基于 OPENMP 的 HOG 特征并行化计算

2021 年 12 月 12 日

1.背景

使用 OpenMP 加速 HOG[1]（梯度直方图）特征计算。HOG 特征是数字图像处理领域一种经典的特征描述子，其描述了图像中一部份区域的梯度方向和强度统计信息，在过去常常被用来结合 SVM（支持向量机）进行行人检测等工作。本文基于 C 语言实现了 HOG 算法，并通过 OpenMP 进行了并行化加速处理，最后通过 Python 接口展示算法结果，并对加速性能做了分析。结果显示使用 OpenMp 实现了有效加速，当输入图像大于 500×500 像素时，使用 4 核处理器的加速比约为 3.72，加速效率约为 0.93。

2.HOG 特征检测

进行 HOG 特征提取时，需要先读取图像并转换为灰度图，本文使用 Python 完成了该过程。HOG 特征提取算法分为以下过程（见图1）：

(1) 提取图像的梯度特征。

(2) 计算梯度直方图。

(3) 区域直方图归一化。

该算法由 C 语言实现并将数据传入由 C 语言编译动态链接库进行处理。下面将简要介绍每个过程的算法原理。

2.1提取图像的梯度特征

使用 3×3 的 Sobel 算子进行梯度检测。Sobel 算子除了能够有效进行梯度检测外，还具有低通滤波器的特性，能够降低高频噪声干扰。Sobel 算子的表达式如下：

$$K_{sobelx}=\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}, K_{sobely}=\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix} \tag{1}$$

(a) (b) (c)

(d) (e)

图 1: 计算 HOG 特征的主要过程:(a) 为原始图像，(b) 为梯度水平分量归一化强度，(c) 为梯度垂直分量归一化强度，(d) 为 HOG 特征，(e) 为 HOG 中主要梯度方向与原始图的叠加（其中为了直观显示将梯度角度进行了旋转 90° 操作，以与边界平行）。

使用算子 K_{sobel} 与图像 I 进行二维卷积可以得到图像梯度水平分量 Gx 和垂直分量 Gy（分别见图1(b) 和 (c)：

$$Gx_{i,j} = \sum_{r=-1}^{1}\sum_{s=-1}^{1} I(i-r, j-s)K_{sobelx}(r,s) \tag{2}$$

同理可以求得 Gx。之后可以进一步求得每个像素 $I_{i,j}$ 的梯度强度和梯度方向。其强度 $G_{i,j}$ 为：

$$G_{i,j} = \sqrt{Gx_{i,j}^2 + Gy_{i,j}^2} \tag{3}$$

其梯度方向 $\theta_{i,j}$ 可以表述为:

$$\theta_{i,j} = arctan(Gx_{i,j}/Gy_{i,j}) \tag{4}$$

对于梯度方向，arctan 函数不允许分母 $Gy_{i,j}$ 为 0，因此使用 atan2 函数替代，该函数结果范围为 $(-\pi, \pi)$，一般将三四象限的梯度增加 π 旋转到一二象限中。

2.2计算梯度直方图

首先将图像划分为若干个 cm×cn 大小的 cell（单元），依次统计每一个 cell 内所有像素的梯度信息。对于一个 cell 中的每个像素，将其按照梯度方向划入若干个 bins 中。bins 实现形式为一个长为 N_{bins} 数组，分别对应梯度方向 $[0, \pi)$。于是，可以遍历 cell 中的每个像素，对于像素梯度方向 θ，找到索引 i, 满足如下关系：

$$\begin{cases} \alpha_i = \dfrac{\pi}{N_{bins}-1} * (i+0.5) \\ \alpha_i \le \theta \\ \alpha_{i+1} > \theta \end{cases} \tag{5}$$

则记 i_0、i_1 分别 i 和 $i+1$。当 $i_0 = -1$ 时，令 $i_0 = N_{bins} - 1$；$i_1 = N_{bins}$ 时，令 $i_0 = 0$, 则该像素对应的数组累加值为:

$$\begin{cases} \delta bins_{i0} = [1 - \dfrac{(\theta - \alpha_i)(N_{bins}-1)}{\pi}]G \\ \delta bins_{i1} = G - bins_{i0} \end{cases} \tag{6}$$

2.3区域直方图归一化

区域直方图归一化可以在保留相对明暗关系的条件下提升对不同光照条件的鲁棒性。其实施方法为将若干个 cell 划归到一个 block 中，并对所有 bins 组成的向量做归一化。本文用 L2 范数进行归一化。用 $bins_{m,n}$ 表示一个 block 中第 m 个 cell 的第 n 个 bin。则每个 block 的 bins$block_n$ 可以表示为

$$block_n = \frac{bins_{m,n}}{(\sum_M \sum_N bins_{m,n}^2)^{0.5}} \tag{7}$$

3.并行优化

OpenMP（Open Multi-Processing）是一个可移植、可拓展的并行程序开发模型，它为用 C/C++、Fortran 开发并行程序提供了轻便简单的接口。本文使用 C 语言实现了 HOG 基本算法，并通过 OpenMP

进行了优化操作。OpenMp 使用灵活，编程简洁，在信息处理、智能识别等诸多需要高性能计算的领域已经得到了广泛的应用。

OpenMP 隐藏了创建子进程，在并行区开始、同步、结束等过程中会产生额外的时间开销。因此，OpenMP 适合对大批量的数据处理操作进行并行化优化操作。下面将结合算法，着重介绍使用 OpenMP 优化 HOG 特征计算的细节。

首先，在图像梯度特征的提取过程中（Eq.[1]-Eq.[2]），使用两个 Sobel 算子对原图像进行二维卷积操作。在对每一个输出结果的计算中，仅仅需要对原始图像和卷积核进行读写。所以说每次循环之间不会发生写冲突，因此可以简单地使用“#pragma omp parallel for”语句来使用多个线程进行并行计算。一般来说在计算卷积时为了保留数据以避免数据损失，常常会进行零延拓、镜像延拓等延拓方式。一般实现这些方式常常用内存拷贝的方法实现，但是这会消耗大量存储带宽，因此这里分别遍历图像内部数据和边沿数据，用增加代码长度的方法来提升运算速度。二维卷积的主要计算代码如下：

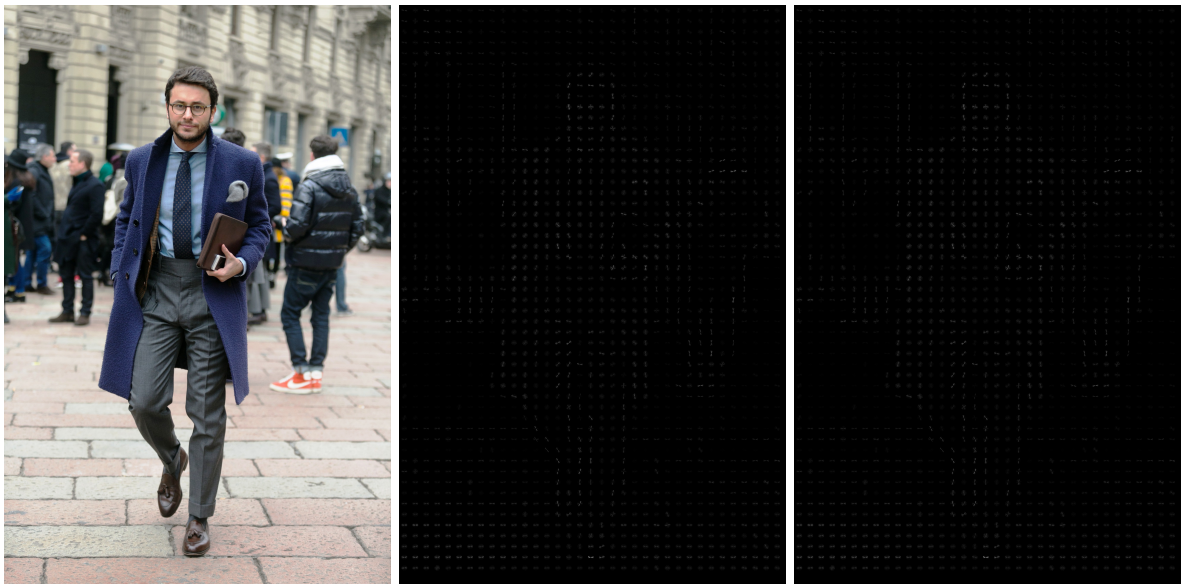
```
#pragma omp parallel for num_threads(4)
for(int i = kernal_rad; i < m - kernal_rad; i++){
    for(int j = kernal_rad; j < n - kernal_rad; j++){
        out[i][j] = 0;
        for(int k = 0; k < kl; k++){
            for(int l = 0; l < kl; l++){
                out[i][j] += kernel[k][l] \
                * data[i+k-kernal_rad][j+l-kernal_rad];
            }
        }
    }
}
```

其次，按照 cell 统计直方图特征的过程（Eq.[3]-Eq.[6]）也可以方便的实现并行优化。在统计每个 cell 的直方图特征时，为了避免写冲突，分配每个线程分别处理一个 cell 的统计特征。于是也可以使用与梯度计算类似的思路，用“#pragma omp parallel for”语句优化。需要注意的是，该过程涉及到了对外部声明的临时变量的处理，因此使用 private 语句对每个线程内的变量进行保护。

最后，对于区域直方图归一化的过程 (Eq.[7])，先计算每个 Block 的 L2 范数 $(\sum_M \sum_N bins_{m,n}^2)^{0.5}$ 作为分母，之后令该 Block 中的每一项分别除以该分母。可见，区域直方图归一化也被分解成了若干个无写冲突的简单过程，两个子过程可以分别用“#pragma omp parallel for”语句优化。

4.结果评估

结果评估包括了对算法有效性的评估以及对优化效果的评估。



(a) (b) (c)

图 2: 算法参数：cell 大小为 50×50 像素，block 大小为 2×2 cells，bins 为 18。(a) 为原始图像，分辨率为 2700×1800；(b) 的并行算法结果；(c) 为串行算法结果。

4.1算法有效性评估

首先在更多的图像上测试了算法的有效性。在不同分辨率的图像分别测试了并行和串行算法（图2、图3），这些测试使用了不同的参数（cell 尺寸、block 尺寸）。可以看到串行和并行算法均有效地提取了 HOG 特征，对于不同图像分辨率、cell 尺寸、block 尺寸均有较好的适应性。经过相关性比较后（图4），结果完全一致，这说明了并行优化算法没有在原算法基础上产生额外误差。

4.2优化效果评估

下面对加速效果进行评价。首先定义加速比 $S_P(q)$，为求解一个问题的最佳串行算法执行时间 T_S 与并行算法的执行时间 $T_P(q)$ 之比，其反映了并行算法的有效性，其中 q 表示核心数量。$S_P(q)$ 数学定义如下：

$$S_P(q) = \frac{T_S}{T_P(q)} \tag{8}$$

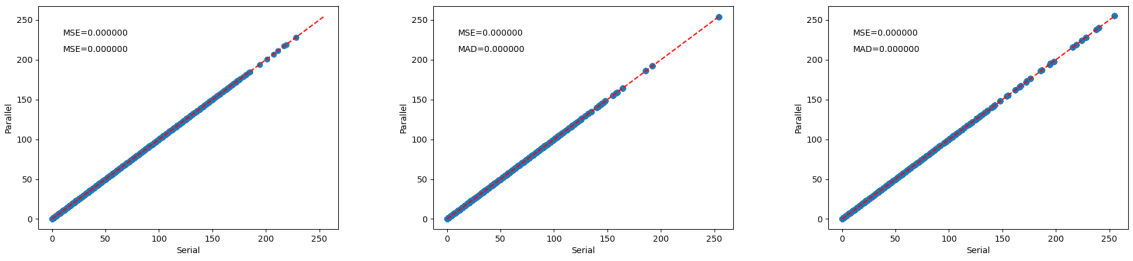
定义并行效率 $E_P(q)$，表明了是否充分发挥了各处理器核心的潜力。其数学定义如下：

$$E_P(q) = \frac{S_P(q)}{q} \tag{9}$$

于是，为了便于评价加速效果，将环境变量“OMP_PROC_BIND”设置为“true”以将每个线程严格按照限制在单个核心中，并通过设置“GOMP_CPU_AFFINITY”将每个线程与不同核心一一绑定。这样

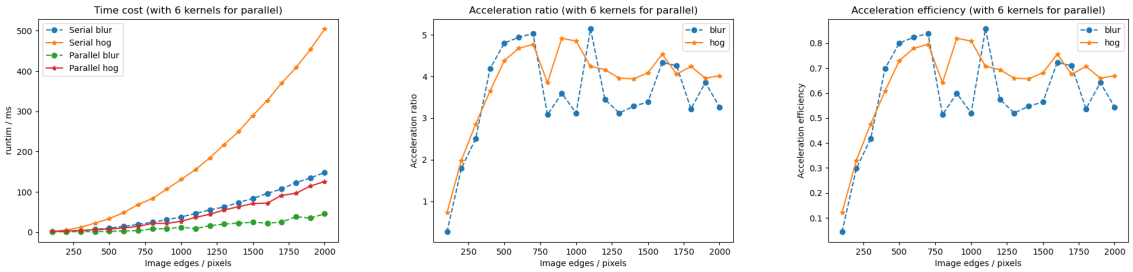
(a) (b) (c)

图 3: 算法参数：cell 大小为 20×20 像素，block 大小为 2×2 cells，bins 为 6。(a) 为原始图像，分辨率为 840×380；(b) 的并行算法结果；(c) 为串行算法结果。



(a) (b) (c)

图 4: 串行和并行结果图像素灰度相关分析，MSE 为均方差，MAE 为平均绝对误差 (a) 为图2的相关分析；(b) 为图3的相关分析；(c) 为图1中使用的 Lena 图像的相关分析。



(a) (b) (c)

图 5: 优化效果评估。(a) 为运行时间；(b) 为加速比；(c) 为并行效率。

通过在程序中使用 “num_threads(USE_THREADS)” 子句来指定线程数，进而控制核心数量，以便于评估并行效率。例如使用如下代码：

```
#pragma omp parallel for num_threads(USE_THREADS)
```

本小节在算法有效的基础上，产生若干具有不同尺寸的正方型随机图像用来测试算法的执行效率。测试环境为：处理器为 Intel(R)Core(TM)i79750H，其主频为 2.6GHz，具有 6 个核心；操作系统为 Windows 10。测试结果见图5。

图5(a) 表示了对于不同尺寸的正方形图像，串行高斯模糊滤波算法（简单的 2 维卷积）、并行高斯模糊滤波算法、串行 HOG 算法、并行 HOG 算法运行所需时间。可以看到，对 2 维卷积和 HOG 算法的优化都大幅提升了程序运行速度，节省时间随着图像尺寸的增大呈现指数性的增加。图5(b) 表示了简单的 2 维卷积以及 HOG 算法的加速比。可以看到随着图像尺寸增大，加速效率不断提高，并最后趋于平缓；对于较小的图像，可能存在管理并行化所需要的多线程管理、数据搬运等步骤的时间耗费高于并行化节省的时间，出现了 $S_P(q) < 1$ 的情况；当图像增大时，并行化管理耗费的时间占比减小，加速比提升不明显，甚至由于内存搬运的原因出现下降。5(c) 则表示了简单的 2 维卷积以及 HOG 算法的并行效率。可见 $E_P(q)$ 最高约为 0.85，平均在 0.6-0.7 之间，实现了较高的并行效率。

参考文献

[1] N. Dalal and B. Triggs, ”Histograms of oriented gradients for human detection,” 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.https://ieeexplore.ieee.org/document/1467360