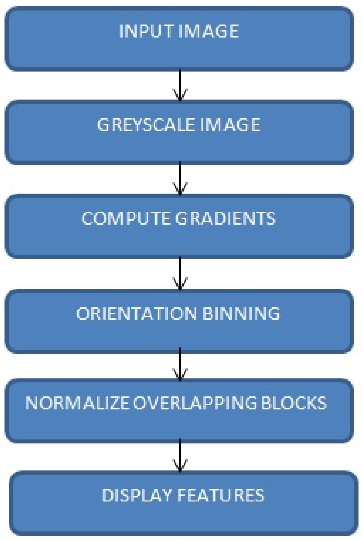
HOG feature extractor in GPU

December 12, 2015

1 Introduction

HOG descriptors or Histogram of Oriented Gradients descriptors are feature descriptors used in computer vision for the purpose of object detection. This method is very similar to that of other feature extraction algorithms like SIFT(Scale Invariant Feature Transform), SURF(Speede Up Robust Features) and others. But it differs in the fact that it is calculated by dividing image into grids.In this project our focus is implement the HOG feature detector function in GPU which is analogous to the function "extractHOGFeatures" in MATLAB. The

Figure 1: Flow chart for HOG feature extractor



central idea of the topic is that shape and the appearance of objects within an image can characterized by the distributions of edge directions. Now this can be achieved by dividing image into smaller connected areas called cells, computing the histogram of gradients within each cell and combining them. In order to get improved performance we can use the normalization procedure applied to larger area called the block. Block contains cells and the block normalized value can be used to normalize the cells. This procedure helps in reducing the effect of illumination changes.

2 Algorithm and Flow chart Description

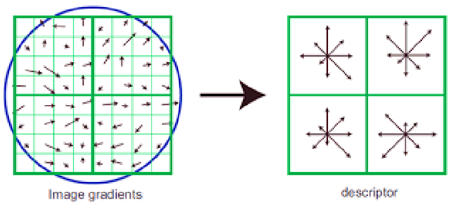
HOG feature extractor algorithm has three main steps which are described as follows:-

1) Gradient Computation- One dimensional gradient filters are used to detect the horizontal and vertical edges. If I is the image then $I_x = I * D_x$ and $I_y = I * D_y$ are the gradients of the image in x and y directions respectively. Its is shown in figure 2.The magnitude and phase of the gradients are given by $\sqrt{I_x^2 + I_y^2}$ and $\theta = \arctan \frac{I_y}{I_x}$. Where Dx and Dy are as follows:

$$Dx = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$Dy = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

Figure 2: Gradient and descriptors

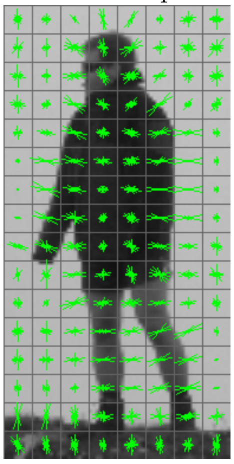


2) Orientation binning- The image is divided into square grids with size being a multiple of two. Each pixel within the cell casts a weighted vote for its orientation with magnitude being that of the gradient. The vote is actually the gradient magnitude or the square root of the gradient magnitude of the particular pixel.

3) Block Descriptor - In order to account for changes in illumination and contrast, the gradient strengths must be locally normalized, which requires grouping the cells together into larger, spatially-connected blocks. The HOG descriptor is then the vector of the components of the normalized cell histograms from all of the block regions. These blocks typically overlap, meaning that each cell contributes more than once to the final descriptor.

4) Display- HOG descriptors can be displayed in multiple ways. Simplest way is to assign color to each orientation or histogram wheel can be utilized.One of the ways to represent is shown in figure 3.

Figure 3: Representation of descriptors on a grayscale image



The flow chart for the above presented algorithm is shown in the figure 1.

3 Implementation

As per the Hog Algorithm mentioned in section 2 Computation of gradient values requires to perform convolution of image with matrix Dx and Dy. This convolution is basically addition and subtraction of 1 from the neighboring pixels of a current pixel. This simple computation eliminates the need to use Cublas library functions.In order to compute Hog Features on GPU user is required to input 7 parameters i.e. input image, output image, Cell size, block size, number of bins , Orientation.

1. Input image: name of the image file used to detect features.
2. Output-image: name of the output image file which will display the features based on the orientation.
3. Cell Size: defines size of each cell histograms to be created. Therefore the number of cell rows and number of cell columns can be calculated as Image Rows/Cell Size, Image Columns/Cell Size.

4. Block Size: defines size of each block. Therefore the number of Block rows and number of Block columns can be calculated as (Cell Row Block Size+1)/(Block Size- Block Overlap), Image Columns/Cell Size, (Cell Columns Block Size+1)/(Block Size- Block Overlap).

5. Block Overlap: Number of Blocks that will overlap.

6. Number of Bins: Weighted number integers to classify the orientation of every pixel.

7. Orientation: Provides orientation to be considered 0 indicates 180 and 1 indicates 360.

Following Kernels are used to calculate features:

- Cal-kernel - 8*8 threads are launched in every block. Number of blocks launched is Image Rows/ threads per block * Image Columns/ threads per block. This kernel calculates gradient and orientation of every pixel of an input image. It also calculated orientation of every pixel for display image.

- Cell-kernel - 8*8 threads are launched in every block. Number of blocks launched is number of Cell Rows/ threads per block * Cell Columns/ threads per block. This kernel performs orientation binning.

- Block-kernel - 8*8 threads are launched in every block. Number of blocks launched is number of Block Rows/ threads per block * Block Columns/ threads per block. This kernel performs normalization of cell histogram.

3.1 Displaying Image

Features are calculated for every cell and recorded in a Feature.txt file based on the cell size given by the user. However a separated set of features are calculated jus to display these features to an output image file. This output image displays only the significant features of the image file. Following parameters which are considered for calculations of display features are as fixed:

1. Number of Bins = 4;

2. Orientation = 0 i.e. 0-180

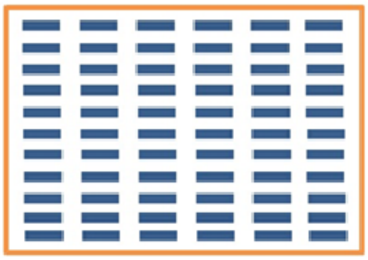
3. Cell Size = Image Rows/32, where 32 is the number of display cell rows in an image and is a fixed value. Therefore Cell Columns is calculated as Image Columns/ Cell Size.

4. Size of the display Image = 17 * Cell rows (32) * 17 * Cell Columns.

Following Kernels are used to calculate features:

- Display-Cell-kernel - 8*8 threads are launched in every block. Number of blocks launched is (Cell Size*Display Cell Row/ threads per block)* Cell Columns/ threads per block. Figure 1.1 shows. This kernel performs orientation binning.
- Display-kernel - 4*4*4 threads are launched in every block. Number of blocks launched is number of Cell Rows/ threads per block * Cell Columns/ threads per block. This kernel performs normalization.

Figure 4: Representation of threads launched in Display-Cell-kernel



3.2 Computational Speed

In order to exploit advantages of GPU following features of CUDA programming are implemented in this project.

- Pinned memory is allocated in CPU to store image, to store features of input image. This reduces transfer time between CPU to GPU and GPU to CPU.
- Two streams are used to launch kernels in GPU. Hence in this program Cal-Kernel and Display-Cell-kernel both are overlapping and computing at the same time. Similarly cell-kernel and display-kernel are overlapping.
- Display-Kernel launches threads in 3 dimensions. As a result many threads are launched per block.

3.3 Results

1. 'dog.bmp' image is used to extract features. Computation time required for extracting features of Dog.bmp image is as shown below:

Note: Command used - salloc -t 5 -A ece406 -p ece406 –gres=gpu:1 srun hogfeature dog.bmp dogH.bmp 8 2 1 4 0

Transfer time from CPU to GPU = 1.99 ms
GPU Execution = 550.73 ms
Transfer time from GPU to CPU = 2.38 ms

Total Execution = 555.10 ms

Figure 5: Input Image

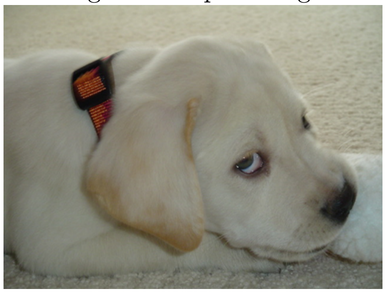
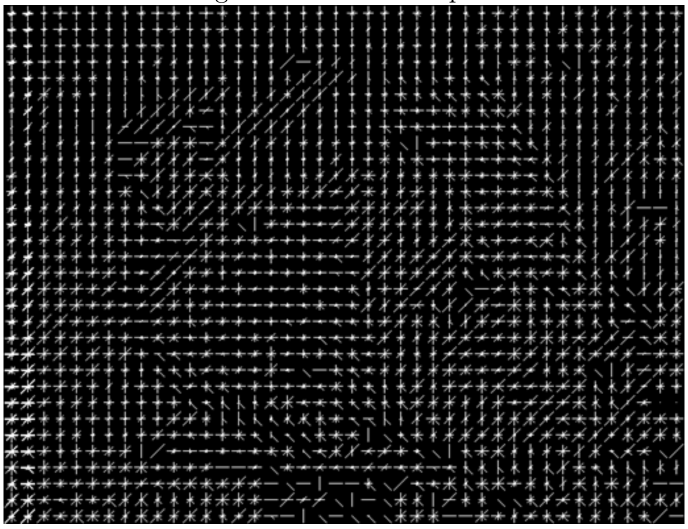


Figure 6: HOG Descriptor



2. 'lena.bmp' image is used to extract features. Computation time required for extracting features of lena.bmp image is as shown below:

Note: Command used - salloc -t 5 -A ece406 -p ece406 –gres=gpu:1 srun hogfeature lena.bmp lenaH.bmp 8 2 1 4 0

Transfer time from CPU to GPU = 0.47 ms
GPU Execution = 15.37 ms
Transfer time from GPU to CPU = 0.12 ms
Total Execution = 15.96 ms

Figure 7: Input Image

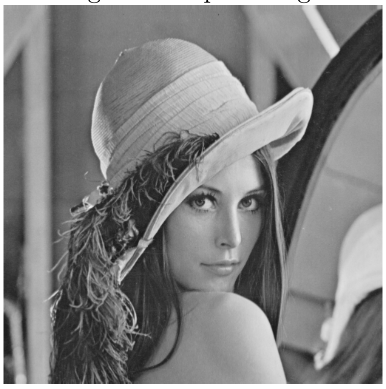
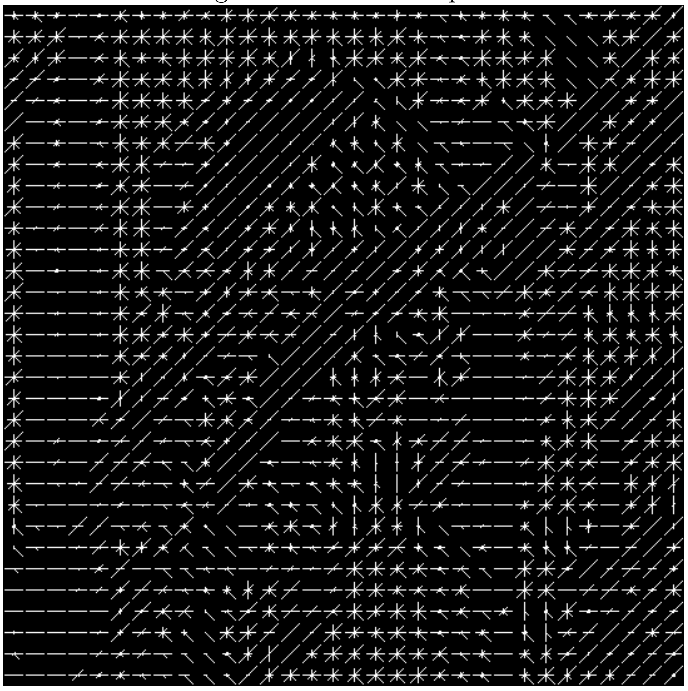


Figure 8: HOG Descriptor



4 Conclusion

We have been successful in extracting the features and displaying features for Dog.bmp image, lena.bmp and few other images. However, we are unable to display the extracted HOG features for certain images.

5 References

Dalal, Navneet, and Bill Triggs. ”Histograms of oriented gradients for human detection.” In Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, pp. 886-893. IEEE, 2005.