CM30080: Computer Vision Report

Erik Philippe Leclercq (epl28)
Julian Jimenez Nimmo (jjn34)
John Steward (js3511)

1 Edge Detection

In order to be able to run the Hough Line Transform algorithm effectively, what had to be done first was to process each image using Canny edge detection. This was done using the "Canny" module in OpenCV.

Since edge detection algorithms are very sensitive to noise in an input image, we first apply a 5x5 Gaussian filter on the image to remove the noise. The smoothed image is then filtered using a Sobel kernel in both the horizontal and vertical directions to get the first derivates G_x and G_y for the horizontal and vertical directions respectively. From these two images, we can find the edge gradient and direction for each using equations 1 and 2:

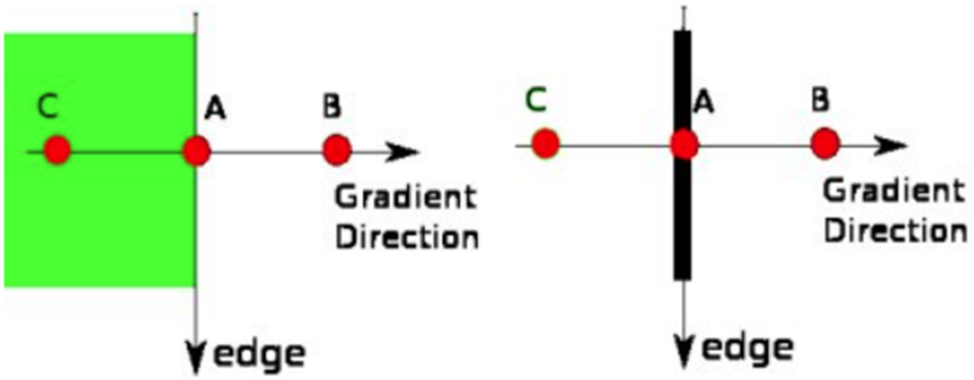
$$\textit{Edge Gradient}(G) = \sqrt{G_x^2 + G_y^2} \tag{1}$$

$$\textit{Angle}(\theta) = \tan^{-1}(\frac{G_y}{G_x}) \tag{2}$$

The gradient direction will always be perpendicular to the edges. It is rounded to one of four angles representing horizontal, vertical and two diagonal directions.

We then use non-maximum suppression (figure 1) to remove any unwanted pixels. At every pixel in the image, it is checked if it is a local maximum in its neighbourhood in the direction of the gradient.

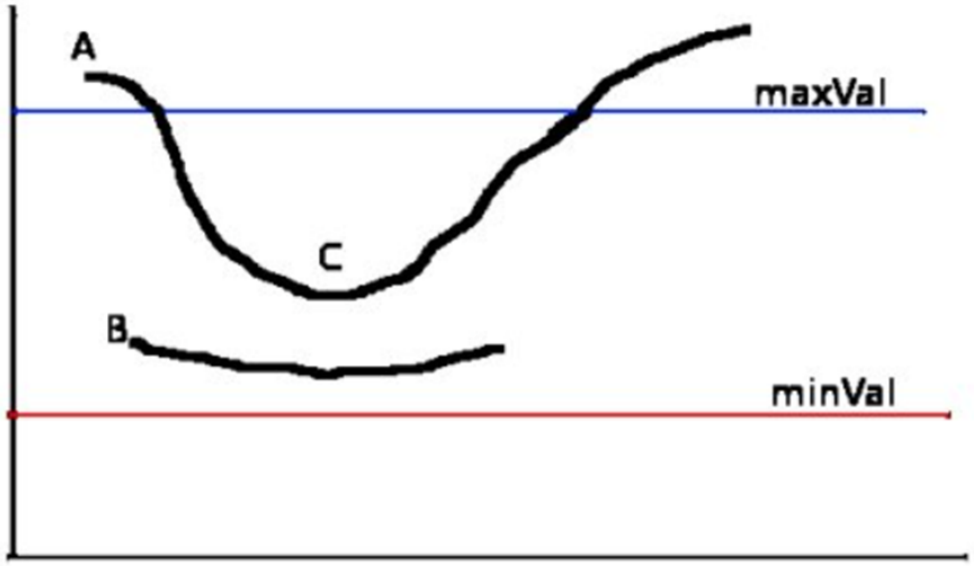
Figure 1: Diagram showing how non-maximum edge detection works, source: [1]



Point A is on the edge (which is vertical), with the gradient direction being perpendicular. Points B and C are in the direction of the gradient, so point A is checked along with B and C to see if it forms a local maximum. If so, it is considered for Hysteresis Thresholding. If not, it is suppressed.

Hysteresis Thresholding (figure 2) is used to detect which candidate edges that were detected during non-maximum suppression are true edges. This utilises two thresholds: minVal and maxVal. Any edges with a gradient larger than maxVal are kept, while any with a gradient smaller than minVal are discarded. For edges that lie between these thresholds, they are checked to see if they are connected to true edges. If they are, then they are kept. If not, they are discarded.

Figure 2: Diagram showing how hysteresis thresholding works, source: [1]



In figure 2, A and C are kept, and B is discarded.

The Canny function in OpenCV uses these steps and returns a greyscale image with only the kept edges.

Once the image has been run through the Canny detection, we can use the "HoughLines" function to detect the lines in the image.

The Hough Line Transform looks at a point (x, y) in the image, and plots it on a graph of r against θ. The family of lines that pass through (x, y) can be defined by the equation 3.

$$r_\theta = x \cos\theta + y \sin\theta \tag{3}$$

Using equation 3, we can plot the family of lines that pass through the point (x, y) on figure 4.

This only considers points where $r > 0$ and $0 < \theta < 2\pi$

This operation is done for all points in the image. If two lines on the graph intersect, then the two points plotted share a line at the intersection. The threshold that is passed into the function defines the number of intersections required on the graph in order for it to detect a line. If the threshold is reached, then the function will return the pair (θ, r), being the line on which all these points lie.

Figure 3: The three points plotted on this graph lie on the line $\theta = 0.925, r = 9.6$, source: [2]

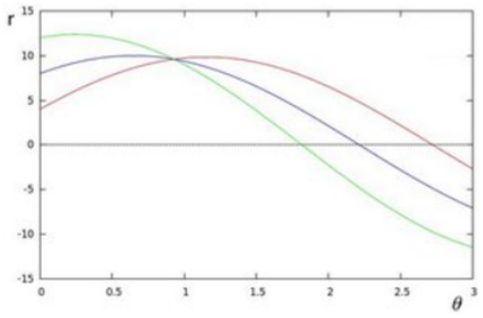
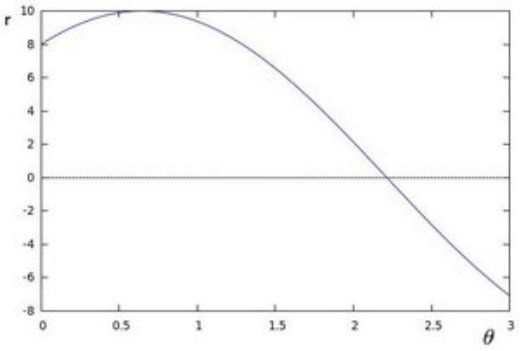


Figure 4: Family of lines that pass through point $(8, 6)$, source: [2]

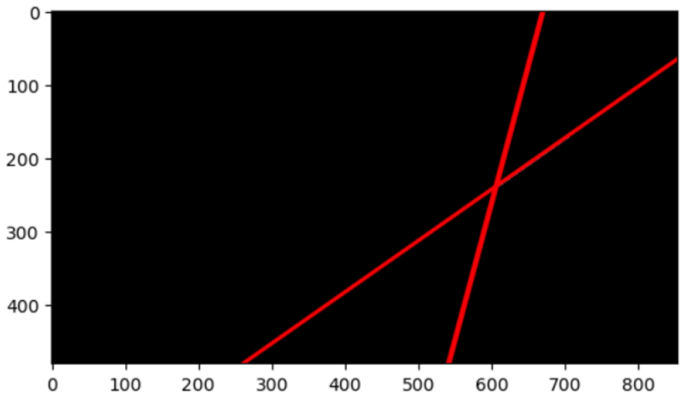


For example, the three points plotted on graph 3 lie on the line ($\theta = 0.925, r = 9.6$).

Since the HoughLines function returns the lines in polar form, we can store θ for the angle of the line. However, since it is measured in radians, we need to convert it to degrees before we calculate the angle between the two lines in the image.

Once we have found the lines in the image and the angles corresponding to them, we calculate the difference between the largest angle in the list and the smallest, storing the result as the angle between the lines. If the value of the largest angle is larger than 90 degrees, it means that the angle measured between the lines is the larger one (figure 5). Since we need to find the smaller angle between the lines, we simply need to store the value of $180 - (\textit{calculated angle})$.

Figure 5: Example image with detected edges



The runtime of this algorithm is around 3.5 seconds for the given dataset, and we achieve 100% accuracy.

The optimal hyperparameters found for the canny detection were a minVal of 150 and a maxVal of 255. To find the correct number of intersections necessary to detect a line, we iterate over a threshold of 1 to 150 intersections necessary until the HoughLines function returns 3 lines. This means that the defined threshold will be different for each test image, finding the optimal threshold for each one individually. Appendix A contains a table with additional results.

2 Template Matching

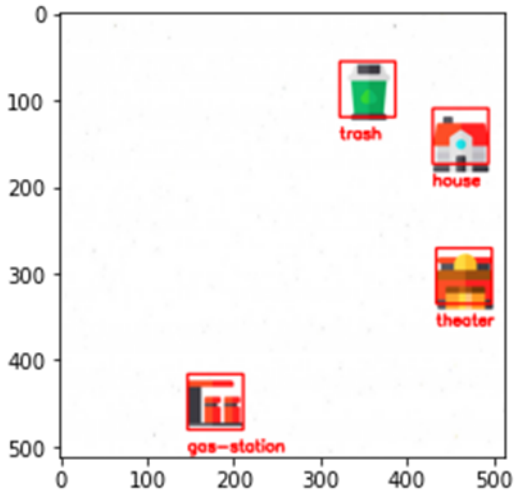
2.1 Approach

An intensity-based template matching approach using Gaussian Pyramids was adopted to complete template localisation and identification at different scales and rotations.

This correlation approach is suitable for finding templates in an image because when a template signal matches with itself it achieves a higher score than when it is matched with another different template signal. To handle intensity changes, the method chosen was to normalise both the template and test window by subtracting the mean and dividing by the standard deviation, and then applying cross correlation. We used OpenCV's 'matchTemplate' function with normalised cross correlation to perform intensity-based template matching, filtering out scores above a threshold. If detected, the

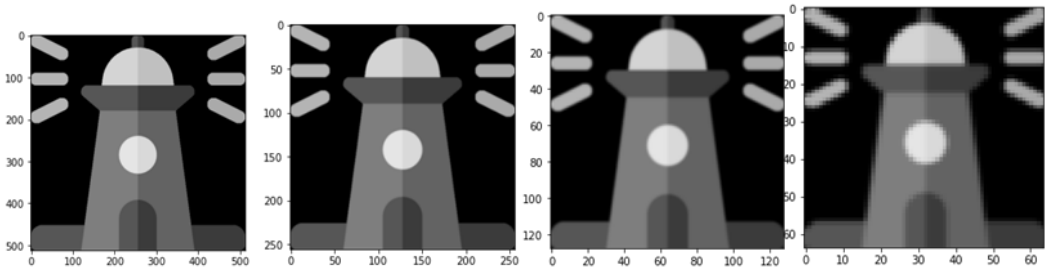
top-left corner of the bounding box is the predicted intensity-based matched pixel, and the bottom-right corner is the addition of the top-left corner x and y coordinates with the width and length of the template, respectively. An example of the output can be seen in figure 6.

Figure 6: Output detection for test image 1



A Gaussian pyramid is created for each template to handle different scales, where each scale is a subsampled and Gaussian blurred image of the previous scale. To perform this, OpenCV's 'pyrDown' function applies a Gaussian blur and downsamples the image. For each scale, the template is rotated by a fixed number of angles to handle rotations in the test set.

Figure 7: Scaled lighthouse. Note the size reduction and increased blur



For both the training templates and test images, the background is set to black with a threshold limit and the images are converted to greyscale to ensure they are in similar value and intensity ranges. The next step is to implement an exhaustive fine-to-coarse scale search strategy, iterating each tem-

plate through each test image with every scale and rotation. Non-Maximum Suppression (NMS) is required as this approach generates many predictions. NMS eliminates overlapping bounding boxes of the same class, preserving the predictions with the highest confidence score. Intersection over Union (IoU) is adopted to measure the overlap of bounding boxes. NMS is used in a template prediction firstly after a rotation, secondly after a scale, and finally after all scales.

2.2 Performance

The approach on the test set without rotations is robust, achieving a best accuracy score of 0.945 with an average runtime of 40-45s on the chosen optimal parameters tested on our designed evaluation method. True Positive Rate (TPR), False Positive Rate (FPR) and accuracy (ACC) measurements were also provided per class in the appendix and can be calculated following the formulas 4, 5, 6.

$$TPR = \frac{TP}{TP + FN} \tag{4}$$

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{6}$$

where

- $TP = True\ Positives$
- $TN = True\ Negatives$
- $FP = False\ Positives$
- $FN = False\ Negatives$

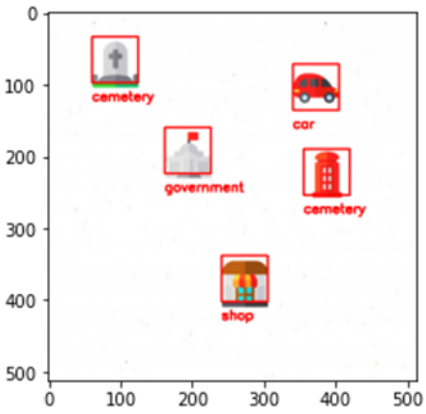
Low TPR values can be observed as there are less True Positives compared to False Negatives, as we treat evaluation per class as a binary classification problem, with the current chosen template as the positive case and all other classes as the negative case. In our task, this means 1 class vs 49 other classes during evaluation. In most classes, the FPR is also quite low, which demonstrates the method's high accuracy. Cases with higher FPR showcase higher recall over precision.

In terms of complexity, this is where this approach falters. For memory complexity, it stores every scale for a template at different rotations. In our

implementation, this consists of a memory complexity of $O(n^2)$ for each template, and for the whole training dataset a complexity of $O(n^3)$. For time complexity, inference with NMS has an estimated $O(n^7)$, as it has to iterate through each template, the scales for the template and the rotations at the scales and apply OpenCV's matchTemplate function. To achieve our optimal results, NMS is applied after the predictions of each rotation template, adding additional time complexity. NMS function has an estimated $O(n^2)$ complexity and 'matchTemplate' a upper bound of $O(n^2)$.

This approach found the following cases difficult to detect. Post-office was confused with supermarket in test image 11, university was confused with bank in test images 10 and 14, telephone-booth was confused with ATM in test image 6 and finally telephone-booth was confused with cemetery in test image 13, as can be seen in figure 8. These errors are reasonable, as templates such as ATM, telephone-booth and cemetery where the length is greater than the width have similar structure, while templates such as university and bank are similar in both square-like structure and architecture, with a pointed roof.

Figure 8: Example of misclassified detection between cemetery and telephone-booth



2.3 Parameters

In terms of parameters, we first find the hyperparameters to form our Gaussian Pyramid. For the number of scales, 4 different scales achieved the best results, with the last scale representing all our test detections. Performance dropped significantly when other scales were used, indicating the importance of this hyperparameter. Each scale downsampled our images by half, and the default Gaussian blur applied by OpenCV's pyrDown was used, as detailed

in equation 7.

$$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix} \tag{7}$$

For rotations, 90-degree rotations achieved the best results, with more orientations substantially increasing the False Positive and False Negative rates. Increasing either the number of stored rotations or the number of scales will also increase the runtime and memory substantially, which is one of the main weaknesses of this approach.

Adding an additional scale doubled the runtime while doubling the rotations, in other words, 45-degree rotation increments increase the runtime by 40s. Each additional scale or rotation significantly increases the required memory to store the templates. Figures 9 and 10 demonstrate the accuracy of the model using different numbers of rotations and scales. For all figures, optimal hyperparameters were used except explicitly stated changed parameters.

Figure 9: Accuracy of model using different stored rotations

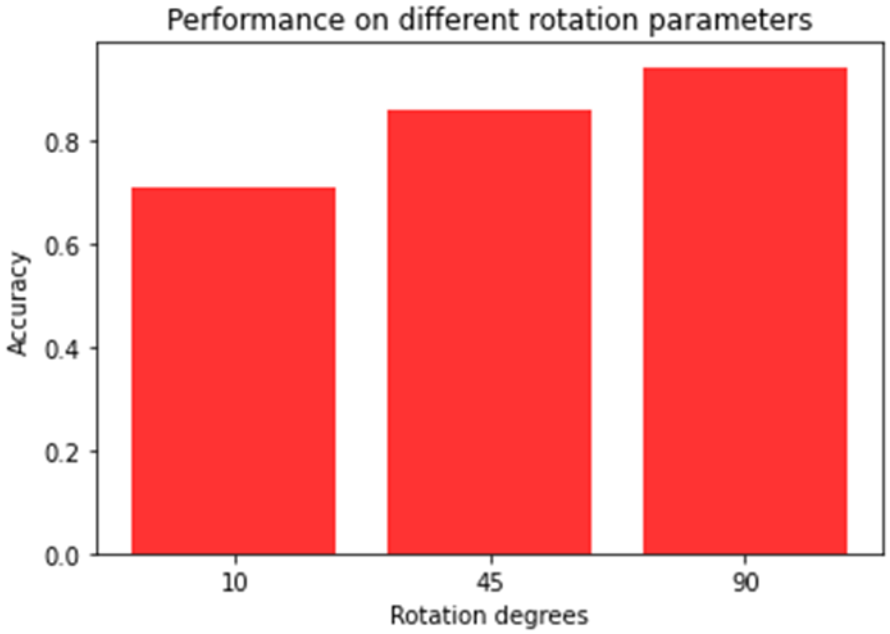
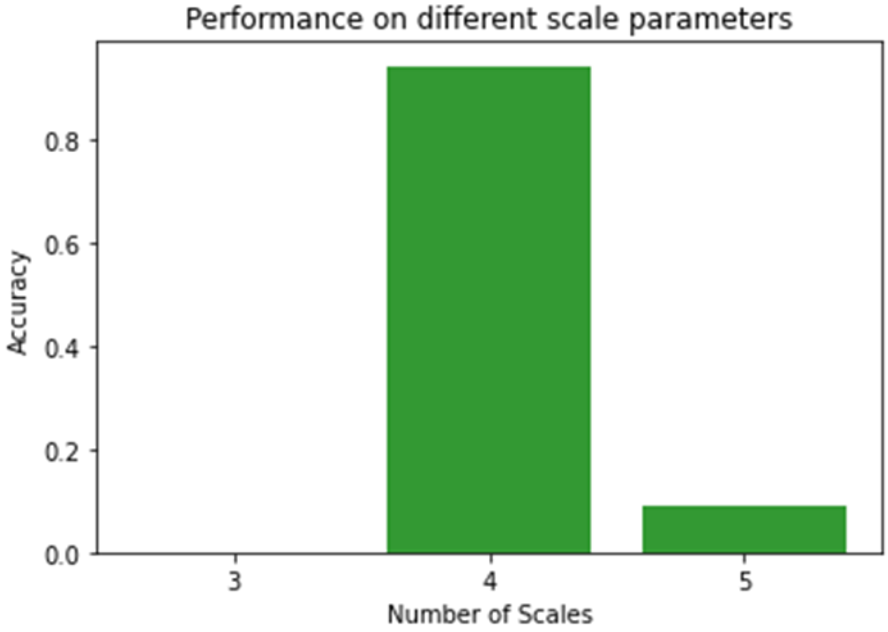


Figure 10: Accuracy of model using different scales for the Gaussian Pyramid



An additional hyperparameter used during pre-processing is the threshold to make the background black. Applying the current threshold of 250 produces an almost completely black background, with some loose white pixels as noise, as seen in Figure 11. An important hyperparameter is the score threshold of the Template matching function. The chosen 0.90 threshold retrieves the most confident predictions, with higher or lower thresholds decreasing the accuracy drastically, as presented in Figure 12.

Figure 11: Test image 12 with white noise pixels

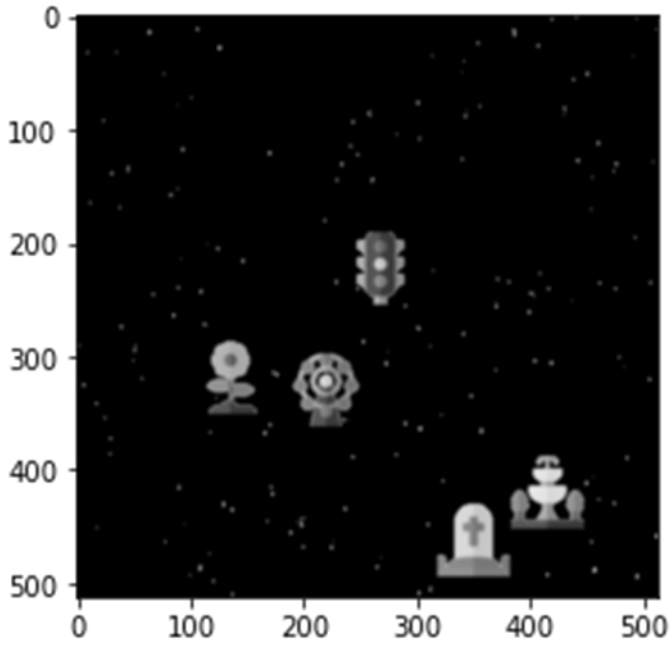
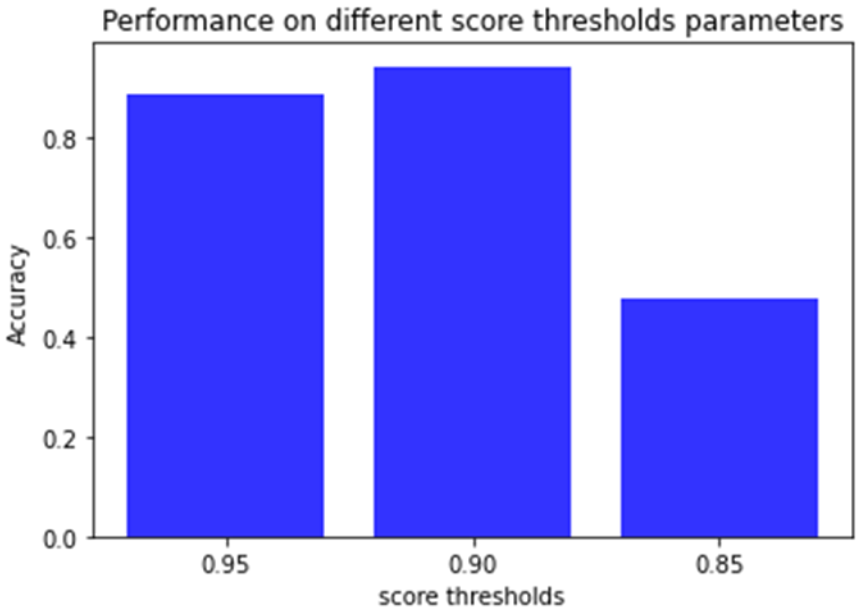


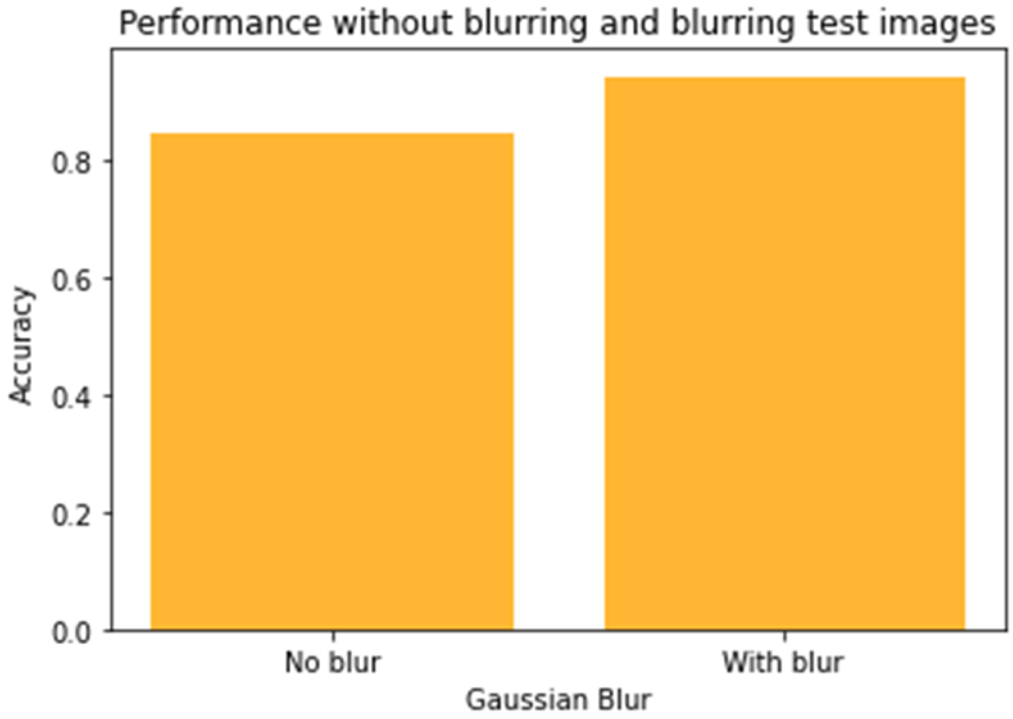
Figure 12: Accuracy of model using different score thresholds



The final hyperparameter is the IoU for NMS. A default value of 0.5 is chosen

as it provides robust performance. Note that the parameters will need to be adjusted for test images with more rotations and scales, especially threshold, rotations, and scale as these are the most impactful parameters on the performance of the model. Using the optimal parameters on the test set without rotations, it achieved a low accuracy of 0.06 on the test set with rotations. This showcases the sensitivity of Template Matching to different scales and rotations when using suboptimal parameters. Additionally, applying a Gaussian blur to the test images, improved detection performance, as seen in Figure 13.

Figure 13: Accuracy of model without blurring and blurring test images (no rotations)



3 Scale Invariant Feature Transform

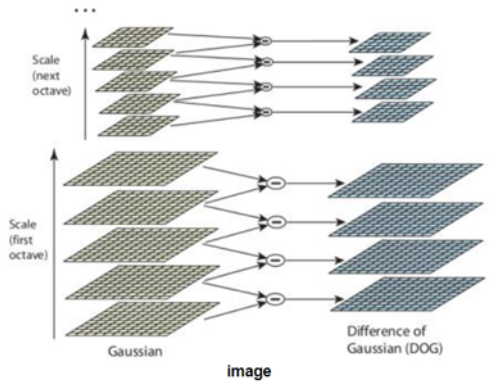
3.1 Algorithm

To extract the features from our training images, we used OpenCV's "detectAndCompute" function. This was done by using Scale-Invariant Feature Transform (SIFT).

In order to detect an image at different scales, space-filtering is used. Laplacian of Gaussian (LoG) is calculated for the image using various σ values, which acts as a blob detector for blobs of different sizes due to changes in σ. We can find the local maxima across the scale and space, giving us a list of (x, y, σ) values, which indicate a potential keypoint at point (x, y) at scale σ.

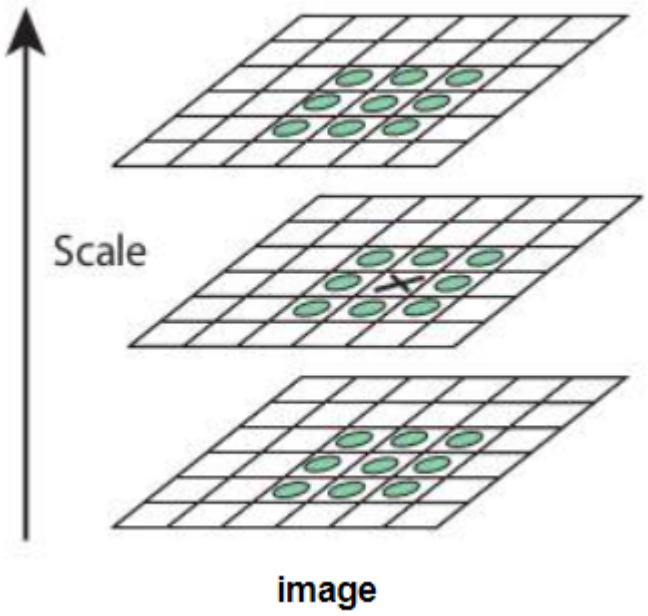
Since LoG is costly to compute, SIFT approximates it by using Difference of Gaussian (DoG). DoG is obtained by computing the difference of blurring an image at two different scales. This process is done for different octaves of the image in the Gaussian Pyramid. This process is illustrated in figure 14.

Figure 14: Difference of Gaussians is used to approximate Laplacian of Gaussian



Once this is done, we search for local extrema in both scale and space. This means a pixel is checked against its eight neighbours, as well as against 9 pixels in the next and previous scales. If it is a local extrema, then it is stored as a potential keypoint. This is illustrated in figure 15.

Figure 15: Search for local extrema



Once the potential keypoints have been found, they need to be refined. A Taylor Series expansion of scale space is used to obtain a more accurate location of extrema, and if the intensity at this extrema is lower than the threshold, it is discarded. Since DoG has a higher response for edges, they need to be removed.

After the keypoints have been refined, an orientation is assigned to each of them to achieve invariance to rotation. A neighbourhood is taken around the keypoint depending on the scale, and the gradient magnitude is calculated in that region, creating an orientation histogram with 36 bins covering 360 degrees.

A 16x16 neighbourhood around each keypoint is then taken, split into 4x4 sub-blocks, and an 8 bin orientation histogram is created for each sub-block, together being represented as a vector to form a keypoint descriptor to be matched between images.

3.2 Approach

After finding the features in each of the templates, we iterated through the test images, using detectAndCompute to find all the features within the test

image. We use a brute force matcher to match the current template to the test image, finding all matching features. This matching will include a lot of False Positives (see figure 16), therefore we filter it out by using the OpenCV's estimateAffinePartial2D which uses RANSAC under the hood to filter outliers. If it finds enough matches (10 in our case), it means that a match has been found. This function returns a transformation matrix that is then applied to the found matches. The new points are passed to the "boundingRect" function to create the bounding box. The result of this approach is shown in figure 17.

Figure 16: All matches found after applying SIFT

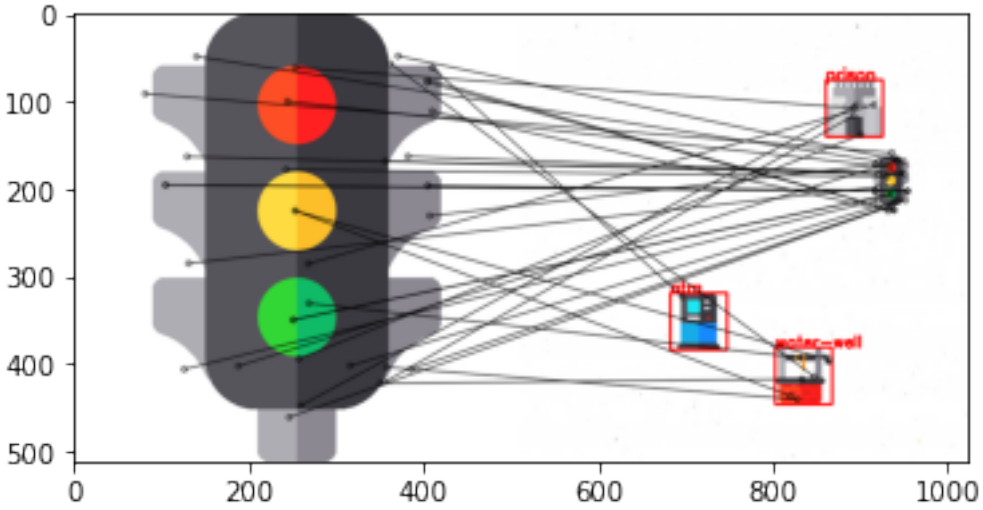
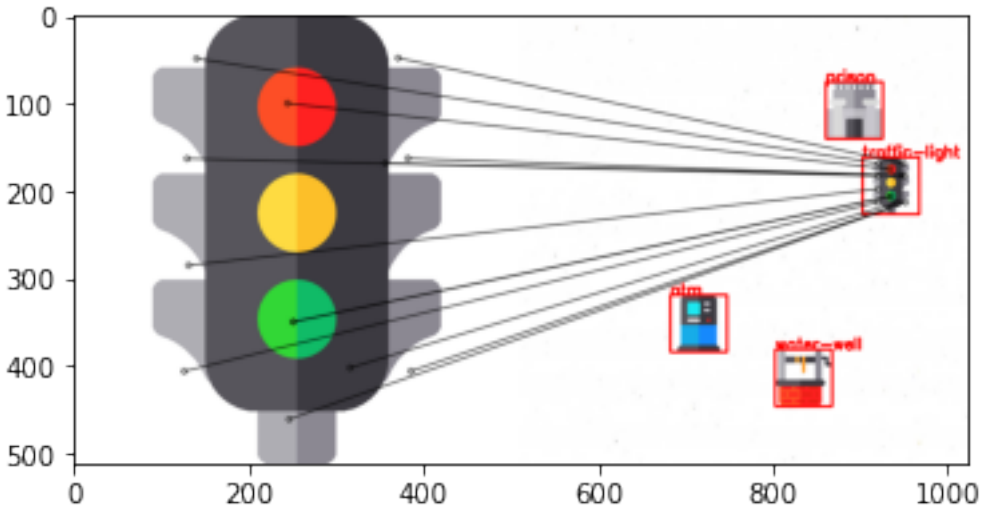


Figure 17: An example of a successful match after performing outlier filtering

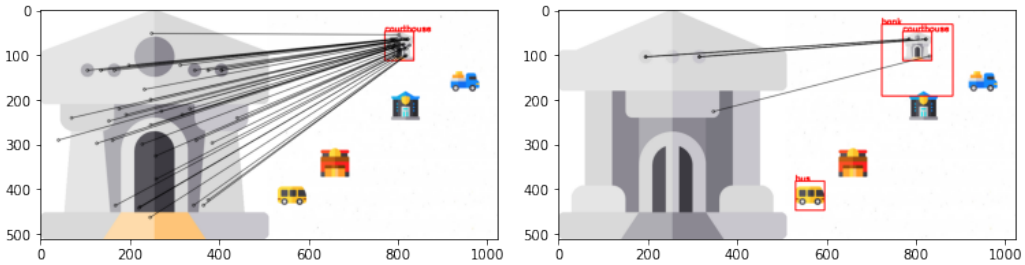


3.3 Accuracy and Comparison

Our implementation of the SIFT algorithm achieves an accuracy of 94.1% on the dataset "TestWithRotations", with the average runtime being around 9 seconds. This is around a fifth the runtime of the task 2 implementation. The reason SIFT performs much quicker than template matching is because, the time complexity of SIFT is $O(n^2)$ as opposed to $O(n^7)$ in task 2.

One of the main cases where our implementation struggles to perform correctly is when the templates look very similar to each other, for example with the bank and the courthouse. Since these templates look similar, they will naturally have similar features, meaning it is more likely for the SIFT algorithm to detect the incorrect icon in the test images. This is shown in figure 18.

Figure 18: An example where our SIFT implementation fails to differentiate between bank and courthouse. Only courthouse should have been detected.



Another case where the algorithm did not perform correctly was when it was detecting a vehicle. For example, in figure 19, the algorithm detected a car when it should have detected a van. This was because it found enough matches in the wheels, as both templates have similar wheels to each other.

A final case where our implementation occasionally struggled was when looking for matches on the very small icons within a test image, shown in figure 20. This is because the image has been scaled down too far for the algorithm to be able to detect enough matches between the icon and the template. This can be remedied with further hyperparameter tuning.

Figure 19: Example where our SIFT implementation matches the wheels of the template to a wrong icon

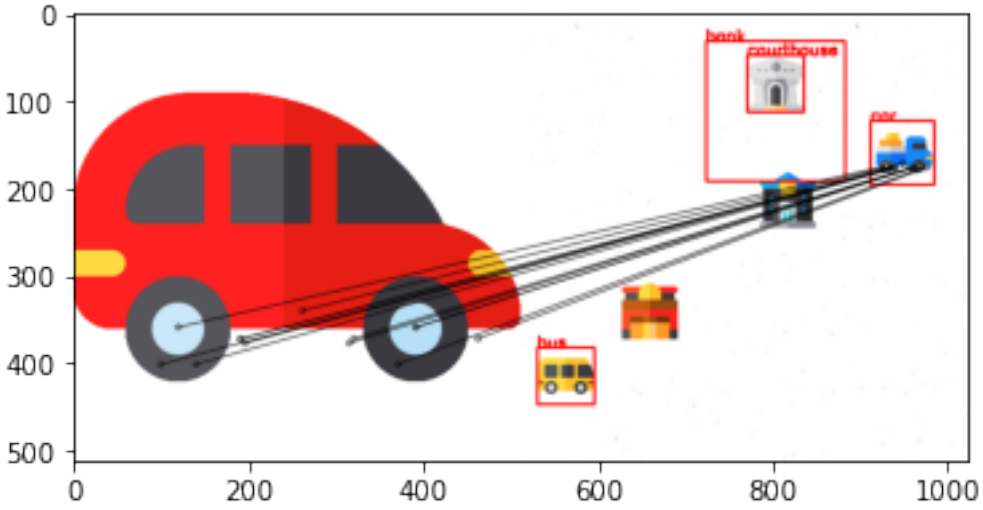
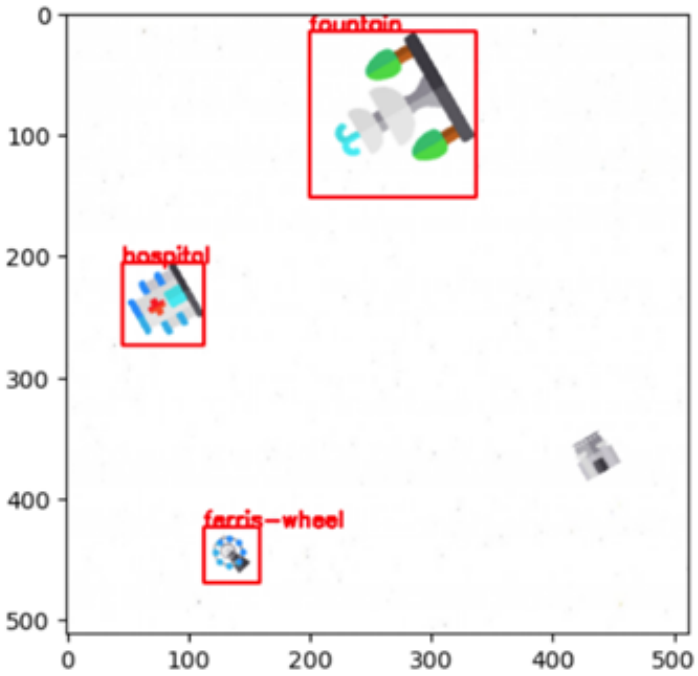


Figure 20: Our SIFT implementation sometimes fails to match small templates



3.4 Hyperparameters

The hyperparameters that we can tune for SIFT are given in table 1. Using the default parameters of SIFT gives us 0.87 accuracy on the non-rotated test set and and 0.74 accuracy on the rotated dataset. Using our tuned hyperparameters, we get 0.95 accuracy for the non-rotated test set and 0.941 on the rotated test set. The tuning of the hyperparameters was done manually by choosing different hyperparameters.

Table 1: Hyperparameters of SIFT

(a) Default Hyperparameters

Name	Values
nFeatures	0
nOctaveLayers	3
contrastThreshold	0.04
edgeThreshold	10
sigma	1.6

(b) Tuned Hyperparameters

Name	Values
nFeatures	0
nOctaveLayers	8
contrastThreshold	0.02
edgeThreshold	20
sigma	1.6

References

[1] OpenCV. https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html.

[2] OpenCV. https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html.

Appendix A: Hyperparameters for Task 1

Table 2 shows different hyperparameters we tried and their achieved accuracy. Changing canny parameters means specifically changing the minimum and maximum values of the hysteresis thresholding defined in section 1. We set minimum to 50 and maximum to 155 and it still achieved accuracy of 1.

Table 2: Different hyperparameters and their results

Changes	Accuracy
Detecting 2 lines	0.6
Detecting 3 lines	1.0
Detecting 4 lines	1.0
Changing canny parameters	1.0

Appendix B: Results for Task 2

Table 3: TPR, FPR and Accuracy for the non-rotated test set

Class Name	TPR	FPR	ACC
Lighthouse	0.0	0.0	0.94
Bike	0.375	0.0	0.94
Bridge-1	0.016	0.0	0.94
Bridge	0.44	0.0	0.94
Silo	0.16	0.0	0.94
Church	0.16	0.0	0.94
Supermarket	0.33	0.01	0.94
Courthouse	0.29	0.0	0.94
Airport	0.38	0.0	0.94
Bench	0.0	0.0	0.94
Trash	0.17	0.0	0.94
Bus	0.17	0.0	0.94
Water well	0.29	0.0	0.94
Flower	0.29	0.0	0.94
Barn	0.0	0.0	0.94
House	0.29	0.0	0.94
Cinema	0.17	0.0	0.94
Bank	0.4	0.02	0.94
Prison	0.17	0.0	0.94
ATM	0.5	0.01	0.94

Solar Panel	0.0	0.0	0.94
Car	0.44	0.0	0.94
Traffic Light	0.44	0.0	0.94
Fountain	0.38	0.0	0.94
Factory	0.17	0.0	0.94
Shop	0.17	0.0	0.94
Gas Station	0.38	0.0	0.94
Government	0.29	0.0	0.94
Theater	0.38	0.0	0.94
Telephone booth	0.17	0.0	0.94
Field	0.17	0.0	0.94
Van	0.29	0.0	0.94
Hydrant	0.0	0.0	0.94
Billboard	0.29	0.0	0.94
Police	0.5	0.0	0.94
Hotel	0.17	0.0	0.94
Post office	0.29	0.0	0.94
Library	0.17	0.0	0.94
University	0.29	0.0	0.94
Bus stop	0.17	0.0	0.94
Windmill	0.17	0.0	0.94
Tractor	0.17	0.0	0.94
Sign	0.0	0.0	0.94
Ferris wheel	0.38	0.0	0.94
Museum	0.0	0.0	0.94
Fire station	0.44	0.0	0.94
Restaurant	0.0	0.0	0.94
Hospital	0.0	0.0	0.94
School	0.17	0.0	0.94
Cemetery	0.5	0.01	0.94

Table 4: TPR, FPR and Accuracy for the rotated test set

Class Name	TPR	FPR	ACC
Lighthouse	0.0	0.14	0.06
Bike	0.0	0.08	0.06
Bridge-1	0.0	0.0	0.06
Bridge	0.0	0.08	0.06
Silo	0.0	0.2	0.06
Church	0.0	0.14	0.06

Supermarket	0.01	0.73	0.06
Courthouse	0.0	0.14	0.06
Airport	0.0	0.0	0.06
Bench	0.0	0.14	0.06
Trash	0.0	0.14	0.06
Bus	0.0	0.08	0.06
Water well	0.01	0.08	0.06
Flower	0.0	0.2	0.06
Barn	0.0	0.14	0.06
House	0.0	0.29	0.06
Cinema	0.0	0.08	0.06
Bank	0.0	0.25	0.06
Prison	0.0	0.57	0.06
ATM	0.01	0.08	0.06
Solar Panel	0.0	0.2	0.06
Car	0.0	0.14	0.06
Traffic Light	0.0	0.0	0.06
Fountain	0.0	0.14	0.06
Factory	0.01	0.0	0.06
Shop	0.01	0.08	0.06
Gas Station	0.0	0.2	0.06
Government	0.0	0.7	0.06
Theater	0.0	0.64	0.06
Telephone booth	0.0	0.2	0.06
Field	0.1	0.5	0.06
Van	0.0	0.0	0.06
Hydrant	0.0	0.0	0.06
Billboard	0.0	0.2	0.06
Police	0.0	0.14	0.06
Hotel	0.0	0.14	0.06
Post office	0.01	0.17	0.06
Library	0.0	0.0	0.06
University	0.0	0.14	0.06
Bus stop	0.01	0.23	0.06
Windmill	0.0	0.2	0.06
Tractor	0.01	0.0	0.06
Sign	0.0	0.0	0.06
Ferris wheel	0.0	0.08	0.06
Museum	0.0	0.08	0.06
Fire station	0.0	0.29	0.06
Restaurant	0.0	0.0	0.06

Hospital	0.0	0.14	0.06
School	0.0	0.14	0.06
Cemetery	0.01	0.08	0.06

Appendix C: Results for Task 3

Table 5: TPR, FPR and Accuracy for the non-rotated test set

Class Name	TPR	FPR	ACC
Lighthouse	0.0	0.0	0.96
Bike	0.43	0.0	0.96
Bridge-1	0.2	0.0	0.96
Bridge	0.5	0.0	0.96
Silo	0.2	0.0	0.96
Church	0.2	0.0	0.96
Supermarket	0.33	0.0	0.96
Courthouse	0.33	0.0	0.96
Airport	0.43	0.0	0.96
Bench	0.0	0.0	0.96
Trash	0.2	0.0	0.96
Bus	0.2	0.0	0.96
Water well	0.33	0.0	0.96
Flower	0.33	0.0	0.96
Barn	0.0	0.0	0.96
House	0.33	0.0	0.96
Cinema	0.2	0.0	0.96
Bank	0.5	0.02	0.96
Prison	0.2	0.0	0.96
ATM	0.5	0.0	0.96
Solar Panel	0.0	0.0	0.96
Car	0.67	0.02	0.96
Traffic Light	0.5	0.0	0.96
Fountain	0.43	0.0	0.96
Factory	0.2	0.0	0.96
Shop	0.2	0.0	0.96
Gas Station	0.43	0.0	0.96
Government	0.33	0.0	0.96
Theater	0.43	0.0	0.96
Telephone booth	0.43	0.0	0.96
Field	0.2	0.0	0.96

Van	0.33	0.0	0.96
Hydrant	0.0	0.0	0.96
Billboard	0.33	0.0	0.96
Police	0.56	0.0	0.96
Hotel	0.2	0.0	0.96
Post office	0.43	0.0	0.96
Library	0.2	0.0	0.96
University	0.5	0.0	0.96
Bus stop	0.2	0.0	0.96
Windmill	0.2	0.0	0.96
Tractor	0.2	0.0	0.96
Sign	0.0	0.0	0.96
Ferris wheel	0.43	0.0	0.96
Museum	0.0	0.0	0.96
Fire station	0.5	0.0	0.96
Restaurant	0.0	0.0	0.96
Hospital	0.0	0.0	0.96
School	0.2	0.0	0.96
Cemetery	0.5	0.0	0.96

Table 6: TPR, FPR and Accuracy for the rotated test set

Class Name	TPR	FPR	ACC
Lighthouse	0.29	0.0	0.94
Bike	0.17	0.0	0.94
Bridge-1	0.0	0.0	0.94
Bridge	0.17	0.0	0.94
Silo	0.33	0.01	0.94
Church	0.17	0.0	0.94
Supermarket	0.29	0.0	0.94
Courthouse	0.29	0.0	0.94
Airport	0.0	0.0	0.94
Bench	0.29	0.0	0.94
Trash	0.2	0.01	0.94
Bus	0.17	0.0	0.94
Water well	0.29	0.0	0.94
Flower	0.38	0.0	0.94
Barn	0.2	0.01	0.94
House	0.38	0.0	0.94
Cinema	0.17	0.0	0.94

Bank	0.44	0.0	0.94
Prison	0.2	0.01	0.94
ATM	0.29	0.0	0.94
Solar Panel	0.38	0.0	0.94
Car	0.29	0.0	0.94
Traffic Light	0.0	0.0	0.94
Fountain	0.2	0.01	0.94
Factory	0.17	0.0	0.94
Shop	0.29	0.0	0.94
Gas Station	0.38	0.0	0.94
Government	0.17	0.0	0.94
Theater	0.0	0.0	0.94
Telephone booth	0.38	0.0	0.94
Field	0.38	0.0	0.94
Van	0.0	0.0	0.94
Hydrant	0.0	0.0	0.94
Billboard	0.17	0.0	0.94
Police	0.29	0.0	0.94
Hotel	0.17	0.0	0.94
Post office	0.38	0.0	0.94
Library	0.0	0.0	0.94
University	0.29	0.0	0.94
Bus stop	0.5	0.0	0.94
Windmill	0.38	0.0	0.94
Tractor	0.17	0.0	0.94
Sign	0.0	0.0	0.94
Ferris wheel	0.17	0.0	0.94
Museum	0.17	0.0	0.94
Fire station	0.5	0.0	0.94
Restaurant	0.0	0.0	0.94
Hospital	0.29	0.0	0.94
School	0.16	0.0	0.94
Cemetery	0.29	0.0	0.94