SRH University Heidelberg

Task 3:
To read an object represented in Image A and detects its occurrences in Image B, by applying the Sliding Window Technique.

Author:	Nikhil Kumar
Matriculation Number:	11037894
Email Address:	nikhil.kumar@stud.hochschule-heidelberg.de
Date of Submission:	26th February 2024

Under the Guidance of Professor
Dr. Ing Milan Gnjatovic

Aim of the Task:

To read an object represented in Image A and detects its occurrences in Image B, by applying the Sliding Window Technique. The input parameters are: two grayscale images, number of chain code directions, grid dimensions M and N, sliding window dimensions window width and window height ,sliding window steps along the x- and y-axis, and threshold value.

Objective:

The objective of this task is to develop an algorithm that reads an object represented in Image A and detects its occurrences in Image B using the sliding window technique. The goal is to accurately identify instances of the object depicted in Image A within Image B by systematically scanning and comparing regions using the sliding window approach. This objective encompasses designing a robust and efficient algorithm capable of handling variations in object appearance, scale, orientation, and background noise to achieve reliable object detection across different scenarios.

Introduction

Image detection is a pivotal aspect of computer vision, tasked with automatically identifying and localizing objects or patterns within digital images. This essential field finds application across diverse domains, including autonomous vehicles, surveillance systems, medical imaging, and facial recognition technologies. At its core, image detection involves employing algorithms and techniques to discern specific objects' presence in an image, irrespective of factors such as orientation, scale, or background clutter. Leveraging methodologies like machine learning, particularly deep learning models such as convolutional neural networks (CNNs), image detection algorithms analyze visual data to accurately classify and localize objects, enabling machines to interpret and comprehend visual information autonomously.

Through the integration of advanced algorithms and techniques, image detection enhances the capabilities of automated systems to perceive and interpret visual data, laying the groundwork for innovative solutions across numerous fields. As technology continues to evolve, the refinement of image detection algorithms remains crucial, driving advancements in accuracy, efficiency, and scalability. By harnessing the power of computer vision, image detection empowers systems to navigate complex visual environments, facilitate object recognition, and contribute to the development of intelligent, perception-driven applications.

What is the Purpose of Object Detection?

The primary purpose of object detection in image processing is to automatically identify and localize specific objects or patterns within digital images. This process plays a crucial role in various applications, including:

- **Automation:** Object detection enables machines to autonomously analyse and interpret visual data, facilitating automation in tasks such as quality control in manufacturing, object tracking in surveillance systems, and inventory management in retail.

- **Enhanced Understanding:** By accurately identifying objects within images, object detection enhances machines' understanding of their surroundings, allowing them to make informed decisions based on visual cues. This capability is essential in applications like autonomous vehicles, where detecting pedestrians, vehicles, and traffic signs is critical for safe navigation.

- **Efficiency:** Object detection streamlines processes by automating tasks that would otherwise require human intervention. This leads to increased efficiency and productivity in various industries, such as agriculture (detecting crops and pests), healthcare (diagnosing medical conditions from medical images), and security (identifying suspicious activities in surveillance footage).

- **Augmented Reality:** In augmented reality (AR) applications, object detection enables virtual objects to interact seamlessly with the real-world environment by accurately detecting and tracking physical objects.

Overall, the main purpose of object detection in image processing is to empower machines with the ability to recognize and understand the visual world, enabling a wide range of practical applications across industries and domains.

What is Object Detection using Sliding Window Technique?

Image detection using the sliding window technique is a fundamental approach in computer vision and image processing for locating objects within larger images. This technique involves systematically scanning through an image at various positions and scales using a fixed-size window, often referred to as a "sliding window." By sliding this window across the image with defined steps along the x- and y-axes, different regions of the image are inspected. At each position, the content within the window is analyzed and compared against a reference object or pattern of interest. The sliding window technique is particularly valuable in scenarios where the object's location, size, or orientation may vary within the image. This method allows for the detection of objects at multiple scales and locations, making it widely applicable in tasks such as object recognition, face detection, and image classification. The effectiveness of image detection using sliding windows depends on careful parameter tuning, selection of appropriate feature descriptors, and robust algorithms capable of handling variations in object appearance and background clutter. Overall, this technique serves as a cornerstone in the development of advanced computer vision systems for real-world applications.

What is Sliding Window Technique?

The sliding window technique is a method for iterating over a sequence of data, typically used in the context of and image processing and computer vision for object detection and localization. It involves systematically scanning an image with a rectangular window of fixed size, moving it across the image in a grid-like fashion.[1a]

Here's how the sliding window technique works:

- **Window Initialization:** Begin by defining a window of a predetermined size and aspect ratio. This window serves as a local region of interest within the image.

- **Grid Scanning:** Starting from the top-left corner of the image, the window is moved horizontally and vertically across the image, one step at a time. At each position, the contents within the window are extracted for further analysis.

- **Feature Extraction:** Once the window is positioned over a region of the image, relevant features are extracted from the contents within the window. These features could include colour histograms, texture descriptors, or deep learning features extracted by a pre-trained neural network.

- **Object Detection:** The extracted features are then fed into a classifier or a machine learning model trained to recognize objects of interest. The classifier determines whether the contents within the window match the characteristics of the object being detected.

- **Sliding Process:** The sliding window process continues iteratively, moving the window across the entire image, typically with a small step size, until the entire image has been scanned.

- **Post-processing:** Once potential object detections have been made at various positions within the image, post-processing steps such as non-maximum suppression may be applied to refine the detections and eliminate duplicate or overlapping detections.

The sliding window technique allows for the detection of objects at different scales and positions within an image, making it a versatile tool for object detection tasks. However, it can be computationally expensive, especially when using large window sizes or scanning high-resolution images. Consequently, optimization techniques such as image pyramid representations and convolutional implementations are often employed to improve efficiency.

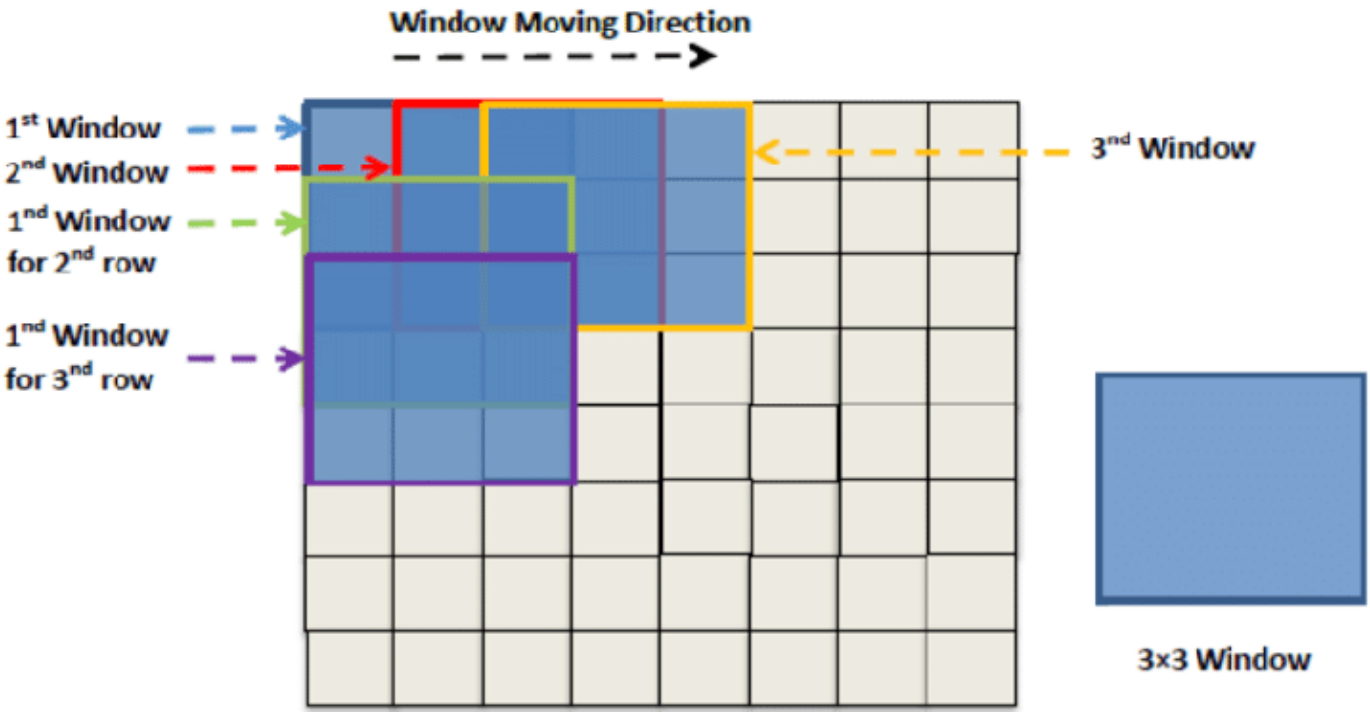


Figure 1 : The sliding window technique is illustrated in the example image above.

What is Histogram of Oriented Gradients?

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image.[1b]

What is the reason for selecting Histogram of Oriented Gradients for image detection?

The Histogram of Oriented Gradients (HOG) feature descriptor is often selected for image detection tasks due to several key reasons:

- **Robustness to Illumination Changes:** HOG features are designed to be robust to variations in illumination within an image. This robustness ensures that the extracted features remain effective even when lighting conditions change, making HOG suitable for real-world applications where lighting may vary.

- **Translation and Rotation Invariance:** HOG features are invariant to translation and rotation of objects within an image. This means that the same object detected at different positions or orientations in the image will produce similar HOG feature representations, facilitating accurate object detection across various scenarios.

- **Effective Representation of Local Image Structure:** HOG captures the local gradient information of an image, providing a concise representation of local image structure. This enables HOG to effectively capture the shape and appearance of objects, making

it well-suited for tasks such as pedestrian detection, where local edge and gradient information is crucial.

- **Computational Efficiency:** Despite its effectiveness, HOG features can be computed relatively efficiently, especially compared to more computationally expensive feature descriptors. This makes HOG suitable for real-time or resource-constrained applications where computational resources are limited.

- **Proven Performance:** HOG has been widely used and extensively studied in the field of computer vision, particularly in object detection tasks. Its effectiveness has been demonstrated across various datasets and applications, making it a trusted choice for image detection tasks.

Working and Procedure steps to read an object represented in Image A and detects its occurrences in Image B:

Step 1 : Converting the input Colour into Gray Scale Image

Grayscale conversion of an image involves transforming it from a multi-color representation (e.g., RGB) to a single-channel representation, where each pixel's intensity corresponds to its brightness level. Grayscale images lack colour information but retain luminance, making them suitable for various image processing tasks.

For simplification of Image, Grayscaled images are simpler than color images, reducing computational complexity and memory requirements for image processing tasks.

To Focus on Luminance of Image, By removing colour distractions, grayscale images emphasize luminance variations, which are often critical for tasks like edge detection, feature extraction, and image segmentation. .

Step 2 : Finding HOG of an image using Sobel filter, Gradient Vector, Magnitude and orientation angle (Theta)

The following procedures are taken once the image has been converted to grayscale:

- **Gradient Computation:**
  Apply the Sobel filter to the grayscale image to compute the gradients in the horizontal (Gx) and vertical (Gy) directions. This can be done by convolving the image with the horizontal and vertical Sobel kernels.

  The Sobel Filter serves the purpose of edge detection, operating by computing the gradient of the image intensity for each pixel. It utilizes a specific formula to compute both the vertical edges (Gx) and horizontal edges (Gy).

Vertical Edge:
Gx ={{-1, 0, 1},
{-2, 0, 2},
{-1, 0, 1}};

Horizontal Edge:
Gy = {{-1,-2,-1}
0, 0, 0}
{1, 2, 1}}

Then Compute the gradient magnitude for each pixel using the formula:
gradient magnitude = sqrt(gx * gx + gy * gy);

Calculate the gradient orientation angle (Theta) for each pixel using the formula:

$$\theta(x,y) = \begin{cases} \arctan\dfrac{g_y(x,y)}{g_x(x,y)}, & \text{if } g_x(x,y) > 0\,, \\ \arctan\dfrac{g_y(x,y)}{g_x(x,y)} + \pi, & \text{if } g_x(x,y) < 0\,, \\ \frac{\pi}{2} \cdot sgn(g_y(x,y)), & \text{if } g_x(x,y) = 0\,. \end{cases}$$

- **Quantization of Gradient Orientation:**

Divide the range of gradient orientations (usually from 0 to 2*PI) into a predefined number of bins or histogram cells. Common choices are 8 or 9 bins, corresponding to angles in 45 or 40 degrees increments.
Assign each pixel's gradient orientation to the nearest bin. This step quantizes the orientations, making them suitable for histogram computation.

Formula : Index = (theta * n) / (2 * PI)

- **Histogram Computation:**

Divide the image into cells of a fixed size (e.g., 8x8 pixels).
For each cell, accumulate gradient magnitudes into orientation histograms based on the quantized gradient orientations of the pixels within the cell. Each histogram bin corresponds to a specific orientation range.
Optionally, apply spatial weighting or normalization to the histograms within each block to improve robustness against lighting variations and contrast differences.

- **Block Normalization:**

Normalizing of HOG (Histogram of Oriented Gradients) refers to a process where the computed HOG descriptors are scaled to make them less sensitive to variations in illumination and contrast. This normalization step helps in improving the robustness of the HOG feature representation for object detection and recognition tasks.

- **Calculating the Cosine Similarity:**
Cosine similarity works by comparing the angle between two vectors in a high-dimensional space. The angle between two vectors is calculated based on the dot product of the two vectors and their magnitudes.
Let's consider two vectors A and B in a 2-dimensional space as an example.

The angle between these two vectors can be calculated using the dot product formula:
A . B = ||A|| ||B|| cos(theta) .
where A. B is the dot product of vectors A and B, ||A|| and ||B|| are the magnitudes of vectors A and B, respectively, and theta is the angle between the two vectors.
Rearranging the above equation, we get:
cos(theta) = (A . B) / (||A|| ||B||)
The cosine similarity between two vectors A and B can be calculated by taking the cosine of the angle between them.
cosine similarity(A, B) = cos(theta) = (A . B) / (||A|| ||B||)

Step 3 : Image Detection using Sliding Window Technique

Sliding Window Approach:
The sliding window technique involves dividing an image into smaller regions by using a window of fixed dimensions. This window is then moved across the image in a systematic manner, typically from left to right and top to bottom, with a predefined step size across X and Y axis. At each position, the contents within the window are analysed to determine whether they contain an object of interest.

Detection and Localization:
When an object is detected within a window, its presence is typically marked by drawing a bounding box around the region containing the object.
The coordinates of the bounding box provide information about the object's location and extent within the image.

Explaining the Program Functionality (What is going in the code):

As explained in the working, each step is defined as a function in the program and it is explained below. (Program File: IP_ProjectTask3.pde)

Grey Scale Function:

The grayscale function takes a single parameter, the input image. Within this function, an image named "img_G" (a placeholder or dummy image) is created using the **createImage** data type, and pixels are loaded for both the original image and the newly created image. Subsequently, a for loop is employed to iterate through the columns (y) and rows (x) of the image. The index is calculated for each pixel using the formula x + y * image width. RGB components of the image pixels are then extracted by their respective indexes using the predefined keywords red, green, and blue.

Next, the colour image is converted to a grayscale image using the luminosity method, which employs the formula: **0.21 * red channel pixels + 0.72 * green channel pixels + 0.07 * blue channel pixels.** This formula preserves the luminance information of the original image while discarding colour information, resulting in a grayscale representation. Finally, the pixels of the "img_G" image are updated with the calculated grayscale values, and the grayscale image is returned.

Computing the Histogram of Oriented Gradients:

- Image Loading and Initialization:

The function takes a **PImage** img as input along with parameters M, N, and n.
It initializes variables including **numpixcels**, representing the total number of cells in the image grid, and main_array, a 2D array to store the computed HOG features.
Two 3x3 matrices filter1 and filter2 are initialized, representing Sobel filters used for gradient computation.

- Pixel-wise Gradient Calculation:

Nested loops iterate over each pixel of the image, excluding the border pixels.
Within these loops, two inner loops traverse a 3x3 neighbourhood centred around each pixel.
For each pixel, the function calculates the gradients f1 and f2 by convolving the pixel neighbourhood with the Sobel filters.

- Gradient Vector Calculation:

Using the computed gradients f1 and f2, the function computes the gradient magnitude mag using the Euclidean distance formula.
It also computes the gradient direction theta using the atan2 function, ensuring the angle falls within the range [0, 2π].

Histogram Accumulation:

The gradient direction theta is mapped to a corresponding bin within the histogram.
The function calculates the index of the bin using the formula (theta * n / (TWO_PI)), discretizing the continuous range of angles into n bins.
The magnitude mag is then added to the appropriate bin in main_array, accumulating gradient magnitudes based on their directions.

Grid Cell Calculation:

The pixel's position within the grid of cells (M rows and N columns) is determined based on its coordinates.
The x_axis and y_axis values represent the grid cell indices for the current pixel.

Histogram Accumulation in Grid Cells:

The computed gradient magnitude mag is added to the corresponding histogram bin within the appropriate grid cell in **main_array[y_axis * gridSize + x_axis][indexarray] += mag.**
This process is repeated for each pixel in the image, resulting in a populated main_array representing HOG features.

Compute Co-Sine Similarity:

After computing the dot product and magnitudes, the cosine similarity is calculated using the **formula: dot product / (sqrt(mag1) * sqrt(mag2)).**

This formula computes the cosine of the angle between the two vectors, which is a measure of their similarity. If both vectors are identical, the cosine similarity will be 1.
If they are not identical, the cosine similarity will be 0.
If they are opposite, the cosine similarity will be -1.

To handle cases where one or both of the magnitudes are zero (which would result in division by zero), a check is performed. If either magnitude is zero, indicating one of the vectors has no magnitude (i.e., it's a zero vector), the function returns 0, indicating no similarity.

Object Detection:

The function object detection takes the following parameters:
img1: The first input image containing the object of interest.
img2: The second input image where the object will be detected.
n: Number of chain code directions.
M and N: Grid dimensions.
windowWidth and windowHeight: Dimensions of the sliding window.
stepX and stepY: Step sizes for moving the sliding window along the x and y axes.
threshold: Similarity threshold above which a detection is considered.

HOG Feature Extraction:

- The function first computes HOG features for img1 using the hog function, storing the result in **hogFeatures1**. HOG features are representations of local gradient orientations in an image.
- HOG features are also computed for each sliding window extracted from img2 using the same hog function, and stored in **hogFeaturesWindow**.

Sliding Window Iteration:

- The function iterates through img2 using a sliding window approach, moving the window with dimensions **windowWidth** and **windowHeight** with steps **stepX** and **stepY**.
- At each window position, a sub-image (window) is extracted from img2.

Cosine Similarity Calculation:
Cosine similarity between the HOG features of img1 and the current window (window) from img2 is calculated using the cosineSimilarity function. This measures how similar the gradient orientations in the two images are.
The computed similarity value is compared against the threshold.
If the cosine similarity exceeds the threshold, it is considered a detection, and a rectangle is drawn around the detected object on the results image using the **drawRectangle** function.

Draw Rectangle function:

The **drawRectangle** function takes an input image img along with the coordinates (x, y) of the top-left corner of the rectangle, as well as its width w and height h. It then draws a rectangle onto the image by modifying its pixel values.

Drawing Borders:

Two loops are used to draw the top and bottom borders of the rectangle:

- The first loop iterates over the width w of the rectangle, starting from x and ending at x + w.
- Within this loop, it sets the colour of the pixel at coordinates **[y * img.width + i]** to black (colour code 0), representing the top border.
- Similarly, it sets the colour of the pixel at coordinates **[(y + windowHeight) * img.width + i]** to black, representing the bottom border.

Another two loops are used to draw the left and right borders of the rectangle:

- This time, the loop iterates over the height h of the rectangle, starting from y and ending at y + h.
- Within this loop, it sets the colour of the pixel at coordinates **[i * img.width + x]** to black, representing the left border.
- It also sets the colour of the pixel at coordinates **[i * img.width + (x + windowWidth)]**to black, representing the right border.

After modifying the pixel values to draw the rectangle borders, the function calls updatePixels() to apply the changes to the image.

Applications:

The sliding window technique is widely used in various applications, including pedestrian detection in autonomous driving systems, face detection in surveillance systems, and object recognition in image retrieval systems.

Observations while developing detection program:

The successful detection of the object in the second image relies significantly on the adjustment of the following parameters.

Chain code direction(n) and grid dimensions(M and N) , window dimension(Width and height) window steps (X and Y) and threshold value.

Minor modifications to each parameter may be necessary when utilizing different images for detection.

Conclusion:

Object detection using the sliding window technique provides a robust and effective method for locating objects within images. By systematically scanning the image space and applying feature extraction and classification techniques, it enables the automated detection and localization of objects of interest, thereby facilitating a wide range of computer vision applications.

Output Result 1:

With Parameters - n = 8;N = 3, M = 2, window dimensions = 80, 110
window steps =16,16 ,threshold value = 0.85

```
HOG Of Image 1 After Normalization:
0.023731295 0.021730924 0.02949506 0.045880087 0.22899513 0.08253775 0.040793788 0.02614769
0.39247575 0.060657088 0.039410125 0.04897175 0.3272507 0.16874176 0.10212776 0.06966021
0.23673598 0.038577996 0.023068357 0.04593047 0.4743665 0.07048503 0.03318415 0.03883819
0.19729237 0.04038131 0.036391713 0.03576295 0.045430373 0.04630595 0.03450301 0.03049203
0.11234995 0.14261226 0.18007725 0.11870336 0.19204873 0.110696554 0.089951575 0.060374714
0.15957578 0.13335975 0.09943791 0.09869741 0.10135738 0.13308094 0.14212859 0.16791858

HOG Of Image 2 After Normalization:
0.015709314 0.24487764 0.15878417 0.01623429 0.01779501 0.24884115 0.13422094 0.0099062035
0.016069332 0.23410164 0.16867368 0.026867004 0.011702206 0.21418059 0.20414737 0.027019443
0.18611653 0.15920769 0.098852865 0.1640925 0.19930565 0.13293228 0.08970753 0.19085468
0.14747259 0.18197839 0.11491434 0.21734054 0.14542957 0.12483797 0.091687955 0.181926
0.02810995 0.18633458 0.10974766 0.036140133 0.021564988 0.24209222 0.17051227 0.05015516
0.039831143 0.17293344 0.10218064 0.02002254 0.05050528 0.24928266 0.14951885 0.017474316

The Co-Sine Similarity between the 2 images are:  0.8953141
```

Output Result 2:

With Parameters - n = 8;N = 3, M = 2, window dimensions = 80, 110
window steps =16,16 ,threshold value = 0.90

```
HOG Of Image 1 After Normalization:
0.024182029 0.3828621 0.09021465 0.0090738535 0.09487185 0.20951056 0.13202664 0.009372164
0.28191823 0.19416009 0.07209963 0.05655249 0.16358007 0.11191458 0.046523605 0.09768121
0.021402428 0.15399502 0.08470749 0.07183822 0.0913876 0.09443669 0.063155755 0.024316192
0.041666094 0.008120726 0.16995826 0.22502705 0.04927858 0.014275915 0.24479716 0.21530613
0.008023339 0.092756286 0.07642995 0.0129016265 0.062363602 0.25283045 0.19080766 0.013891226
0.25100845 0.2868141 0.010453357 0.011081511 0.21871224 0.19042177 0.0059812353 0.017794896

HOG Of Image 2 After Normalization:
0.015709314 0.24487764 0.15878417 0.01623429 0.01779501 0.24884115 0.13422094 0.0099062035
0.016069332 0.23410164 0.16867368 0.026867004 0.011702206 0.21418059 0.20414737 0.027019443
0.18611653 0.15920769 0.098852865 0.1640925 0.19930565 0.13293228 0.08970753 0.19085468
0.14747259 0.18197839 0.11491434 0.21734054 0.14542957 0.12483797 0.091687955 0.181926
0.02810995 0.18633458 0.10974766 0.036140133 0.021564988 0.24209222 0.17051227 0.05015516
0.039831143 0.17293344 0.10218064 0.02002254 0.05050528 0.24928266 0.14951885 0.017474316

The Co-Sine Similarity between the 2 images are:  0.96872824
The Co-Sine Similarity between the 2 images are:  0.93808854
```

Co Sine Similarity is printed twice as Object **“3”** is detected twice

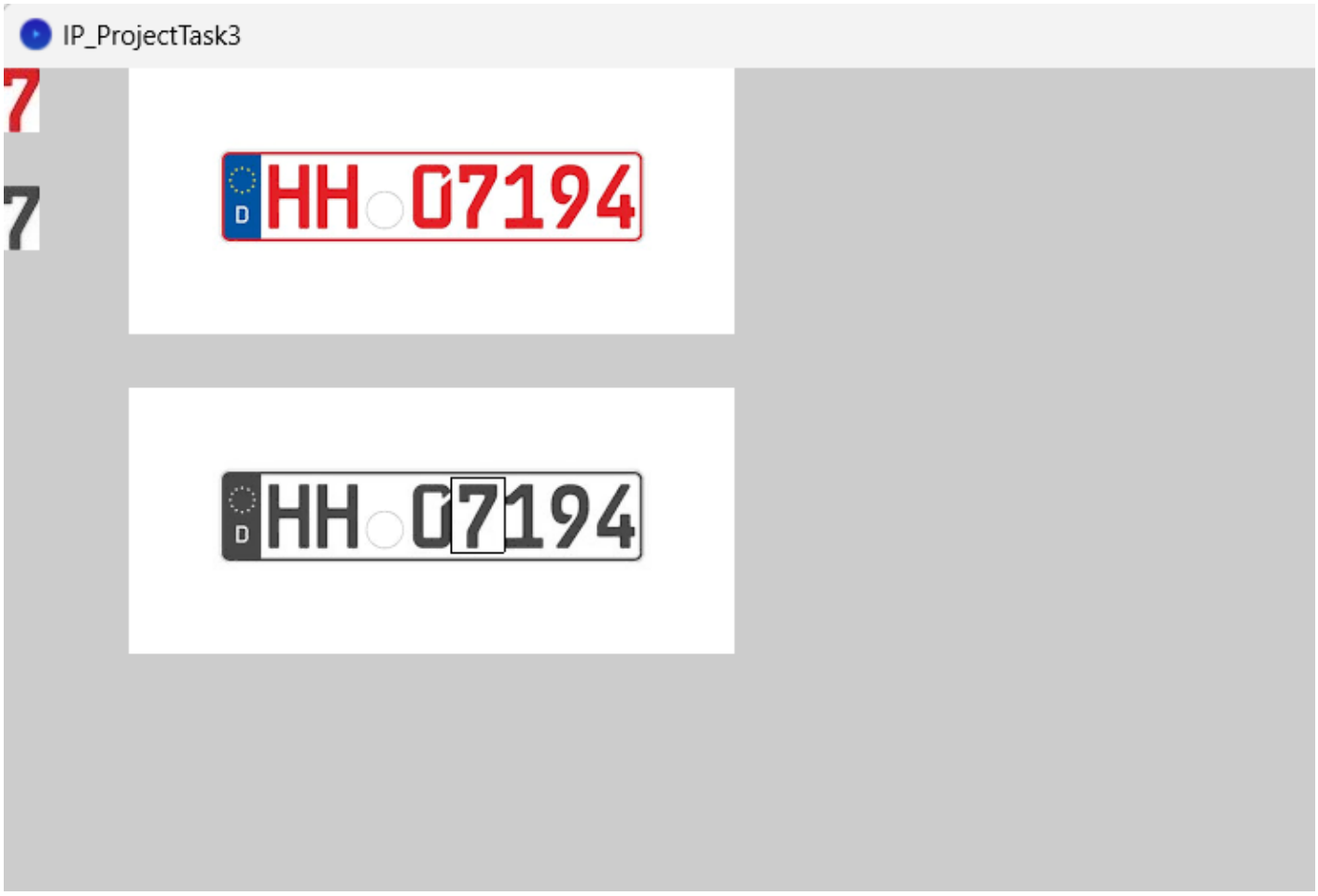
Output Result 3:

With Parameters - n = 4 ;N = 2, M = 3, window dimensions = 30, 42
window steps =15,10 ,threshold value = 0.10

```
HOG Of Image 1 After Normalization:
0.26120687 0.08639772 0.040492613 0.041645892
0.03370638 0.2285522 0.46344072 0.013538726
0.36057088 0.015513028 0.021834271 0.006297113
0.008851724 0.08590279 0.5100189 0.0051971567
0.48859775 0.009426321 0.032375842 0.11558115
0.031528343 0.0025860374 8.60449E-4 0.0014380993

HOG Of Image 2 After Normalization:
0.16573322 0.12581986 0.24178445 0.20612152
0.27522445 0.29704502 0.27403104 0.3166032
0.16294938 0.15488632 0.17603905 0.2002009
0.18440391 0.1762956 0.17392036 0.09259172
0.17269163 0.30710426 0.19435759 0.21381208
0.14666137 0.17165206 0.12910353 0.12511969

The Co-Sine Similarity between the 2 images are:  0.8864543
```



References:

1. Text Reference :

1a) https://medium.com/@data-overload/sliding-window-technique-reduce-the-complexity-of-your-algorithm-5badb2cf432f#:~:text=The%20sliding%20window%20technique%20is,and%20processing%20each%20window%20independently

1b) https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

2. Source Code Reference :

- http://gnjatovic.info/imageprocessing/ip.code.snippets.txt
- Lecture Notes.

3. Image Reference:

References of Image used in the Code:

- http://gnjatovic.info/imageprocessing/
- https://fuwong.com/wp-content/uploads/2017/08/German-license-plate-101-Europlates-wiki-11.jpg
- https://upload.wikimedia.org/wikipedia/commons/thumb/4/48/Danish_number_plate.jpg/640px-Danish_number_plate.jpg

References of Image used in the Report:

- https://www.researchgate.net/profile/Hafida-Benhidour/publication/345431594/figure/fig1/AS:955188156325889@1604746011536/An-illustration-of-the-sliding-window-technique.png