的时候，往往又会检测出多个目标；背景差法通过训练运动场景的背景，得到背景的表达模型，再通过将图像与背景模型做差来获得目标的位置。这种算法的缺点显而易见，就是检测的准确率受到模型训练情况的直接影响。

而跟踪算法根据算法定位点的不同也分为几类：基于区域的跟踪，基于特征的跟踪，基于轮廓的跟踪以及基于模型的跟踪。

基于区域的跟踪算法将跟踪目标作为一个区域，通过在图像中寻找与这个区域最为相似的目标进行跟踪，该算法受遮挡等情况的影响比较大；基于特征的跟踪算法通过在跟踪目标中选取一定的特征表达，然后在图像中搜索这样的特征来确定目标的位置，这种算法可以比较好的处理遮挡的情况，因为此时仍然能够搜索到目标的部分特征；基于轮廓的跟踪方法，利用一个轮廓将目标包含在其中，然后构造一个能量函数来表示这个轮廓，在图像中求解这个函数的极小值来进行跟踪，算法的主要缺点是受噪声的影响很大；基于模型的跟踪通过对跟踪目标建立模型然后在图像序列中通过查找匹配来进行跟踪，缺点也是受到模型的影响较大。

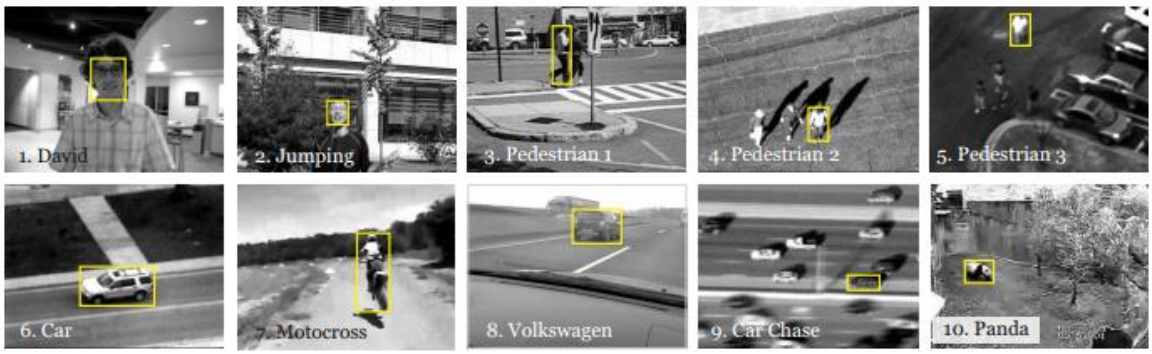


图 1. 4　TLD 算法实际运行效果

TLD（Tracking-Learning-Detection）算法[21]是 2013 年由捷克大学的 Zdenek Kalal 提出的一个优秀算法，算法不仅将传统的跟踪与检测结合起来，并且还引入了新的学习模块来处理目标在长时间运动中发生的形变。因此，算法在单目标长时间跟踪情况下表现优异。TLD 算法凭借自己的结构以及准确性，引起了跟踪算法研究领域的兴趣。Georg Nebehay[22]在算法基础上进行了扩展，使之能够适用于多目标的跟踪领域；国防科技大学的郭鹏宇通过将算法结构进行改造简化，提升算法的适应性，同时利用 CPU 与 GPU 协同工作来加速 TLD 算法[23]；南京航空航天大学的周欣通过改造算法的局部检测器提升了 TLD 算法的鲁棒性以及准确性[24]；程立英等[25]通过将金字塔光流法引入 TLD 算法，使得改进后的算法可以处理物体在

长时间运动中出现的孔径问题。

### 1.2.3 移动 GPU 通用计算研究

图形处理单元（GPU）[26]在计算机中的应用首先是在游戏的图像渲染方面，之后，人们又利用 GPU 的多核架构来进行通用的高性能计算。人们对于开发桌面级 GPU 在通用计算加速上的潜力进行持续研究已有几十年之久，但是在移动 GPU 领域，高性能计算仅仅是一个刚刚引起注意的研究点。这不仅是由于移动 GPU 的出现以及性能的提升较晚，也是由于 OpenCL、CUDA 这样的并行加速框架也是最近才开始对移动 GPU 的支持。

一个典型的移动GPU由数个计算单元组成，在每个计算单元中又有数个ALU，这种结构与桌面 GPU 类似。两类 GPU 之间最大的区别在运行内存方面，桌面 GPU 拥有自己单独的内存，而移动 GPU 的内存与 CPU 所使用的内存是同一块。但是每个计算单元依然具有可以自己单独使用的本地存储器，这部分存储器其它计算单元并不能够访问。

当前的移动 GPU 主要分为三类：高通的 Adreno 系列[27]，NVIDIA 的 Tegra 系列[28]以及 ARM 的 Mali[29]系列。在软件的支持方面，Adreno GPU 以及 Mali GPU 支持使用 OpenCL 1.1 框架，Tegra GPU 则支持自己的 CUDA 编程框架。与桌面 GPU 往往支持各类编程框架不同，移动 GPU 留给开发人员可选择的空间通常比较小。

在移动 GPU 的并行计算加速研究方面，王国辉[30]展示了在移动 GPU 上利用 OpenCL 来实现去除对象的修复算法，可以实现性能上的提升；刘广婷[31]在手机上实现了人脸识别应用，并将 CPU 与 GPU 版本进行了比较，GPU 版本取得了 2.3 倍的加速比。

## 1.3 本文工作与创新点

### 1.3.1 本文工作

本文选取移动 GPU 为研究平台，针对计算机视觉领域的核心热点问题——图像分类以及目标跟踪，在移动平台进行了实现并且针对移动 GPU 的结构特点进行了加速优化，最后在分类以及跟踪的整个过程上分别取得了 2.1 与 2.09 倍的加速效果。总的来说，本文的主要工作分为以下几个方面：

（1） 将分类算法与跟踪算法在移动平台上进行了实现

对于这 CNN 以及 TLD 两个热点的算法，借鉴 MXNet 中的 CNN 实现以及基础版本的 TLD 代码，分别选取高通 801 芯片[32]以及 Tegra K1[33]芯片，在移动平台

上实现了这两个算法的核心功能。

（2） 对于图像分类中的卷积神经网络以及图像跟踪 TLD 算法进行了评测与分析。在之前算法实现的基础之上，对于两种算法各个模块的运行情况进行测试分析，得到算法在各个阶段的耗时情况，找到算法执行的瓶颈阶段，方便下一步进行针对性加速实现。

（3） 针对移动平台 GPU 的特点，利用 GPU 对两个核心算法进行加速实现与测评。对于高通 801 芯片上分类算法中耗时巨大的卷积操作，通过 OpenCL 转移到 GPU 上进行计算；而对于 Tegra K1 平台上的目标跟踪算法使用 CUDA 对于分类器进行加速实现，保证算法正确性的同时，取得了 2.09 倍的加速效果。同时，两种平台也覆盖了移动 GPU 的两种编程框架支持。

### 1.3.2 本文创新点

本文的创新点主要有以下几个方面：

（1） 通过对于 CNN 以及 TLD 算法在移动平台的测试，分析了算法各个阶段的计算负载情况

（2） 对于 CNN 图像分类算法，在手机上利用 OpenCL 进行 GPU 加速，取得了预期的加速效果

（3） 对于 TLD 图像跟踪算法，使用 CUDA 框架调用移动 GPU，进行针对性优化，提升了跟踪算法的性能表现。

## 1.4 论文组织结构

本课题论文的内容主要分为五章，具体组织结构如下：

第一章为绪论部分，主要介绍了本课题的研究背景以及当前的研究现状。在当前移动设备性能飞速发展的大背景下，移动设备端的应用也是不断丰富。但是对于移动设备上一个非常强大的计算设备——移动 GPU 的利用却还不够。尤其是在计算需求比较大的计算机视觉领域，如何进行高性能计算更是重要。本课题的研究现状中主要介绍了对于移动 GPU 通用计算的加速研究现状以及计算机视觉核心应用分类与跟踪的发展趋势。此外，本章还介绍了本文的主要研究工作、创新点以及论文的组织结构。

第二章分析介绍了 CNN，TLD，移动 GPU 以及深度学习编程框架 MXNet。讲述了 CNN 算法以及 TLD 算法的具体实现细节，并且对于课题所选用的两种移动 GPU 进行了结构分析，介绍了它们的基本特点。

第三章主要是针对 CNN 在 Adreno330 GPU 上的加速实现。本章首先测试分析了 CNN 在执行时各个网络层的耗时情况，找到了最值得注意的是卷积层。然后利

用 OpenCL 将这一部分计算转移到了 GPU 上，通过对于 GPU 实现的优化，实现整个过程的加速。最后，对于加速后的实现与基础实现进行了比较测评。

第四章则介绍了 TLD 算法在 Tegra K1 平台上的加速过程。首先先对 TLD 在 Tegra K1 上的基础实现进行了分析，耗时最久的是跟踪模块中的三个分类器，因此这是我们加速的重点。于是我们将这一部分分类计算转移到了 GPU 上执行，并且针对 GPU 计算快、访存慢的特点进行了算法优化，根据任务的负载情况设定线程组织。最终，在整体跟踪时间上取得了 2.09 倍的加速效果。

第五章为结论与展望部分，主要是对于本文工作的一个概括性总结，并且对于以后的研究方向提供了一个可行的方案。

# 第二章　CNN，TLD 与移动 GPU 介绍

## 2.1 CNN 算法原理与 MXNet 框架

卷积神经网络是由基础的浅层人工神经网络发展而来，因此，本节首先介绍人工神经网络的相关背景知识，然后在人工神经网络的基础上介绍深度学习与卷积神经网络。

### 2.1.1 人工神经网络

人脑是一个极其准确高效的分析系统，生理学家与计算机学家一直希望人工模拟人脑的结构来达到类似人脑的功能。人工神经网络就是这样一个模型，它将处理数据的结构看做一个个的神经元，并在这些神经元之间进行类似人脑神经元[34]的连接。图 2.1 中就是神经网络最基本结构——神经元的模型。

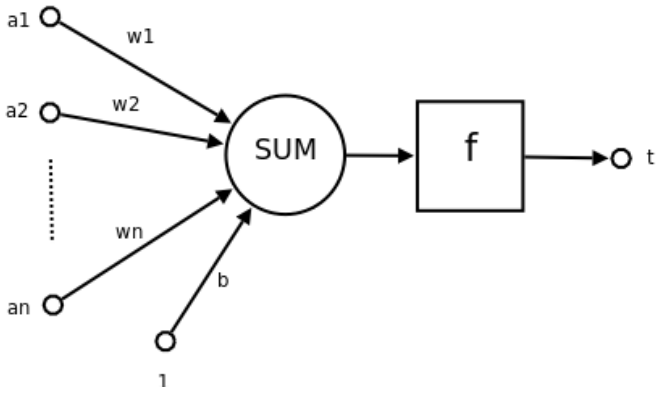


图 2. 1　神经元的基本模型

这样的结构表示的内容实际是一种数学运算：这里 a1 到 an 就是某个输入向量每一维度上的一个分量，w1~wn 则是这个神经元每个神经突触的权值，b 为附加在这个神经元上的偏置权值，f 为这个神经元的传递函数，通常为一个非线性函数。最后，t 就是这个神经元的输出。

整个结构就是输入向量信息进入神经元，经过神经元的处理（权重的计算），得到一个标量结果。整个的数学表示就是：

$$t = f(W * A + b) \qquad (2.1)$$

其中，f 为这个神经元传递函数，W 为权重向量（w1,w2…wn），A 为输入向量(a1,a2…an)，b 为这个神经元的偏置。

一个简单的浅层神经网络，就是由这样的一些神经元组成，上一层神经元的输出组合成一个新的向量作为下一层神经元的输入。这样的浅层神经网络如图 2.2 所示：

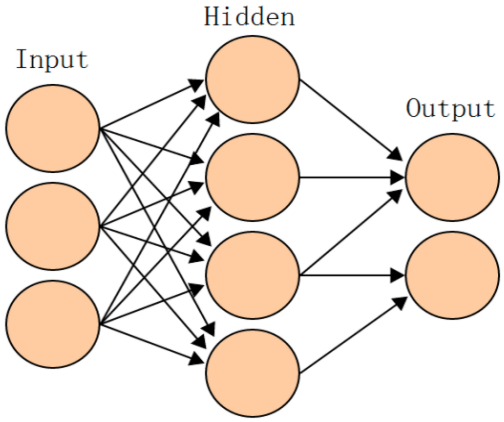


图 2.2　单隐含层的简单网络

上图所示的简单网络只有一个隐含层，用类似的网络构建方法，我们很容易就可以构造出含有更多隐含层的网络。神经网络的参数——即各个神经元的突触连接权重以及偏置，都可以通过对神经网络进行训练来求得。但是像图 2.2 中这样的网络，输入层与隐藏层之间的神经元都是全连接的，这种结构就会造成整个神经网络的参数数量十分巨大。比如我们考虑处理一张 256*256 的图像，由于图像中一共有 256*256 个像素，如果每个像素点作为一个输入值的话，那么在第一个隐含层神经元数目与输入向量维度相同的情况下，这一层的权值就回多达 $256^4$=4,294,967,296 个。这对于神经网络的训练来说，是一个巨大的挑战。为了解决参数规模过大的问题，计算机学家们提出了一种新的网络模型——卷积神经网络[35]。

### 2.1.2 卷积神经网络

卷积神经网络，顾名思义，其中的核心就是卷积运算操作。卷积计算在网络中承担的责任是提取高维度特征。卷积操作的数学过程如图 2.3 所示：

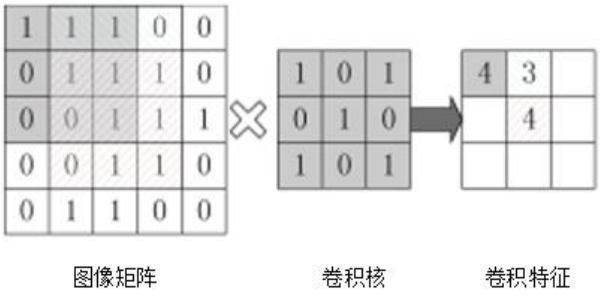


图 2.3　卷积运算数学过程

为了解决之前提到的网络参数数量过大的问题，卷积神经网络主要采用以下四种方法来降低参数数量并保持网络的有效性：局部感知，卷积核重用，多卷积核以及池化采样。下面依次进行介绍：

#### 2.1.2.1 局部感知

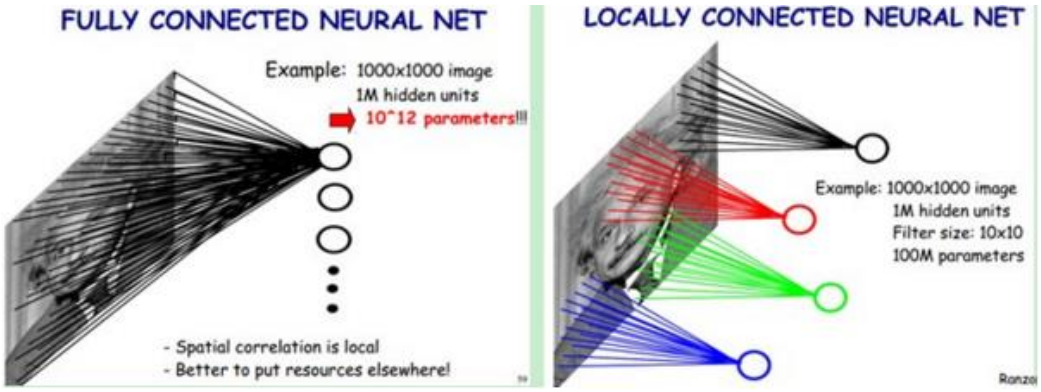


图 2. 4　局部感知示意图

图 2.4[16]就是卷积神经网络中局部感知的示意图，第一个隐含层中的某一个神经元不再与输入向量中的每一个连接，它只会感知固定大小的一片区域内的输入信息。比如在图中，这个神经元就只负责一种颜色神经元连接内的输入。通过将用局部区域的连接来替代全连接，局部感知的引入大大减少了神经网络中的参数数量。

#### 2.1.2.2 参数共享

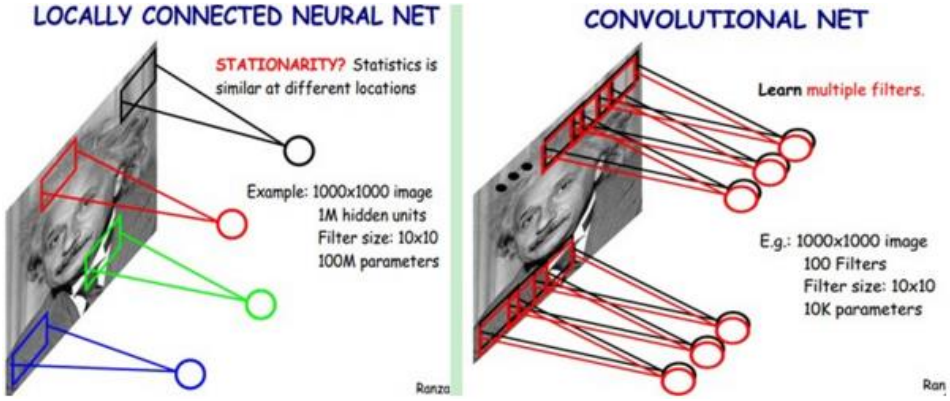


图 2. 5　参数共享示意图

在局部感知的基础上，为了避免输入信息的浪费造成神经网络对特征的提取不够准确，这里又采用了参数共享策略[16]。对应于图 2.5，参数共享就是第一个隐藏层中的神经元将自己的局部感知视野划过整个输入图像，即对于输入图像中每一个相连的区域进行卷积运算。这所有的相连区域都采用相同的神经元突触权值，也就是卷积框的参数。通过这个方法，既可以避免神经网络的参数数量过大，又可以有效的保持网络的特征提取能力。

#### 2.1.2.3 多卷积核

在图 2.5 中，每一个这样的局部视野计算都可以看做是一个卷积核所进行的操作，类比于人类的神经网络，我们可以将其设想为每一个卷积核负责提取图像中某

一类的特征。那么，为了增强卷积神经网络的特征提取能力，自然应当有多个不同的卷积核来负责提取图像中不同的特征，这就是卷积神经网络中的多卷积核。在识别手写数字的经典网络 LeNet-5[36]中，卷积层中就有 6 个这样的卷积核。输入图像经过每一个卷积核的计算都会产生一个输出矩阵，而所有这样的输出矩阵叠加就是图像经过卷积层的处理结果。

#### 2.1.2.4 池化采样

在卷积网络中，往往还包含了一个名叫池化层的层级结构。最经常使用的池化层类型是最大池化层，如图 2.6 所示，在卷积层的输出中，对于 2*2 大小的方块，选取其中 4 个值中的最大值作为这个方块的特征代表输出。假设池化采样的方块大小为 2*2，那么池化后的数据量就只有池化之前数据量的四分之一。

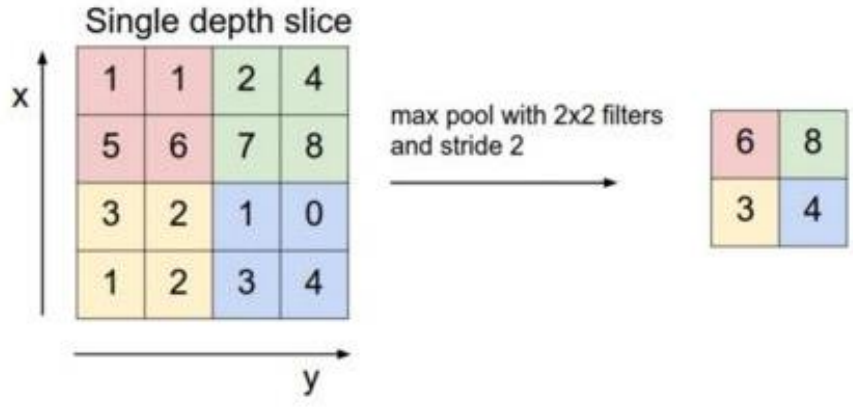


图 2. 6　池化层操作示例

池化层的操作可以设想为确认某个特征是否存在，而不关心这个特征到底出现在个具体位置上。池化层的好处也是显而易见的，它会大大减小其后网络层中参数的数量。

除此之外，通过池化层还可以使输出大小保持固定，这样，不论前面使用了什么样结构的计算，都可以通过池化层将输出归一到我们需要的格式上去；池化层用在图像分类领域还有一个好处是它可以神经网络分类结果提供平移不变性与旋转不变性。如果我们对某一个区域做了池化操作，那么这个区域的平移或者旋转就不会影响池化结果。这会大大提升分类网络的适应性与准确性。

将以上的技术综合起来，我们就得到了一个完整的卷积神经网络。图 2.7 就是利用一个卷积神经网络对于一张图片进行分类的过程示意图：

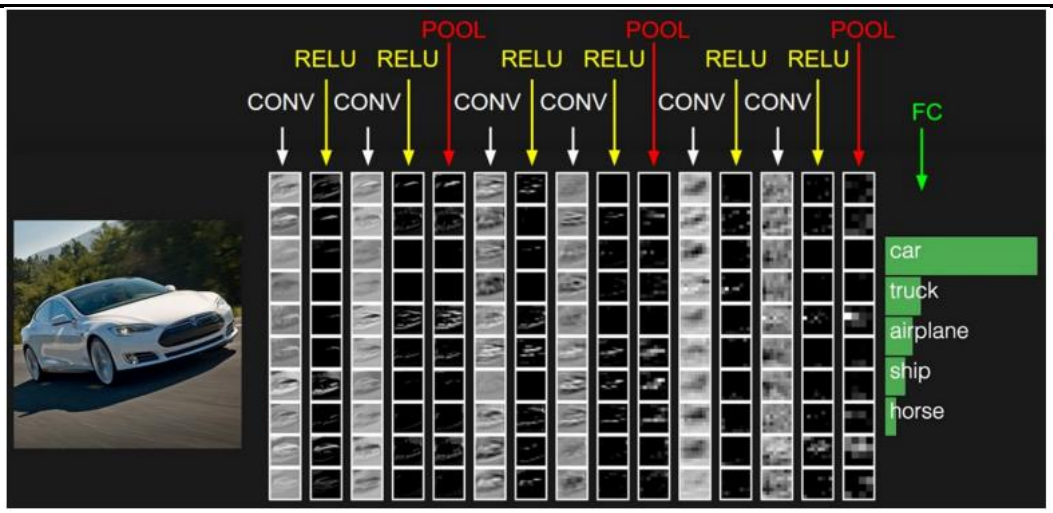


图 2.7　卷积网络进行图像分类实例

### 2.1.3　MXNet 框架

MXNet[37]是一个开源高效的深度学习编程框架，是当前的主流深度学习框架之一。下面从功能、结构以及性能三个方面对 MXNet 进行介绍。

MXNet 的低层开发语言使用的是 C++，上层提供 Matlab、Python 以及 Ruby 等多种编程语言接口；提供多种对于深度学习网络的训练支持，比如卷积神经网络、递归神经网络等等；并且整个框架原生支持多加速卡以及多机的训练过程；提供 Nvidia GPU 加速支持。MXNet 提供了一整套完整的深度学习工具集合，可以进行模型训练、网络测试、参数微调等等，并且官方提供了非常详尽的文档说明以及代码示例。除此之外，MXNet 还拥有一个十分活跃的开发者社区，在这里使用者不仅可以与其他使用者进行交流，而且可以直接联系到 MXNet 的开发人员以及提交对 MXNet 的修改。

在主流的深度学习框架中，实现过程中所使用的编程思想主要分为两种，一种是命令式编程，另一种则是声明式编程[38]。以 a=b+1 这个运算为例，采用命令式编程执行则首先需要 b 已经被赋值，然后立即执行加法，最后将结果保存在 a 中；而声明式编程则会返回一张计算图来代表这个运算，在需要得到 a 具体数值的时候，再对 b 进行赋值，然后执行计算。命令式编程的优点是容易理解、各种语言无缝交互，缺点则是过于受限与算法的细节，很难在更加宏观的层面对于算法进行整体上的优化。声明式编程的优点是在计算开始之前就构造一个整体的运算图，对于整个算法的理解更加全面，伴随而来的缺点则是实现复杂，查错困难。

以往的编程框架往往会在这两种编程思想中选择一种来指导框架的开发，而 MXNet 选择将两个策略综合起来进行实现：在命令式编程上提供张量运算，而在声明式编程中也有符号表达式支持。使用者在使用的时候就可以自由的根据自己

的需要来进行混合式编程，提升框架的开发以及训练效率。

在使用深度学习进行开发的时候，尤其是在训练过程中，会占用极大的内存。因此，对于内存的控制效果是评价一个深度学习框架优秀与否的重要标准。在这一方面，MXNet 提供了两种启发式方法来减少框架的内存使用——Inplace 与 Co-share。下面依次进行介绍：

（1） Inplace 方法：在这个方法中，对于之前根据网络运算所构造的计算图，对每一个变量进行标注，标注数值的根据是还有多少数据会依赖它进行计算。当发现这个标注变为零的时候，就意味着这个变量不会再被使用，就可以对分配给它的内存进行回收。

（2） Co-share：这个方法允许两个不同的变量使用一段相同的内存空间。当然，这种共享不能是任意的，比如两个数据同时需要使用的时候，这一段内存中就只能够表示其中一个数据。为此，框架考虑计算图中的一条计算通路，在这样的一条通路中，变量彼此之间由于具有依赖关系而不能并行计算，可以利用 co-share 策略来公用内存空间。

下面，对于 MXNet 的整体架构设计分为几个模块进行介绍：

（1） 读入数据模块：由于深度学习中训练集的数据量通常都很大，达到几十 GB 甚至更多，因此数据读取的速度对整个系统性能的影响很大。MXNet 中提供了工具可以对任意大小的样本数据进行打包压缩，转化成单个或多个文件来提升顺序与随机读取的速度。

在读取数据的时候，MXNet 使用数据迭代器来控制这个过程。迭代器每次只会将当前所需要的数据读入内存。在迭代器中采用多线程实现，利用多线程预取来隐藏文件读取的开销。

（2） 训练模块：在训练模块中，MXNet 实现了常用的优化算法来实现模型，训练模块的核心是一个训练引擎以及代表网络的符号象征。对于使用者来说，只需要将输入数据的迭代器准备好，然后将网络符号传递给训练引擎，就可以开始网络的训练过程。与通常的数据流引擎不同的是，MXNet 的引擎只允许有一个任务可以修改现有的资源。因此，为了保证训练过程引擎自动优化调度的正确性，使用者在提交任务的时候需要标明哪些资源是只读，哪些资源会被修改。

（3） 数据通信模块：MXNet 框架中的数据通信使用参数服务器[39]实现，具体来说就是一个 KVStore[40]。在 KVStore 中使用一个两层的通信结构，如图 2.8 中所示。第一层的服务器管理单机内部的多个设备之间的通讯。第二层服务器则管理机器之间通过网络的通讯。第一层的服务器在与第二层通讯前可能合并设备之间的数据来降低网络带宽消费。同时考虑到机器内和外通讯带宽和延时的不同性，对于不同的通信层次可以使用不同的一致性模型。例如第一层我们用强的一致性模

型，而第二层则使用弱的一致性模型来减少同步开销。

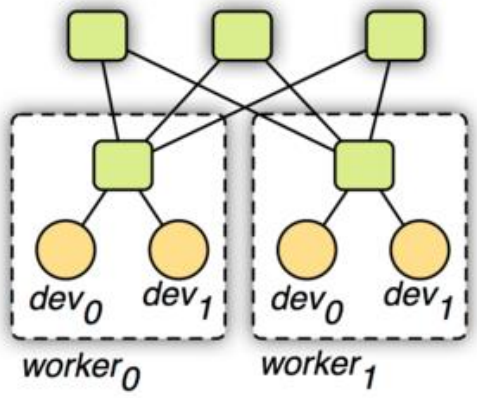


图 2.8　两层参数服务器

接下来，我们通过一些实验结果来比较一下 MXNet 与其它深度学习框架之间的性能差异。

首先，使用三种流行的卷积神经网络 Alexnet、Googlenet[41]以及 Vggnet[42]来测试 TensorFLow[43]、Caffe[44]、Torch7[45]与 MXnet 在单卡 GTX980[46]上的速度表现。测试结果如下图：

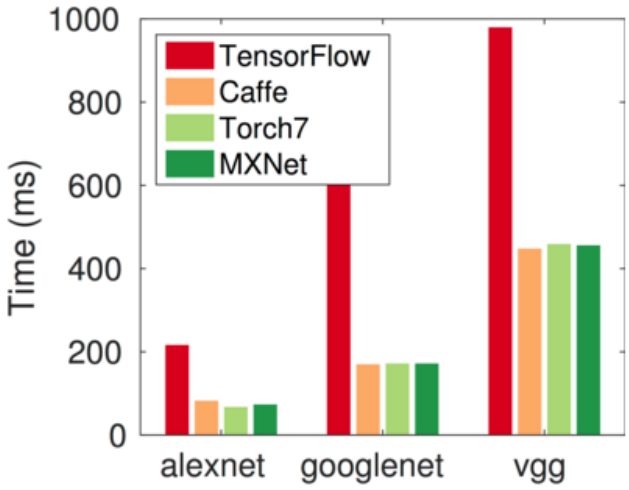


图 2.9　深度学习框架单卡测试结果

可以看出 MXNet、Caffe 以及 Torch7 在这个测试中的速度表现相差不大，这是由于在单卡上的测试条件下几个网络的绝大部分运算都是由 GPU 运算标准库 CUDA 以及 CUDNN 完成。Tensorflow 在这里要比其它几个框架慢三倍，这可能是由于其支持的 CUDNN 版本过低以及项目刚刚开源，并未针对性优化的缘故。

再来看一下不同框架在网络预测中的内存使用情况，依然采用上述三种网络，每次处理的数据 patch 大小为 128，预测过程中的内存开销如图 2.10：

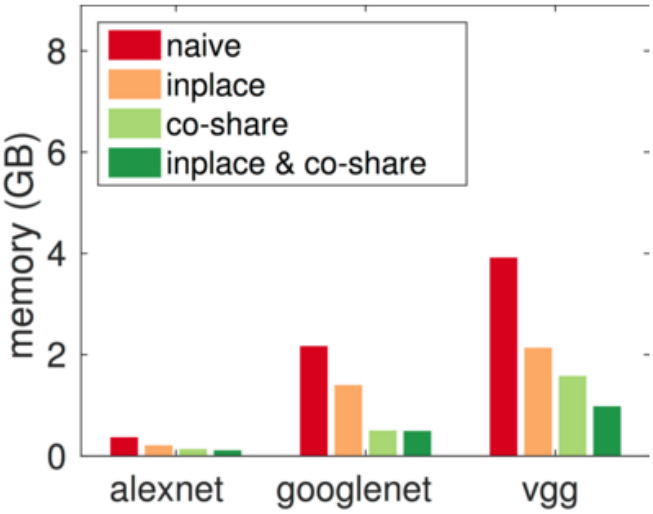


图 2. 10　MXNet 内存优化效果

由图中可以看出，MXNet 采用的内存优化策略效果十分明显：Inplace 以及 Co-share 方法均可以极大减少内存开销。将两者结合起来，在预测的时候可以减少约 75%的内存使用。特别的，若是对于单张图片进行预测，即使是采用最为复杂的 vggnet，在整个系统的基础上也只需要 6MB 的额外内存。

MXNet 在内存优化上的优异表现也是本课题最终选用它作为基础来进行手机上的 CNN 实现的主要原因，因为相比于桌面设备，移动设备中的内存限制更为严格。

## 2.2 TLD 算法原理

TLD 算法的提出是为了解决单目标的长时间跟踪问题。算法在执行的时候，可以任意指定一个跟踪目标，而且不需要预先训练过程。算法模型中将检测与跟踪结合了起来，因此共有三个模块，如图 2.11 所示，有检测模块、跟踪模块以及学习模块。下面对于每个模块的功能以及实现逐个进行介绍。

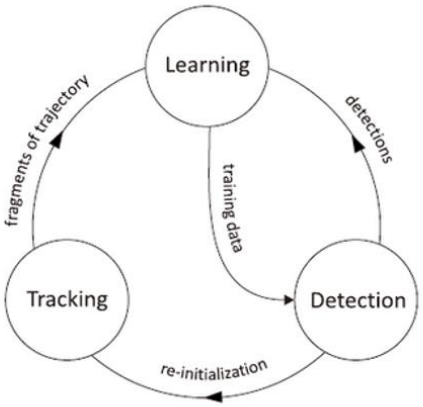


图 2. 11　TLD 算法的基本结构

### 2.2.1 检测模块

TLD 中检测模块就是由一个检测器组成，检测器的原理是对于当前图像中每一个图像块进行检测，分析其是否是要跟踪的目标。这个检测器的核心就是一个级联分类器，由方差分类器、集成分类器以及最近邻分类器组成。

方差分类器是一个简单有效的分类器。首先，需要计算原始跟踪目标图像元中每个像素值的灰度值方差，然后对于当前帧图像中的每一个样本框体同样也计算灰度值方差。最后将方差值小于原始方差值一半的样本标记为非目标。通过这一步，就可以排除掉大约一半的样本。因此，将其放在级联分类器的第一阶段具有最高的效率。

集成分类器的核心是根据图像的某类特征的先验概率分布以及检测图像元的特征来判断这个图像元是不是目标图像，可以看做是很多个随机蕨分类器。如图 2.12，用一个数字图像的检测为例：在图中选取五对特征点，每对特征点 A、B 中，比较该点对的亮度值。若 A 的亮度大于 B，特征值为 1；否则为 0.。那么图中的五个点对的特征值就为 01011，转化为十进制则为 11。


图 2. 12　随机蕨分类器原理

将不同的数字样本经过这样的特征点进行检测，就可以得到不同的先验概率分布。

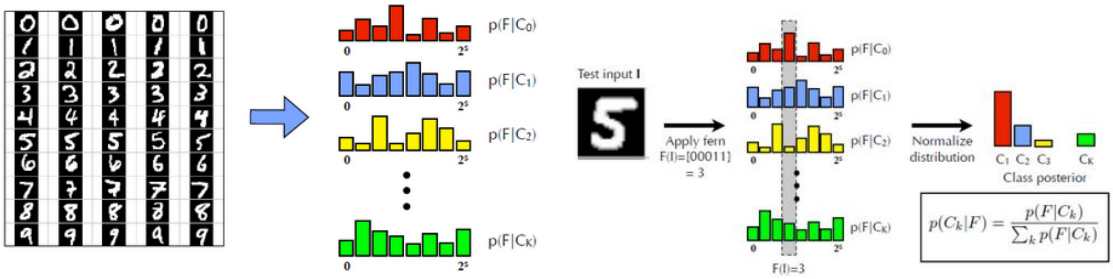


图 2. 13　基层分类器原理

图 2.13 中过程就可以看做对这个集成分类器的训练过程。之后，当有未分类的样本进去时，若其将经过分类器的结果为 3，则从学习模块中的分类概率模型中找出后验概率为 3 的最大的一类，就是样本的分类。以上就是集成分类器的原理。

最近邻分类器是级联分类器的最后一个部分。它的原理很简单，通过计算目标

样本集合中的框体与当前图像中的样本框体的相似程度来判定当前框体是否为要跟踪的目标。相似度[47]的计算过程如下：

存放目标样本的集合我们记做 T 集合，设 T 中共有 n 个目标样本，则 T = {T1，T2，…， Tn}。

对于当前图像中的一个框体 P，P 与某一个目标样本的相似度为：

$$S(P,Ti) = 0.5 * (N(P,Ti) + 1) \quad (2.2)$$

其中，N 为进行的规范化操作，以保证相似度的取值不小于 0 且不大于 1。

P 与 T 集合的正相似度 S1 计算公式如下，其中 Ti 为 T 中某一样本：

$$S1(P,T) = MaxS(P,Ti) \quad (2.3)$$

类似的，P 与 T 集合的负相似度 S2 计算公式如下：

$$S2(P,T) = MinS(P,Ti) \quad (2.4)$$

最后，P 与目标样本的相似度取值为：

$$S(P,T) = 1 - \frac{S2}{S1+S2} \quad (2.5)$$

在这一模块中，分类器的出口设定了一个阈值，以 0.6 为例，如果 S(P,T)大于 0.6，则分类器认为检测框体就是样本，否则不然。在检测模块阶段，可能会输出多个、一个或者零个正样本，这在算法之中都是允许的。

### 2.2.2 跟踪模块

TLD 算法的跟踪模块的核心算法是算法作者提出的 Median-Flow 追踪算法[48]。该算法的流程如图 2.14 所示：

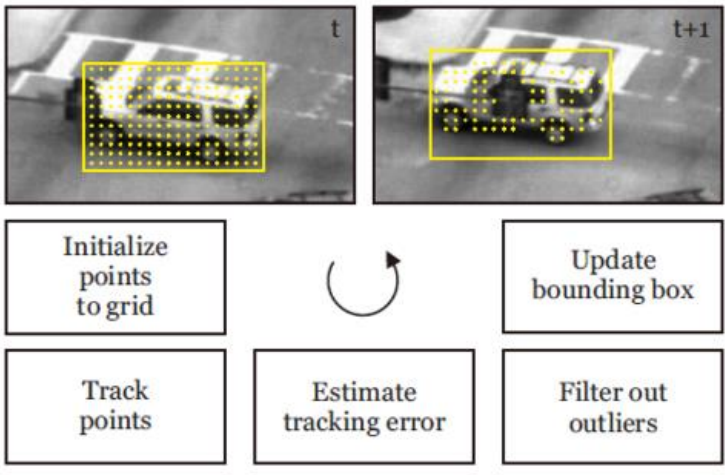


图 2. 14　追踪算法流程

首先在上一帧时刻 T 的图像中对于跟踪目标的样本框内均匀产生一些随机点，然后用 Lucas-Kanade 追踪器[49]追踪每一个追踪点在 T+1 时刻的位置，然后以 T+1 时刻的位置为起点，反向追踪这些点在 T 时刻的位置，此时计算重新回到时刻 T

的这些点与本来位置之间的误差，选取误差小于平均误差一半的那些点来作为追踪器的最佳跟踪点。最后根据这些最佳跟踪点的位置以及形变信息来确定 T+1 时刻的目标包围框体的位置和大小。

### 2.2.3 学习模块

TLD 中的学习模块负责对追踪以及检测模块中的模型进行更新学习，在学习模块中的核心是两个纠正器，分别称为 P 专家与 N 专家。对于一个物体的长时间运动来讲，它出现的位置应当具有结构性，这个结构性分为时间结构性与空间结构性。

P 专家的作用是保证时间局部性，即目标在不同帧图像之间的位置应当可以构成连续的轨迹。对于某帧图像中由追踪器预测的一个位置，如果它在检测器中的分类结果为负，P 专家就会将这个位置改为正，以此保持物体的位置形成连续轨迹。

N 专家的作用是保证空间局部性，即每一帧图像中物体最多只能够出现在一个位置上。检测器与 P 专家可能一起产生了多个正样本，但是同一时刻只能有一个目标，N 专家就从这多个正样本中挑选出最可能的一个，然后将这个唯一的结果作为 TLD 算法的追踪结果输出，同时也用这个位置的样本框来初始化追踪器。

所以，在整个 TLD 算法中，检测器是负责监督追踪器，来改正发生在追踪过程中的错误。追踪器则负责监督检测器的训练过程，用自己的结果对检测分类的结果进行纠正。算法的每一次跟踪计算的流程如图 2.15 所示：

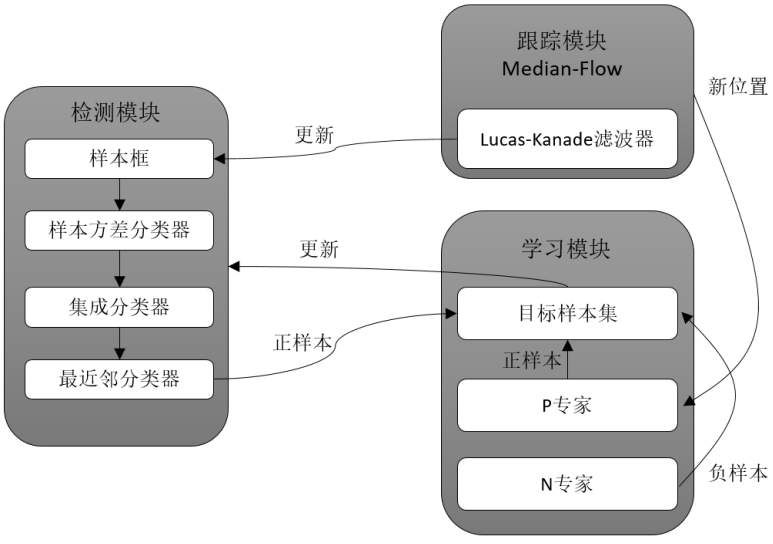


图 2. 15　TLD 每次跟踪算法执行流程

首先，检测器根据目标框的大小在一帧图像中划分样本框体，然后利用级联分类器进行分类，对于通过分类器的框体标记为正样本。追踪器负责计算得到物体在这一帧图像中的位置，P 专家负责在这个位置附近挑选出正样本。以上两阶段所产

生的的正样本一起交给 N 专家，N 专家从中筛选最为可信的一个作为算法的输出，同时利用这个最新的样本来更新检测器的相关参数。TLD 算法在这种不断学习更新的过程中可以保持对于物体在长时间运动情况下跟踪的准确性。

## 2.3 移动 GPU 结构

本课题的关注以及实验平台为移动设备上的 GPU。为了更好地针对移动 GPU 进行加速工作，本课题分别选取所搭载 GPU 支持 OpenCL 以及 CUDA 的两款代表性芯片——高通 801 以及 Tegra K1。下面对这两款芯片的结构特点以及其 GPU 进行介绍。

### 2.3.1 高通 801 与 Adreno 330

高通 801 芯片是美国高通公司 2014 年发布的一款晓龙系列移动处理器中的产品，很多中高端手机都采用了这一款芯片，比如三星 Galaxy S5[50]，小米 NOTE[51] 以及索尼 Xperia Z3[52]等。图 2.16 中展示了高通 801 处理器的基本模块结构：

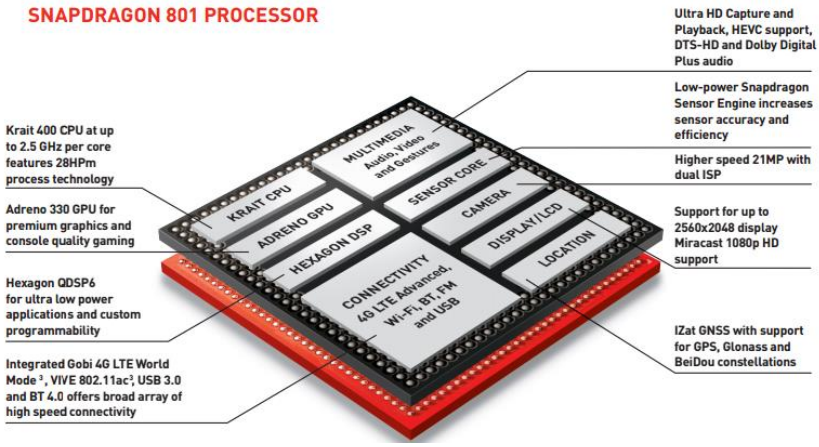


图 2. 16　高通 801 处理器基本模块结构

高通 801 芯片搭载 Krait 400 架构 CPU，针对多线程以及多任务做了定制优化。缓存采用 LR-DDR3 来加速数据存取，提升处理器性能。CPU 的峰值计算能力约为 3.6GFlops 每秒，峰值频率 2.5GHz。而对于本课题重点关注的 GPU，高通 801 采用的是 Adreno330GPU，这款 GPU 的基本参数以及软件支持如表 2.1 所示：

表 2.1　Adreno330GPU 基本参数

| 指标名称 | 主频 | 峰值性能（半精度） | 制 程 工艺 | ALU 数量 | 支持 API |
|---|---|---|---|---|---|
| 指标参数 | 450MHz | 129.8GFlops | 28nm | 128 | OpenCL1.1<br>Direct3D 11.1 |

最终，考虑到 Adreno 330GPU 对 OpenCL1.1 的支持以及多达 128 的 ALU 数量，本课题选取高通 801 芯片作为实验平台来实现 CNN 在移动设备上的运行以及 GPU 加速。

### 2.3.2 Tegra K1 与 GK20A GPU

Nvidia 公司的 Tegra 系列芯片是移动端市场上另外一款非常通用的移动芯片。本课题选取的是 Nvidia 于 2014 年发布的一款高端处理器 Tegra K1，它是 Nvidia 为了应对移动端设备对于计算的增长需求而专门开发的一款高性能产品。现在已经被广泛使用在了比如平板电脑、无人飞行器甚至 Tesla 电动汽车这些对计算能力要求很高的设备之上。Tegra K1 的整体架构如图 2.17：

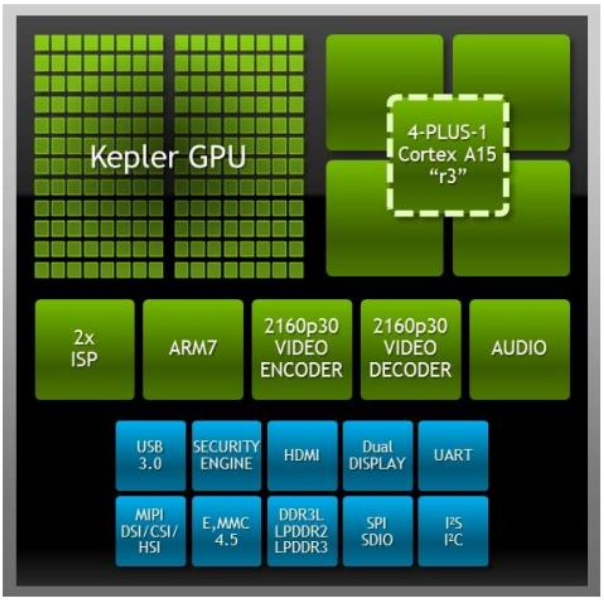


图 2. 17　Tegra K1 模块结构

从图 2.17 中可以看出，Tegra K1 采用的是 Cortex A15 “r3” 四核 CPU 并搭载一颗 Kepler 架构的 GK20A GPU。A15 的主频为 2.3GHz，共有 8GB LP-DDR3 内存。配合 Nvidia 第三代的节电核心，可以实现 CPU 的高性能与高续航能力并存。整个 CPU 的 SMP（对称多重处理）架构可以让 CPU 灵活的进行切换，在 CPU 负载比较重的时候，启用全部四个高性能核心以实现超强性能，而每一个核心可以根据工作负荷自动独立的启用和关闭。

更加值得注意的是 Tegra K1 的 GK20A GPU，与以往的移动 GPU 不同，Nvidia 在 Tegra K1 上采用与桌面级 GPU 相同的 Kepler 架构来构建移动 GPU，并且针对系统进行了一系列的优化来实现移动计算领域的超高性能同时考虑功率消耗问题。与桌面 GPU 动辄多达几千个单精度 CUDA 浮点计算核心不同，Tegra K1 的 GK20A GPU 包含 192 个 CUDA 计算核心在保证计算能力的同时功耗仅仅只有不到 2W。

192 的计算核心数也已经超越了几年前的桌面 GPU 的计算核心数目。

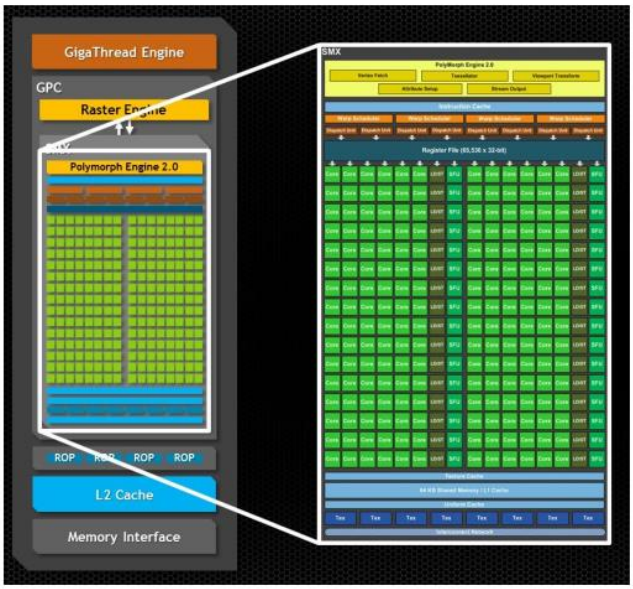


图 2. 18　GK20AGPU 模块结构

Kepler 架构的 GK20A GPU 由 GPC（图形处理集群），SMX（流处理器群）以及内存控制器组成，如图 2.18。以 Nvidia 的桌面 GPU GeForce GTX 680 GPU 为例，它包含 4 个图形处理集群、8 个流处理器群以及 4 个内存控制器。Tegra K1 上的 GK20A GPU 则由 1 个图形处理集群、1 个流处理器群与 1 个内存控制单元构成。GK20A GPU 中包含了 4 个 ROP 单元并且在 ROP 与内存接口中有 128KB 大小的二级 Cache 做中间缓冲。由于 GPU 结构的相同，Tegra K1 GPU 支持的编程特性与桌面级 GPU 也相同，有 OpenGL ES 3.0、OpenGL 4.4、DX11、Tessellation 以及 CUDA 等。

除了目标高性能的设计之外，GK20A GPU 也关注了整个芯片低功耗特性的保持。为此，在桌面级 GPU 通常采用的功耗效率设计之外，NVIDIA 还采用了诸如色彩压缩、元素提出等技术来降低 GPU 的功耗。

本课题经过调研，鉴于 Tegra K1 对于 CUDA6.0 的开发支持以及其所具备的优异性能，选取搭载 GK20A GPU 的 Tegra K1 来作为开发平台之一，对于 TLD 算法进行高性能实现。

## 2.4 本章小结

本章主要介绍了课题研究内容相关的一些背景知识。首先介绍了深度学习领域取得的巨大成功，特别是在图像分类领域大放异彩的卷积神经网络。重点介绍了它的组成单元、卷积原理，并对其中使用的一些提升分类准确度与降低参数数目的策略。然后又引出了我们在移动 GPU 上实现卷积神经网络的基础——MXNet 框

架。对 MXNet 的设计思想与内存优化进行了重点介绍，这也是我们在众多主流框架中选择 MXNet 的原因。接下来又介绍了我们研究的另一个计算机视觉核心算法——TLD 算法。我们对 TLD 的跟踪、分类以及学习模块分解进行了介绍，探讨了它在单目标长时间跟踪问题上表现优异的原因。最后介绍了我们的实验平台，高通 801 芯片以及 Tegra K1 芯片，给出了这两款芯片所搭载的 Adreno 330GPU 与 GK20A GPU 的主要参数以及提供的编程 API 支持，为接下来工作中的优化打下了基础。

# 第三章　CNN 的高性能实现

本章主要介绍了本课题的主要工作之一：在含有一个高通 801 芯片的小米 note 手机上，实现了卷积神经网络的前向分类过程，然后介绍了一些 OpenCL 的应用背景以及如何利用 OpenCL 使用手机上的 Adreno 330 移动 GPU。最后依据手机上对于程序计算负载的测试结果，重点加速了卷积层网络的算法实现，并且做了分块矩阵乘法、向量化等优化操作，最终取得了 2.1 倍的加速比。

## 3.1　CNN 计算负载分析

对于卷积网络在移动设备上的应用，需要实现的是一个图像经过神经网络得到分类结果的过程，而不需要训练这个神经网络，也就是我们通常所称的前向过程。与前向过程想对应的则是后向过程，主要负责通过训练来更新网络的参数，这一部分工作往往是在高性能服务器上进行的。为了简化在手机上卷积神经网络的前向过程实现代码，我们首先从 MXNet 框架中分离出实现前向过程的主要函数，然后在 Ubuntu 主机上通过交叉编译器进行编译，生成可在移动设备上执行的程序，再将程序通过 ADB[53]接口发送到移动设备，检验执行结果。

在保证算法结果正确的同时，为了之后进行针对性的加速工作，我们首先测试了不同网络在手机上进行前向分类过程的执行时间，为了适应手机上的内存以及计算能力情况，我们选取了三种不同的网络——FastPoorNet，Inception-full 以及 Inception-sub 进行了计算负载的测试。FastPoorNet 是 MXNet 开源社区为计算能力较弱的设备所设计的网络结构，在保证分类精度与 AlexNet 相当的同时尽量减少计算规模；Inception-full 则是由 Google 提出的首个在图像分类测试中达到 95%准确率的网络；Inception-sub 是 Inception-full 网络的一个精简版本。关于这三种网络的测试结果如表 3.1 所示：

表 3.1　卷积神经网络计算负载情况测试

| 网络类型 | FastPoorNet[54] | Inception-sub | Inception-full[55] |
|---|---|---|---|
| 前向执行时间 (ms) | 1307 | 12050 | 21930 |
| 卷积层执行时间 (ms) | 1037.16 | 11610 | 20130 |
| 卷积层时间占比 | 79.34% | 96.35% | 91.79% |

从测试所得数据可以看出，对于三种不同的网络，卷积层所消耗的时间都占总时间的绝大部分，最低有 79.34%，最高到了 96.35%。为了确定卷积层内的计算任务大小以及具体每一次卷积层的耗时情况，我们对 FastPoorNet 网络的卷积层计算

进行了更加细致的测试，在 FastPoorNet 中含有 7 个卷积层，测试结果如表 3.2：

表 3.2　FastPoorNet 各卷积层具体执行参数

| 网络层 | 执行时间 | M | N | K |
|---|---|---|---|---|
| 卷积层 0 | 245.99 | 2916 | 64 | 363 |
| 卷积层 1 | 242.82 | 625 | 192 | 576 |
| 卷积层 2 | 83.98 | 144 | 96 | 1728 |
| 卷积层 3 | 69.67 | 144 | 160 | 864 |
| 卷积层 4 | 105.85 | 144 | 144 | 1440 |
| 卷积层 5 | 103.48 | 144 | 160 | 1296 |
| 卷积层 6 | 185.347 | 144 | 256 | 1440 |

其中，M、N 以及 K 代表了 MXNet 在使用矩阵乘法计算卷积层时矩阵规模的大小情况（设矩阵乘为 A*B=C，则 A 的规模为 M 行 K 列，B 的规模为 K 行 N 列，C 的规模为 M 行 N 列）。从测试结果可以看出，卷积计算涉及到的矩阵规模还是相当大的，适合在 GPU 上进行并行加速执行。综合上面的测试情况，我们决定对于卷积层使用 GPU 进行高性能实现。

卷积层中具体的操作流程在第二章我们已经介绍过，而在进行算法功能的编程实现的时候，卷积层中的运算主要有以下三种实现方法：直接卷积法、矩阵乘方法以及 FFT 法。

直接卷积法就是按照卷积操作的基本计算原理来进行计算，优点是计算规则简单标准、不会产生多余的数据计算，缺点则是由于卷积计算时卷积核在整个图像上移动，对应到内存上来说就是很多不连续的数据访问，会造成访存时间的巨大开销。权衡利弊，很少有深度学习框架采用这种方式来实现卷积层的具体操作。

矩阵乘方法是在主流深度学习框架中应用较为广泛的一种方法，通过对图像数据与卷积核数据进行重组来将卷积运算转化为矩阵乘法运算，最终达到减少乱序存储访问，加速算法执行速度的目的。关于矩阵乘方法，Szegedy[55]给出了详细的说明，下面我们以一个简单的卷积计算为例来理解一下这个转换过程：图 3.1 中给出了一个卷积计算的原始方法以及矩阵乘方法。输入是一个 3 通道的 3*3 图像，输出为 2 个 2*2 的特征矩阵，因此共有 2*3=6 个卷积核。若采用矩阵乘法来计算，则将三个通道的图像数据依据进行卷积操作的顺序进行重新组合，得到一个新的较大的图像数据矩阵，并且将六个卷积核组合成为另一个矩阵，每一组卷积核构成一列，则卷积之后的图像特征就在两个矩阵相乘所得的结果之中。途中两个卷积后的特征矩阵分别就是矩阵乘法最终矩阵的两列。矩阵乘方法计算卷积的缺点则是由于必须要重组图像与卷积核数据，因此会带来额外的计算开销；同时，由于数据

有重复存储，也增加了存储开销。

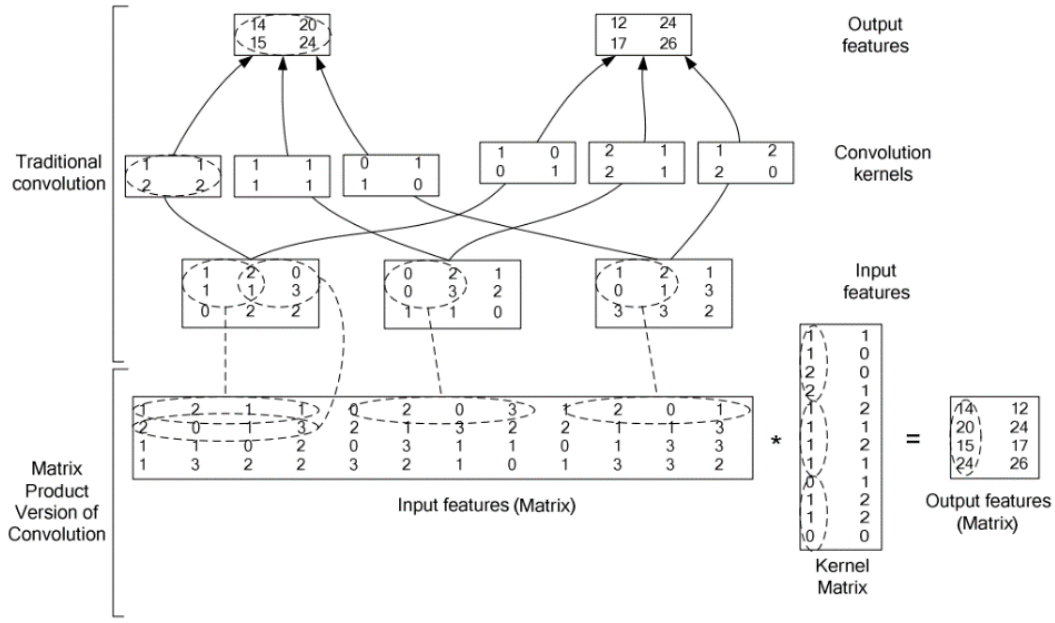


图 3.1　矩阵乘法计算卷积原理

FFT 法[56]是另一种计算卷积的方法。其所用的原理是时域卷积与相应的频域相乘结果是对应的。输入数据经过离散傅里叶（DFT）变换转换到频域，然后与滤波器的频域响应相乘，最后再用反离散傅里叶变换返回时域。从傅里叶的时代开始人们就知道了有这样一种计算卷积的方法，但是一种没有采用，原因就是离散傅里叶变换是比卷积计算更加耗时的操作。直到 1965 年 FFT（快速傅里叶变换）的出现，局面才有所改变，DFT 可以通过 FFT 来计算，这样，整体来看，FFT 计算卷积的速度更快，因此，FFT 卷积计算方法通常也被称为高速卷积。

但是相比于矩阵乘法，很少有深度学习框架会采用 FFT 来计算卷积层，原因是由于矩阵乘法的底层函数支持相比于 FFT 来说更加完备，因此，在我们的课题研究实现中，也选择使用矩阵乘法来实现卷积层的计算。

在 MXNet 的框架实现中，完成前向过程以及其中主要的卷积层计算的流程如下：

（1） 根据网络结构，调用 Init_Map()函数来构造整个网络的计算图。

（2） 调用 Bind_Map()函数，来将训练好的网络参数输入网络计算图中。

（3） 调用 Forward_Op()函数，开始前向过程的运算。这里的 Op 代表网络层的类型。在卷积层中，首先调用 Im2col()函数来将图像数据与卷积核数据进行重组，然后调用 cblas_sgemm()来完成矩阵计算，得到结果。

## 3.2 算法基础实现

高通 801 芯片所搭载的移动 GPU——Adreno330 支持 OpenCL 1.1 标准框架，

因此，我们可以使用 OpenCL 将卷积计算转移到 GPU 上来执行。OpenCL 是一个开放的、无版权的跨平台并行编程标准，它提供了一个标准接口来给程序员进行编程。制造计算设备的硬件厂商来实现这个接口并将其实现作为运行库封装在生产的设备之中。OpenCL 最初由苹果公司开发，拥有其商标权，并在与 AMD、IBM、英特尔和 NVIDIA 技术团队的合作之下逐步完善。在 OpenCL 中，把进行运算的处理器称做设备(devices)。一个 OpenCL 设备拥有一个或者多个计算单元(compute units)。属于同一个工作组(workgroup)的线程一起放在一个计算单元上执行。一个计算单元又由一个或多个本地处理单元（processing element）和本地内存组成。图 3.2 就是 OpenCL 编程模型的架构示意图。在 OpenCL 中，主机处理器 Host（通常是 CPU）来管理 OpenCL 的运行环境上下文以及计算设备，程序员需要进行选择来将计算密集的任务编程成为一个 kernel 文件，然后选择一个计算设备来执行这个任务。OpenCL 模型中有两个规模（Size），局部规模用来指定在每个计算单元中有多少个线程，全局规模则用来指定线程空间中一共有多少个线程。对于线程空间的组织，可以是一维、二维或者三维。每个计算单元都会独立的执行编写的 kernel 程序，于是就达到了并行计算的效果。

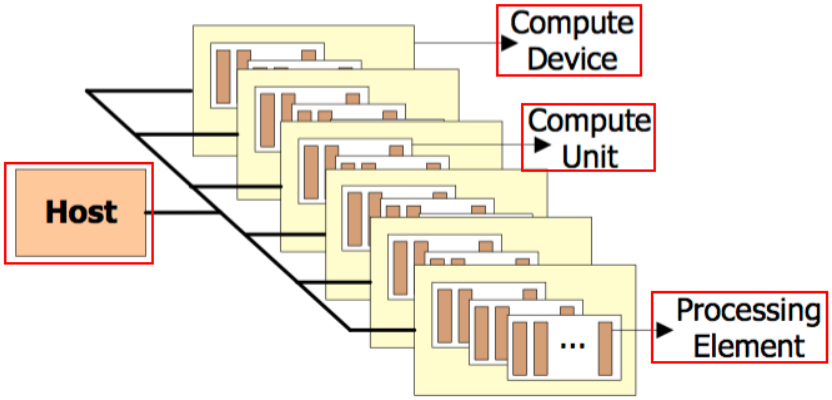


图 3. 2　OpenCL 编程架构

对于我们在 GPU 实现的矩阵乘法，首先完成了一个确保功能正确的基础实现。在矩阵乘计算卷积方法中共有三个矩阵 A、B 与 C。矩阵 A 表示图像数据，矩阵 B 是卷积内核，矩阵 C 则用来存储卷积计算的结果。这里 A 具有 M 行和 K 列，B 具有 K 行和 N 列以及 C 具有 M 行和 N 列。在 MXNet 对于卷积层的前向计算设定中，矩阵 A，B 和 C 均使用列主序来存储数据。我们设置了局部大小为{1,1}和全局大小{M，N}，在此条件下，我们有 M * N 个工作线程，每个工作组包含一个线程，每个线程计算 C 的一个元素，这就是我们 GPU 矩阵乘法的基本内核。

## 3.3 基于 Adreno 330 优化

本节将针对算法的负载以及移动 GPU 的结构特点来进行一些优化来提高计算内核的性能。我们先介绍一些在数据准备阶段进行的基本优化。

首先，由于 GPU 的结构决定了它在处理 ALU 运算时速度快，处理条件分支指令时却相当缓慢，我们可以通过对矩阵规模进行归一化，来减少由于矩阵尺寸所引起的条件分支。同时，为了使添加的无效数据尽可能的少，我们用零填充矩阵使之行列数都是 32 的倍数。

第二，我们需要恢复矩阵 A 为行主序存储而不是列主序存储。在矩阵乘法计算中，计算 C 中的一个元素，需要 A 的一整行和 B 的一整列。如 3.2 节所述，在 MXNet 的前向实现中卷积层的三个矩阵都是使用列主序来进行存储的。换言之，进行矩阵乘法的时候，B 的每一列中的元素可以被连续地访问，A 中每一行的元素却会被打乱。所以这种存储顺序调整有助于提高矩阵乘法中的连续访存比例，加快算法执行速度。

### 3.3.1 优化任务划分

在我们的基本实现 GPU 内核中，每个线程仅计算 C 矩阵中的 1 个元素而且局部规模的大小为 1，并没有对于 GPU 的计算资源进行充分的利用。因为 Adreno330 有 4 个计算单元而且在每一个计算单元中都有 32 个 ALU，并支持 16 个线程同时运行。但按照现在的算法，当某一工作组被装载到某个计算单元。其中只有一个线程正在运行，并且它仅计算一个元素。每个线程工作太少会使得线程结束很快，接着会引起过多的线程和工作组的交换，这都会增加时间成本。因此，我们采取分块矩阵乘法方法[57]来将 C 的子块分配到不同的工作组，增加每个计算单元的任务负载，达到更加均衡的并行。

该方法原理示意如图 3.3，为了计算矩阵 C 的一个子块（紫色），需要 A 的相应的行（绿色）和 B 的相应的列（蓝色）。现在，如果我们利用分块（AUB 和 BSUB）来划分 A 和 B，那么我们就可以将 C 的子块的计算转化为了两次矩阵块的乘法计算，然后将两个矩阵块的计算结果相加即可。

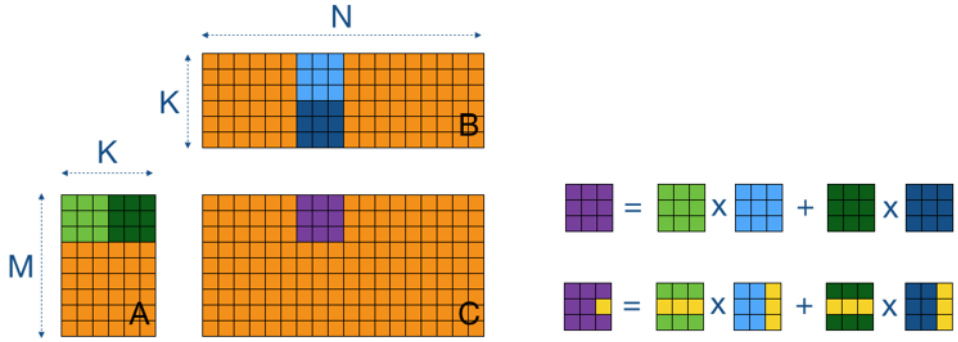


图 3. 3　分块矩阵乘法原理

为了达到最好的任务划分效果，对于局部规模大小我们进行了很多尝试与测验，最终设置局部大小为{1，32}和全局大小{M /32，N}。这意味着现在每 32 个线程组成一个工作组，那么我们总计拥有 M * N /1024 个工作组。在这种情况下，每个工作组负责矩阵 C 的 1 个子块，这个分块被分成 32 个列。工作组中的每个线程将计算包含 32 个元素的一列，也就是 32 个元素。通过这样的任务划分，我们使得 GPU 负载更加均衡、合理。

### 3.3.2 片上存储优化

现在，一旦一个线程需要 A 或 B 的数据，将直接从全局存储器中来获取值，这将导致过多的全局内存访问并且会减慢速度。在移动 GPU 中，每个计算单元独立具有的存储器称为本地存储器，本地存储器中的数据访问速度高于全局存储器，并且在本地存储器中的数据可以在此工作组中的线程之间共享。这促使我们设法使用本地内存来重用数据。

我们首先使用 OpenCL-Z[58]应用对于实验平台上的 GPU Adreno330 存储速度进行了测试，测试结果如图 3.4：

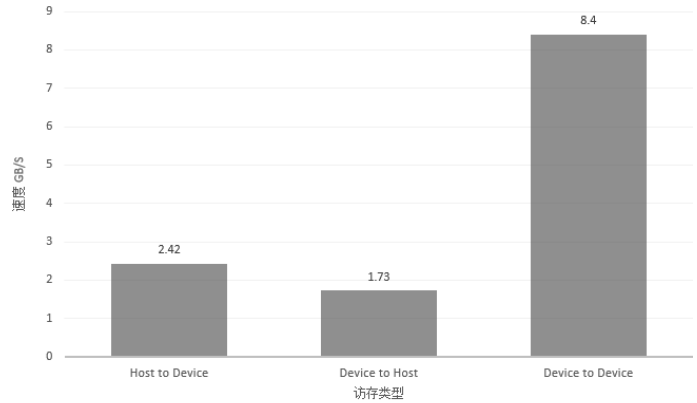


图 3. 4　Adreno330 访存测试结果

从测试数据可以看出，片上存储（Device to Device）的速度 8.4GB/S 是最快的，约为从 CPU 拷贝数据到 GPU（Host to Device）的 3.47 倍，同时也是从 GPU 拷贝数据到 CPU（Device to Host）的 4.85 倍。所以，减少 CPU 与 GPU 之间的数据通信，尽量利用片上存储可以提升算法实现的性能。

在 Adreno 330 中片上存储器的大小为 8192 KB，而计算过程中采用单精度浮点数进行，每个单精度浮点数需要 4B 进行存储，那么片上存储可以容纳 2048 个单精度浮点数。那么对于每个线程块的计算，我们将每次它会使用到的 ASUB 与 BSUB 两个子块共 2*32*32=1024 个数据从 CPU 读取到 GPU 的片上存储，之后线程块中每个线程从片上读取数据，而不是从全局存储器来读取计算的数据。通过这种方式，在本地存储器中的每个元素将被用于工作组中的 32 个线程。也就是说，

我们成功的将全局存储访问量减少到只有原来的 1/32，通过优化数据的读取方式与效率来提升算法执行的速度。

### 3.3.3 循环展开与向量化

在矩阵乘法的实现中，由于有很多按序的乘加操作，所以，循环就成为了算法执行的主体部分。而在每一次循环的时候都会产生一次条件判断，由于 GPU 结构的原因，我们希望尽量减少这样条件判断的次数。循环展开就是一种解决方案。

循环展开是一种循环变换技术，可以加快循环执行的速度，其代价则是代码二进制文件的大小会增加。由于 GPU 对于条件分支的执行很慢，而循环展开会减少甚至消除分支的数量，所以循环展开在 GPU 上将带来可观的效益。而另一方面，更大的循环体给编译器提供了做更多优化的机会。值得一提的是，并不是越大的循环体就会取得越好的性能，这里面还有一个高速缓存交换的问题。

在具体实现上，简单来说循环展开就是将多个循环体的操作写在同一个循环体中，通过修改循环控制变量以及增量来保持算法执行的正确性。在本课题的实现中，经过对性能以及代码文件大小的综合考量，我们选择以 8 为距离进行循环展开，即将 8 个循环体进行合并成为一个新的较大的循环体。

目前，在我们的卷积计算内核中的数据类型为 4 个字节的单浮点类型。我们希望将指令转化成为 SIMD 指令流，这可以通过向量化来做到。向量化是一种特殊的并行计算方式，相比于一般的程序，与同一时间内只会对一个操作数进行一种操作不同，向量化的计算可以在同一时间内对多个数据进行多各种操作。向量化是高性能计算中一种重要的优化策略。

在 Adreno 330 的硬件层次不支持向量运算（如向量乘或向量加），但他们确实有专为全局和本地存储器特殊设计的更宽的读取和存储指令。我们可以利用这些指令对于数据的存取操作进行向量化。OpenCL 还支持简单的向量数据类型，这使得我们也不必将矩阵转换成向量类型就可以享受到向量化带来的收益。Adreno 330 GPU 所支持的浮点向量类型有 float4、float8 与 float16 三种，我们对于这三种数据类型都进行了实验，其中 float4 的性能提升最大，所以，我们用 float4 类型重写了数据的存取代码。图 3.5 中就是这一阶段优化前与优化之后的代码示例：

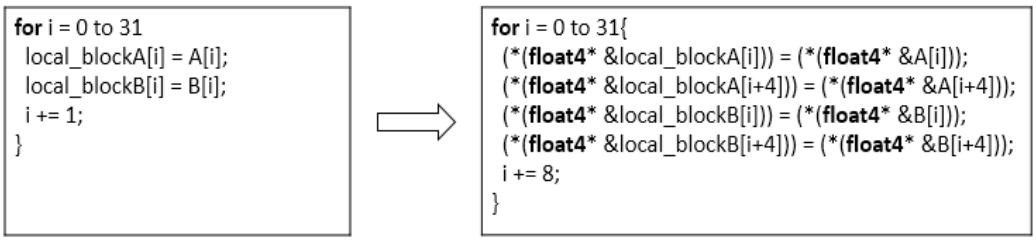


图 3.5　读取数据阶段优化对比

## 3.4 Android 应用的构建

在使用 C++与 OpenCL 在移动设备上正确实现了卷积神经网络的功能之后，程序的功能是完备的，但是并不是很方便使用。在手机的 ADB shell 命令行中运行程序，使用者并不能看到自己要分类的图片是什么样子，也只能够分类已经存放在手机中的图片。为了解决这个问题，我们将算法的实现放入一个 Android 应用程序之中，使用者可以通过这个交互性比较好的程序来利用 CNN 的分类功能。

由于我们的算法实现使用 C++代码，调用 GPU 使用 OpenCL 代码，而 Android 应用的开发使用的是 Java 语言，因此，在功能的整合上，需要进行一定的工作。为了在 Android 应用中调用已经实现好的 C++代码，需要使用 JNI[59]方法。JNI 即 Java Native Interface 的缩写，是连接上层应用实现与底层函数功能的桥梁。早在 Java 诞生之前，很多函数或者工作已经用早一些的语言实现并且优化成功了，但是在 Java 中，难道我们还要重新再次实现一遍么。JNI 就解决了这个问题，通过它我们可以自由使用 Native 语言所编写的函数功能。

通过 JNI 进行 C++函数功能的调用，主要有以下几个步骤：

（1） 编写带有 JNI 关键字的函数接口，这是 Java 层与 C++层通信的通道。

（2） 在 C++卷积神经网络的算法实现中，添加上面定义的函数接口，并且将图像分类的功能封装在其中。

（3） 利用交叉编译工具，将带有 JNI 接口的 C++文件编译生成静态链接库.so 文件。

（4） 在 Android 的 Java 应用实现中，首先加载这个.so 文件，然后就可以在代码中调用第（1）步中所定义的接口函数了。

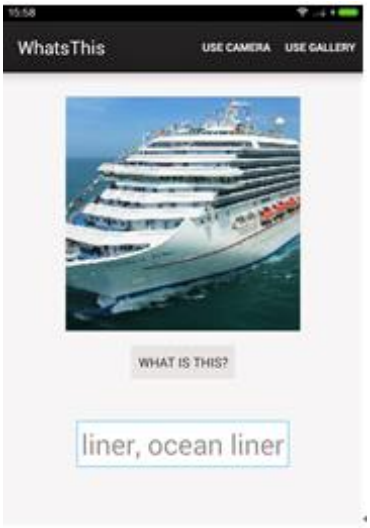


图 3. 6 Android 程序运行截图

图 3.6 中就是我们实现的 Android 应用在进行图像分类时候的屏幕截图。中间

就是进行分类的图片，下面的文本框中显示经过卷积神经网络的分类结果。我们的应用实现不仅能够从相册中选取某一张图片进行分类，还能够直接从摄像头拍照进行分类，大大增强了卷积神经网络在图像分类领域在手机上的适用性。

## 3.5 评测与分析

本课题采用的实验平台是小米 Note 手机，采用高通 801 处理器，搭载 Adreno330 GPU。关于测试主要分为三个方面：

### 3.5.1 通信带宽测试

GPU 的优点表现在对于大规模的计算速度很快，而它与 CPU 之间的数据通信往往是算法性能的瓶颈所在。所以，为了检验我们对于卷积层计算的加速效率，我们对于 GPU 的带宽利用进行测试。

首先，我们需要测试 GPU 的理论带宽。由于在 GPU 中，存储主要分为全局存储与共享存储，所以，对于带宽的测试也要分为这两个方面。带宽的理论值已经在之前有 OpenCL-Z 测出，我们这里还需要卷积层实现的带宽使用情况，因此，我们通过矩阵计算的规模以及程序运行的时间，达到了我们高性能实现的带宽情况，并且与理论值进行了对比，这部分测试的结果如图 3.7：

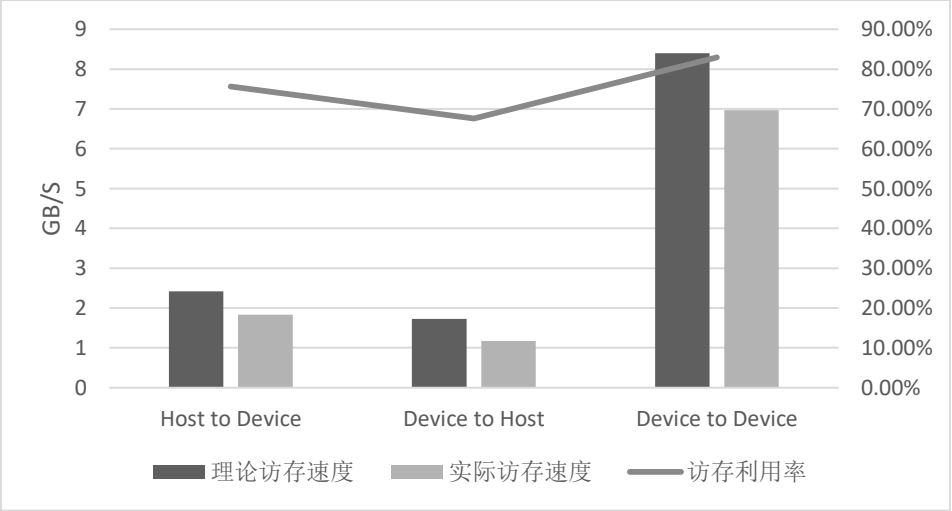


图 3. 7　算法实现访存效率测试结果图

从图中可以看出，我们实现的程序在访存的利用率上取得了令人满意的表现，访存的利用率稳定在 70%到 80%左右，其中从 GPU 到 CPU 的访存利用率最低，GPU 的片上存储利用率最高。这可能是由于 GPU 片上存储总的数据访问量比较大，而计算后从 GPU 拷贝回 CPU 的数据量比较小所导致。

### 3.5.2 矩阵乘法速度测试

我们首先使用 FastPoorNet 来测试矩阵乘法在不同内核上的执行时间。结果展示在图 3.8。从实验结果中，我们可以看到，我们利用 GPU 比使用 CPU 获得了更好的性能。对于移动 GPU，带宽的利用可能是限制程序加速效果最重要的因素，因为我们使用片上存储对程序进行优化之后速度提升了 3 倍；使用向量加载/存储指令将性能提升了 1.25 倍；而循环展开又将性能提升了 1.16 倍。总的来说，我们的在移动 GPU 上的高性能实现是非常有效的。

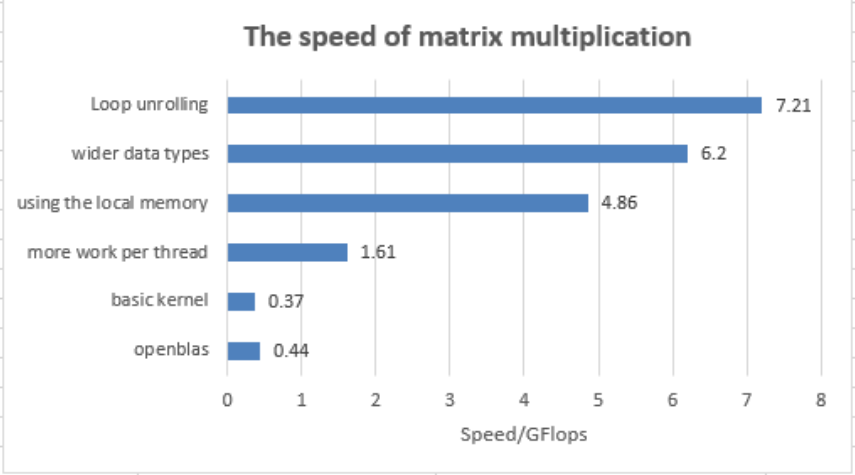


图 3. 8　算法实现矩阵乘速度测试图

### 3.5.3 分类时间测试

表 3.3　CNN 高性能实现不同网络测试结果

| CNN Type | FastPoorNet | Inception-sub | Inception-full |
|---|---|---|---|
| CPU version Time / s | 1.30 | 12.05 | 21.93 |
| GPU version Time / s | 0.61 | 5.66 | 10.31 |
| Speed Up | 2.131 | 2.129 | 2.127 |

这一部分的测试中，我们采取了不同的 CNN 网络来对于整个图像分类时间进行测试。CPU 版本即为 MXNet 中提供的 CPU 实现，使用 openblas 进行加速；GPU 版本即为我们采用 GPU 加速的高性能实现。尽管在卷积计算中使用移动 GPU 会带来额外的工作，比如说数据的重新组织以及传输等，但是在整个图像分类的前向过程中，我们也取得了很好的效果。我们的测试结果显示在表 3.3 中，可以看到，对于不同的 CNN，我们的程序都能取得大约 2.1 倍的前向过程加速比，我们的高性能实现可以明显加速算法的执行，并且对于不同的卷积神经网络类型有着比较稳定的加速效果。

## 3.6 本章小结

在本章中我们针对移动 GPU 提出了一个 CNN 前向过程的加速算法，将最耗时的卷积操作利用矩阵乘实现，然后通过使用 OpenCL 将其转移到计算性能更加强大的 GPU 上来实现加速。此外，我们也探索了诸如优化任务划分、利用片上存储、向量化以及循环展开等等技术来取得更好的结果。最终，我们将矩阵乘的速度由 CPU 版本的 0.44GFLOPS 提升了整整 16 倍，达到了 7.21GFLOPS。实验结果表明，该算法可以将推断任务的执行时间减少到只有一半。对于我们使用移动 GPU 加速之后的算法，由于卷积层所花费时间的下降，造成全连接层等其它神经网络层耗时所占比例就会相应的增加，为此，在之后的研究方面，可以考虑对所有网络层进行加速。

# 第四章　TLD 算法的高性能实现

本章主要讲述了在 Tegra K1 移动平台上针对 TLD 单目标长时间跟踪算法进行 GPU 加速实现。我们首先测试分析了 TLD 算法中不同模块的不同部分在计算负载上的差别，然后简要介绍了将要使用的 CUDA 编程框架。最后利用 CUDA 编程框架将耗时最多的跟踪模块中的过滤器部分进行了并行加速实现，并且讨论了一些针对算法以及 Tegra GPU 的优化策略。最后对于加速后的算法与原始算法进行了测试比较分析，取得了令人满意的加速效果。

## 4.1　TLD 算法计算负载分析

目标跟踪作为计算机视觉中的一个复杂度比较高的核心问题，它的算法往往需要执行比较长的时间。为了在移动平台上达到更好的加速效果，我们首先对于 TLD 算法进行测试，统计各个阶段的执行时间，找到计算任务最重的部分。

我们在测试的时候使用的台式机搭载 Intel 的 I7-2600 处理器，主频 3.4GHz，内存 8GB。代码采用 Orthocenter 基于 OpenCV 所做的 C++算法实现[60]。表 4.1 中是程序测试的结果：

表 4.1　TLD 基准算法计算负载测试结果

| 视频种类 | 分辨率 | 跟踪耗时 | 检测模块 | 跟踪模块 | 学习模块 |
|---|---|---|---|---|---|
| Car | 320*240 | 33.21 | 24.36 | 1.20 | 2.44 |
| Data1 | 252*288 | 40.13 | 32.21 | 2.48 | 3.57 |
| Data2 | 640*480 | 96.45. | 60.57 | 3.21 | 2.88 |

从表中的测试数据我们可以看出：

（1） 不同分辨率的视频的目标跟踪算法计算时间不同。分辨率越高的视频所需要的时间越长，这主要是由于高分辨率的视频图像中每一帧画面里有更多需要检查的样本框，导致检测模块与跟踪模块的任务增加所致。

（2） 在 TLD 算法的三个主要模块中，耗时最久的是检测模块，跟踪模块与学习模块的耗时与检测模块相比差距很大；而且跟踪模块和检测模块的耗时多少随分辨率的增长变化不大，但是检测模块所需要的计算时间随着视频分辨率的增长显著增加。

表 4.2　TLD 基准实现检测模块计算负载测试

| 视频种类 | 分辨率 | 方差分类器 | 集成分类器 | 最近邻分类器 |
|---|---|---|---|---|
| Car | 320*240 | 1.14 | 3.65 | 17.58 |
| Data1 | 352*288 | 2.66 | 15.41 | 16.45 |
| Data2 | 640*480 | 2.17 | 45.95 | 11.15 |

因此，我们得出结论，在 TLD 算法中，决定算法运行速度的瓶颈阶段就是检测模块。之后，为了更进一步的了解，我们又对检测模块中的不同分类器进行了测试，得到的结果如表 4.2。

从测试结果中我们可以得出结论：在 TLD 算法检测模块的三级级联分类器中，一般来说集成分类器与最近邻分类器的耗时比较长，方差分类器的耗时相对较少而且随视频分辨率的增长变化不大。这与我们预期的结果也是一致的，因为方差过滤器的计算原理最为简单，而集成分类器与最近邻分类器的计算过程则相对比较复杂。

通过上面的测试分析，我们决定将检测模块中的级联分类器部分作为我们的加速重点，通过提升这一瓶颈段的执行速度来提升算法性能。

## 4.2 CUDA 编程模型

与第三章中所介绍的统一编程框架 OpenCL 类似，CUDA（Compute Unified Device Architecture）也是一种被广泛采用的高性能编程模型，它是 Nvidia 公司为了促进在 GPU 上通用高性能计算的发展而在 2007 年推出的。凭借着 CUDA 编程模型在 NVIDIA 所生产的 GPU 上的出色表现，其成为了最受欢迎的高性能编程框架。

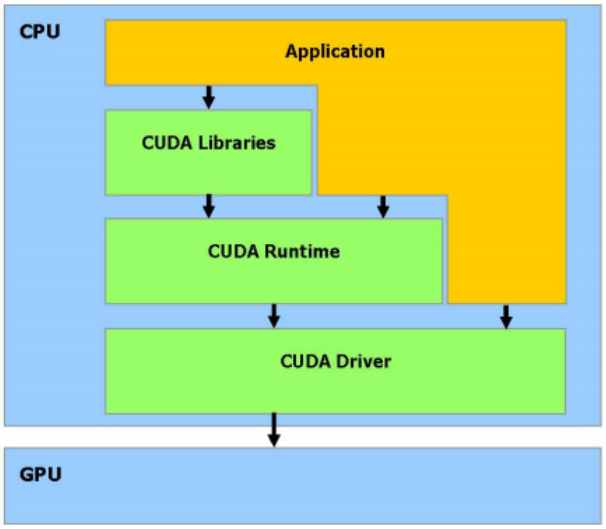


图 4. 1　CUDA 软件堆栈结构示意图

在 CUDA 推出之前，尽管 GPU 中包含着强大的计算能力，但是它主要支持的是图像相关的应用，比如各种 3D 游戏等。GPU 只能通过图形 API 来编程，导致 GPU 的应用范围比较狭窄，并不是通用计算的加速选择。CUDA 的出现使得 GPU 的开发脱离了图形 API 的束缚，应用背景也扩展到了整个通用计算领域。图 4.1 就是 CUDA 的软件堆栈结构示意图。

整个 CUDA 软件结构分为几个层次，首先是硬件驱动程序 CUDA Driver 和应用程序编程接口 API 以及运行时环境 Runtime。在这之上便是高一层次的 CUDA 运算库，比较典型的有 CUBLAS 以及 CUDNN 等。CUDA 的编程接口与 C 语言非常类似，因此对于程序原来说学习成本很低，这也是它受到欢迎的重要原因之一。

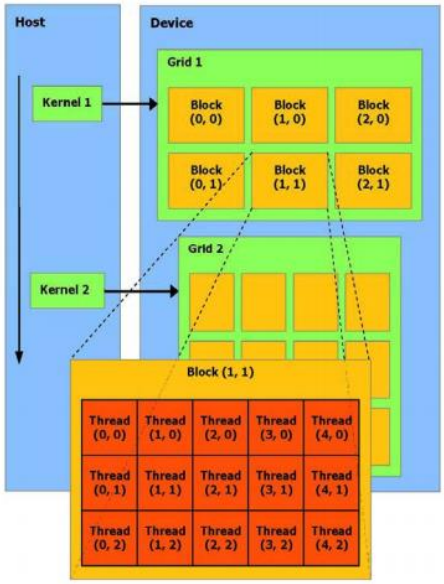


图 4. 2　CUDA 线程组织结构图

在线程组织方面，CUDA 与 OpenCL 类似，多个线程被聚集在一起成为线程块，数个线程块又被归纳在一起成为一个更大的线程网格。整个线程组中的所有线程都执行相同的代码，这样就实现了大规模的并行计算。CUDA 同样支持一维、维以及三维的线程结构。图 4.2 就是 CUDA 线程组织的示意图：

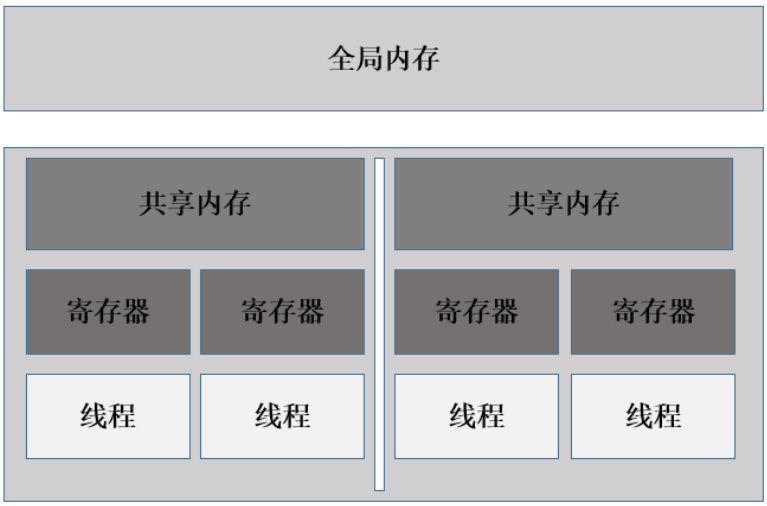


图 4. 3　CUDA 内存管理示意图

在存储支持上，CUDA 编程模型也主要分为全局内存、共享内存与寄存器三个层次。他们的访问速度依次为寄存器>共享内存>全局内存。这些存储器的组织

如图 4.3 共享内存可以被一个线程块中的线程所访问；不加指定的话，数据从 CPU 拷贝进入全局内存，也就是速度最慢的一级。因此，在对于算法进行高性能实现的时候，我们需要尽量多使用全局内存以及寄存器。

## 4.3 基于移动 GPU 的高性能实现

在 TLD 算法中，我们已经知道最耗时的部分是检测模块之中的级联分类器。这一部分会占用大量计算时间的主要原因是对于每一帧图像，级联分类器都需要遍历检测图像中所有可能的样本框，而样本框的数目是巨大的，以 640*480 分辨率的视频为例，每一帧图像中都有大约 300000 个样本框需要检验。在 TLD 算法的原始实现中，每一个样本框的检验都是串行执行的，但是这些计算之间并没有什么相关性，因此，我们可以通过 GPU 多线程来并行执行这些任务。下面具体介绍级联分类器中每一个部分在 GPU 上的高性能实现。

### 4.3.1 方差分类器实现

方差分类器是 TLD 级联分类器的第一个部分，因此，它需要检测的样本数目最多，效果也是最为显著，它大约能够淘汰一半的负样本。方差分类器的计算原理也是最简单的，图像样本灰度值的方差作为一个分散度的度量，它是我们判断样本是否为目标的依据。

由于视频中的图像都是 RGB 三通道的图像，因此，首先需要将其转化为灰度图像，对于某一像素点，从 RGB 值到灰度值的转换计算为：

$$\mathrm{Gray} = \mathrm{R} * 0.299 + \mathrm{G} * 0.587 + \mathrm{B} * 0.114 \qquad (4.1)$$

其中，Gray 代表计算出的灰度值，R 代表 Red 通道的取值，G 代表 Green 通道的取值，B 代表 Blue 通道的取值。

对于某一图像样本 X，设其中有 N 个像素点，$\bar{X}$代表这些像素点灰度的平均值，$X_i$代表其中某一像素点的灰度值，那么，这个图像的灰度值方差计算公式为：

$$\mathrm{V} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - \bar{X})^2} \qquad (4.2)$$

要讲计算样本图像框灰度值方差的任务转移到 GPU 上执行，最简单也最直接的实现方法就是将每个样本框的计算作为一个线程，每个线程读取每个属于本样本的像素点进行计算。这也是我们最基本的实现，伪代码如下：

```
__global__ void PVCompute(){
    int index = get_patechNum();
    float average = get_sumGray()/N;
    float variance = 0.0;
```

```
        for (int j = 0;j < N;j++)
            variance += (x[i] – average) * (x[i] - average)

        variance = sqrt(variance / N);
    }
```

但是，每一帧图像的样本框很多，而且这些样本框是在图像上平移生成的，因此，最基本的 GPU 实现每个线程单独计算，不会共享其它线程的计算工作，这在 TLD 的算法场景中会存在很多的重复计算。为此，我们改进了 GPU 的算法，采用“差值法”[61]来计算某样本框的方差。

首先我们定义对于某个图像样本 I，I 的灰度值积分定义式为：

$$I(x,y) = \sum_{\substack{x'\le x \\ y'\le y}} i(x',y') \qquad (4.3)$$

这里，坐标(x,y)定义了某个像素点在图像中的位置，i(x,y)就是这个像素点的灰度值，即 I(x,y)为图像样本 I 中所有像素点的灰度值的和。一旦计算得到了某个图像样本灰度值的积分，那么，任意一个样本框内的所有像素点灰度值的和就可以转化为三次简单的加减法，转化公式如下：

$$I(x,y)=I(C)+I(A)-I(B)-I(D) \qquad (4.4)$$

公式的示意图如图 4.4：

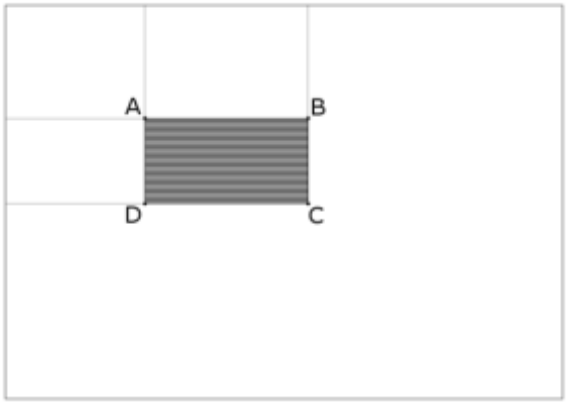


图 4. 4　插值法计算方差示意图

为了计算某个框体内的灰度值之和，我们只需要知道它的四个顶点坐标就可以用简单的加减法得到结果。这就是“差值法”的原理，提前计算好的像素和矩阵在之后的计算中会被多次重用结果，这特别适合用在图像中样本框数目很多的情况。

此外，还有一个有利的因素就是在一次目标跟踪的整个过程中，所有样本框的顶点坐标都是不变的，因此，我们可以提前计算好所有的顶点坐标，并将其作为所有样本框体的地址索引，在算法开始之初就复制到 GPU 端进行存储。这样，就避

免了每次计算时的重新拷贝。

还有一个问题，利用“差值法”计算方差有两种方式：一是将积分图像在 CPU 端就计算完毕，然后再拷贝到 GPU 端，另外一种方式则是将原始图像拷贝到 GPU 端，然后再 GPU 上进行积分操作。在实现中，由于第一种方案会产生更多的数据拷贝操作，第二种方案则会增加在 GPU 端的计算负载。考虑到 GPU 的结构设计使得它与 CPU 之间的数据通信远慢于在 GPU 上执行计算任务，在我们的实现中选取第二种方案。

在“差值法”实现下，方差计算的高性能实现如下：

（1） 对于每一帧新的图像，将图像信息传送到 GPU 端，GPU 进行积分图像的计算。

（2） 在 GPU 端，每个线程根据自己的全局编号计算出线程负责的样本框索引，从顶点数组中取出该样本框的顶点坐标

（3） 根据公式，计算样本框的灰度值方差，若其小于跟踪目标样本框方差值的一半，则标记为正样本；否则，标记为负样本。

（4） 汇总所有图像样本框的计算标记结果，将标记结果数组传送回 CPU 端。CPU 筛选正样本送至下一个分类器。

### 4.3.2 集成分类器实现

集成分类器是级联分类器的第二部分，其可以看做由数个随机蕨分类器组成，原理在第二章中已有介绍。

在算法的实际实现中，为了保证检测模块的准确性，分类器的输入并不是原始图像，而是经过高斯过滤的模糊图像，这样可以减弱图像中具体细节的影响。这是算法在进入随机蕨分类器之前的准备工作。

由于之前已经经过一次方差分类器的分类标记，因此，到达第二级集成分类器的样本数目比原始样本数目要少；而且在这一级分类器中，为了减小算法分类结果的偶然性，会设置多个随机蕨，所以每个样本框具有多组特征需要进行计算。因此在这一级集成分类器中，我们设定每个线程块负责计算一个样本框，线程块中的每一个线程负责计算某一个随机森林的特征。

在集成分类器的高性能实现中另一个值得注意的问题是像素点访问的随机性。如图 4.5 所示[62]，对某一个随机蕨所要计算的特征对（直线相连的两个点），它所需要访问的数据在原始样本框中并不是聚集在一起连续分布的，而是分散在整个样本框体中。GPU 的结构特征决定了它对于类似随机的数据存储是十分缓慢的，因此，对于随机蕨特征对数据的读取会是算法效率的瓶颈所在。

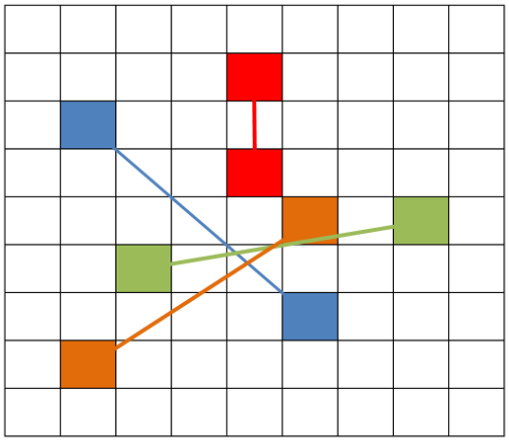

图 4.5　随机蕨分类器的特征点分布

我们注意到随机蕨的比较特征点是在跟踪开始的时候确定的，并且在整个跟踪过程中不会再发生改变。所以为了降低程序在访存上的开销，我们在 CPU 端根据随机蕨的特征点对位置信息，将图像数据重新组合，将某个随机蕨分类器中会访问到的点连续存放在一起，并且将不会使用到的点剔除出去。通过这一步转化，就会增加访存的效率，更好地对算法实现加速。这一部分的示意图如 4.6：

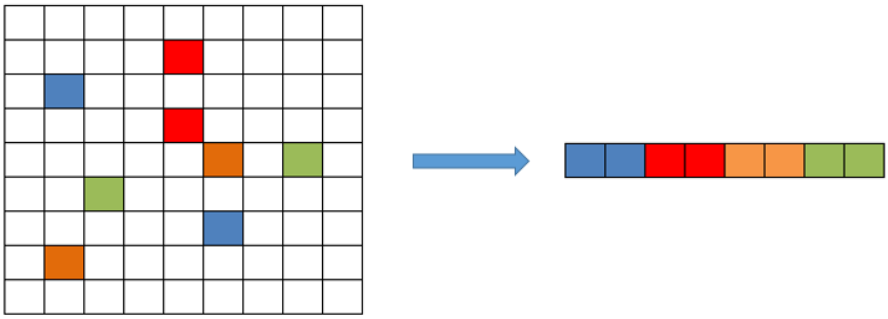

图 4.6　特征点重新组织

如图 4.6 所示，在一个图片框体中有关于一个随机蕨的四组特征点（以不同的颜色标记），它们的分布是乱序的。在我们的实现中将这些特征点提取出来，组合成为一个新的连续数组。这样，在随机蕨的计算过程中，访存就会转变为连续访存。

综合以上的讨论，设有 M 个通过方差分类器到达集成分类器的样本框，N 个随机蕨分类器，每个随机蕨分类器中有 S 组需要进行计算的特征点对。整个集成分类器的计算过程如下：

（1）　在 CPU 端，对于图像进行高斯滤波，得到消除细节的模糊图像。对于通过了方差分类器的图像样本框，根据随机森林中特征点的选择，将需要参加比较的像素点挑选出来，重新生成一个新的图像矩阵，拷贝到 GPU 端。同时，还需要从学习模块中拷贝关于集成分类器的后验特征

矩阵到 GPU 端。

（2） 在 GPU 端，线程组织为 M 个线程块，每个线程块中有 N*S 个线程，每个线程计算出一个随机蕨分类器中一组特征的结果。

（3） 汇总计算的特征结果，通过查找后验概率分布，得到该线程块所计算的框体样本应当标记为正样本还是负样本。

（4） 将标记为正样本的框体索引返回 CPU，等待下一步分类。

在算法的实现中，由于随机蕨特征点的位置信息已经通过在 CPU 端的数据重组包含在了新构造的图像矩阵中，因此我们就不再需要将森林的信息拷贝到 GPU 端，这会加速程序的执行速度。

### 4.3.3 最近邻分类器实现

最近邻分类器作为级联分类器的最后一个阶段，主要任务就是对于通过前两个分类器检测的样本框体进行更细致的评估，量化它们与目标样本集合中目标样本框的相似程度，从中选取最为相似的样本框体，作为检测模块所最终确定的目标位置。

目标样本库在长时间的跟踪过程中由学习模块负责更新，添加最新的正负目标样本以保持算法准确性。对于我们的目标样本库，假设其中有 A 个标记为正的样本，B 个标记为负的样本，那么，对于最后经过前两阶段检测，到达第三阶段检测的 X 个样本框来讲，就共需要进行(A+B)*X 次相似度的计算。

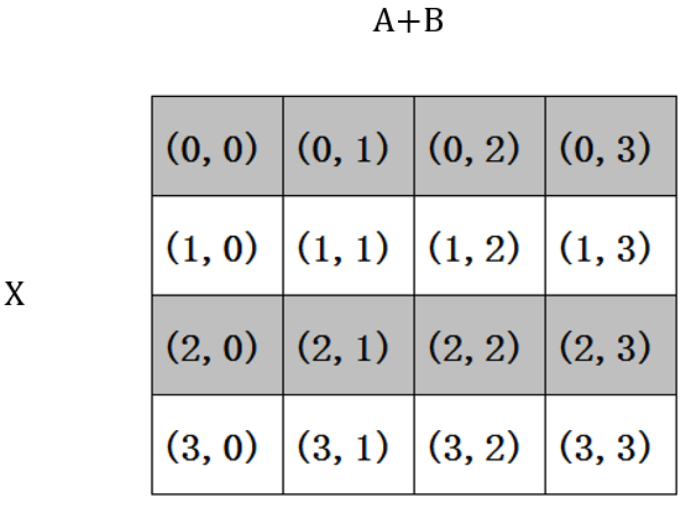


图 4.7 最近邻分类器线程组织

因为到达这一阶段检测的样本框数目通常不是太多，因此，我们选择每一个线程块负责计算一个样本框与目标样本集合中某一个样本的相似度。这里以 X=4，A+B=4 为例来介绍具体的线程组织，图 4.7 为示意图。

图中每一个方块代表一个线程块，X 方向代表有 4 个到达最近邻分类器的样本框体；A+B 方向代表目标样本库中的 4 个样本。（0,0）线程块负责计算第 0 号样本框体与第 0 号目标样本的相似度，（1,1）线程块负责计算第 1 号样本框体与第 1 号目标样本的相似度，以此类推。

在每一个线程块内部，由于我们所使用的 GK20A GPU 最多支持 1024 = 32*32 个线程，所以，线程的工作划分分为两种情况：如果框体内的像素点数目小于 1024，那么，我们就使每个线程块线程的数目与像素点的个数相同，每个线程负责计算一个像素点；如果框体内像素点的数目大于 1024，那么每个线程块内设定 1024 个线程，首先每个线程至少计算一个像素点，多出来的像素点再从头依次分配到每一个线程之上。

对于相似度的计算，每个框体像素点上的计算比较复杂。首先，我们需要计算每个像素点的灰度值与样本对应像素点灰度值的乘积，然后还需要计算框体内这个像素点灰度值的平方以及样本内对应位置像素点的平方。这样，每个像素点我们就会计算出三个值，最后，我们还需要将所有像素点上计算出来的这三个值进行累加求和。

由于这里涉及到的求和数据数目比较多，我们首先对于每一个像素点将这个像素点所计算出的三个数值进行求和，然后放入线程块的共享存储中。对于所有像素点的求和使用“归并求和”算法，这一部分算法伪代码如下：

```
//得到线程编号
int thread_num = get_thread_num();
//将每个像素点出计算出的三个值累加，存入共享存储
shared[thread_num] = mult_0 + mult_1 + mult_2;
//调用前一半的线程进行归并求和
for (int i = thread_total;i > 1;i = i/2)
{
    //线程编号是前一半线程
    if (thread_num < i/2)
        shared[thread_num] += shared[thread_num + i/2];
    //线程编号不是前一半线程
    else
        break;
}
```

在前面所述的归并求和操作完成之后，最后求得的值就会保存在每个线程块共享内存的 shared[0]中，然后依据第二章中的公式来计算框体与跟踪目标的相似度。在 TLD 算法基础设定中，相似度阈值设定为 0.6，因此，最后将相似度分类器

得到计算结果大于 0.6 的框体作为正样本返回 CPU。

这些框体经过学习模块的 P-N 专家检测，得到的唯一最终结果就是 TLD 算法计算出的跟踪目标位置。

## 4.4 实验评测与分析

本节将对之前所实现的 TLD 算法进行测试，我们查看利用移动 GPU 之后的加速效果。测试所采用的的实验平台为 Nvidia Tegra K1 开发板，搭载 GK20A GPU。图 4.8 是所采用的开发板实况以及 GPU 的相关参数。

GK20A GPU 所支持的 CUDA 版本为 6.5，计算能力值为 3.2。所支持的最多线程数目为 2048，每个线程块内的共享内存大小为 49152Bytes，可用寄存器数目为 32768。Tegra K1 开发板配备 HDMI 接口、USB 接口以及网口，这些接口支持为实验提供了很大的便利。

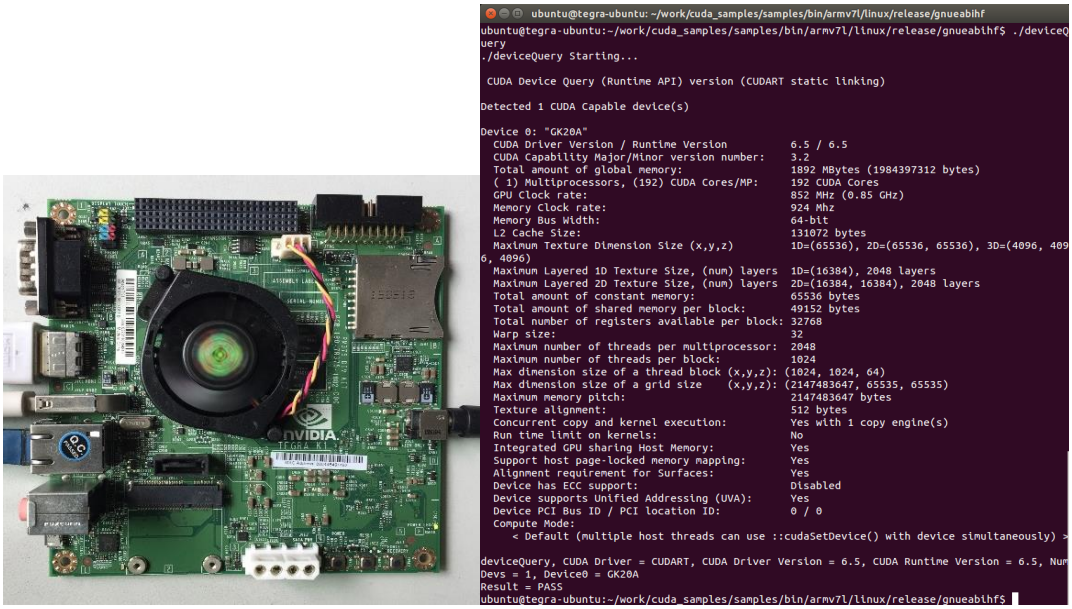


图 4. 8　Tegra K1 开发板以及 GK20A GPU 测试

我们课题中基础 CPU 版本的 TLD 实现采用的是 Orthocenter 的 C++版本。对于我们的高性能实现，我们测试了检测模块每一个分类器的性能以及整体跟踪过程的性能。测试结果如下图：

（1）方差分类器高性能实现加速结果

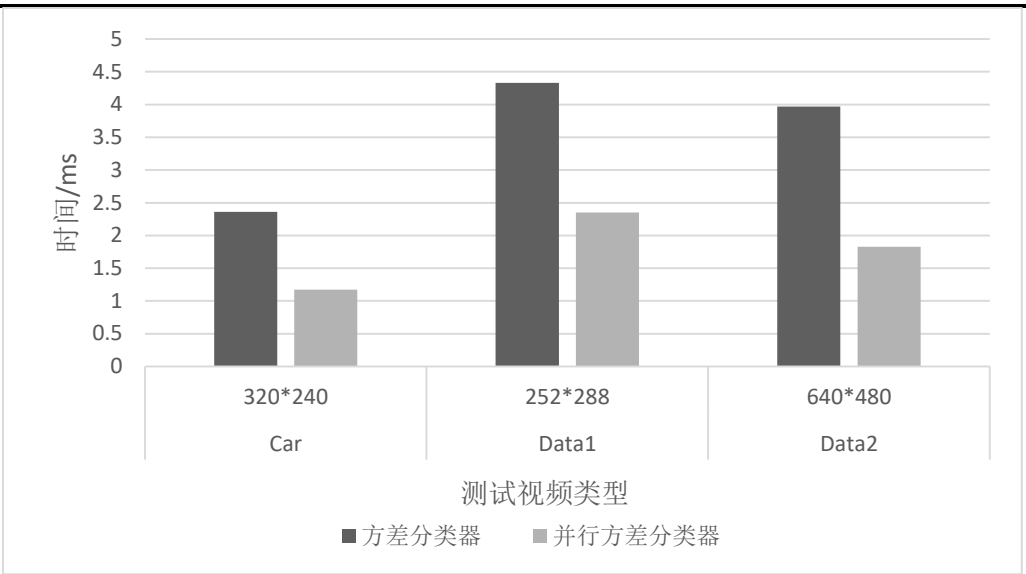


图 4. 9　方差分类器高性能实现测试

从实验结果可以看出，对于不同分辨率的视频，我们在移动 GPU 上实现的高性能方差分类器均可以显著减少这一分类器阶段的时间。

（2）集成分类器高性能实现加速结果

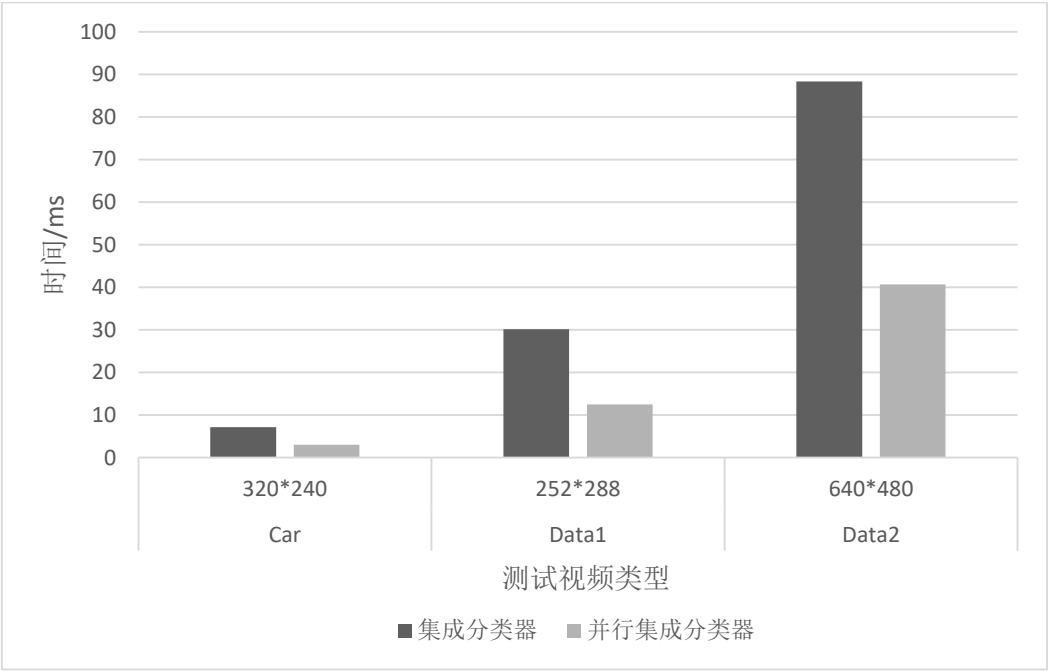


图 4. 10　集成分类器高性能实现测试

集成分类器是级联分类器中耗时比较长的一个阶段，在这个阶段，对于不同分辨率的视频，均取得了 2 倍左右的加速效果。

（3）最近邻分类器高性能实现加速结果

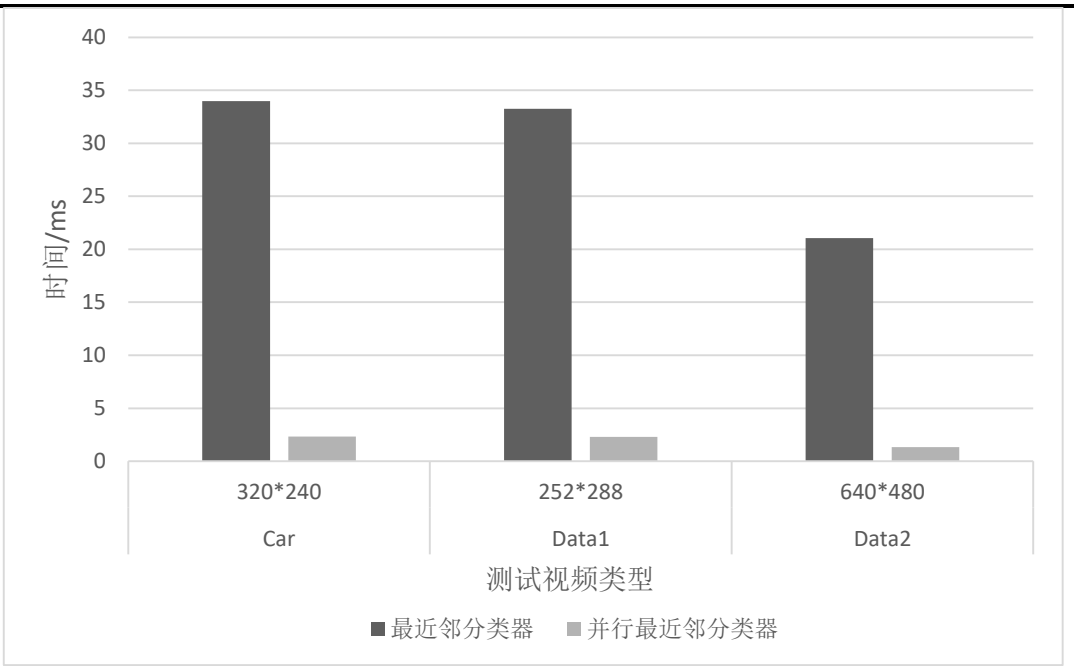


图 4. 11　最近邻分类器高性能实现测试

最近邻分类器是集成分类器中另一个比较耗时的阶段，对于这一部分的加速效果也是最明显的。这可能是由于到达这一部分的样本框体比较少，我们将每个像素的计算分散到了线程之上，达到了比较好的并行效果。

（4） TLD 算法整体高性能实现加速结果

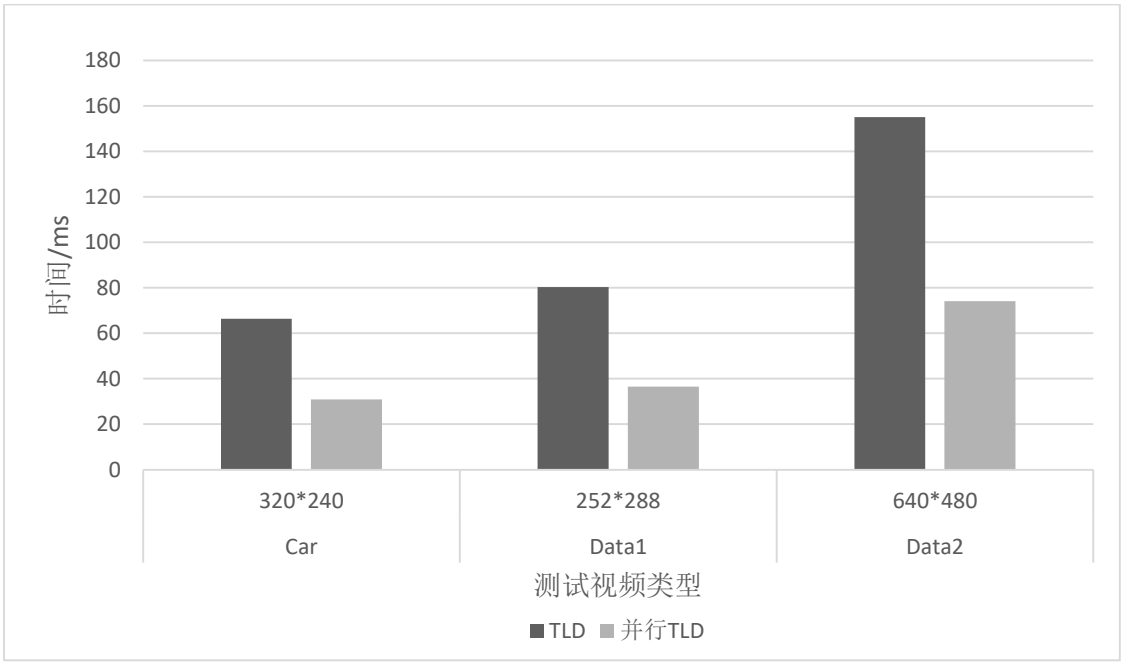


图 4. 12　TLD 高性能实现的整体测试

从测试结果看出，除了在每一部分分类器上取得了明显的加速效果，对于整个跟踪过程，在不同图像分辨率的情况下，都可以明显减少整个过程的时间。

(5)　TLD 算法的高性能实现的加速比情况：

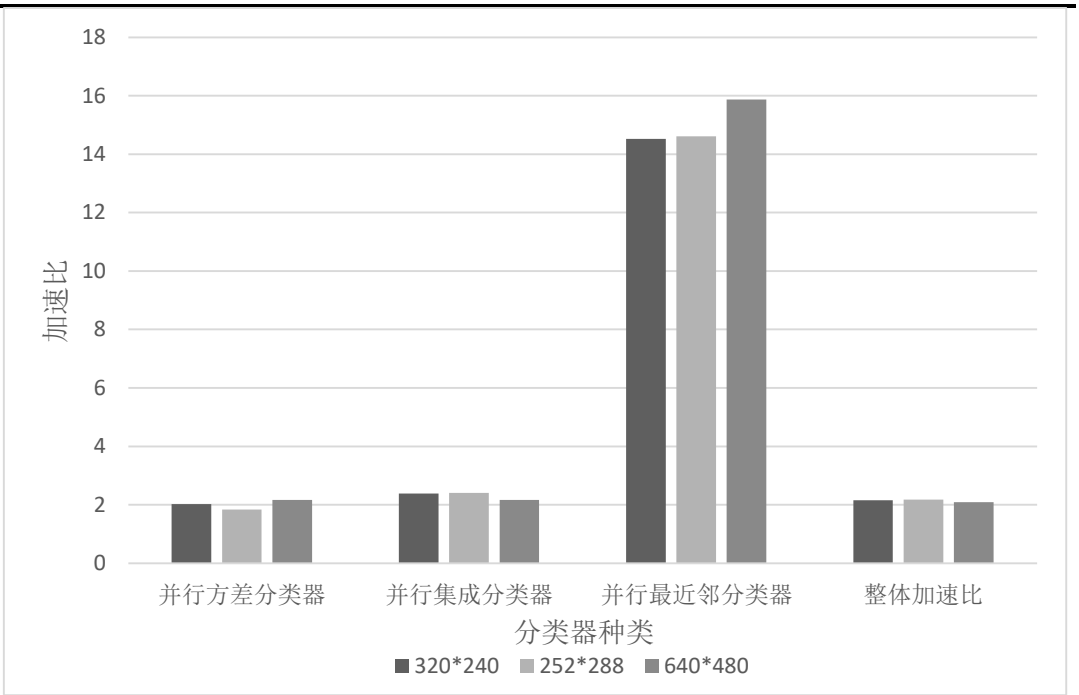


图 4. 13　TLD 高性能实现加速比测试

这里，我们比较了我们的实现在每个分类器以及整体过程上的加速比情况。从测试结果们可以看出，并行方差分类器、并行集成分类器以及整体过程上的加速比都稳定在 2 左右，在并行最近邻分类器上取得了最为明显的加速效果，加速比达到 15 左右。

## 4.5 本章小结

本章主要介绍了将 TLD 算法在移动 GPU 上进行高性能实现的情况。首先，我们在移动平台上针对基础版本的 TLD 实现进行了测试，分析出在算法执行过程中最耗时的部分集中在检测模块。因此，我们决定对于检测模块中的核心部分——级联检测器进行并行加速。针对级联检测器中三个不同监测器的特点以及所需要处理样本的数目多少，我们分别执行了不同的优化算法与任务分配策略使得 GPU 的计算核心之间负载更加均衡，访存更加连续。最后，在 Tegra K1 移动平台上的测试显示我们在不同的加速器上都取得了不错的加速效果，对于整体的跟踪过程而言，在保证算法正确性的同时，我们实现了 2.09 倍的加速。

# 第五章　总结与展望

本章是对论文主要内容的一个总结以及对未来可能研究方向的一些设想。首先回顾总结了课题在计算机视觉核心算法在移动 GPU 上应用的主要工作与创新点，然后展望了一些值得以后继续研究的内容。

## 5.1 工作总结

本论文的主要工作以及创新点可以总结为以下几个方面：

（1） 研究了计算机视觉中的两个核心算法，卷积神经网络与 TLD 目标跟踪算法，并且针对移动平台进行评测与分析。

对于利用卷积神经网络进行图像分类的深度学习算法，我们选取了 MXNet 深度学习框架作为基础，通过针对 ARM 平台的交叉编译在高通 801 芯片上成功执行；对于 TLD 目标跟踪算法，我们采用了 Orthocenter 所实现的 C++版本，同样在 Tegra K1 平台上得到了实现。然后，对于移动设备上的算法执行情况进行了测试与负载的分析，为下一步的改进提出方案。

（2） 为了加速卷积神经网络在移动平台上的执行速度，改进了卷积层的实现，利用 GPU 进行并行加速。

针对卷积神经网络在卷积层计算负载过于集中的问题，利用移动 GPU 对这一层的实现进行了并行加速。针对 GPU 计算能力强的特点，我们采用了矩阵乘法来实现卷积层的具体操作，之后还根据具体 GPU 的结构特性以及性能参数进行了诸如任务划分优化，使用片上存储，循环展开以及向量化的优化策略。测试结果显示我们最终在整个分类过程上取得了 2.1 倍的加速效果。

（3） 针对 TLD 算法在检测模块的瓶颈，对于其中核心级联分类器的算法实现进行了改进，根据计算负载设计了 GPU 上的并行方案。

TLD 算法主要分为三个模块，其中检测模块占据了执行时间的绝大部分。对于检测部分的级联分类器，我们分别针对其要处理的样本数目以及计算负载进行了 CUDA 上的线程组织，并且对于方差分类器改进了方差的算法。最后在 Tegra K1 平台上跟踪算法取得了 2.09 倍的加速比。

（4） 利用卷积神经网络的高性能实现结果，进行了手机上的软件实现，完整实现了利用卷积神经网络的分类功能。

将我们在高通 801 上的卷积神经网络高性能实现进行编译封装为.so 文件，我们在 Android 应用中调用我们实现的算法文件，成功在 Android 应用中实现了卷积神经网络分类的功能，应用支持从相机或者相册获取图片进行分类，更加易用，达到了预期的效果。

## 5.2 未来研究方向

根据本课题的研究进展以及结合最近计算机视觉以及移动设备方面的发展趋势，在以后的研究工作中，我们还可以对以下几个方面开展更加深入细致的研究：

1. 对于深度学习神经网络进行更加细致全面的加速，进一步提升性能

目前，在我们的高性能实现中，对于卷积神经网络我们加速了其中最为耗时的卷积层部分，但是除了这一瓶颈，其他网络层比如池化层，全连接层等也有可以提升的空间。推而广之，对于其他类型的深度学习神经网络，诸如递归神经网络，增强学习神经网络等，均可以研究在移动平台上的加速实现，使得深度学习在移动平台上有更加广阔的应用空间，这一工作将会促进深度学习在移动平台的发展。

2. 对于 TLD 算法，仍然有一些可以改进的方面，使之更加适应移动设备这一平台。

在 TLD 的算法实现中，目标样本库是不断扩大的，随着时间的推移，新的正样本会不断添加进入样本库，这会带来两个负面影响。首先，过多的样本数目会占据很多不必要的内存；其次，时间过于久远的目标样本可能会对检测起到负面的影响。因此，可以尝试添加一个替换算法，保持样本库中样本数目的稳定，并且用更加可信的样本替换掉可能有负面作用的样本。这样在保证算法准确性的同时还能降低算法在移动平台上的内存占用，具有较大的研究意义。

# 致　谢

研究生阶段两年半的学习比预想的过得要快得多，在论文即将撰写完毕的时候，回想自己的研究生学习生活，心里面都是感谢与成长。作为一名国防生来到这里，自己的军政素质需要很大的提升；刚开始自己的研究生阶段，如何发现问题、解决问题也是一个挑战。庆幸自己身边这么多愿意帮助自己的师长、领导以及同学，他们的帮助使我受益良多。

首先，我十分感谢我的导师张春元教授。研究生第一个学期我就上了张老师的体系结构课程，生动的讲述、信手拈来的示例以及对于各种经验的总结，都让我印象深刻。进入实验室，自己有了更多与张老师接触的机会，也学到了更多。尽管张老师担任着学院的行政职务，每天都很忙，仍然一有时间就参加实验室的组会讨论，和我们交流指导，带给我们当前领域最新的研究态势。在分析问题的时候，张老师一丝不苟的态度，高瞻远瞩的大局观让我明白科研之路不是一个短跑的赛场，而需要长久的坚持与不断进步、积累。张老师不仅是我的研究生导师，也是我学习的好榜样。

其次，我十分感谢文梅师姐。师姐今年也是博士生导师了，但是因为平时一直在实验室和大家一起学习科研，实验室里的同学都亲切地称呼为“大师姐”。师姐平时负责实验室的各项工作，在科研中一旦有什么解决不了的问题大家也都积极向师姐请教。师姐还会帮助我们把握研究方向与研究进度，这对于我们这些研究生来说帮助很大。师姐还会组织实验室的集体活动，大家在欢声笑语中感情更加融洽。

我要特别感谢蓝强师兄，师兄在高性能计算领域研究了多年，有非常扎实的理论基础与丰富的编程经验。在我的研究过程中，遇到的问题总能在师兄这里找到解决的思路，师兄也总是不厌其烦的和我讲解，在毕业设计的工作上帮助了我很多。蓝强师兄刻苦学习的态度也非常值得我学习。

感谢苏华友师兄、荀长庆师兄、乔寓然师兄、黄达飞师兄、董辛楠师姐、薛云刚师兄。作为师兄师姐，他们是实验室的主心骨，也会在学习生活的各个方面给我帮助，帮我顺利的完成研究生阶段的学习。

感谢 MASA 课题组的其他成员：陈照云、肖涛、沈俊忠、范方圆、辛思达、曹壮、王自伟、邓皓文、方皓、薄乐、吕倩茹、王彦鹏和王泽龙。大家一起在课题组中学习生活，一起讨论问题，一起开展各种活动，使我的研究生生活更加丰富多彩。

感谢学院五队的队干部宋浩队长和胡浩政委。他们尽心尽力的将五队塑造成了一个优秀的队伍，一个温馨的集体，为我们提供了全方位的关心，让我们能够踏踏实实的进行课题的研究工作。

感谢我两年半研究生生活中的的室友：赵云祥、赵杨、汪志、李宁、包涵和吴金磊。大家一起学习上课，一起聚餐娱乐，让寝室里始终都是温暖的感觉。

最后我还要衷心感谢我的父母和亲朋好友。他们不会对我提任何要求，但是有任何困难他们都会帮助我解决。他们是我最坚强的后盾也是我的力量来源，我会努力不辜负他们的期望。

感谢所有帮助我、关心我的人，我将心怀感激，继续前行！

# 参考文献

[1]WIKIPEDIA.Nokia.N70[EB/OL].https://en.wikipedia.org/wiki/Nokia_N70,2016-10-29.

[2]WIKIPEDIA.Nokia.N8[EB/OL].https://en.wikipedia.org/wiki/Nokia_N8,2016-10-19.

[3]XIAOMI.小米 5[EB/OL]. http://www.mi.com/mi5/,2016-11-01.

[4]KHRNOS GROUP.The open standard for parallel programming of heterogeneous systems. https://www.khronos.org/opencl/,2016-06-12

[5]NVIDIA.What'sCUDA.http://www.nvidia.com/object/cuda_home_new.html,2008-06-19

[6] De La Escalera A, Moreno L E, Salichs M A, et al. Road traffic sign detection and classification[J]. IEEE transactions on industrial electronics, 1997, 44(6): 848-859.

[7] Rui Y, Huang T S, Chang S F. Image retrieval: Current techniques, promising directions, and open issues[J]. Journal of visual communication and image representation, 1999, 10(1): 39-62.

[8] Lipton A J, Fujiyoshi H, Patil R S. Moving target classification and tracking from real-time video[C]//Applications of Computer Vision, 1998. WACV'98. Proceedings., Fourth IEEE Workshop on. IEEE, 1998: 8-14.

[9] Antonie M L, Zaiane O R, Coman A. Application of Data Mining Techniques for Medical Image Classification[J]. MDM/KDD, 2001, 2001: 94-101.

[10] Møller M F. A scaled conjugate gradient algorithm for fast supervised learning[J]. Neural networks, 1993, 6(4): 525-533.

[11] Foresti G L. Object recognition and tracking for remote video surveillance[J]. IEEE Transactions on circuits and systems for video technology, 1999, 9(7): 1045-1062.

[12] Yilmaz A, Javed O, Shah M. Object tracking: A survey[J]. Acm computing surveys (CSUR), 2006, 38(4): 13.

[13] Paul G V, Beach G J, Cohen C J, et al. Realtime object tracking system: U.S. Patent 7,684,592[P]. 2010-3-23.

[14] LeCun Y, Bengio Y, Hinton G. Deep learning[J]. Nature, 2015, 521(7553): 436-444.

[15] Benvenuto N, Piazza F. On the complex backpropagation algorithm[J]. IEEE Transactions on Signal Processing, 1992, 40(4): 967-969.

[16] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[17]Stanford Vision Lab.ImageNet. http://image-net.org/[EB/OL],2016-10-23

[18] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[J]. arXiv preprint arXiv:1512.03385, 2015.

[19] Bouguet J Y. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm[J]. Intel Corporation, 2001, 5(1-10): 4.

[20] Baraldi P, Sarti A, Lamberti C, et al. Evaluation of differential optical flow techniques on synthesized echo images[J]. IEEE Transactions on Biomedical Engineering, 1996, 43(3): 259-272.

[21] Kalal Z, Mikolajczyk K, Matas J. Tracking-learning-detection[J]. IEEE transactions on pattern analysis and machine intelligence, 2012, 34(7): 1409-1422.

[22] Nebehay G, Pflugfelder R. Consensus-based matching and tracking of keypoints for object tracking[C]//IEEE Winter Conference on Applications of Computer Vision. IEEE, 2014: 862-869.

[23] Guo P, Li X, Ding S, et al. Adaptive and accelerated tracking-learning-detection[C]//ISPDI 2013-Fifth International Symposium on Photoelectronic Detection and Imaging. International Society for Optics and Photonics, 2013: 89082H-89082H-10.

[24] Xin Z, Qiumeng Q, Yongqiang Y, et al. Improved TLD visual target tracking algorithm[J]. Journal of Image and Graphics, 2013, 18(9): 1115-1123.

[25] 程立英, 张丹, 赵姝颖, 等. 一种基于 TLD 改进的视觉跟踪算法[J]. 科学技术与工程, 2013 (9): 2382-2386.

[26] Owens J D, Houston M, Luebke D, et al. GPU computing[J]. Proceedings of the IEEE, 2008, 96(5): 879-899.

[27]WIKIPEDIA.Adreno[EB/OL].https://en.wikipedia.org/wiki/Adreno,2016-11-01.

[28] WIKIPEDIA.Tegra[EB/OL]. https://en.wikipedia.org/wiki/Tegra,2016-11-04.

[29]WIKIPEDIA.Mali(GPU)[EB/OL].https://en.wikipedia.org/wiki/Mali_(GPU),2016-11-04.

[30] Wang G, Xiong Y, Yun J, et al. Accelerating computer vision algorithms using OpenCL framework on the mobile GPU-a case study[C]//2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013: 2629-2633.

[31] Cheng K T, Wang Y C. Using mobile GPU for general-purpose computing–a case study of face recognition on smartphones[C]//VLSI Design, Automation and Test (VLSI-DAT), 2011 International Symposium on. IEEE, 2011: 1-4.

[32]Nvidia.Snapdragon801processor[EB/OL].https://www.qualcomm.com/products/snapdragon/processors/801.2015-03-15

[33]WIKIPEDIA.TegraK1[EB/OL].https://en.wikipedia.org/wiki/TegraK1,2016-11-04.

[34] Aleksander I. A probabilistic logic neuron network for associative learning[J]. 1987.

[35] Ciresan D C, Meier U, Gambardella L M, et al. Convolutional neural network committees for handwritten character classification[C]//2011 International Conference on Document Analysis and Recognition. IEEE, 2011: 1135-1139.

[36] LeCun Y, Jackel L D, Bottou L, et al. Comparison of learning algorithms for handwritten digit recognition[C]//International conference on artificial neural networks. 1995, 60: 53-60.

[37] Chen T, Li M, Li Y, et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems[J]. arXiv preprint arXiv:1512.01274, 2015.

[38]DMLC.MXNet[EB/OL].https://github.com/dmlc/mxnet/blob/master/docs/zh/overview.md.2016-10-20

[39] Li M, Andersen D G, Park J W, et al. Scaling distributed machine learning with the parameter server[C]//11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014: 583-598.

[40] Rak M, Venticinque S, Echevarria G, et al. Cloud application monitoring: The mOSAIC approach[C]//Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. IEEE, 2011: 758-763.

[41] Malik O. Googlenet going global[J]. Online at http://gigaom. com/2007/09/21/googlenet-going-global, 2007.

[42] Wang L, Guo S, Huang W, et al. Places205-vggnet models for scene recognition[J]. arXiv preprint arXiv:1508.01667, 2015.

[43] Abadi M, Agarwal A, Barham P, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015[J]. Software available from tensorflow. org, 2015, 1.

[44] Jia Y, Shelhamer E, Donahue J, et al. Caffe: Convolutional architecture for fast feature embedding[C]//Proceedings of the 22nd ACM international conference on Multimedia. ACM, 2014: 675-678.

[45] Collobert R, Kavukcuoglu K, Farabet C. Torch7: A matlab-like environment for machine learning[C]//BigLearn, NIPS Workshop. 2011 (EPFL-CONF-192376).

[46]Nvidia.GTX980[EB/OL].http://www.geforce.cn/hardware/desktopgpus/geforce-gtx-980.2015-06-08.

[47] Keller J M, Gray M R, Givens J A. A fuzzy k-nearest neighbor algorithm[J]. IEEE transactions on systems, man, and cybernetics, 1985 (4): 580-585.

[48] Kalal Z, Mikolajczyk K, Matas J. Forward-backward error: Automatic detection of tracking failures[C]//Pattern recognition (ICPR), 2010 20th international conference on. IEEE, 2010: 2756-2759.

[49] Baker S, Matthews I. Lucas-kanade 20 years on: A unifying framework[J]. International journal of computer vision, 2004, 56(3): 221-255.

[50] 维基百科.三星 GalaxyS5[EB/OL].https://zh.wikipedia.org/wiki/三星

GalaxyS5.2016-09-03

[51] XIAOMI.小米 NOTE[EB/OL]. http://www.mi.com/minote/,2015-04-01

[52]Sony.XperiaZ3[EB/OL].http://www.sonymobile.com/globalen/products/phones/xperia-z3/.2015-06-17

[53] Vidas T, Votipka D, Christin N. All Your Droid Are Belong to Us: A Survey of Current Android Attacks[C]//WOOT. 2011: 81-90.

[54]DMLC.FastpoorNet[EB/OL].https://github.com/dmlc/mxnet/blob/master/docs/how_to/smart_device.md.2016-07-13

[55] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015: 1-9.

[56] Burrus C S S, Parks T W. DFT/FFT and convolution algorithms: theory and implementation[M]. John Wiley & Sons, Inc., 1991.

[57] Coppersmith D, Winograd S. Matrix multiplication via arithmetic progressions[C]//Proceedings of the nineteenth annual ACM symposium on Theory of computing. ACM, 1987: 1-6.

[58] Byleas.OpenCL-Z[EB/OL]. https://sourceforge.net/projects/opencl-z/.2013-04-22

[59] Gordon R, Essential J N I. Java Native Interface[J]. Prentince Hall PTR, 1998.

[60] Github.OpenTLD[EB/OL]. https://github.com/Orthocenter/TLD.2014-11-04

[61] 张平. 基于 CUDA 的 TLD 视觉跟踪算法研究[D]. 北京交通大学, 2014.

[62] GÜRCAN İ. Hybrid CPU-GPU Implementation of Tracking-Learning-Detection Algorithm[D]. MIDDLE EAST TECHNICAL UNIVERSITY, 2014.

# 作者在学期间取得的学术成果

## 发表的学术论文

[1] Yang Shi, Qiang Lan, Hao Fang, Mei Wen. Accelerating CNN's Forward Process on Mobile GPU Using OpenCL[C]. The 8th International Conference on Digital Image Processing (ICDIP2016), 2016, Chengdu, China. (EI 检索)