基于 GPU 的卷积检测模型加速

刘 琦 黄 咨 陈璐艳 胡福乔
(上海交通大学自动化系系统控制与信息处理教育部重点实验室 上海 200240)

摘 要 近年来,形变部件模型和卷积神经网络等卷积检测模型在计算机视觉领域取得了极大的成功。这类模型能够进行大规模的机器学习训练,实现较高的鲁棒性和识别性能。然而训练和评估过程中卷积运算巨大的计算开销,也限制了其在诸多实际场景中进一步的应用。利用数学理论和并行技术对卷积检测模型进行算法和硬件的双重加速。在算法层面,通过将空间域中的卷积运算转换为频率域中的点乘运算来降低计算复杂度;而在硬件层面,利用 GPU 并行技术可以进一步减少计算时间。在 PASCAL VOC 数据集上的实验结果表明,相对于多核 CPU,该算法能够实现在单个商用 GPU 上加速卷积过程 2.13 ~ 4.31 倍。

关键词 卷积检测模型 计算机视觉 GPU

中图分类号 TP3191 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2016.05.057

CONVOLUTION-BASED DETECTION MODELS ACCELERATION BASED ON GPU

Liu Qi Huang Zi Chen Luyan Hu Fuqiao
(*Key Laboratory of System Control and Information Processing, Ministry of Education of China, Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China*)

Abstract In recent years, convolution-based detection models (CDM), such as the deformable part-based models (DPM) and the convolutional neural networks (CNN), have achieved tremendous success in computer vision field. These models allow for large-scale machine learning training to achieve higher robustness and recognition performance. However, the huge computational cost of convolution operation in training and evaluation processes also restricts their further application in many practical scenes. In this paper, we accelerate both the algorithm and hardware of convolution-based detection models with mathematical theory and parallelisation technique. In the aspect of algorithm, we reduce the computation complexity by converting the convolution operation in space domain to the point multiplication operation in frequency domain. While in the aspect of hardware, the use of graphical process unit (GPU) parallelisation technique can reduce the computational time further. Results of experiment on public dataset Pascal VOC demonstrate that compared with multi-core CPU, the proposed algorithm can realise speeding up the convolution process by 2.13 to 4.31 times on single commodity GPU.

Keywords Convolution-based detection model Computer vision GPU

0 引 言

目标检测是计算机视觉领域中一项计算密集的工作。现实生活中的绝大多数应用,如智能监控中的行人检测、汽车安全中的车辆检测和图像检索中的特定类别目标检测等,都要求检测速度足够快以便达到实时性,同时也希望尽可能降低误报率。近年来,研究人员提出了一批优秀的卷积检测模型 CDM(Convolution-based detection model),并在诸如 PASCAL VOC[1]、ImageNet[2]等极具挑战性的标准评测数据集上取得了出色计算性能。这些模型以卷积运算为基础,用特定的模板与输入图像或特征图进行卷积得到响应图,进而通过最大响应值确定目标位置或将其作为特征图再次进行卷积运算,如图 1 所示。然而,尽管卷积检测模型简单有效,但卷积运算高强度的计算消耗阻碍了其在现实生活中的应用。

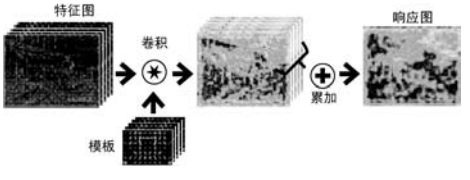

图 1 卷积检测模型中的卷积运算流程

针对卷积运算耗时这一问题,一种有效的方法是利用卷积定理将空间域中的卷积运算通过傅里叶变换到频率域中更为高效的点乘运算,进而显著地降低计算复杂度[3]。尽管如此,在面对实际尺寸目标检测任务时,即使采用多线程并行技术,这种方法在 CPU 平台上的运行效率依然低下。而随着高性能计算

收稿日期:2014 - 11 - 24。国家自然科学基金项目(61175009);上海市产学研合作项目(沪 CXY - 2013 - 82)。**刘琦**,硕士生,主研领域:计算机视觉,并行计算。**黄咨**,硕士生。**陈璐艳**,硕士生。**胡福乔**,副教授。

技术的快速发展,GPU 的通用科学计算能力得到了越来越多的重视。快速傅里叶变换和点乘运算中蕴含了丰富数据独立性,使得 GPU 上的计算并行化成为可能,可以进一步提高计算效率,节省处理时间。

本文从算法加速和硬件加速两个角度入手,首先将卷积定理推广至卷积检测模型,通过对比应用卷积定理前后的模型计算复杂度,从数学上论证了算法有效性。接下来,本文分析了算法的并行性,在此基础上提出了基于 OpenCL 的 GPU 加速策略并进行实现。实验结果表明,本文算法能够有效利用 GPU 的计算能力,相对于多线程 CPU 实现,取得了 2.13 ~4.31 倍的加速比。

1　相关工作

一直以来,卷积操作在统计、信号处理、计算机视觉等领域都是最基础的数学算子之一,在空间域上通常以滑动窗口的形式实现。在许多早期的研究工作中,卷积算子常被用来调整信号的频率特性,如高斯平滑、平均平滑和 Sobel 滤波等。此后,模板匹配利用卷积算子在空间域上搜索特定的模式,这被认为是一种简单的检测模型。然而,这类模型难以在包含诸多类别的静态图片中准确地定位特定目标。为解决目标检测中存在的姿态和形态各异的问题,绝大多数常用的检测模型都采用了大量的模板和卷积运算,如形变部件模型 DPM(Deformable partbased models)和卷积神经网络 CNN(Convolutional neural networks)。尽管这类模型的检测效果显著提升,但其计算量也随卷积算子的增加而线性增长。

1.1　形变部件模型

Felzenszwalb 等人[4]提出的形变部件模型采用 HOG(Histograms of oriented gradient)描述子[5]作为底层特征,使用树状图模型[6]来表示目标结构,并据此将根模板和若干可形变部件模板组合到一起,形变部件用于反映目标局部形态特性。该模型通过滑动窗口的方式,在 HOG 特征金字塔上逐层卷积估计目标可能出现的位置。这种计算方式需要处理全部的搜索空间,从而运行效率非常低下。

为了解决此问题,Felzenszwalb 等人[7]进一步提出了采用部分假设剪枝策略的级联实现。但由于级联中存在过多控制流和负载不均衡等问题,难以适用于 GPU 架构。Song 等人[8]和 Hirabayashi 等人[9]进行了形变部件模型的 CUDA 实现,但同时也将其应用局限于 Nvidia GPU 平台。De Smedt 等人[10,11]对构建 HOG 特征金字塔进行了 OpenCL 实现,将卷积遗留在 CPU 端,以流水线在 CPUs/GPUs 异构体系下实现形变部件模型。相较于特征提取,形变部件模型中的卷积运算计算量更大,也更适于采用 GPU 并行计算架构。

1.2　卷积神经网络

卷积神经网络[12,13]源于对动物视觉皮层的研究,是多层感知器 MLP(Multilayer perceptions)的变形。不同于形变部件模型采用固定的 HOG 特征,卷积神经网络将模板直接运用于原始图像上,通过大量卷积操作模拟视觉识别系统。卷积层和采样层是卷积神经网络的核心。卷积层主要用于产生不同层次的特征编码,在前馈和反向传播过程中都消耗了绝大部分的计算资源。采样层通过降采样操作,针对局部畸变和简单几何变换等问题实现了一定的不变形。

实际上,形变部件模型也是一种卷积神经网络。Girshick 等人[14]简化了由 Krizhevsky[15]等人提出的参数规模非常庞大的卷积神经网络,用于构建特征金字塔,以替代 HOG 特征。同时他们将距离变换推广为采样层,将形变部件模型的推断过程重构为一个卷积神经网络。

受限于网络深度和计算速度,卷积神经网络很难应用于实际尺寸的目标检测任务中。当前主流的卷积神经网络库都是使用 CUDA 实现 GPU 加速,如 Caffe[16]等。Cecotti 等人[17]将傅里叶变换引入到卷积神经网络,但其主要目的是为了便于脑电信号分析而非加速。

2　频率域算法加速

为加速线性目标检测系统的评估速度,Dubout 等人[3]引入傅里叶变换,实现了 6 ~8 倍的加速比。我们可以将此方法进行推广,用于加速卷积检测模型。

算法加速的核心思想是傅里叶变换的卷积定理,即原始信号在空间域的卷积可由其对应的傅里叶变换乘积的反变换求得,可表示为:

$$f * g = \mathscr{F}^{1}(\mathscr{A}(f) \times \mathscr{A}(g)) \tag{1}$$

算法加速的基本流程主要包括以下 3 个步骤:

(1) 计算所有空间域特征图和模板的傅里叶变换;

(2) 将频率域特征图和模板进行点乘和累加运算,得到频率域响应图;

(3) 计算傅里叶逆变换得到空间域响应图。

为深入分析算法加速的有效性,假定卷积检测模型的输入为具有 L 维特征的 J 张尺寸为 $M \times N$ 的特征图和 K 个尺寸为 $P \times Q$ 的模板。同时,将特征图和模板的第 l 维分别标记为 $x^l \in R^{M\times N}$ 和 $y^l \in R^{P\times Q}$,则由特征图和模板卷积而得的响应图 $z \in R^{(M-P+1)\times(N-Q+1)}$ 可表示为:

$$z = \sum_{l=0}^{L-1} x^l * \bar{y}^l \tag{2}$$

其中,$\bar{y}$ 表示翻转的模板。

卷积运算的计算复杂度为 $O(MNPQ)$。由于特征图和模板的每个系数之间都需要进行一次乘法和一次加法运算,因此每一维所需的浮点数运算量为:

$$C_{conv} = 2MNPQ \tag{3}$$

假定每张特征图都需要与所有 K 个模板进行卷积,则每张特征图所需的浮点数运算量共为:

$$C_{spatial} = KL(C_{conv} + MN) \tag{4}$$

其中,MN 表示最终为得到 1 维响应图而进行的累加运算数。为简便起见,假定响应图的大小仍然保持为 $M \times N$。

为了在频率域中顺利进行点乘运算,首先需要在傅里叶变换之前将模板填补成与对应特征图同一大小。对于实数傅里叶变换,其变换结果满足 Hermitian 冗余,即其中一半的变换结果与另一半共轭。因此傅里叶变换后的特征图与模板大小均为 $M \times (N + 2)$,共包含 $M \times (N/2 + 1)$ 个复数系数。同样,简记其大小为 $M \times N$,共包含 $M \times N/2$ 个复数系数。

两维傅里叶变换的计算复杂度为 $O(M^2N^2)$,快速变换算法可将其降低至 $O(MN\log_2 MN)$。根据文献[18,19],可得到每一维实数特征图傅里叶变换所需的浮点数运算量为:

$$C_{FFT} = 2.5MN\log_2 MN \tag{5}$$

对于频率域中的点乘和累加运算,特征图和模板对应位置上的复数系数之间需要各进行一次复数乘法(6 次浮点数运算)

和复数加法(2 次浮点数运算),故总运算量为:

$$C_{mul+acc} = 4MN \tag{6}$$

在实际系统中,通常只需要载入一次模板即可,因此模板的傅里叶变换可以离线进行。同时,根据傅里叶变换的线性特性,多维响应的累加运算可以在频域进行,故只需要计算一维的逆变换。因此,引入傅里叶变换后,对于每张特征图,共需要进行 L 次正变换、K 次逆变换和 KL 次点乘和累加运算,所需的浮点数运算量共为:

$$C_{frequency} = LC_{FFT} + KLC_{mul+acc} + KC_{FFT} \tag{7}$$

对比空间域和频率域的浮点数运算量,则理论上的加速比为:

$$\frac{C_{spatial}}{C_{frequency}} = \frac{KL(2PQ+1)}{2.5(K+L)\log_2 MN + 4KL} \tag{8}$$

根据式(8),易知若模板数量越多,特征维度越高,即 $KL \gg K+L$,则能够获得越高的加速比。同时,对于更大的模板也能够获得更高的加速比。这两点都是当前主流卷积检测模型的共同特性。

3 GPU 硬件加速

为了进一步加速卷积检测模型,本文在 GPU 上基于 OpenCL 对前述算法进行了并行实现,使硬件加速不再受限于 GPU 平台类型。OpenCL 是一个面向异构系统,用于通用并行计算的免费开放标准,也是一个统一的编程环境,可广泛适用于各类不同架构的 CPU、GPU、DSP 及 FPGA 等并行处理器。

图 2 所示的算法加速的串行实现伪代码如算法 1 所示。

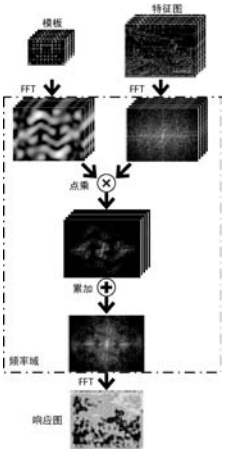

图 2 利用卷积定理进行算法加速的基本流程

算法 1 频率域算法加速的串行实现

```
输入:特征图 F、模板 T
输出:响应图 R
FOR Tj ∈ T DO
    fTj = forwardFFT ( Tj )                    //模板正变换
  END
FOR Fi ∈ F DO
    fFi = forwardFFT (Fi)                      //特征图正变换
    FOR fTj ∈ fT DO
      fSij = fFi · fTj                         //频率域点乘
      fRij + = fSij                            //频率域累加
      Rij = backwardFFT(fRij)                  //响应图逆变换
    END
END
```

在算法 1 中,存在十分明显的并行性,主要有两个方面:

(1) 数据并行性:无论是傅里叶变换、点乘,还是累加,这些基于像素的运算之间都不存在数据依赖性。因此,可以将输入数据映射到 OpenCL 的 NDRange 索引空间,利用一个线程处理一个像素的运算过程。

(2) 任务并行性:算法 1 中同类型的任务之间不存在数据依赖性,如所有模板的傅里叶变换。而同一 FOR 循环内的不同类型任务则需要顺序执行,如点乘、累加和响应图傅里叶逆变换。

通过调整 FOR 循环的顺序,可以使算法 1 更适宜 GPU 并行化,如算法 2 所示。将同一 FOR 循环中的任务依次发送至 GPU,插入到不同的 OpenCL 命令队列,可以使之并行执行。若存在数据依赖性,则将其插入同一命令队列,并通过 OpenCL 事件进行通信,保证执行次序。不同 FOR 循环间通过同步命令保证内存的一致性。

算法 2 频率域算法加速的并行实现

```
输入:特征图 F、模板 T
输出:响应图 R
FOR Tj ∈ T DO
    event = writetoGPU ( Tj )
    fTj = fFFTkernel ( Tj,event )                  //模板正变换
END
Synchronization( )
FOR Fi ∈ F DO
    event = writetoGPU (Fi)
    fFi = forwardFFT (Fi,event )                   //特征图正变换
END
Synchronization( )
FOR fFi ∈ fF & fTj ∈ fT DO
    fRij = mul&acckernel ( fFi,fTj )               //点乘、累加
END
Synchronization( )
FOR fRij ∈ fR DO
    Rij,event = backwardFFT( fRij )                //响应图逆变换
    readtoCPU (Rij,event )
END
Synchronization( )
```

3.1 快速傅里叶变换

N 点一维离散傅里叶变换(DFT)可表示为:

$$y_k = \sum_{j=0}^{N-1} x_j \omega_n^{jk} \tag{9}$$

其中,$0 \leqslant k < n$,$\omega_n = exp(-2\pi i/n)$。在诸多快速傅里叶(FFT)算法中,Cooley-Tukey 算法[20]最为常用。其以分治法为策略,递归地将长度为 $n = n_1 n_2$ 的 DFT 分解成为两个长度分别为 n_1 和 n_2 的较小规模 DFT 以及旋转因子的复数乘法。式(9)可重写成:

$$y_{k_1+k_2 n_1} = \sum_{j_2=0}^{n_2-1} \left[\left(\sum_{j_1=0}^{n_1-1} x_{j_1 n_2 + j_2} \omega_{n_1}^{j_1 k_1} \right) \omega_n^{j_2 k_1} \right] \omega_{n_2}^{j_2 k_2} \tag{10}$$

其中,$j = j_1 n_2 + j_2$,$k = k_1 + k_2 n_1$,$\omega_n^{j_2 k_1}$ 称作旋转因子,n_1 或 n_2 称为基。计算示例如图 3 所示。

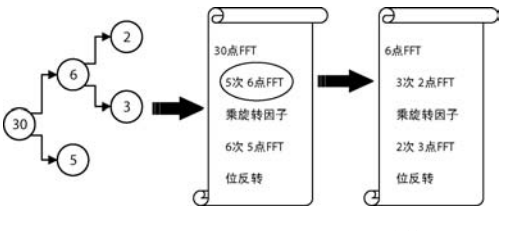

图 3 以 2、3、5 为基的 30 点 DFT 快速计算流程

由于输出并非顺序,Cooley-Tukey 算法中存在显式位反转操作,这并不适合在 GPU 上实现。Stockhan 算法[21]通过在每次分解中穿插进行多维转置,可以避免这一问题而直接得到顺序输出,适于 GPU 实现[22]。

在利用 GPU 并行计算 2 维 FFT 时,通常的方法是使用 1 维 FFT 分别变换其所有行和列。在分解完成后,FFT 核函数将数据从显存加载到寄存器中,利用 GPU 的局部内存存储中间数

据,每个线程递归地计算一个基的 FFT。

3.2　拼接策略和优化

前文曾提及需要在傅里叶变换之前将模板填补成与对应特征图同一大小。而简单的填补方法或将导致过高的内存开销,或需要消耗较多的计算资源,特别是对于部分需要构建特征金字塔的卷积检测模型。

对于上述问题,可以通过利用一种快速启发式左下装箱算法[23]拼接特征图来解决。其基本思路是将所有特征图组合成为若干特征拼图,后将模板填补成同一大小[3],如图 4 所示。特征拼图的大小通常等同于最大的特征图。

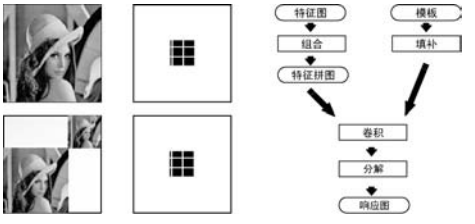

图 4　特征拼图示意图及基本流程

为避免 GPU 产生不必要的调度开销,不仅需要控制 GPU 上任务的数量,也需要合理设置每个 OpenCL 核函数运行时所启动的线程数。因此,本文将特征图和模板的点乘运算和累加运算合并到一个核函数中,使用 3 维 NDRange 索引空间,每一个工作组计算一个像素上 L 维特征的相关运算。这样不仅可以避免过多的调度开销,还能够更为规律地进行内存合并访问。同时有计划地发送任务,适当的同步也可降低调度开销。

4　实验结果与分析

本文选择形变部件模型作为基础,进行 OpenCL 实现,GPU 上的 FFT 计算使用 clMath 数学库中的 clFFT 函数库。基准 CPU 多线程实现为 Idiap 研究所的 FFLD[3],其使用 CPU SIMD 指令集和 OpenMP 实现多线程并行计算,FFT 计算使用 FFTW 函数库,矩阵运算使用 Eigen 函数库。

本文在两个不同硬件平台上进行了实验,一个配置为 Intel Core i7 3770(3.4 GHz,4 核)和 Intel HD Graphics 4000(0.35 GHz,128 个流处理器);另一个配置为 Intel Core i5 3570K(超频 4.2 GHz,4 核)和 Nvidia GeForce GTX 670(超频 1.17GHz,1344 个流处理器)。

测试数据来自 PASCAL VOC 2007 数据集,检测模板来自基准 CPU 实现。以自行车为例,其检测模板共 54 个,模型构建的 32 维 HOG 特征金字塔共 17 层,组成 6 个特征拼图。根据算法 2 可将整个卷积过程(从输入特征图和模板到输出响应图)分为四部分:模板正变换、特征图正变换、点乘和累加以及响应图逆变换。表 1 给出了各部分总的计算时间,包括 GPU 计算时间、数据传输时间,以及各类在 CPU 上进行的预处理或后处理的计算时间,以便更客观地对比本文算法的加速效果。

表 1　CPU 多线程实现与 GPU 实现的性能对比

计算设备	计算时间(ms)			
	模板	特征图	点乘累加	响应图
i7 3770	874	366	1696	35
HD 4000	1519	440	398	146
i5 3570K	750	296	1837	29
GTX 670	828	312	73	117

在 Intel GPU 平台上,点乘和累加运算上取得了 4.26 倍的加速比,而其他三部分则更慢一些,模板正变换、特征图正变换和响应图逆变换的计算时间分别为基准的 0.58、0.83 和 0.23 倍。同样地,在 Nvidia GPU 平台上,本文实现了 25.16 倍的点乘和累加运算加速,而其他三部分则分别为基准的 0.91、0.94 和 0.25 倍。

实验结果表明,点乘和累加运算能够充分利用 GPU 中的大量流处理器,从而实现较高的加速比。而有关 FFT 运算的其他三部分较基准更慢的一个主要原因是其中数据传输以及 CPU 上的各类预处理或后处理操作所需的计算时间较高。例如,响应图逆变换速度远慢于基准的原因便是分解响应拼图操作中数据传输耗时较多。另一个主要原因在于所选用的 FFT 函数库。由于目前便于使用的以 OpenCL 实现的 FFT 函数库较少,本文选用的 clFFT 函数库仍处于开发初期,其功能和效率还有待提升。事实上,基准 CPU 实现所使用的 FFTW 函数库能够利用批处理操作,同时精确计算 32 维 HOG 特征图和模板的快速傅里叶变换。而 clFFT 函数库在 HD 4000 上却只能同时精确计算 5 维,在 GTX670 上也不过增加至 11 维。对此,在具体实现中不得不先将特征图和模板进行拆分,计算 FFT 后再将其融合。

另外,GTX 670 上的各部分计算时间均少于 HD 4000,加速比分别为 1.83、1.41、5.45 和 1.25 倍。这主要得益于 GTX 670 拥有更高的工作频率和更多的流处理器,能够产生更多的并发线程。

由于模板的傅里叶变换可以离线进行,所以若从实际应用的角度出发,可将特征图正变换、点乘和累加,以及响应图逆变换这三部分计算时间的总和作为标准,来衡量卷积在线流程的性能。在 HD 4000 上,卷积在线流程的计算时间由 2097 ms 降至 984 ms,提速 2.13 倍,其中点乘和累加运算的计算时间占比从 80.88% 降至 40.45%,如图 5 所示,其中饼图的面积表示卷积在线流程所需要的计算时间。在 GTX 670 上,卷积在线流程的计算时间由 2162 ms 降至 502 ms,提速 4.31 倍,而点乘和累加运算的占比更是明显下降,由 84.97% 降至 14.54%,如图 6 所示。

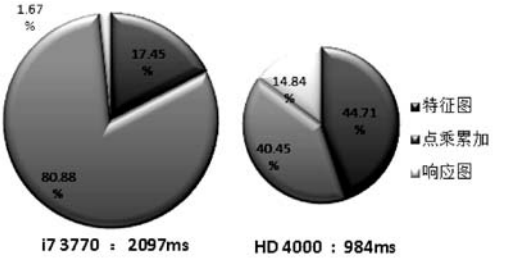

图 5　HD 4000 上卷积在线流程的性能对比

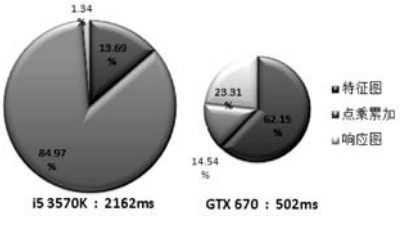

图 6　GTX 670 上卷积在线流程的性能对比

综上所述,尽管在 OpenCL 实现中所选用的第三方函数库的效率并不理想,从而导致部分运算时间较 CPU 实现有所增加。但从整体上来看,对于卷积检测模型本文的 GPU 实现在不

同硬件平台上仍然能够取得不错的加速效果。

5 结 语

本文提出了将傅里叶变换推广至卷积检测模型,通过将空间域中的卷积运算转换到频率域中的点乘运算以大幅降低计算复杂度,并从数学上论证了其有效性。同时,本文在 GPU 上使用 OpenCL 实现了进一步的硬件加速。实验结果表明,尽管本文所选用的第三方函数库效率并不理想,但本文的 GPU 实现仍然能够在不同架构的硬件平台上取得不错的加速效果。相对于 4 核 CPU 平台下的多线程实现,本文的 GPU 实现能够达到 2.13 ~ 4.31 倍的加速比。

未来的工作将主要集中于研究目前耗时较多的预处理或后处理操作的加速、提高 FFT 运算效率等问题,同时将本文所提出的加速方法应用于卷积神经网络。

参 考 文 献

[1] Everingham M, Van Gool L, Williams C K I, et al. The pascal visual object classes (voc) challenge[J]. International journal of computer vision, 2010, 88(2): 303 - 338.

[2] Deng J, Dong W, Socher R, et al. Imagenet: A large-scale hierarchical image database[C]//Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009: 248 - 255.

[3] Dubout C, Fleuret F. Exact acceleration of linear object detectors [C]//Computer Vision - ECCV 2012. Springer Berlin Heidelberg, 2012: 301 - 311.

[4] Felzenszwalb P F, Girshick R B, McAllester D, et al. Object detection with discriminatively trained part-based models[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2010, 32(9): 1627 - 1645.

[5] Dalal N, Triggs B. Histograms of oriented gradients for human detection [C]//Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. IEEE, 2005, 1: 886 - 893.

[6] Felzenszwalb P F, Huttenlocher D P. Pictorial structures for object recognition[J]. International Journal of Computer Vision, 2005, 61(1): 55 - 79.

[7] Felzenszwalb P F, Girshick R B, McAllester D. Cascade object detection with deformable part models[C]//Computer vision and pattern recognition (CVPR), 2010 IEEE conference on. IEEE, 2010: 2241 - 2248.

[8] Song H O, Zickler S, Althoff T, et al. Sparselet models for efficient multiclass object detection[C]//Computer Vision - ECCV 2012. Springer Berlin Heidelberg, 2012: 802 - 815.

[9] Hirabayashi M, Kato S, Edahiro M, et al. GPU implementations of object detection using HOG features and deformable models[C]//Cyber-Physical Systems, Networks, and Applications (CPSNA), 2013 IEEE 1st International Conference on. IEEE, 2013: 106 - 111.

[10] De Smedt F, Struyf L, Beckers S, et al. Is the game worth the candle? Evaluation of OpenCL for object detection algorithm optimization[C]// International Conference on PECCS, 2012: 284 - 291.

[11] De Smedt F, Van Beeck K, Tuytelaars T, et al. Pedestrian detection at warp speed: Exceeding 500 detections per second[C]//Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on. IEEE, 2013: 622 - 628.

[12] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278 - 2324.

[13] Ouyang W, Wang X. Joint deep learning for pedestrian detection[C]// Computer Vision (ICCV), 2013 IEEE International Conference on. IEEE, 2013: 2056 - 2063.

[14] Girshick R, Iandola F, Darrell T, et al. Deformable Part Models are Convolutional Neural Networks [R/OL]. Berkeley, CA, USA: UC Berkeley, 2014 [2014-10-01]. http://arxiv.org/abs/1409.5403.

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

[16] Jia Y, Shelhamer E, Donahue J, et al. Caffe: An open source convolutional architecture for fast feature embedding[CP/OL]. 2014. http://caffe.berkeleyvision.org.

[17] Cecotti H, Graeser A. Convolutional neural network with embedded fourier transform for EEG classification[C]//Pattern Recognition, 2008. ICPR 2008. 19th International Conference on. IEEE, 2008: 1 - 4.

[18] Frigo M, Johnson S G. The design and implementation of FFTW3[J]. Proceedings of the IEEE, 2005, 93(2): 216 - 231.

[19] Li Y, Zhang Y Q, Liu Y Q, et al. MPFFT: an auto-tuning FFT library for OpenCL GPUs[J]. Journal of Computer Science and Technology, 2013, 28(1): 90 - 105.

[20] Cooley J W, Tukey J W. An algorithm for the machine calculation of complex Fourier series[J]. Mathematics of computation, 1965, 19(90): 297 - 301.

[21] Stockham Jr T G. High-speed convolution and correlation[C]//Proceedings of the April 26 - 28, 1966, Spring joint computer conference. ACM, 1966: 229 - 233.

[22] Dotsenko Y, Baghsorkhi S S, Lloyd B, et al. Auto-tuning of fast fourier transform on graphics processors[J]. ACM SIGPLAN Notices. ACM, 2011, 46(8): 257 - 266.

[23] Chazelle B. The bottomn-left bin-packing heuristic: An efficient implementation[J]. Computers, IEEE Transactions on, 1983, 100(8): 697 - 707.

(上接第 214 页)

参 考 文 献

[1] 马建设,赵雪江,苏萍,等. 基于 Android 系统的视频播放器开发[J]. 计算机应用与软件,2013,30(11):136 - 137,175.

[2] 唐敏. 基于 Android 平台的通讯帮手的设计与开发[J]. 计算机科学,2012,39(6):573 - 576.

[3] 李兴华. 名师讲坛—Android 开发实战经典[M]. 北京:清华大学出版社,2012.

[4] 吴亚峰,索依娜. Android 核心技术与实例详解[M]. 北京:电子工业出版社,2010.

[5] 刘凡俊,李登有. 球面的距离公式及其应用[J]. 数学教学研究,2013,32(3):39 - 43.

[6] 李刚. 疯狂 Android 讲义[M]. 2 版. 北京:电子工业出版社,2013.

[7] 范亚琼. 短信报警与定位系统的研究与实现[D]. 北京交通大学,2008.

[8] 庄翠翠,李成荣,韦玮,等. 基于 Android 系统的多传感器体感应用[J]. 计算机系统应用,2013,22(8):72 - 75.

[9] 中国地震信息网. 中国地震烈度表(GB/T 17742-2008)[EB/OL]. [2014-07-05]. http://www.csi.ac.cn/publish/main/847/1114/index.html.

[10] 中国地震信息网. 震级和震中烈度有何关系[EB/OL]. (2010-04-15). [2014-07-05]. http://www.csi.ac.cn/publish/main/720/721/20131025102955 - 797568625/index.html.