VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

NGUYEN QUANG HUY - 523H0140

NGUYEN DANG NAM KHANH - 523H0148

MIDTERM ESSAY

DIGITAL IMAGE PROCESSING

HO CHI MINH CITY, 2025

VIETNAM GENERAL CONFEDERATION OF LABOR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

NGUYEN QUANG HUY - 523H0140

NGUYEN DANG NAM KHANH - 523H0148

MIDTERM ESSAY

DIGITAL IMAGE PROCESSING

Advised by

Dr. Nguyen Chi Thien

HO CHI MINH CITY, 2025

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Dr. Nguyen Chi Thien, our instructor and mentor, for his valuable guidance and support throughout the mid-term report. He has been very helpful and patient in providing us with constructive feedback and suggestions to improve our work. He has also encouraged us to explore new technologies and techniques to enhance our system's functionality and performance. We have learned a lot from his expertise and experience in web development and software engineering. We are honored and privileged to have him as our teacher and supervisor.

Ho Chi Minh city, 2nd November 2025.

Author

(Signature and full name)

Huy

Nguyen Quang Huy

Khanh

Nguyen Dang Nam Khanh

DECLARATION OF AUTHORSHIP

We hereby declare that this is our own project and is guided by Dr. Nguyen Chi Thien. The content research and results contained herein are central and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the main author from different sources, which are clearly stated in the reference section.

In addition, the project also uses some comments, assessments as well as data of other authors, other organizations with citations and annotated sources.

If something wrong happens, we'll take full responsibility for the content of my project. Ton Duc Thang University is not related to the infringing rights, the copyrights that We give during the implementation process (if any).

Ho Chi Minh city, 2nd November 2025.

Author

(Signature and full name)

Huy

Nguyen Quang Huy

Khanh

Nguyen Dang Nam Khanh

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. PROBLEM SOLVING METHOD

1.1 Task 1 - Detection and bounding of traffic signs in a video stream

This section outlines the methodology for locating and drawing bounding boxes around traffic signs for a video stream. The difficulty of this task stems from accurately extracting signs from a moving and often noisy background under changing illumination. Achieving reliable results largely relies on an early preprocessing phase, which standardizes the image and maps it to a color domain resilient to lighting variations. Hence, the processing pipeline is structured around a crucial normalization and color transformation step, followed by hue-based segmentation and refined shape filtering to separate and detect the signs.

Input and Output: The input for this task is a frame from a video stream, and the output is the same frame with detected traffic signs enclosed in bounding boxes.

Figure 1.1.1: Example frame from the video (Input)

The foundational step in this pipeline is the conversion of the source frame from the BGR color space into the HSV (Hue, Saturation, Value) representation. The selection of this color space is the single most influential decision for this task,

as it decouples the color information (Hue) from brightness and illumination (Value). This directly dictates the quality and reliability of the color-based segmentation for all subsequent processing, making the system resilient to environmental changes like shadows and glare.

1.1.1 Frame-level preprocessing

The initial phase of the detection pipeline is dedicated to image conditioning and normalization. This is a critical sequence of operations designed to mitigate the challenges posed by variable lighting, shadows, glare, and sensor noise, which are ubiquitous in real-world driving scenarios. The process is a synergistic pipeline where each step prepares the image for the next, culminating in an output that is optimized for reliable feature extraction.

a. Gamma Correction

Gamma correction is a non-linear operation used to adjust the brightness and contrast of an image to account for the non-linear way human eyes perceive light.

For an input pixel intensity $I \in [0, 255]$, the output intensity O is calculated as: $O = 255 \times (I / 255)^{(1/\gamma)}$. There is a Look-Up Table (LUT) that pre-computed for all 256 possible intensity values to make this operation highly efficient.

b. Gaussian Blurring

After brightness adjustment, a Gaussian blur is applied. This is a low-pass filtering technique that smooths the image by convolving it with a Gaussian kernel, reducing high-frequency noise. The two-dimensional Gaussian function used to generate the kernel is given by:

$$G(x, y) = (1 / (2\pi\sigma^2)) \times e^{(-(x^2 + y^2) / (2\sigma^2))}$$

Where (x, y) are the coordinates relative to the kernel's center and σ is the standard deviation.

The GaussianBlur function in **openCV** library creates a kernel (in my case, 5x5) where each element's value is determined by the Gaussian function. The output value of each pixel in the image is then calculated as a weighted average

of its neighboring pixels, with the weights being the values from the Gaussian kernel. This convolution operation effectively **blurs** the image.

c. Contrast Limited Adaptive Histogram Equalization (CLAHE)

CLAHE is an advanced form of histogram equalization used to improve local contrast. Unlike global histogram equalization, CLAHE operates on small regions of the image (tiles), preventing the over-amplification of noise that can occur in relatively uniform areas.

The function follow:

- The image is divided into a grid of small, contextual regions (tiles).
- For each tile, a standard histogram is computed.
- To limit contrast, each histogram is "clipped" at a predefined value (clipLimit). The excess pixel counts are then redistributed evenly across all histogram bins.
- Histogram equalization is then applied to the clipped histogram of each tile.
- The final pixel values are determined by bilinear interpolation between the mappings of the four nearest tiles.

By converting the image to the HSV color space, brightness (Value) is separated from color (Hue and Saturation). Applying CLAHE only to the V channel enhances local contrast and detail in light and dark areas without distorting the image's original colors, which are crucial for the subsequent color segmentation stage.

d. Color Space Conversion to HSV

The final preprocessing step is to convert the image to the HSV color space and merge the channels after enhancing the V channel.

The HSV (Hue, Saturation, Value) color space is used for its robustness to illumination changes. The BGR color model is highly sensitive to lighting, but HSV separates the color information (Hue) from intensity (Value). This allows for reliable color detection across various lighting conditions by thresholding a

narrow range of Hue values, which is essential for a robust traffic sign detection system.

Figure 1.1.2: The preprocessed image

The enhanced image exhibits higher clarity in both bright and shadowed regions, allowing for more accurate traffic sign segmentation and recognition.

1.1.2 Apply kernel for extract masks

This stage executes the core segmentation logic by isolating pixels corresponding to the characteristic colors of traffic signs. This is achieved through color thresholding and a series of morphological operations to refine the resulting binary masks.

a. Color Range Selection for HSV Segmentation

This is the primary segmentation step, where the preprocessed image is filtered to isolate pixels that fall within the color profile of target traffic signs. Based on the robust HSV color space, this is achieved by defining specific ranges for Hue, Saturation, and Value.

The `cv.inRange` function is used to create a binary mask, where pixels within a specified lower and upper bound are set to white (255) and all other pixels are set to black (0).

Two primary colors are targeted: red and blue.

- Red Sign Detection: Detecting the color red in the HSV space is complex because the hue value is cyclical, and red straddles the 0/179 boundary (in OpenCV). To capture the full spectrum of red, two separate ranges are defined:
  - Range 1 (lower_red1, upper_red1): Captures the lower end of the red spectrum (Hue values roughly 0-20).
  - Range 2 (lower_red2, upper_red2): Captures the upper end of the red spectrum (Hue values roughly 110-180).
  - These two resulting binary masks are then combined using a `cv.bitwise_or` operation to create a single, comprehensive mask_red that identifies all red pixels.
- Blue Sign Detection: Blue occupies a continuous and stable range within the HSV spectrum. A single range (Hue values roughly 100-130) is defined to isolate the characteristic blue used in mandatory and informational traffic signs.
- Error range color: use only for debugging and testing, not affected by the process.

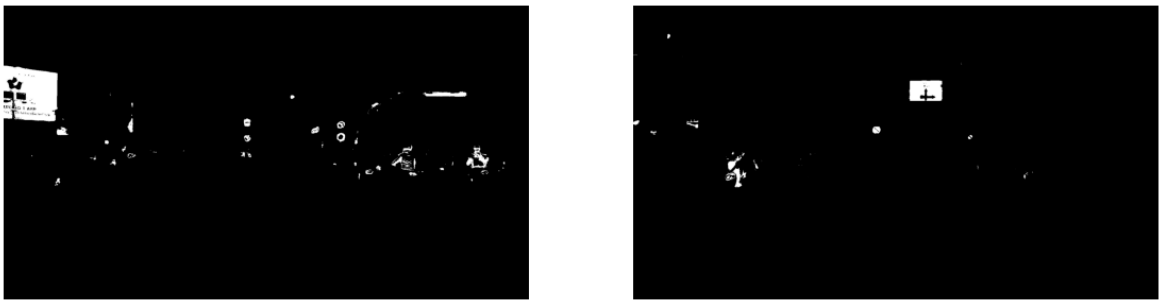

Figure 1.1.3: Binary mask only apply color bitwise color, red mask (left), blue (right)

b. Morphological Operations for Mask Refinement

The raw binary masks produced by color segmentation are imperfect. They often contain small, spurious regions of noise (salt noise) and may have gaps or holes within the larger, valid sign regions (pepper noise). Morphological

operations are applied to clean these masks and make them a more accurate representation.

- Morphological Erosion: The structuring element slides over the image, and for each position, if all the pixels under the structuring element match the foreground, the pixel in the output image is set to the foreground. Otherwise, it is set to the background.
- Morphological Dilation: The structuring element slides over the image, and for each position, if any pixel under the structuring element matches the foreground, the pixel in the output image is set to the foreground.
- Morphological Opening: First, the image undergoes erosion, which removes small objects and noise. Then, dilation is applied to restore the size of the remaining objects to their original dimensions.
- Morphological Closing: First, dilation is applied to the image, filling small holes and gaps. Then, erosion is applied to restore the original size of the objects.
- Merge Mask: The red and blue masks described above are processed using morphological opening and closing operations to remove small noise regions and fill small gaps, respectively. Then, dilation is applied to slightly enlarge the mask regions, ensuring that the detected areas fully cover the traffic signs.

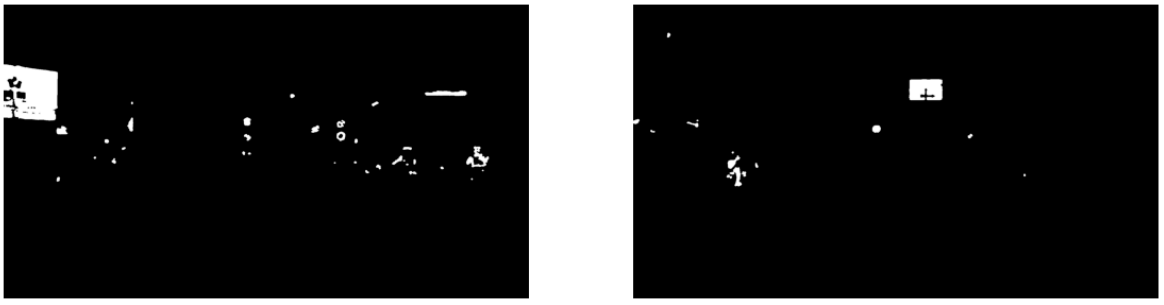

Figure 1.1.4: Binary mask apply morphological technique

The cleaned and merged masks provide a more accurate and continuous representation of the red and blue regions.

1.1.3 Contouring and bounding detect

The final stage of the detection pipeline is dedicated to the identification and geometric localization of potential traffic signs within the cleaned red and blue mask regions. This is achieved through contour extraction, shape analysis, and bounding operations that mark detected signs with green rectangles on the output frame. The process not only delineates the sign regions but also filters out irrelevant shapes, ensuring that only valid circular or triangular signs are preserved for later recognition stages.

a. Contour Extraction

After mask refinement, contours are extracted using the cv.findContours() function with the `cv.RETR_EXTERNAL` retrieval mode and `cv.CHAIN_APPROX_SIMPLE` contour approximation. This operation traces the boundaries of connected components in the binary masks, returning a set of coordinate points representing each contour. Only the outermost boundaries are retained to minimize redundancy.

Figure 1.1.5: Contour extraction, visualize by cv.`drawContours`

b. Shape Filtering and Classification

Each detected contour is analyzed to determine its geometric characteristics. Two primary shape types are targeted: triangular and circular.

- Triangular Detection:
  - For each contour, the perimeter is calculated using cv.arcLength(). The contour is then approximated using `cv.approxPolyDP()` with a proportional error tolerance (ε × perimeter). Contours with exactly three vertices are treated as potential triangles. To ensure geometric validity, the edge lengths of the approximated triangle are compared. The distances between all vertex pairs (d_1, d_2, d_3) are computed, and their average is used to assess regularity. A contour is accepted as a valid triangular sign if the deviation of each edge length from the mean is less than 20%:

$$|di - d| < 0.2d, i = 1, 2, 3$$

  - Once validated, a bounding rectangle is drawn using `cv.boundingRect()` to enclose the detected triangle.
- Circular Detection:
  - Circularity is evaluated by comparing the contour area *A* A to the area of its minimum enclosing circle. The circle is obtained using `cv.minEnclosingCircle()`, providing a center point and radius *r*. Circularity C is defined as: $C = \frac{A}{\pi r^2}$
  - Contours with C > 0.67 are considered circular candidates. This threshold effectively distinguishes traffic sign circles from irregular blobs or partial arcs. Each valid circular region is enclosed within a bounding rectangle centered on the circle.

c. Area Filtering

To eliminate small, irrelevant regions (e.g., noise or reflections), contours with areas smaller than a predefined minimum threshold (600 pixels) are

discarded before shape evaluation. This ensures that only visually significant regions are processed for shape classification.

d. Output Visualization

For each validated sign contour, a bounding box is drawn in green (0, 255, 0) on the input frame using the `cv.rectangle()` function. The resulting frame displays all detected traffic signs, both circular and triangular, as enclosed regions, ready for potential classification or labeling in subsequent stages.

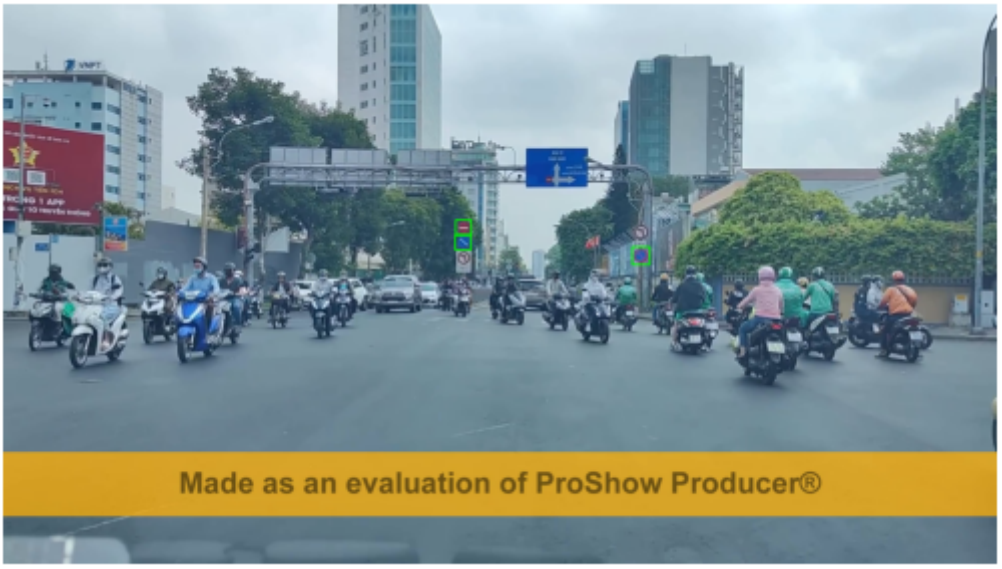

Figure 1.1.6: Final frame with bounding box (output)

1.2 Task 2 - Detection of numerical digits in a static image

This section outlines the methodology for locating and drawing bounding boxes around each individual digit in the provided high noise image. A successful outcome is highly dependent on the initial conversion of the image into a clean binary format. The pipeline is therefore architected around an image alignment, a critical binarization stage, followed by contour analysis and precise geometric filtering to isolate each digit.

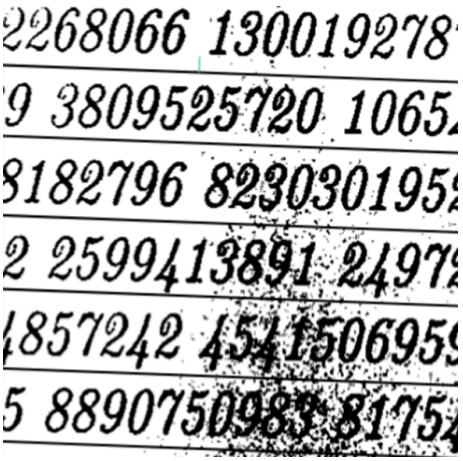

Figure 1.2.1: Provided image (Input)

The foundational step in this pipeline is the conversion of the source image into a binary representation where the digits are unambiguously separated from the background. The selection of the binarization technique is the single most influential decision for this task, as it directly dictates the quality of the input for all subsequent processing

1.2.1 Image gray-scale conversion

a. Average method

A simple arithmetic mean of the three color channels:

$$Y = \frac{R + G + B}{3}$$

This approach treats all channels equally, producing a balanced but less perceptually accurate representation.

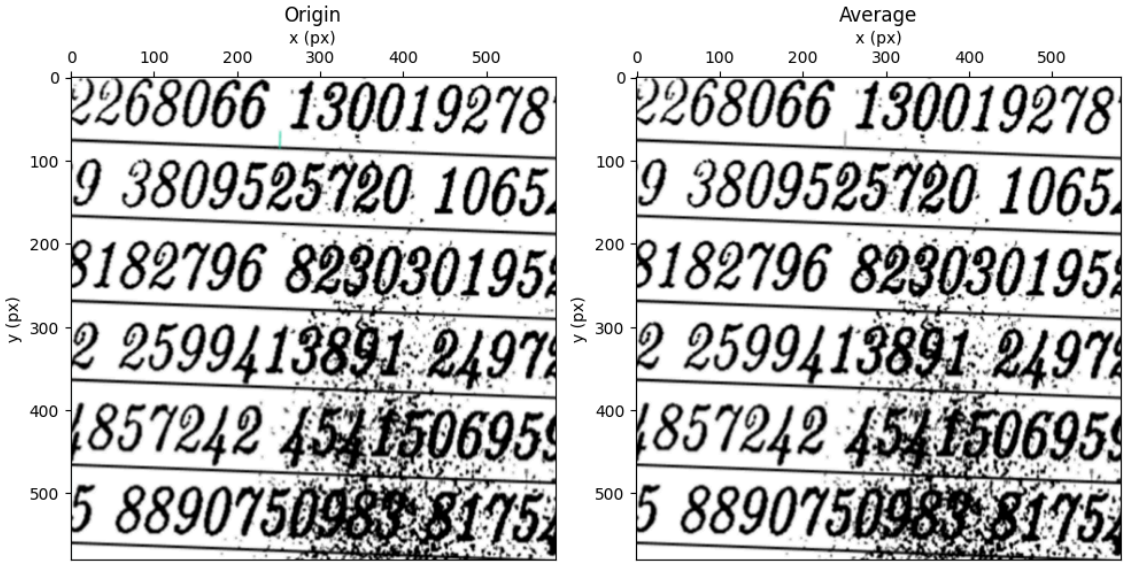

Figure 1.2.2: Origin image (Left) and Gray-scaled image with Average method (Right)

b. Lightness method

Uses only the extremes of color intensities:

$$Y = \frac{max(R, G, B) + min(R, G, B)}{2}$$

This method better preserves high-contrast regions but loses subtle midtone information.

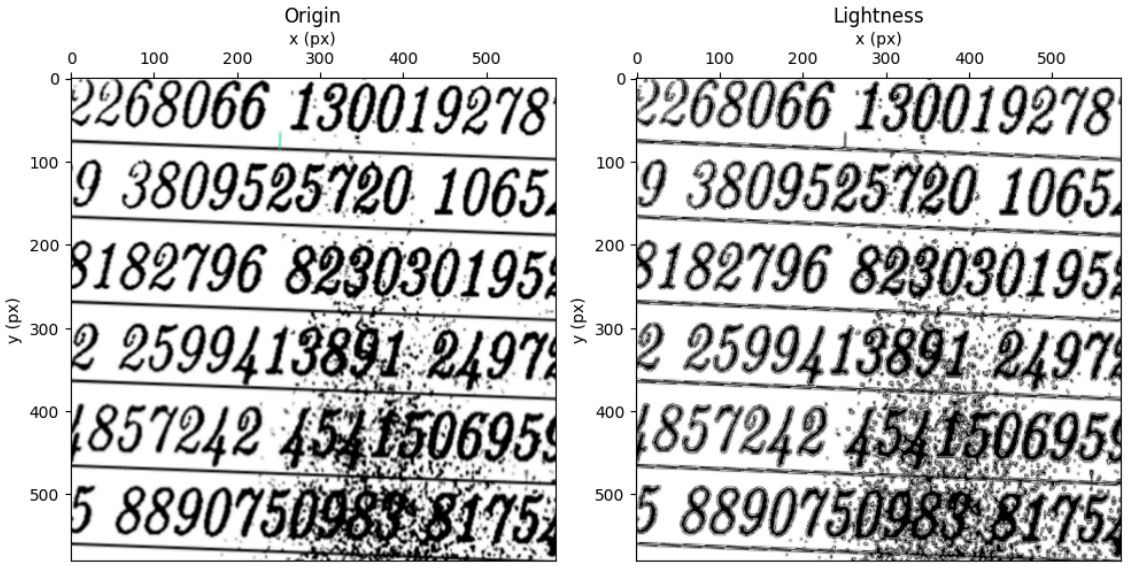

Figure 1.2.3: Origin image (Left) and Gray-scaled image with Lightness method (Right)

c. Luminosity method

Adopted in OpenCV's built-in function cv2.COLOR_BGR2GRAY:

$$Y = 0.299R + 0.587G + 0.114B$$

This formula reflects the human eye's higher sensitivity to green light and lower sensitivity to blue, producing a perceptually accurate gray-scale image.

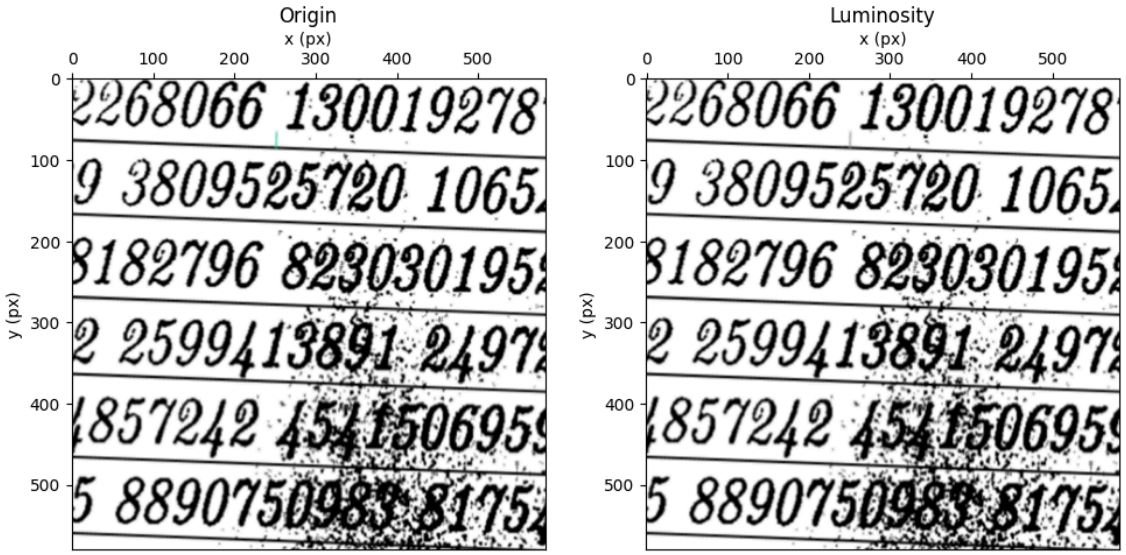

Figure 1.2.4: Origin image (Left) and Gray-scaled image with Luminosity method (Right)

d. Single channel extraction

In some cases, a single color channel (usually the green channel) is selected directly:

$$Y = G$$

This method is computationally fastest but may cause information loss if the chosen channel does not represent overall luminance well.

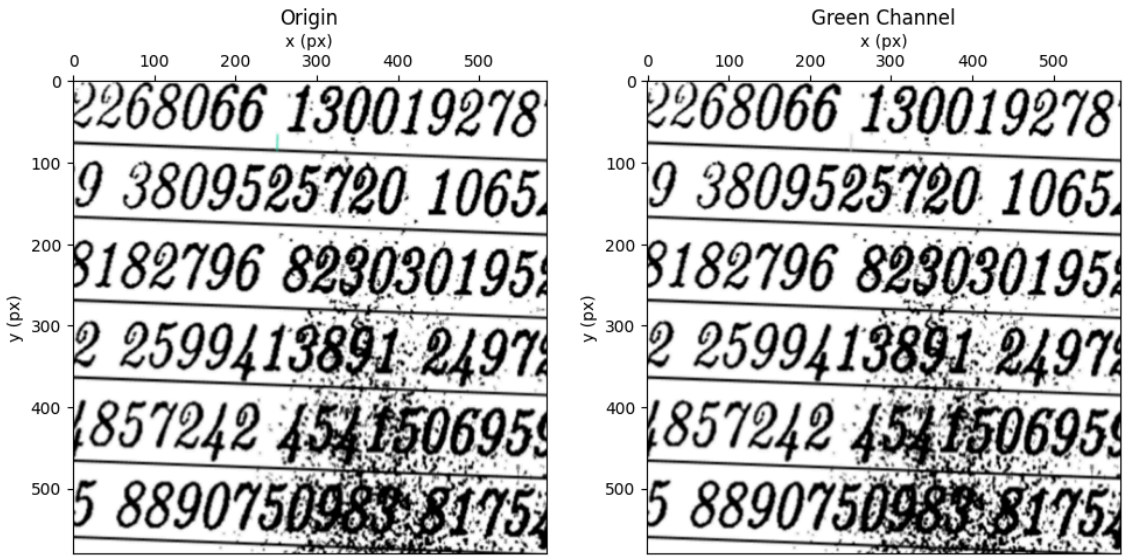

Figure 1.2.5: Origin image (Left) and Gray-scaled image with Single channel (Green) extraction method (Right)

e. Comparison among above methods

Among the above methods, the Luminosity method (OpenCV default) provides the most consistent and perceptually valid result for natural images.

While the Average method is simpler, it disregards human visual sensitivity, and the Lightness method tends to exaggerate contrast in noisy regions.

Since the subsequent stages (partitioning and entropy-based segmentation) rely heavily on accurate intensity representation to distinguish noise patterns, **the Luminosity-based conversion was selected as the standard approach for gray-scale transformation in this work.**

Figure 1.2.6: Comparison among 4 gray-scale conversion methods. Average (top-left), Lightness (top-right), Luminosity (bottom-left), Single Channel Extraction (bottom-right)

1.2.2 Image alignment

This rotation adjustment ensures that textual structures are horizontally aligned, thereby improving the accuracy and stability of subsequent preprocessing and binarization stages. Two complementary approaches are considered: Hard-coded and Rule-based method.

a. Hard-coded rotation

In this implementation, a fixed rotation angle of 2.16° was applied after adding a 20-pixel white padding around the image to prevent boundary loss during transformation. Because of this hard-coded rotation, the parameter only works best with the exam given image.

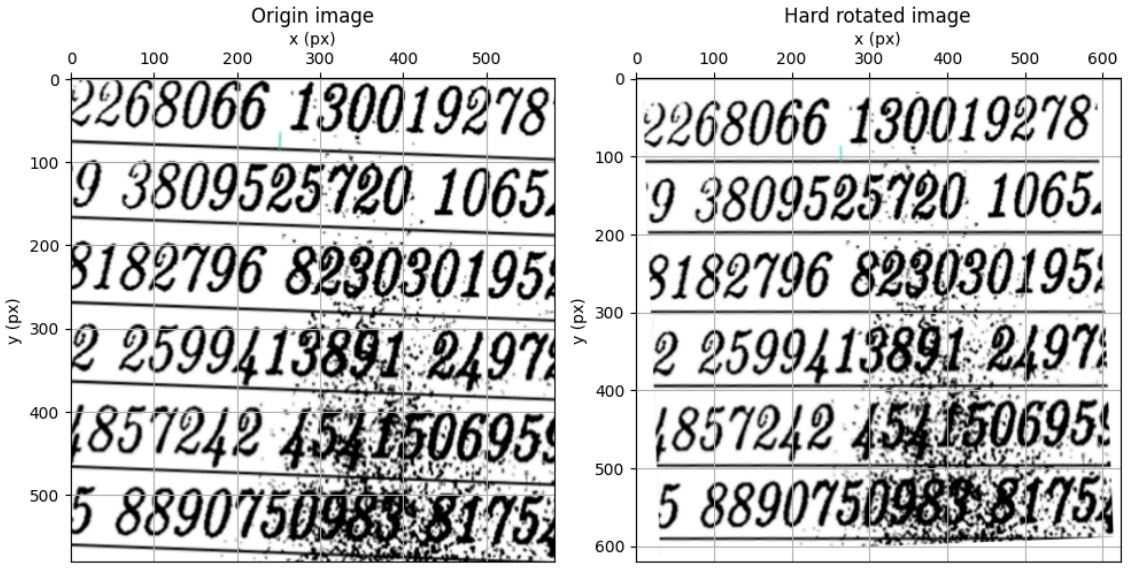

Figure 1.2.7: Origin image (Left) and Hard rotated image (Right)

b. Rule-based rotation

In addition to the fixed-angle rotation, a more adaptive alignment method was used based on the orientation of straight lines detected within the image. Since the image contains multiple prominent linear structures, the dominant line direction can be used as a reliable reference for estimating the skew angle. This approach computes the rotation angle automatically from the average orientation of detected lines and then applies the same affine transformation as in the hard-coded rotation function.

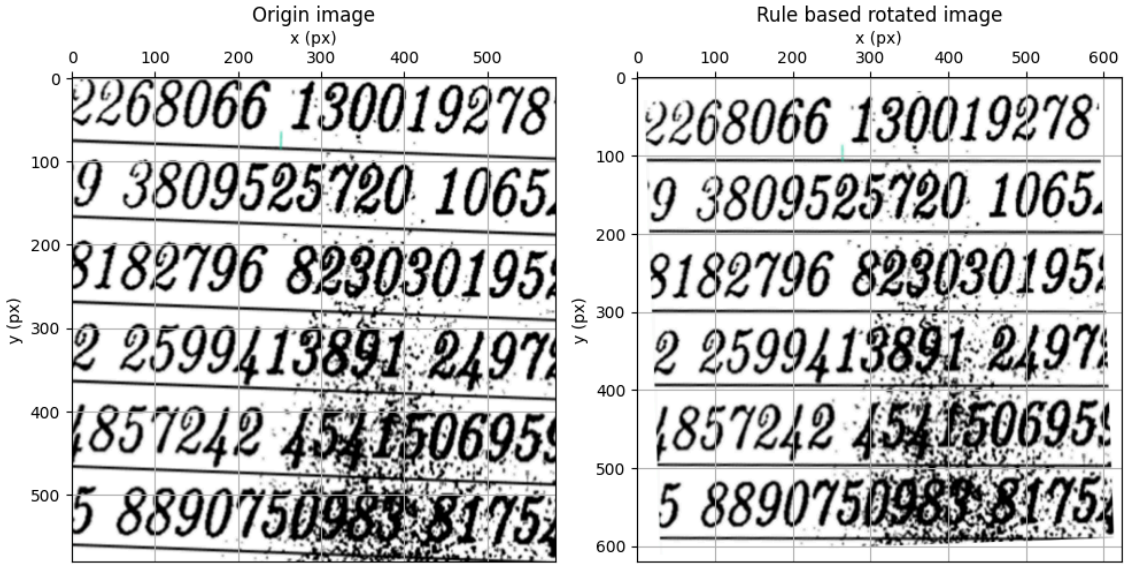

Figure 1.2.8: Origin image (Left) and Rule based rotated image (Right)

c. Comparison between Hard rotate and Line based rotated

Two different rotation methods were implemented to align the numerical digit images: a hard-coded rotation and a rule-based rotation using the Hough Transform.

Experimental comparison on the given image showed no significant improvement in alignment quality between the two methods. Since the image tilt across samples was nearly constant, the dynamically estimated rotation angles closely matched the fixed empirical angle. Furthermore, the rule-based approach occasionally produced unstable angles when noise or text patterns interfered with line detection.

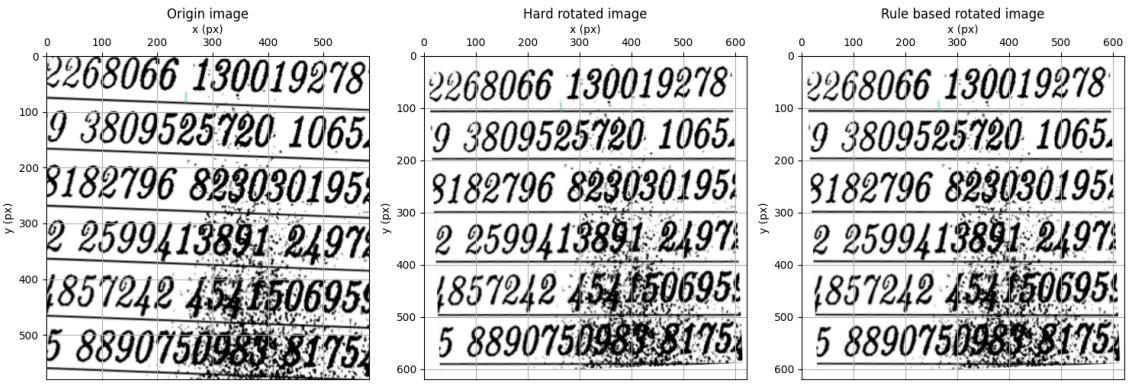

Figure 1.2.9: Origin image (Left), Hard rotated Image (Center) and Rule based rotated Image (Right)

For subsequent stages of implementation, the hard rotation method was adopted for its simplicity, stability, and computational efficiency.

1.2.3 Image partitioning

Because the input image exhibits varying levels of noise across different regions, an additional partitioning step is introduced before binarization. This target is to isolate areas with distinct noise characteristics and apply suitable preprocessing operations locally rather than globally. Two complementary approaches are considered: Hard-coded and Rule-based method.

a. Hard-coded segmentation

In this implementation, the hard-coded segmentation is defined through a fixed coordinate system that divides the image into four equally sized rectangular regions. The segmentation boundaries are determined by halving both the image width ***(w // 2)*** and height ***(h // 2)***, effectively creating a 2×2 grid. Each sub-image is represented by its top-left corner coordinates ***(x, y)*** and its corresponding width ***(w)*** and height ***(h)***, calculated directly from the global image dimensions.

Cell	**x**	**y**	**w**	**h**	**Noise level**
1	0	0	***w//2***	***h//2***	Low
2	***w//2***	0	***w//2***	***h//2***	Medium
3	0	***h//2***	***w//2***	***h//2***	Medium
4	***w//2***	***h//2***	***w//2***	***h//2***	High

Table 1.2.1: Coordinate definition

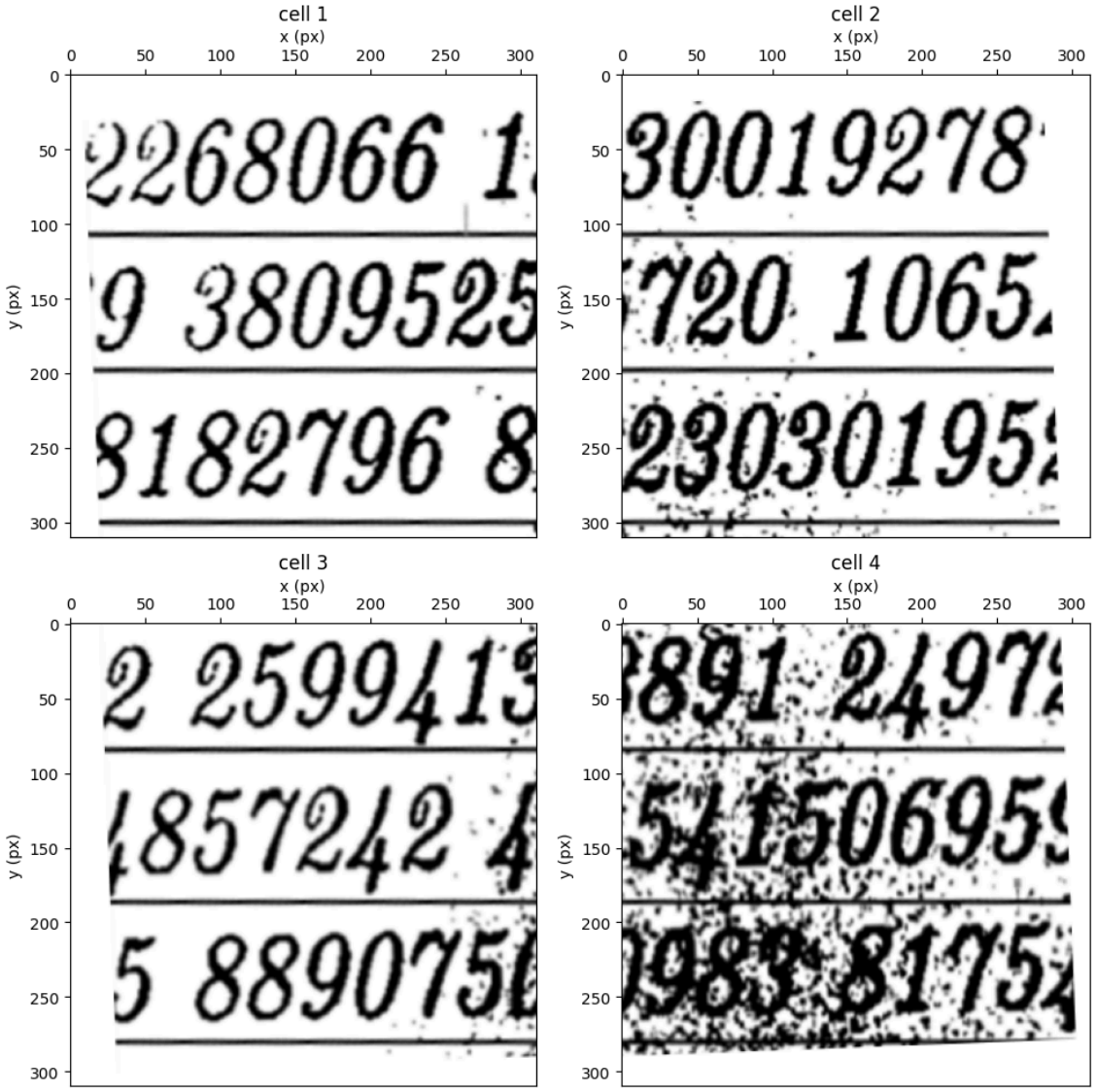

Figure 1.2.10: 4 rectangular regions

b. Entropy-based segmentation

Entropy, in this context, is used as a statistical measure of local texture variation – higher entropy values correspond to regions with greater randomness and noise, while lower entropy indicates smoother and more homogeneous areas. By scanning the image through non-overlapping rectangular patches, the entropy of each patch can be estimated from the normalized intensity histogram.

The segmentation process then follows a rule-based grouping mechanism: patches are assigned to discrete categories (e.g., Low, Medium, High noise)

based on predefined entropy thresholds rather than a learned model. Neighboring patches that belong to the same category are merged into larger rectangular regions, forming a small set of coherent sub-images that capture the dominant noise patterns across the frame.

This rule-based strategy maintains the geometric consistency of rectangular partitions while introducing a degree of adaptivity absent in fixed-coordinate segmentation. It provides a lightweight, data-driven foundation for localized filtering and thresholding without relying on external machine learning libraries or probabilistic clustering algorithms.

However, due to the midterm exam restriction – *"Only OpenCV and basic Python libraries are allowed in the programming part"* – the implementation of this segmentation method is not included here and is instead presented conceptually.

1.2.4 Image binarization

a. Threshold techniques

Several thresholding methods are available in OpenCV, each with distinct characteristics and suitability for different imaging conditions. A thorough evaluation is necessary to select the most robust method.

- **Method A: Glocal thresholding (cv2.THRESH_BINARY)**

$$dst(x, y) = \{255, if\ src(x, y) > T; 0, otherwise\}$$

This is the simplest form of thresholding, where a single, manually specified threshold value ***T*** is applied across the entire image. Any pixel with an intensity greater than ***T*** is set to the maximum value (e.g., 255 for white), and all other pixels are set to 0 (black). While computationally efficient, this method is highly brittle. It assumes uniform illumination across the image, an assumption that rarely holds true in practice. After a few experiments, we found that with ***T = 100*** for low noise region, ***T = 25*** for medium noise region and ***T = 5*** for high noise region.

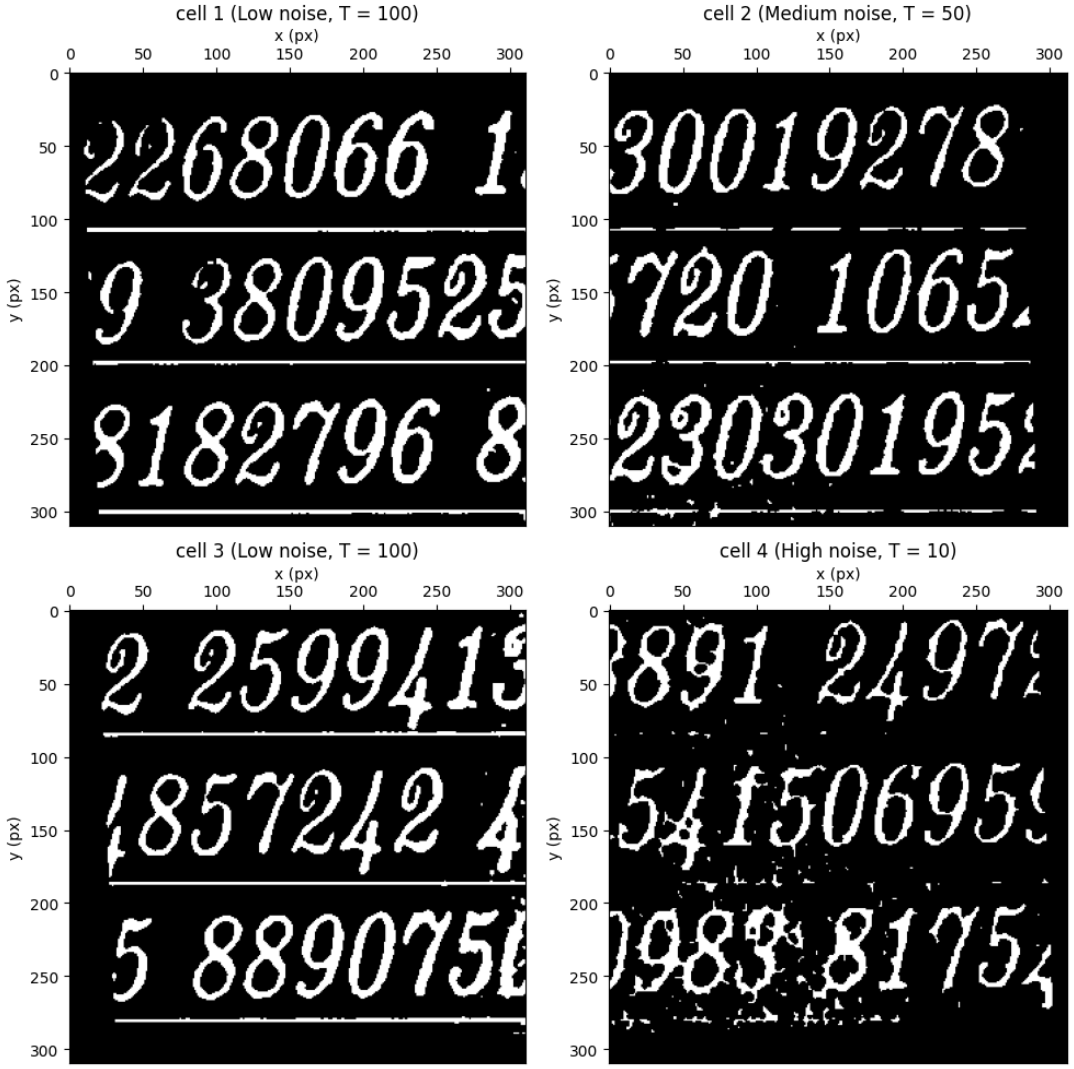

Figure 1.2.11: 4 regions with 3 noise group (Low noise: cell 1, 3 - T = 100; Medium noise: cell 2 - T = 25, High noise: cell 4 - T = 5) after applied Global Thresholding

- **Method B: Otsu's binarization (cv2.THRESH_OTSU)**

Otsu's method automates the threshold selection process. Instead of requiring a manually chosen threshold, it analyzes the image histogram and determines an optimal global threshold T^* that minimizes the intra-class variance (or equivalently, maximizes the inter-class variance).

This is expressed as:

$$T^* = \arg\min_T [\omega_1(T)\,\sigma_1^2(T) + \omega_2(T)\,\sigma_2^2(T)]$$

where ω_i and σ_i^2 denote the class probabilities and variances for the two pixel groups divided by threshold $\boldsymbol{T}$.

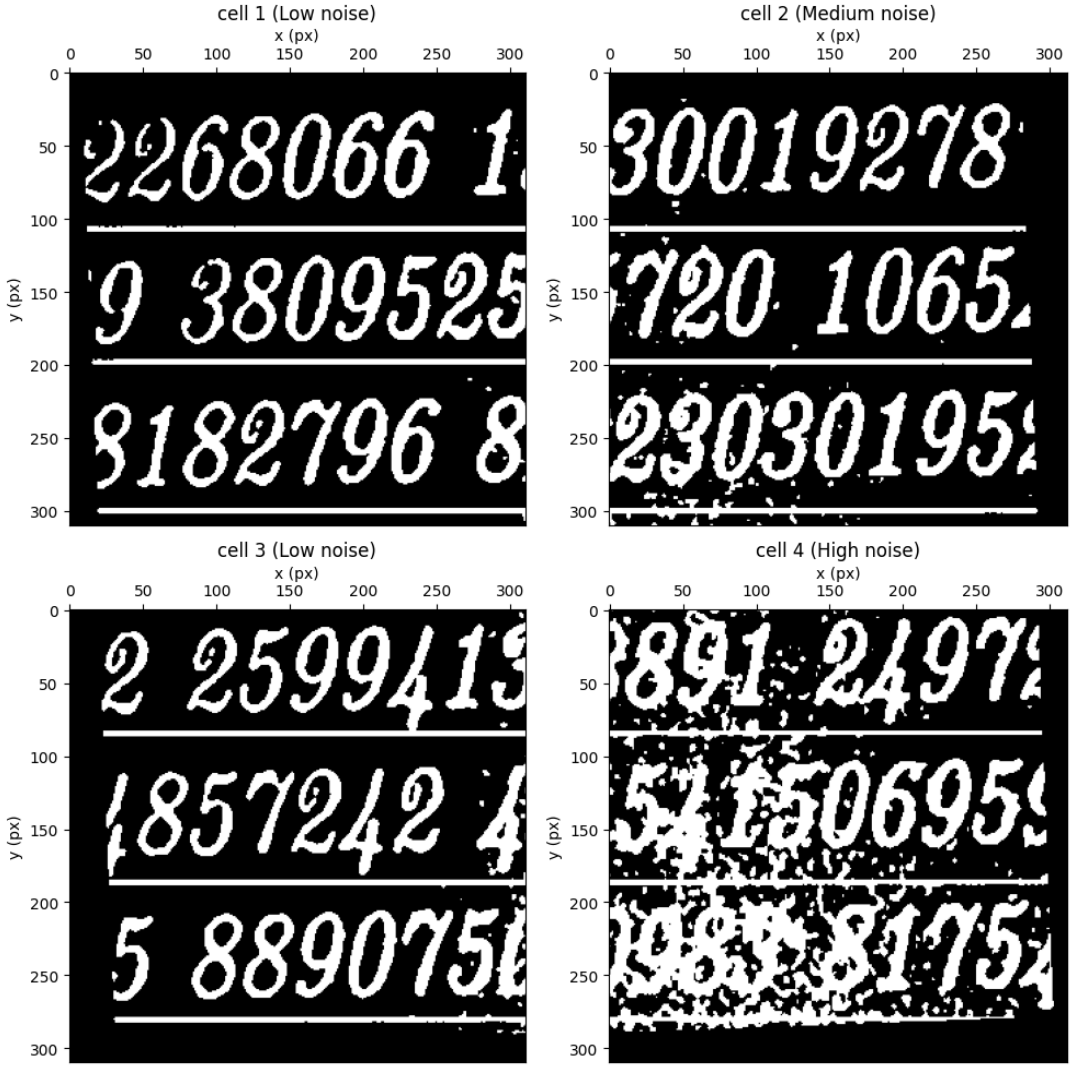

Figure 1.2.12: 4 regions with 3 noise group (Low noise: cell 1, 3; Medium noise: cell 2, High noise: cell 4) after applied Otsu's Thresholding

Although Otsu's method provides a principled way to select $\boldsymbol{T}$, it is still a global approach and performs poorly under non-uniform illumination or shadows.

Cell	1	2	3	4
Thresh	149	150	145	137

Table 1.2.2: Otsu's thresholding

- **Method C: Adaptive thresholding (cv2.adaptiveThreshold)**

This approach overcomes the limitations of global methods by computing a local threshold for each pixel based on the intensities within a small neighborhood window. This makes it robust against illumination variation and noise.

1. v2.ADAPTIVE_THRESH_MEAN_C: The threshold for a pixel is the mean of its neighborhood pixel intensities minus a constant ***C***.

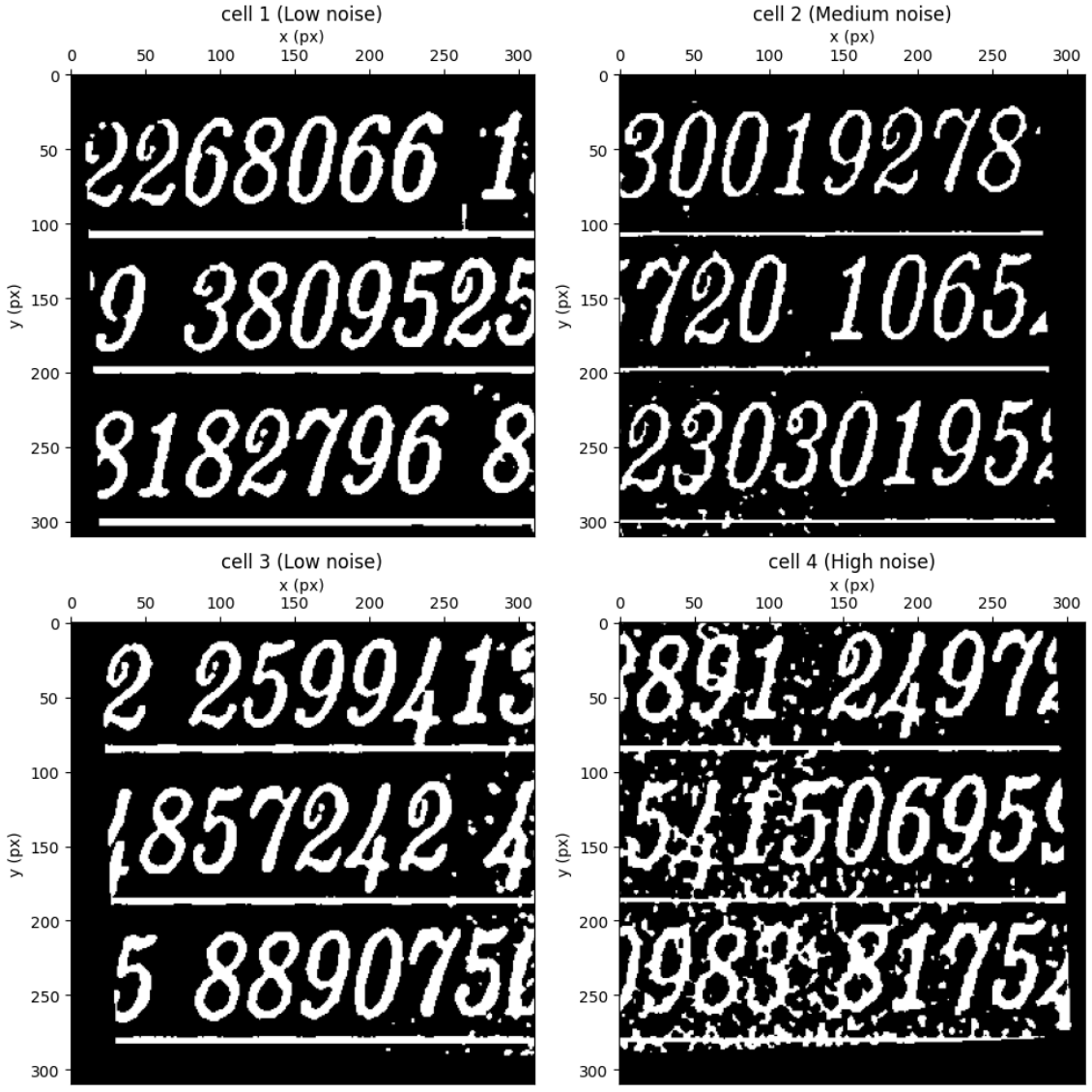

Figure 1.2.13: Apply Adaptive Mean thresholding

2. cv2.ADAPTIVE_THRESH_GAUSSIAN_C: The threshold is a Gaussian-weighted sum of the neighborhood pixel intensities minus the constant ***C***.

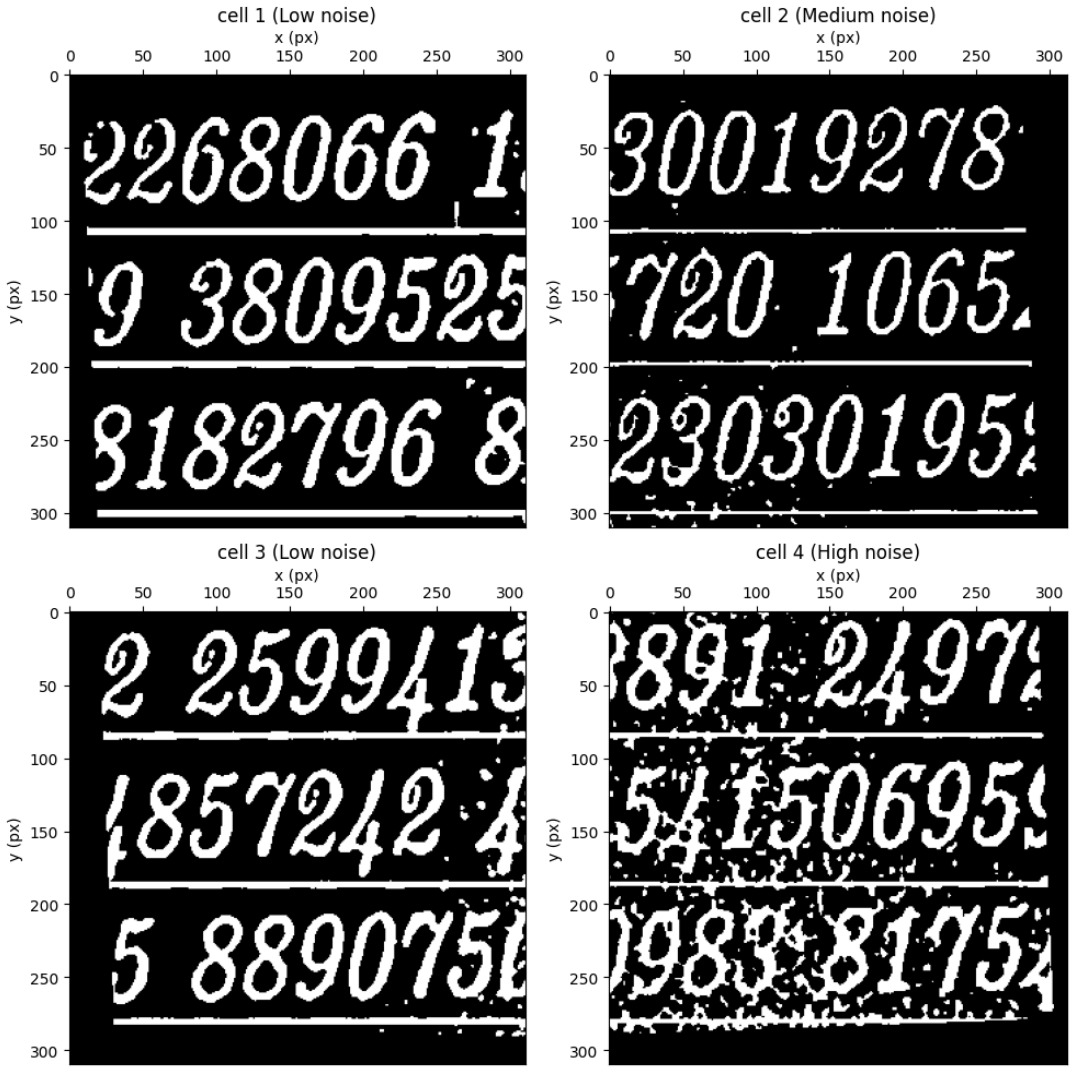

Figure 1.2.14: Apply Adaptive Gaussian thresholding

b. Comparison among above methods

The Gaussian variant is selected as the optimal approach. By giving higher weight to central pixels, it preserves fine structural details of digits while being less sensitive to local noise. This ensures consistent and reliable segmentation performance, especially for real-world scanned or photographed documents.

1.2.5 Digit segmentation and filtering

After obtaining a clean binary image through thresholding, the next step is **digit segmentation** – isolating individual digits from the image – followed by

filtering, which removes unwanted noise or false contours. This step is essential for preparing the dataset for digit recognition or OCR (Optical Character Recognition) tasks.

a. Find contours

Contours are continuous curves or boundaries that represent object outlines. In OpenCV, the function cv2.findContours() is used to detect all connected white regions in a binary image. Each contour can potentially correspond to a digit, a noise cluster, or other artifacts.

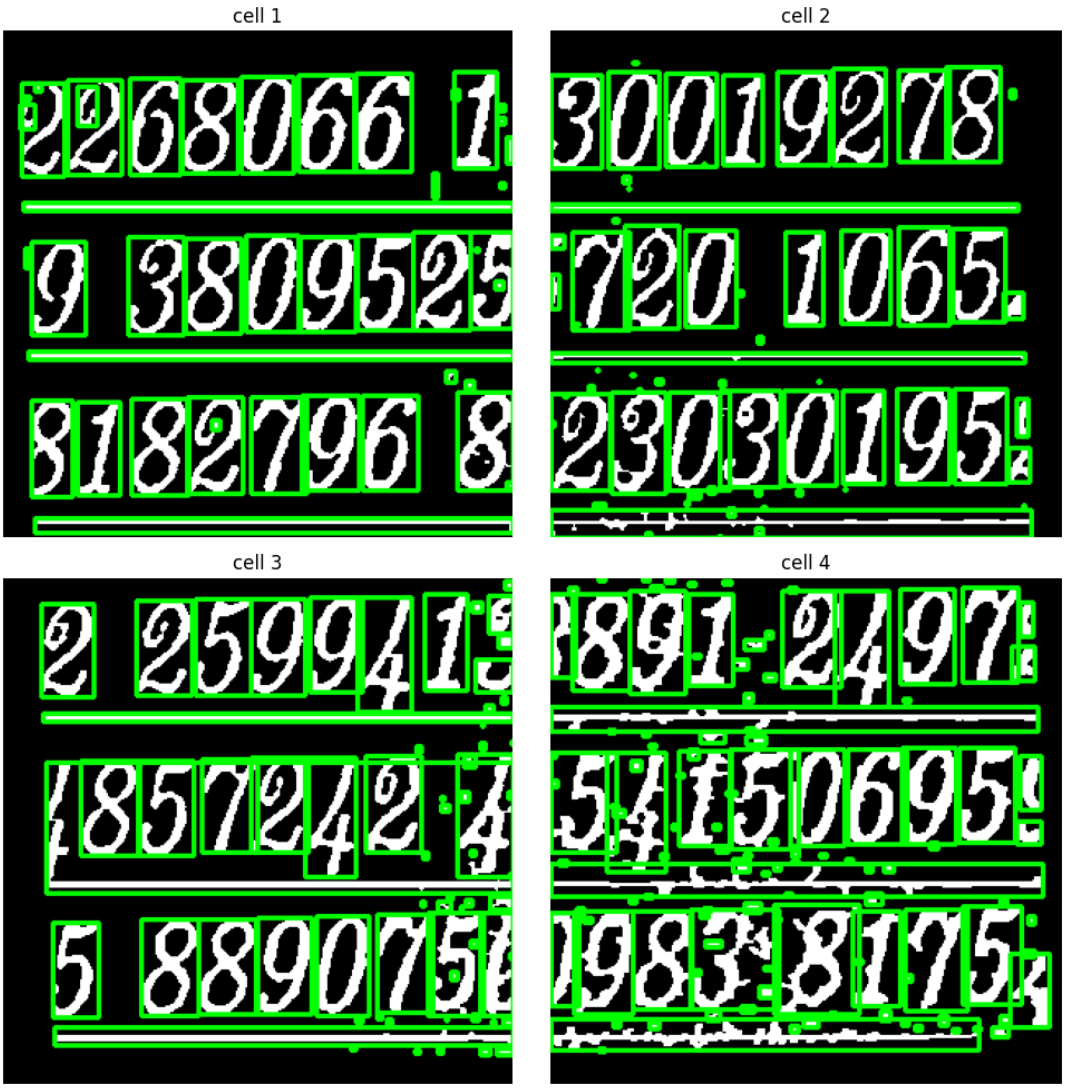

Figure 1.2.15: Contours finding

This operation outlines all possible connected components. However, not every contour corresponds to a valid digit; many are just noise or artifacts.

b. Filter contours

Filtering is necessary to retain only meaningful contours that represent digits. Typical filtering criteria include:

- Area threshold: remove contours that are too small or too large (e.g., 200 - 5000).
- Aspect ratio: digits usually have reasonable width-to-height ratios (e.g., between 0.2 and 1.5).
- Bounding box height: used to exclude lines or dots (e.g., larger than 15).

c. Draw bounding box

Once valid contours are identified, we can draw bounding boxes around each digit to visualize the segmentation result. Each bounding box defines a region of interest (ROI) that can be extracted for classification or further processing.

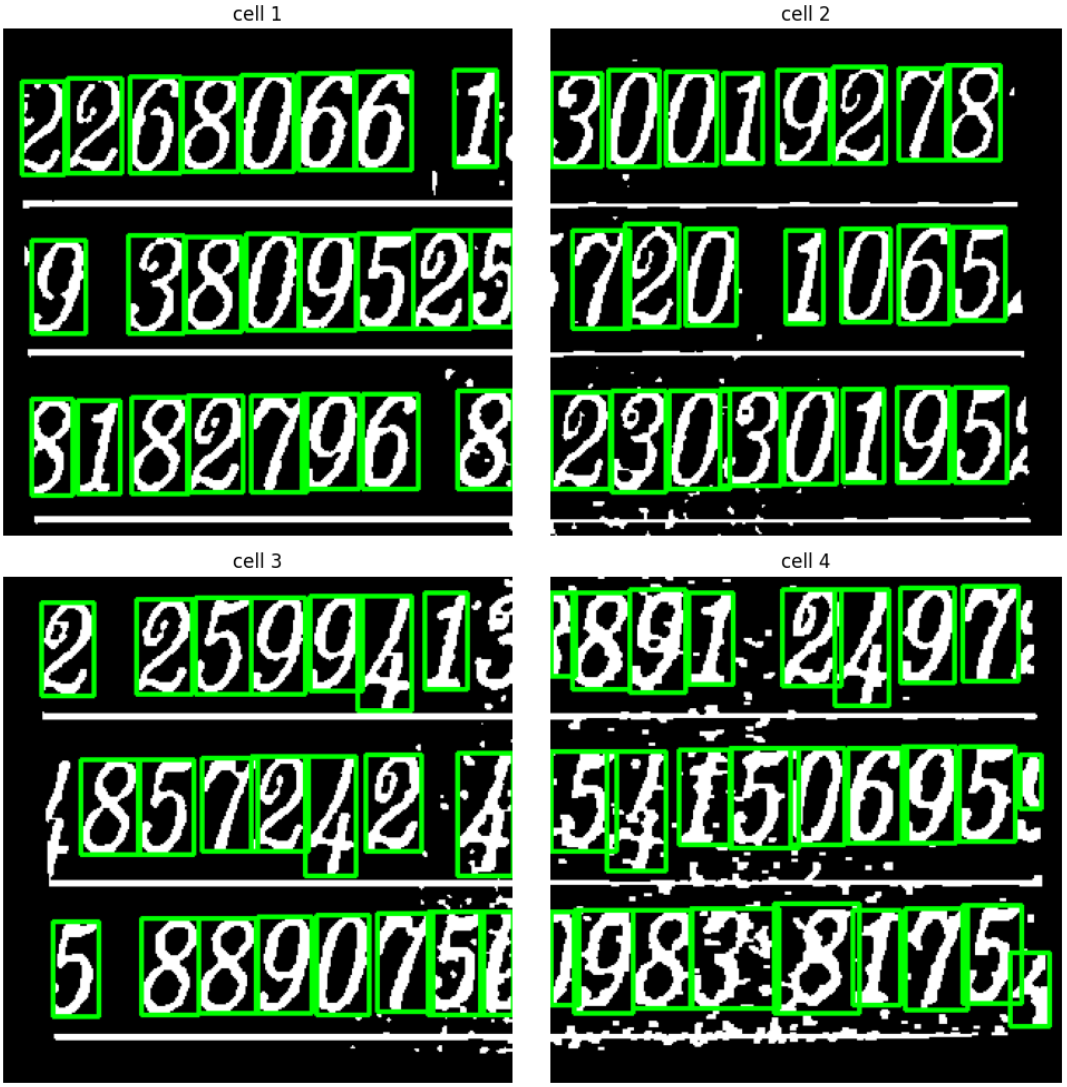

Figure 1.2.16: Contours filtering results

d. Morphological operations

Morphological transformations refine the binary image before contour extraction or after segmentation.

- Opening (erosion followed by dilation): removes small white noise, apply to cell 4.
- Closing (dilation followed by erosion): fills small black gaps inside digits, apply to cell 2, 3.

CHAPTER 2. TASKS RESULTS

2.1 Task 1 - Detection and bounding of traffic signs in a video stream

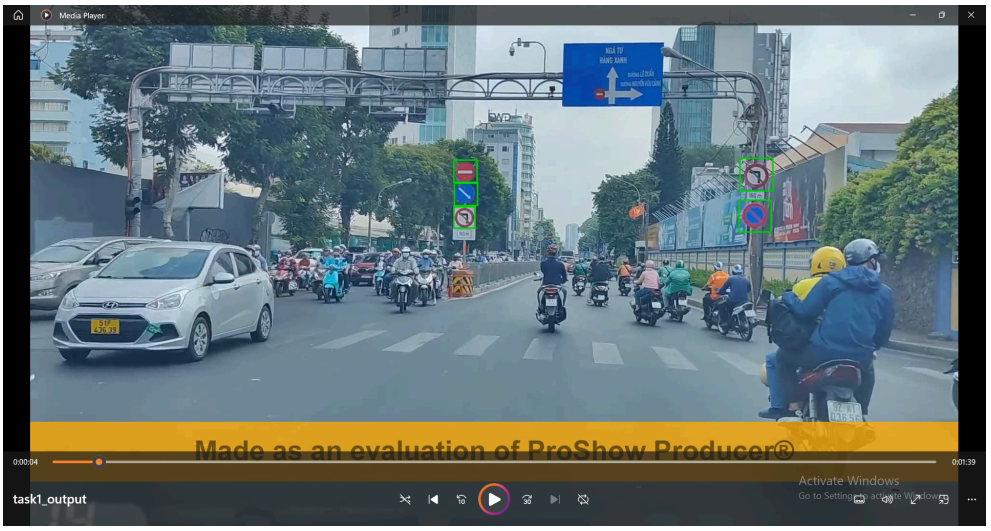

Figure 2.2.1: Output frame at 00:04

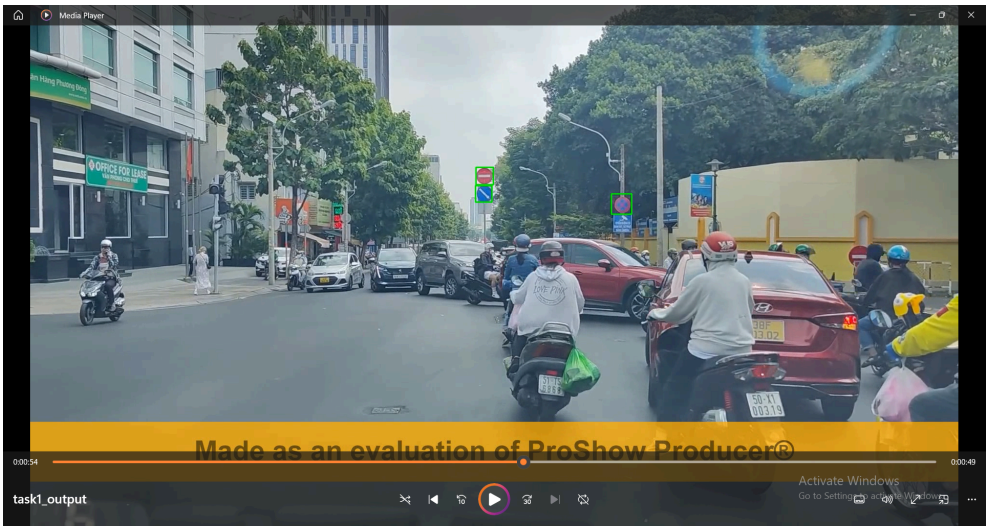

Figure 2.2.2: Output frame at 00:54

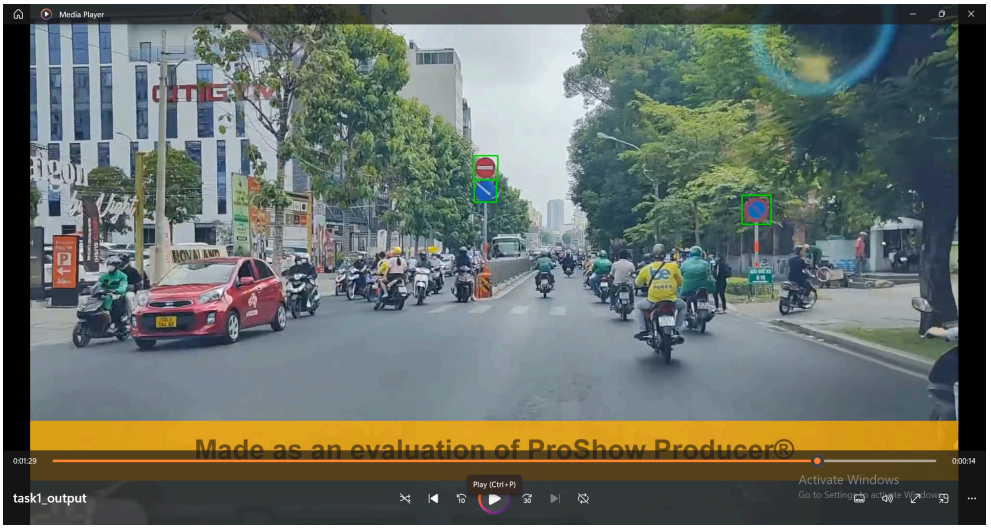

Figure 2.2.3: Output frame at 01:29

2.2 Task 2 - Detection of numerical digits in a static image

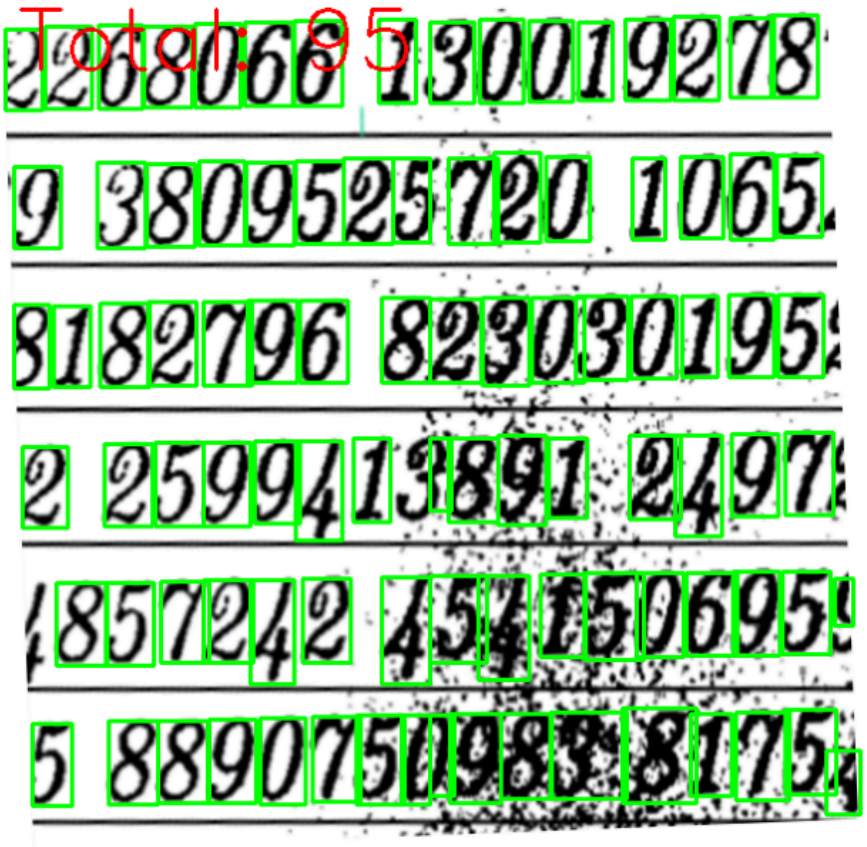

Figure 2.2.4: Final output with 95 digits detected