INDIANA UNIVERISTY BLOOMINGTON

Assignment 1

B657 Computer Vision

REPORT

Submitted by

Rohil Bansal
Masters Candidate in Computer Science
School of Informatics and Computing
Indiana University
bansalro@indiana.edu

Gurleen Singh Dhody
Masters Candidate in Computer Science
School of Informatics and Computing
Indiana University
gdhody@umail.iu.edu

Chitesh Tewani
Masters Candidate in Computer Science
School of Informatics and Computing
Indiana University
ctewani@iu.edu

Under the Guidance of:

Professor David Crandall
Asst. Professor, School of Informatics and Computing
Indiana University

Part 1 - Watermark

Constants:

Alpha: 10

Radius: (image_size/2 – 50)

Watermark constant: 0.25

Check for Watermark:

Function signature:

Check_image(SDoublePlane inputImage, int N) where N is the user input

The function checks whether a watermark is present in the image or not by calculating the Pearson's correlation coefficient and compares it with a threshold value (in our case it is 0.25).

The correlation coefficient calculates the relation between the vector v (that is calculated based on the user input N) and the vector c which is calculated by calculating the real parts of the circle in the frequency domain.

Add Watermark:

Function signature:

Mark_image(SDoublePlan inputImage, int N) where N is the user input.

The function adds a watermark in an image by creating a circle on the image of radius (length of image/2 – Radius) and with alpha value 10. The circle is created of N number of pixels where N is provided by the user. Now this circle is correlated to the vector v which is a binary vector. The binary vector v = (v1, v2…..v_l) (with v_i = (0, 1) and the length l a parameter of the watermarking algorithm) that appears to be a

random sequence. We can seed a random number generator with N, and then use the random number generator to produce the l binary digits. So, if the vector's value is 1, then the pixel for the circle is highlighted else nothing happens.

Note:

While testing the program on a set of about 50 different images, we found out that one set of parameters is not applicable for each type of image. The alpha values need to be adjusted according to the image to get the appropriate result. Some images have more high frequency areas and in those images, large alpha values produce noise while the same alpha value works fine for images having a good mixture of low and high frequencies.

Qualitative Analysis:

While doing qualitative analysis on a set of about 30 images, we found the anomaly as described above. However, following are the few screenshots of the images after watermarking them with alpha value 10:

Also, while checking whether the watermarks were present in a set of 30 images, the program could identify 25 images correctly.

Quantitative Analysis –

To test this, we generated a water-marked image using N = 50 and constant parameters. Ranging values of N from 1 to 100, we checked if the model is able to check the watermark in the water-marked image (of N = 50).

Results of which are follows –

True Positives – 1

False Positives – 10

False Negatives – 89

True Positives – 0

To perform the test, execute –

```
./watermark 1.3 input_image.png watermarked_image.png quantCheck 50
```

Part 2 – Car Object Detection

Procedure to run the program:

After cloning the GitHub repo to a particular directory and then move to part2 directory type the command “make detect” and then compile the application

./detect <Image-Name.png>

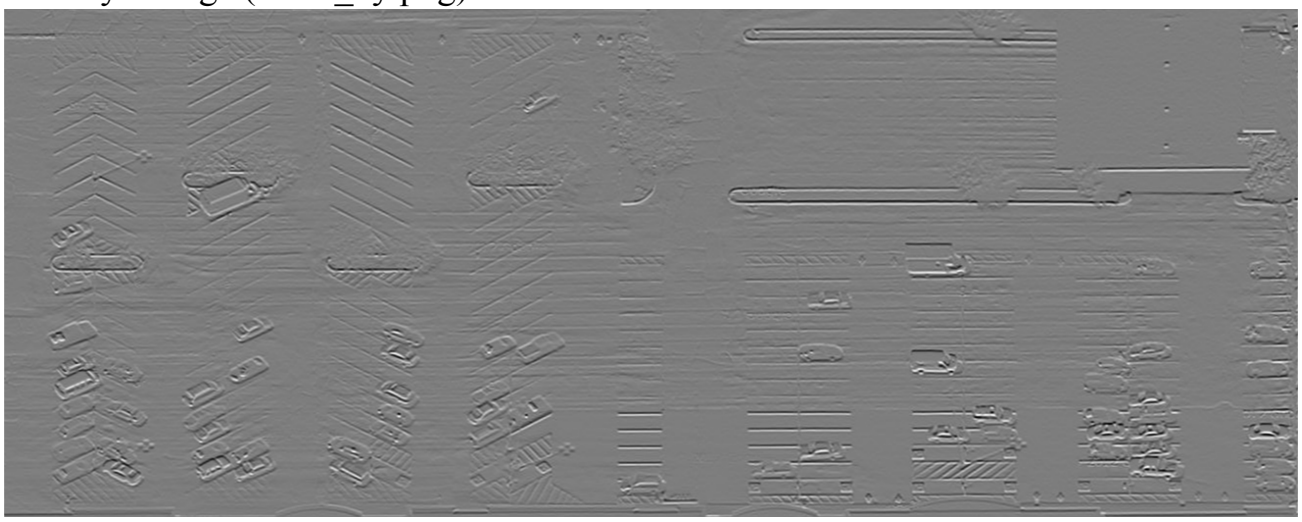
Helper ssh files:
./run-informatics.sh Cleans and Builds the project with Informatics.png image
./run-srsc.sh Cleans and Builds the project with SRSC.png image
./run-plaza.sh Cleans and Builds the project with Plaza.png image
./clear-text-file.sh removes all text files in the current folder.

The Two approaches

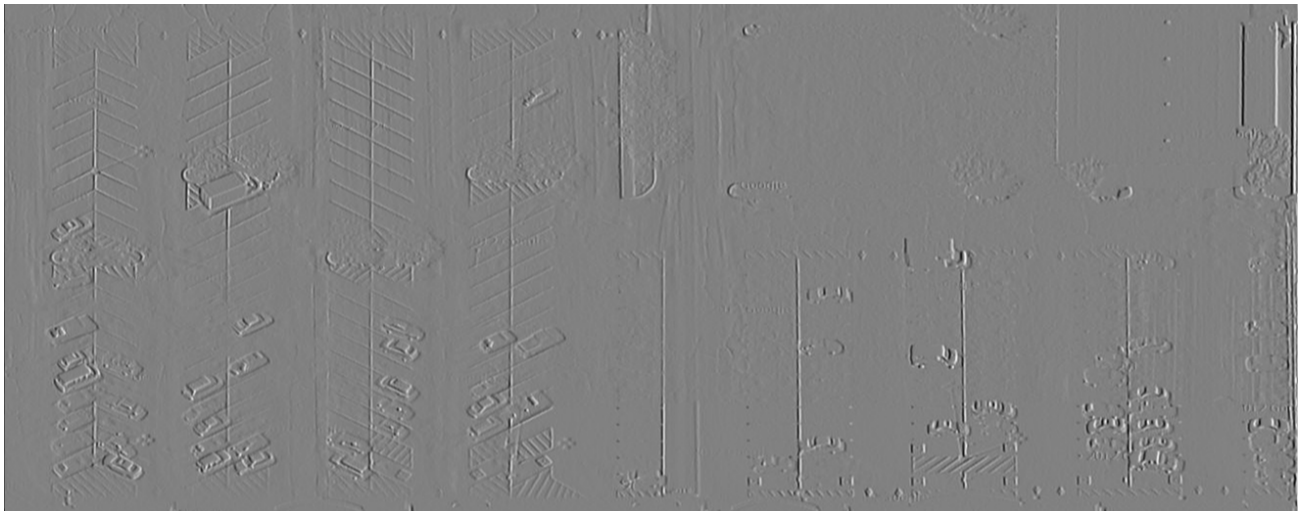
- Morphological Filter – Template Matching
- HOG Detector

Morphological Filter – Template Matching [5][2]

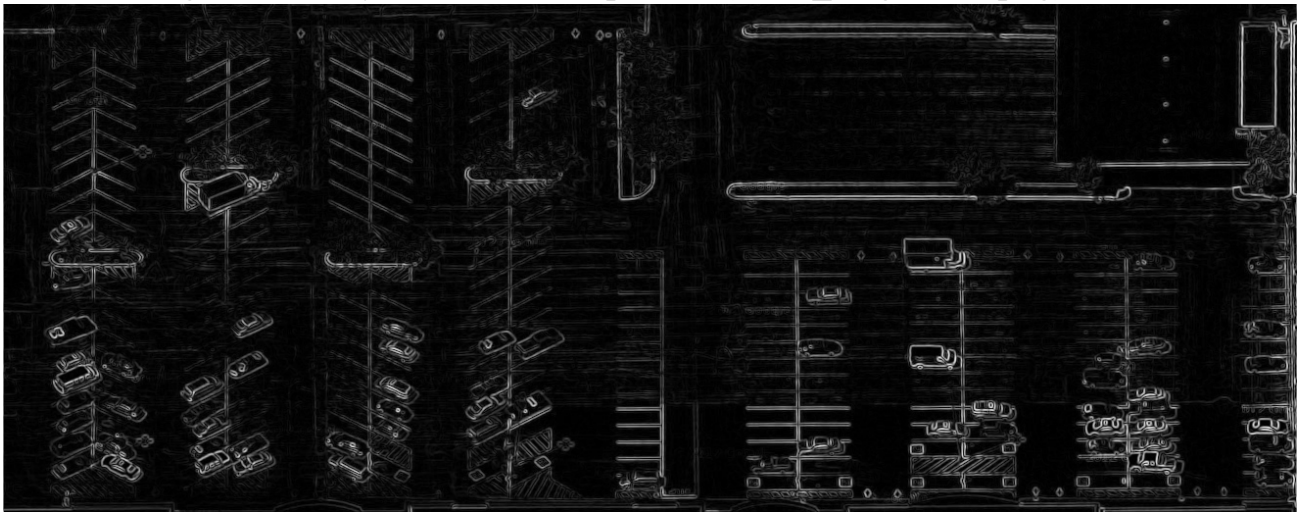
Model Flow:
1) Load Normal Image (Example images shown for Plaza)
2) Using the Sobel edge detector
   a) Find dy change (sobel_dy.png)



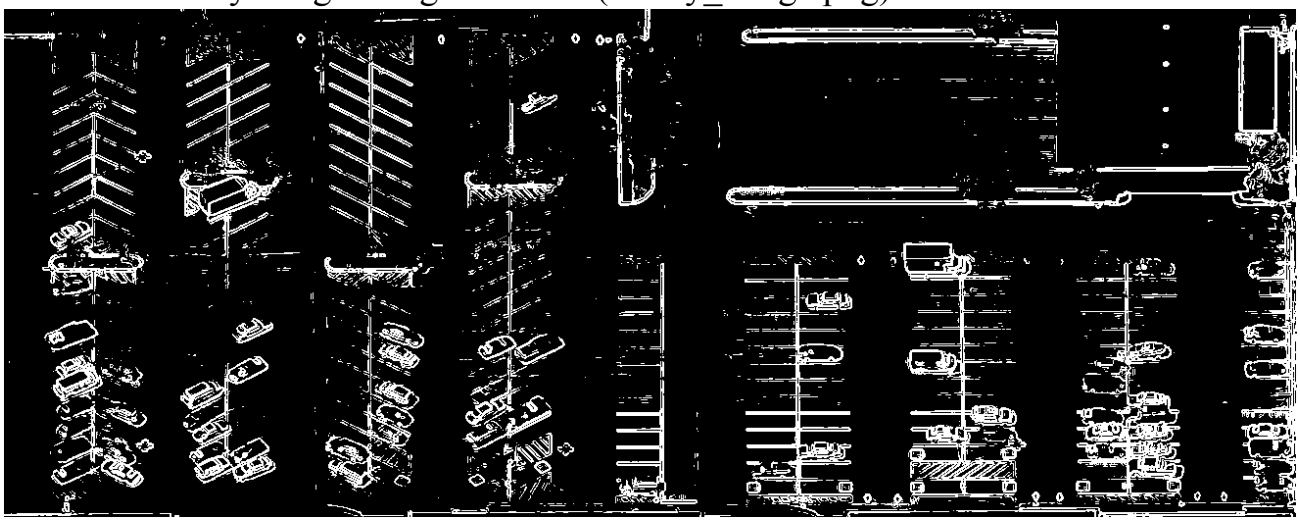
   b) Find dx change (sobel_dx.png)



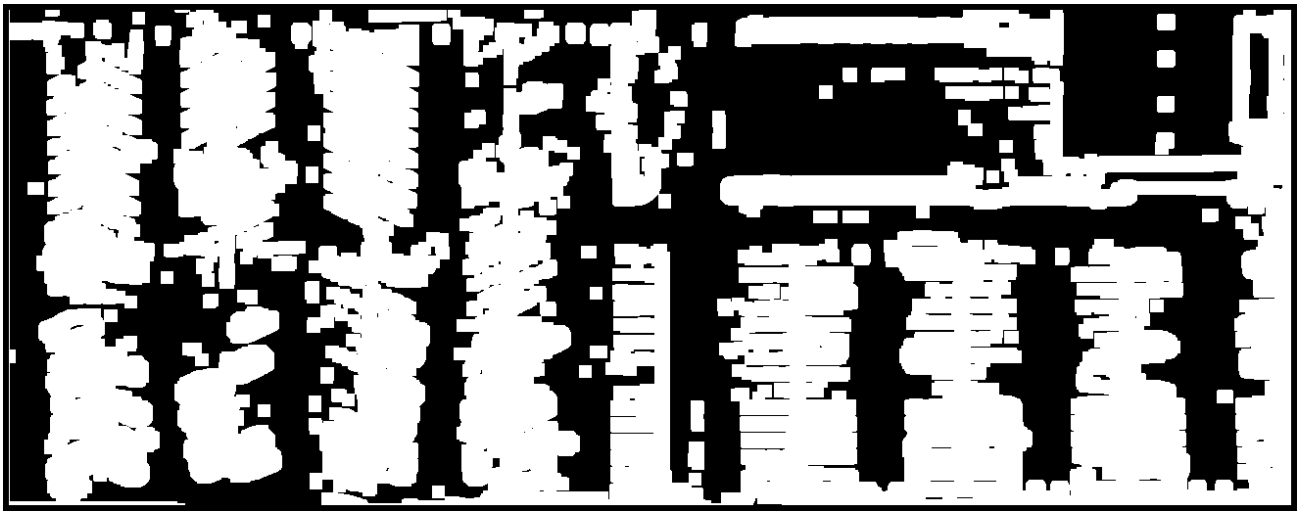
3) Find the magnitude of dy and dx sobel operator (sobel_magnitude.png)



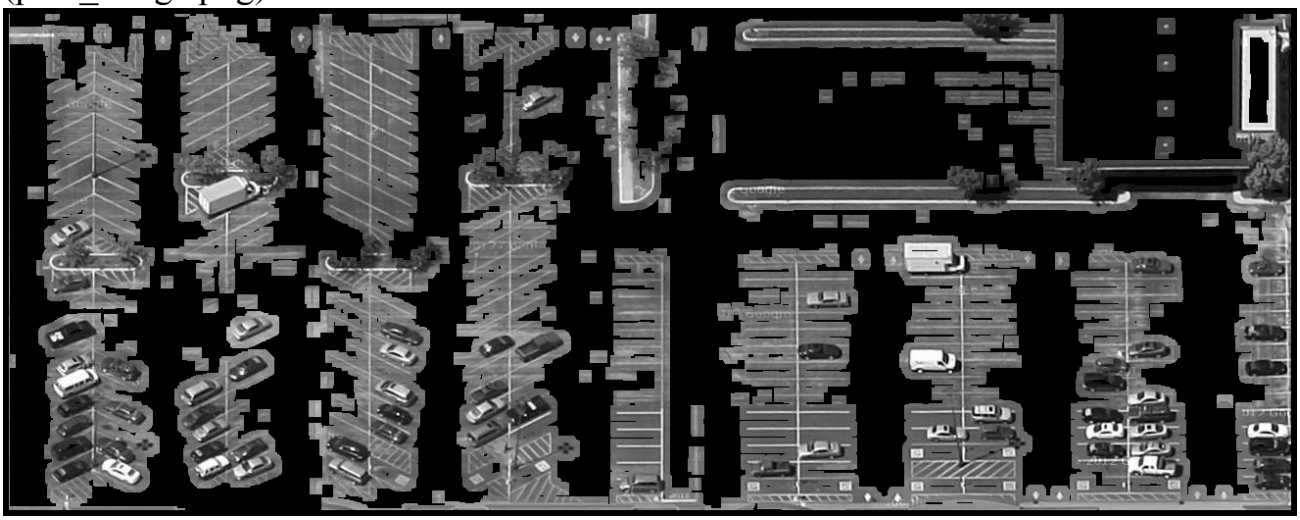
4) Covert to binary image using threshold (binary_image.png)



5) Apply morphological filter, do the dilation process (morph_dilation.png)



6) Using Image produced at Step 5 we multiply it with the Image in Step 1 (pass_image.png)



7) Using Image in Step 6 we convolve the templates using sliding window technique
8) The Pearson's correlation coefficient is used to find match between template and a window
9) Setting a threshold, we get the windows in image where car is detected
10) Applying non-maximum suppression algorithm multiple window detection for same car are removed
11) All windows are written to output image and text file (final_overlay_output.png)

Design Decisions:

Sobel Operator

The sobel operator used is as follows:

$[-1\ 0\ 1]^T * [1\ 2\ 1]$ for dy | $[1\ 2\ 1]^T * [-1\ 0\ 1]$ for dx

The implementation for convolving used a separable kernel which was faster than a general convolution. Through this we were able to achieve the first three steps in 'Model Flow'.

Binary Threshold

Since in the given image we are only concerned with detecting cars, the remaining area in image is just noise for us. After applying the sobel detector we can use the binary threshold to remove the unwanted edges. The edges of a car provide a sharp contrast change which help the sobel to detect bright thick edges. Applying a low threshold to convert sobel to binary image preserves the car edges. We played with different values. Values between 30-60 serve the general purpose.

Morphological Filter

The morphology filter helps to do a dilation or erosion in binary image. Dilation increases the thickness of lines. Whereas erosion reduces the thickness of lines. We understood the concept by reading articles. We applied dilation on our image to fill

the rectangles of our car detected by sobel. The dilation filter is a 13*13 matrix of 1. Where the size of the square matrix can be defined in the constant.h file.

Pass Image

Using the morphological Filter output M[x,y], and the original Image I[x,y]. We multiplied each pixel with the corresponding pixel.
M'[x,y] = M'[x,y] * I[x,y]
This provided a image that had the original image with all the cars. Whereas the remaining part of the image is blacked out. This step helps us to reduce false positives. Since when template runs over a black/semi-black (noise – no-car) zone the correlation between the two will be very less.

Pearson's Correlation

This correlation is between the template and the selected window in the image, if the window contains a car will give a high value. By setting a threshold we can somewhat guarantee that the given window contains a car. **We use the correlation coefficient as confidence in our detectedBox class**

Non-maximum Suppression

Since we are using a sliding window technique it is more than possible that a car gets detected more than once. For that we applied an intuitive algorithm. Remember as each window detected stores its top left corner pixel value. Using this we calculate the Euclidian distance between the top left corner points of each window. By setting a threshold(|T|) we can declare that these windows point to the same car. Given the case multiple windows get detected for the same car. We select the one with the highest confidence(correlation).
W = width of the template; H = height of the template; sqrt is the square root function.
|T| = sqrt(W*W + H*H) / 3.00

Functions Implemented:

gaussianValue_xy:

The function is used to produce a value for a 2D Gaussian template

gaussianValue_x:

The function is used to produce a value for a 1D Gaussian template

create_gaussian_kernel_1D:

Creates a 1D Gaussian kernel, requires window size and sigma value

create_gaussian_kernel _2D:

Creates a 2D Gaussian kernel, requires window size and sigma value

extend_image_boundaries:

Extend the resolution (column * row) of an image by copying the boundary pixel values into the extended column and rows. Requires the original SDoublePlane 2D array of the original image, length to extend, and name of file that will be written to disk after operation performed on original image

min_max:

Finds the minimum and maximum value in a 2D SDoublePlane plane, function used generally by write normalized image. Takes the input SDoublePlane and min max reference variables

toString:

Function hacked by Gurleen Singh that takes a number and converts to a string I n cpp

write_normailzed_image:

Normalizes a SDoublePlane values in the range of 0-255 (grayscale) so it can be written to an image. Take the input SDoublePlane of original image to be normalized and filename used to write output file to disk.

convolve_seperable:

Convolve Separable uses 2 1D separate kernel for convolution. Generally, the 2 1D kernel multiply to form one 2D kernel. But since the operation of 1D kernel running 2 times is faster than the conventional method it is preferred. Giving the same results.

convolve_general:

Convolve Separable uses a 2D kernel for convolution. Easy to implement but slower than 2 times of 1D kernel

binary_image:

Using a threshold value in image converts any pixel above that to 255 whereas remaining pixel are left at 0 intensity. Thus, converting a binary (0-1) image.

erosion:

morphological filter that does the operation of erosion

dilation:

morphological filter that does the operation of dilation

sobel_gradient_filter:

Applies the sobel filter over the image to detect edge gradient map. In input, we can specify the dy or dx operation of the sobel over image.

hog_implementation:

Histogram of Oriented Gradient is implemented. The function takes input of a file name and outputs its HOG vector in a text file. Uses HOG defined constants

window_mean:

Calculates the sliding window mean intensity value

window_variance:

Calculates the sliding window variance intensity value

window_template_variance:

Calculates the sliding window & template co-variance intensity value

slide_window:

The function that governs the movement of the template over the image and continuously tries to find a correlation between templates and sliding window above a given threshold

remove_duplicate_box:

Each or may templates may detect the same car. We don't want multiple windows of detection on the same car. Implemented a naïve algorithm that does a good job of removing duplicate windows.

magnitude_image:

Used to find the magnitude of the image based on the dy and dx components of sobel operator

sliding_window_car_detection:

The function is a wrapper that calls the slide_window function and remove_duplicate_box function. And then outputs the desired file.

cubicInterpolate:

In progress implementation of cubic Interpolating of rescaling an image

biCubicInterpolate:

In progress implementation of biCubic Interpolating of rescaling an image

Constants Defined in constants.h

JUMP_X

The jump/translation sliding window takes in X direction

JUMP_Y

The jump/translation sliding window takes in Y direction

CORR_COEFFICIENT_THRESHOLD

The threshold used by slide_window to detect if the window in the image is a car or not when compare to the given template. The higher the value the difficult it is a window to be selected as a car. Optimum values lie between .4 to .5. A higher value prevents black contrast cars to get detected, which we don't want.

MEAN_THRESHOLD

The mean threshold used to precisely select a sliding window based on mean to be correlated with the template. Helps reduce redundant windows of cars and false positives. Also, helps in efficient computation. Optimum values lie between 30 to 60. If MORPH_DILATION_FILTER and BINARY_IMAGE_THRESHOLD are kept high this can be kept low as most of the noise gets removed by morphology operation, playing a less important role.

MORPH_DILATION_FILTER

The square matrix filter of 1's. This constant defines its size. Optimum values lie between 13 to 17. (only takes odd numbers)

BINARY_IMAGE_THRESHOLD

Intensity Pixel value that helps to convert a grayscale sobel edge map to binary image. Difficult to find optimum value for all images. Optimum lies between 40 to 95. A significant increase in BINARY_IMAGE_THRESHOLD must be compensated with the increase in MORPH_DILATION_FILTER. As more reconstruction of pass image is required. Though a high value makes sense to remove noise, but many times the edge intensity detected by sobel for cars is not bright especially that are in a slanted angle.

CAR_TEMPLATE_NAME

Constant to specify the location of template images used.

HOG_CELL_SIZE

HOG Detector constant, suitable value is 8

HOG_BIN_SIZE

HOG Detector constant, currently works with only 9

HOG_BIN_SEPERATION

Standard value taken to be 20

HOG_BLOCK_SIZE

Standard value taken to be 2

PI

Constant PI value

NUMBER OF TEMPLATES

Number of templates it picks from the templates folder. 7 Specifies first 7 templates to pick. The name of the templates should be sequential as in "Templates/template-car<Serial-Number>.png". If NUMBER_OF_TEMPLATES is 3 then there should be 3 files in Templates folder as
template-car1.png, template-car2.png, template-car3.png

BLACK_PIXEL_COUNT

A naïve implementation of removing big box shaped areas from being detected as cars. Though the big boxes do remain after binary thresholding and dilation but their inner area has some left out black pixels. While calculating mean of a sliding window if black pixels encountered are more than threshold then it ignores the window altogether.

When It Works Well

- The intuitive steps involve keeping only that part of the image that contains cars in the pre-processing step. If the image contains mostly car images it will work well without false positives. Though any structure in the image that has a shape

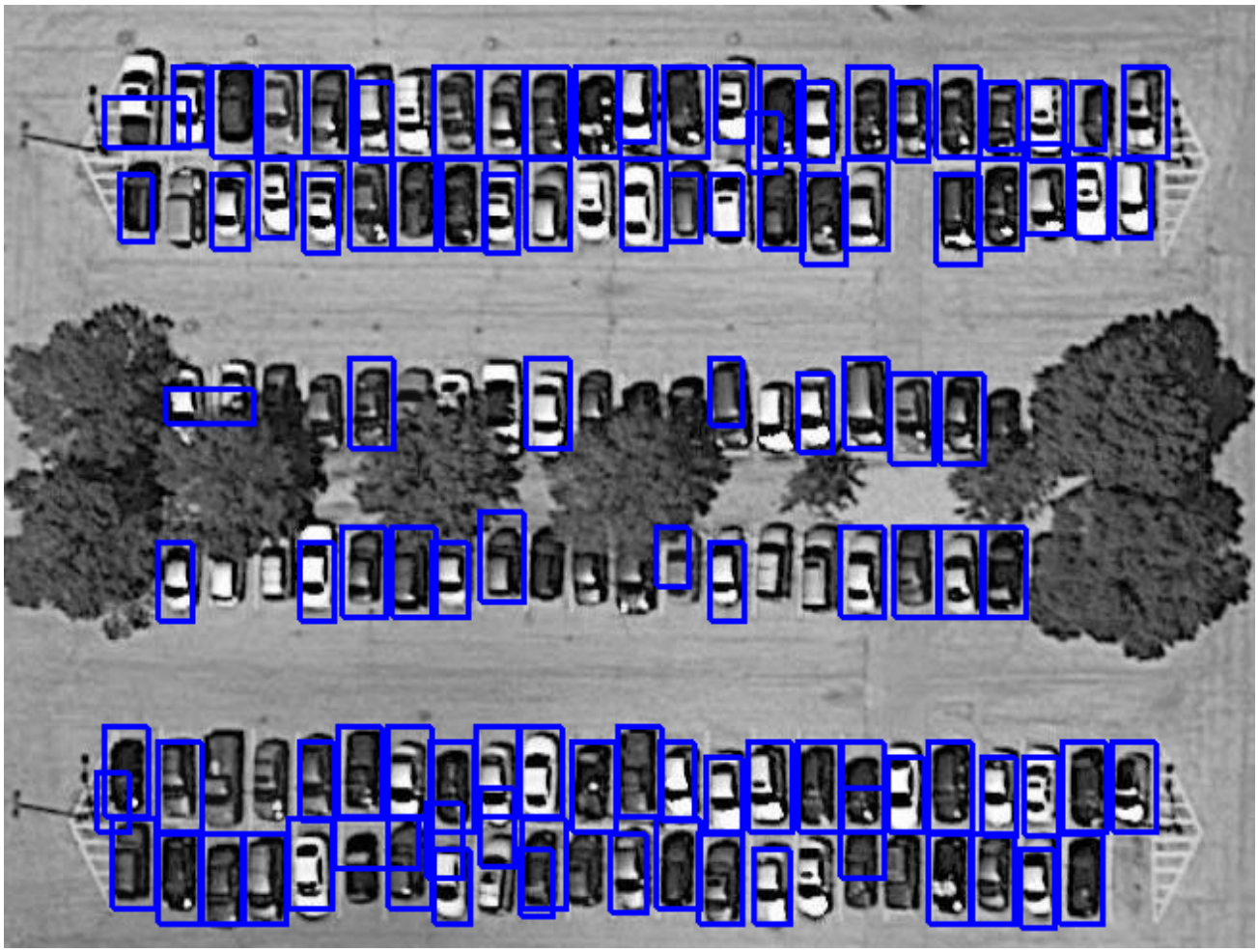
of the car, remains even after dilation and has a good chance of getting detected as a car.

- Very less or 0 false positives if parameters are strictly tuned to detect only cars. If we relax the hyper-parameters, then the number of cars detected increases substantially but at the cost of negligible amount of false positive (2-3). The configuration of parameters can be left to the user depending on his desired application.
- Using several templates help us to detect multiple cars. More the number of templates higher the number of cars detected.
- Templates and image have cars of the same scale

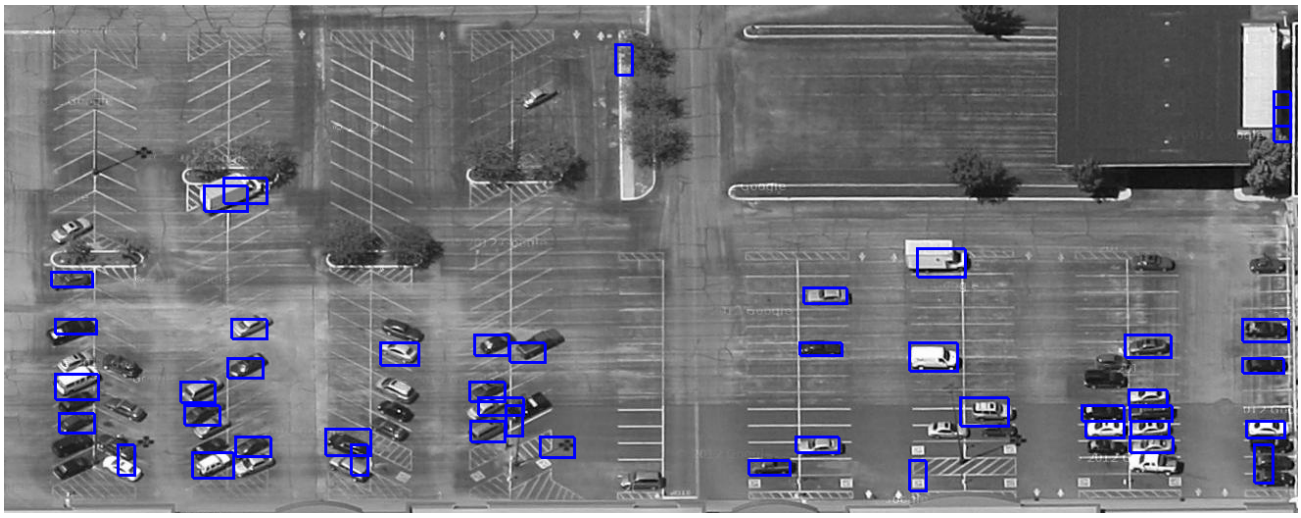
Result –

Using 24 different car templates, we detected the cars in provided images.

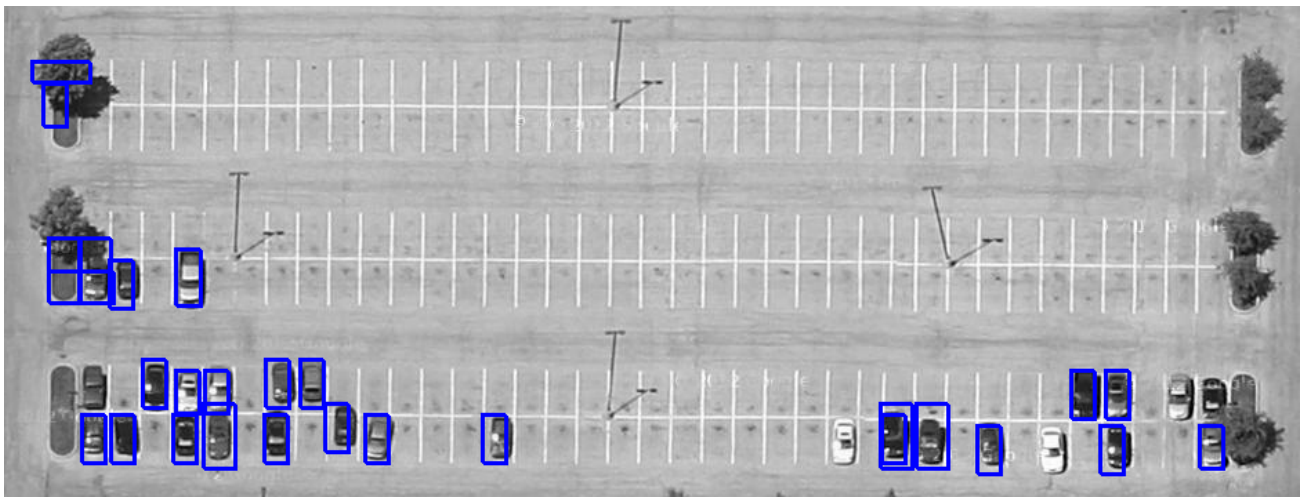
Informatics.png – 102 cars detected, 4 overlap



Plaza.png – 37 cars detected, 5 overlap, 5 false positives



SRSC.png – 23 cars detected, 1 overlap, 4 false positives



Drawbacks

- The templates are of fixed sizes. If we rescale the templates their quality reduced and we get poor results. Rescaling the image to make it fit with ratios of the template has been left as future work.
- The morphological filter dilates the boundaries of parking spot. So using a high filter is a double edge sword, when there is no car present in that area. As even that area gets included to be tested under given templates for car detection.

- The templates define the types of car we can match. Any vehicle that may have an uneven shape or contrast nature with respect to the templates might not get detected. Though including the undetected templates may solve the future problems, but no way it's an efficient solution.
- More the templates higher is the running time (24 templates take around 5 minutes)

Future Work

- Finding templates that give maximum result for many images. Removing templates that have a low contribution score
- Fine tuning the constants to find a pair that produces best result in different parking lots
- Implementing the Image sub-sampling "down sampling" of images constructing the Image pyramid to match templates to find cars. Thus, providing a solution of finding cars in images where template does not match the cars scale in a test image.

HOG Detector (Incomplete) – Future Work [1]

With the template approach, we had an early intuition that no matter how many templates we got they won't be sufficient to work with all the cars in the images. Plus, an overhead of selecting many templates and running across a single image would be an overhead compared to computation cost. To resolve this the team explored a different option with respect to templates. That would do a similar & better job of template comparison but more in terms of mathematics and numbers than just plain images, and also being much faster since only one pass of the image is required compared to the N passes for N number of templates. The team came across Histogram of Oriented Gradients Detector.

It could collect a large sample of +ve images (car templates) and equally number of -ve images (neutral areas in parking lots) and train a SVM/logistic classifier to provide a confidence for object – car – detection. The best part is that the first designed approach 'morphological' is as per the framework we needed to implement HOG. Means, which is easily replaceable. Using most of the earlier templates (24px*48px) from the first approach as +ve data we could train the machine learning model. And the sliding window detector which detected the correlation factor will now instead

detect the hog vector and find a numerical similarity in our trained ML model. We also need the Image down-sampling listed as future work in the first approach.

What we have done so far

Implemented the HOG detector
The function is a standalone function, which when passed a template generates a HOG vector which it outputs in a text file

What we still need to do

Provide the function with multiple +ve -ve images so it could output the HOG vector for each sample & Load all the vectors in a dataset (manual labelling)
Run an ML algorithm somewhere on the lines of SVM or logistic to train our model on car detection
Modify the sliding window code in our framework that generates a call to the hog function to provide a hog vector that can be used by our trained ml model to classify

References

1) http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf Histograms of Oriented Gradients for Human Detection Navneet Dalal and Bill Triggs INRIA Rhone-Alps, ˆ 655 avenue de l'Europe, Montbonnot 38334, France
2) https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm
3) http://www.cs.cornell.edu/courses/cs4670/2013fa/projects/p5/
4) https://www.youtube.com/watch?v=7S5qXET179I&t=3171s
5) VEHICLE DETECTION FROM PARKING LOT AERIAL IMAGES Huan Weia,b, c Guoqing Zhou*,a Zezhong Zhengc, b,a Xiaowen Li b, c Yalan Liud Ying Zhangc Shang Lia, c, b Tao Yuea a Guangxi Key Laboratory for Spatial Information and Geomatics, Guilin, Guangxi, PRC 541004 b State Key Laboratory of Remote Sensing Science, Jointly Sponsored by Beijing Normal University and the Institute of Remote Sensing and Digital Earth of Chinese Academy of Sciences, Beijing, PRC 100101 c School of Resources and Environment, University of Electronic Science and Technology of China, Chengdu, Sichuan, PRC 611731 d Institute of Remote Sensing and Digital Earth of Chinese Academy of Sciences, Beijing, PRC, 100094 https://www.researchgate.net/publication/269330076_Vehicle_detection_from_parking_lot_aerial_images
6) https://en.wikipedia.org/wiki/Pearson_correlation_coefficient