DATA-MINING PROJECT – CIA-2

KEERTHIVAS L -23011101063 [AIDS A, 2nd Year]

MITHUNAKUMAR V G -23011101078 [AIDS A, 2nd Year]

DATATYPE : IMAGE

Vehicle Detection Using Image-Based Data Mining

Challenges Faced

During the vehicle detection project, some challenges were encountered:

Image Quality Issues:

Some images had unclear vehicles, making it difficult for annotation and model learning.

Several images included unnecessary elements such as traffic, pedestrians, and background objects, reducing the clarity of the road surface.

Lighting Conditions:

Images had inconsistent brightness, contrast, and shadows, making preprocessing important for standardization.

Manual Annotation:

All images were manually annotated using RoboFlow, which was time-consuming .

Time Taking:

It was time taking to preprocess and detect the vehicle on the given image

Few images were attached for reference:

Images that were similar to a plain road

Images with unnecessary elements:

Application of Vehicle Detection Model

Use case of our model: Model can process incoming images and flag roads with vehicles.

Where can a vehicle detection model be used?
In smart city development, to automatically detect vehicles from road surveillance footage for quicker repairs.

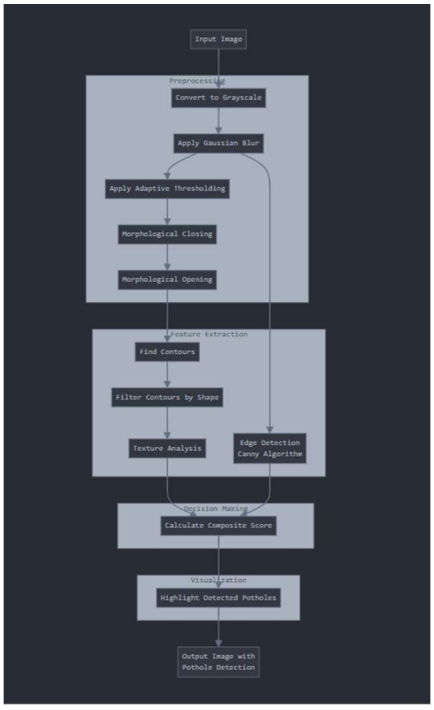
What happens if the model is trained at a larger scale?
With more data, the model can become highly accurate, adapting to different road textures, lighting conditions, and traffic environments across countries.

Can this be used by municipal bodies?
Yes, the model can assist municipal corporations to automate road inspection and maintenance alerts.

Is this model scalable to real-time systems?
With the use of real-time object detection algorithms (like YOLO), the model can be scaled to live video analysis from road-side cameras or dashcams.

Can it help with accident prevention?
Definitely! Early vehicle detection can help reduce accidents, especially for two-wheelers and night driving.

Architecture Diagram Workflow



Vehicle Detection Architecture Explanation

1. Convert the Image to Grayscale

Why: Reduces the complexity of the image by eliminating color information while preserving structural details. This simplifies processing and reduces computational overhead.

Implementation: Uses the standard grayscale conversion formula (0.299×R + 0.587×G + 0.114×B) to weight each color channel according to human perception.

Benefit: Focuses the analysis on intensity variations rather than color differences, which is more relevant for detecting structural irregularities like vehicles.

2. Apply Gaussian Blur

Why: Reduces noise and smooths out small irrelevant details that might interfere with the detection process.

Implementation: Convolves the image with a Gaussian kernel, which gives more weight to pixels near the center.

Benefit: Improves the robustness of subsequent steps by removing high-frequency noise while preserving the overall structure of the image.

3. Apply Adaptive Thresholding

Why: Separates the foreground (potential vehicles) from the background (road surface) by converting the grayscale image to binary.

Implementation: Calculates a threshold for each pixel based on the mean of its local neighborhood, accounting for lighting variations across the image.

Benefit: Works better than global thresholding for outdoor scenes where lighting conditions vary within the image.

4. Perform Morphological Operations

Why: Refines the binary image by removing noise and filling small gaps.

Implementation:

Closing (dilation followed by erosion): Fills small holes in the foreground objects.

Opening (erosion followed by dilation): Removes small objects and noise without affecting larger structures.

Benefit: Creates cleaner regions of interest for contour detection, reducing false positives.

5. Find Contours

Why: Identifies boundaries of potentially connected regions in the binary image.

Implementation: Labels connected components and extracts boundary points.

Benefit: Provides shape information needed to identify potential vehicle regions.

6. Filter Contours Based on Shape Factors

Why: Not all detected contours are vehicles; this step eliminates contours that don't have vehicle-like characteristics.

Implementation: Calculates and evaluates multiple shape metrics:

Area: Must be large enough to be a vehicle

Perimeter: Used to calculate circularity

Circularity: Vehicles are somewhat circular but not perfectly so ($0.2 < \text{circularity} < 0.9$)

Aspect Ratio: Most vehicles aren't extremely elongated ($0.4 < \text{aspect\_ratio} < 2.5$)

Extent: Ratio of contour area to bounding rectangle area (must be > 0.3)

Benefit: Greatly reduces false positives by applying domain knowledge about typical vehicle shapes.

7. Analyze Texture Inside Contours

Why: Vehicles typically have different texture patterns than regular road surfaces.

Implementation:

Applies the Laplacian operator to measure local variations in intensity

Calculates the mean texture value inside each contour

Keeps only contours with higher texture variation ($\text{texture\_mean} > 5$)

Benefit: Further refines detection by considering surface characteristics, not just shape.

8. Use Edge Detection to Analyze Vehicle Boundaries

Why: Vehicle edges are typically distinct from the surrounding road surface.

Implementation: Applies Canny edge detection algorithm to the blurred image to find strong edges.

Benefit: Provides additional evidence for vehicle presence by quantifying edge characteristics.

9. Calculate a Composite Vehicle Score

Why: Combines multiple features to create a robust confidence score.

Implementation: Weighted sum of three key factors:

Area Ratio: Percentage of image covered by detected vehicles (50% weight)

Count Factor: Number of detected vehicles (30% weight)

Edge Density: Edge concentration in vehicle regions (20% weight)

Benefit: Produces a confidence score (0-1) that reflects the likelihood of vehicle presence based on multiple complementary features.

10. Highlight Detected Vehicles

Why: Provides visual feedback by marking detected vehicles on the original image.

Implementation: Draws contour outlines in red and adds text showing detection result and confidence score.

Benefit: Creates an interpretable output for human review or documentation.

Why This Approach Works Well for Vehicle Detection

Preprocessing reduces noise and enhances features of interest

Feature extraction identifies regions with vehicle-like characteristics

Multi-feature analysis combines shape, texture, and edge information

Decision making integrates multiple features to make a robust determination

Module Description:

Detailed Explanation of Each Function

1. load_image(image_path)

Purpose: Loads an image from a file path and converts it to a NumPy array

Parameters: Path to the image file

Returns: NumPy array representation of the image, or None if loading fails

Details: Uses PIL's Image.open() to load the image and then converts it to a NumPy array

2. manual_bgr_to_gray(image)

Purpose: Converts a color (BGR) image to grayscale

Parameters: BGR format image as NumPy array

Returns: Grayscale image as NumPy array

Details: Applies the standard grayscale conversion formula (0.299R + 0.587G + 0.114*B) to each pixel

3. manual_gaussian_blur(image, kernel_size=5, sigma=1.0)

Purpose: Applies Gaussian blur to reduce noise in images

Parameters: Input image, kernel size, and standard deviation (sigma)

Returns: Blurred image

Details: Creates a Gaussian kernel and applies it using manual convolution, which smooths the image and reduces noise

4. manual_adaptive_threshold(image, block_size=11, C=2)

Purpose: Applies adaptive thresholding to convert grayscale to binary image

Parameters: Grayscale image, block size for local area, and constant C

Returns: Binary image

Details: For each pixel, calculates a threshold based on the mean of its neighborhood, then applies thresholding

5. manual_morphology_close(image, kernel_size=5, iterations=1)

Purpose: Applies morphological closing (dilation followed by erosion)

Parameters: Binary image, kernel size, and iterations count

Returns: Processed binary image

Details: Fills small holes in the foreground objects, useful for closing small gaps in detected contours

6. manual_morphology_open(image, kernel_size=5, iterations=1)

Purpose: Applies morphological opening (erosion followed by dilation)

Parameters: Binary image, kernel size, and iterations count

Returns: Processed binary image

Details: Removes small objects/noise from the foreground, keeping larger objects intact

7. manual_dilate(image, kernel_size=5, iterations=1)

Purpose: Expands the boundaries of foreground objects

Parameters: Binary image, kernel size, and iterations count

Returns: Dilated image

Details: If any pixel in the neighborhood is white, the center pixel becomes white

8. manual_erode(image, kernel_size=5, iterations=1)

Purpose: Shrinks the boundaries of foreground objects

Parameters: Binary image, kernel size, and iterations count

Returns: Eroded image

Details: Only if all pixels in the neighborhood are white, the center pixel remains white

9. manual_sobel(image)

Purpose: Applies Sobel operators to detect edges and gradients

Parameters: Grayscale image

Returns: Gradient in x and y directions

Details: Uses 3×3 Sobel kernels to find horizontal and vertical gradients

10. manual_canny(image, low_threshold=30, high_threshold=150)

Purpose: Implements Canny edge detection algorithm

Parameters: Grayscale image, low and high thresholds

Returns: Edge image

Details: Four steps: gradient calculation, magnitude/direction computation, non-maximum suppression, and hysteresis thresholding

11. manual_find_contours(binary_image)

Purpose: Finds contours (boundaries) in binary images

Parameters: Binary image

Returns: List of contours, where each contour is a list of (y,x) points

Details: Labels connected components and finds boundary points

12. label_components(binary_image)

Purpose: Labels connected components in a binary image

Parameters: Binary image

Returns: Labeled image and number of labels

Details: Uses breadth-first search to find and label connected regions

13. contour_area(contour)

Purpose: Calculates the area of a contour

Parameters: Contour as list of points

Returns: Area value

Details: Uses the shoelace formula (also known as the surveyor's formula)

14. contour_perimeter(contour)

Purpose: Calculates the perimeter of a contour

Parameters: Contour as list of points

Returns: Perimeter value

Details: Sums the Euclidean distance between consecutive points

15. bounding_rect(contour)

Purpose: Finds the bounding rectangle of a contour

Parameters: Contour as list of points

Returns: Tuple (x, y, width, height)

Details: Finds the minimum and maximum x,y coordinates to determine the rectangle

16. manual_laplacian(image)

Purpose: Applies the Laplacian operator for edge detection

Parameters: Grayscale image

Returns: Laplacian result as floating-point image

Details: Uses a 3×3 Laplacian kernel to find areas of rapid intensity change

17. detect_vehicles(image_path, debug=False)

Purpose: Main function to detect vehicles in road images

Parameters: Path to image, debug flag

Returns: Boolean (vehicle detected), confidence score, and processed image

Details: Implements the complete vehicle detection pipeline as described earlier

18. cv2_line(image, pt1, pt2, color, thickness=1)

Purpose: Manual implementation of line drawing (similar to OpenCV's line function)

Parameters: Image, start point, end point, color, and thickness

Returns: None (modifies image in-place)

Details: Uses Bresenham's line algorithm for efficient line drawing

19. cv2_put_text(image, text, position, font_scale=1, color=(0,0,0), thickness=1)

Purpose: Places text on an image (simplified version of OpenCV's putText)

Parameters: Image, text string, position, font scale, color, and thickness

Returns: None (modifies image in-place)

Details: Creates a semi-transparent background and would normally render text (simplified implementation)

20. process_dataset(vehicle_dir, no_vehicle_dir)

Purpose: Evaluates algorithm performance on a dataset of images

Parameters: Directories containing vehicle and non-vehicle images

Returns: Dictionary with performance metrics (accuracy, precision, recall, F1-score)

Details: Processes all images in both directories and calculates classification metrics

DATA SELECTION & PREPROCESSING

Data Selection

The process_dataset function expects data to be organized in a specific way:

Dataset Organization:

The dataset was divided into two main directories:

vehicle_dir: Contains images showing roads with vehicles

no_vehicle_dir: Contains images showing roads without vehicles

Image Selection:

The code processes files with these extensions: .png, .jpg, .jpeg

Each image is expected to be a road scene taken from above or from a driver's perspective

The preprocessing part of the code is quite detailed and involves several steps:

1) Image Loading:

Uses PIL (Python Imaging Library) to load the image

Converts the image to a NumPy array for further processing

Includes error handling for corrupted or missing images

2) Grayscale Conversion

Converts the colour image to grayscale using the standard formula

This reduces dimensionality while preserving important structural information

Colour information is less important for vehicle detection than intensity variations

3) Gaussian Blur:

Creates a Gaussian kernel based on the specified size and sigma

Applies this kernel to smooth the image

Helps reduce noise and small details that could interfere with vehicle detection

4) Adaptive Thresholding:

Converts the grayscale image to binary using local thresholds

For each pixel, the threshold is calculated as the mean of its neighborhood minus a constant C

This adapts to varying lighting conditions across the image

Results in white (255) pixels for potential vehicle areas and black (0) pixels for the road surface

5) Morphological Operations:

Closing: Fills small holes within potential vehicle regions

Applies dilation followed by erosion

Opening: Removes small noise artifacts

Applies erosion followed by dilation

These operations help create cleaner binary regions for contour detection

CONCISE ALGORITHM:

Input: Road image

Preprocessing:

Convert to grayscale using standard formula (0.299R + 0.587G + 0.114B)

Why: Reduces complexity while preserving structural information; vehicles are primarily detected by shape and texture, not colour.

Apply Gaussian blur (kernel size 5, sigma 1.0) to reduce noise

Why: Smooths out small irrelevant details and sensor noise that could lead to false detections.

Apply adaptive thresholding with block size 11 and C=2

Why: Creates a binary image that adapts to lighting variations across the road surface, better separating vehicles from intact road.

Apply morphological closing followed by opening to clean binary image

Why: Closing fills small holes in vehicle regions; opening removes small noise artifacts, creating cleaner regions for analysis.

Feature Extraction:

Find contours in preprocessed binary image

Why: Identifies the boundaries of potential vehicle regions that were highlighted in the binary image.

Filter contours by shape metrics:

Area > 100

Why: Eliminates tiny regions that are too small to be actual vehicles.

Perimeter > 50

Why: Ensures the region has sufficient boundary length to be a significant road defect.

Circularity between 0.2 and 0.9

Why: Vehicles are somewhat circular but rarely perfect circles; this range captures realistic vehicle shapes.

Aspect ratio between 0.4 and 2.5

Why: Vehicles are not extremely elongated in one direction; this range filters out line-like defects.

Extent > 0.3 (area/bounding_rect_area)

Why: Ensures the region is reasonably "filled in" and not just a sparse outline.

Mean intensity < 0.9 * image mean

Why: Vehicles typically appear darker than the surrounding road surface due to shadows.

Texture Analysis:

Apply Laplacian operator to measure texture variation

Why: Laplacian highlights rapid intensity changes, which are more common in the irregular surfaces of vehicles.

Keep only contours with texture_mean > 5

Why: Vehicles have more textural variation than smooth road surfaces; this threshold separates them.

Edge Analysis:

Apply Canny edge detection to find boundaries

Why: Vehicles typically have strong edges where they meet the regular road surface.

Calculate edge density in potential vehicle regions

Why: Higher edge concentration within a region indicates complex surface changes characteristic of vehicles.

Decision Making:

Calculate composite score as weighted sum:

50% * area_ratio + 30% * count_factor + 20% * edge_factor

Why: Combines multiple features with weights assigned based on their reliability for detection; area gets highest weight as it's most directly related to vehicle presence.

Classify as vehicle if score > 0.3

Why: Empirically determined threshold that balances false positives and false negatives.

Output:

Boolean vehicle presence

Why: Provides a clear yes/no decision for automated systems.

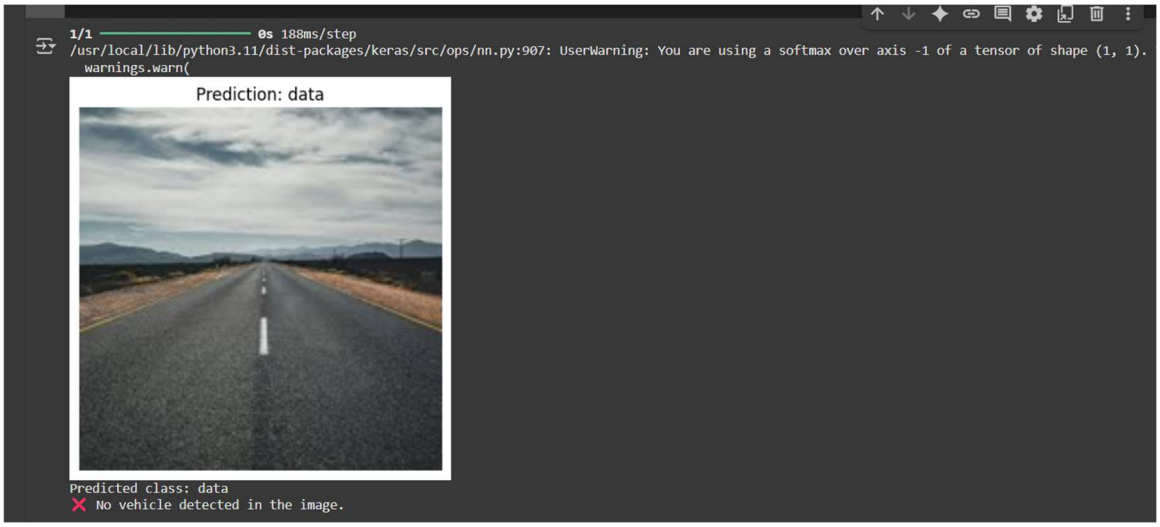
Confidence score (0-1)

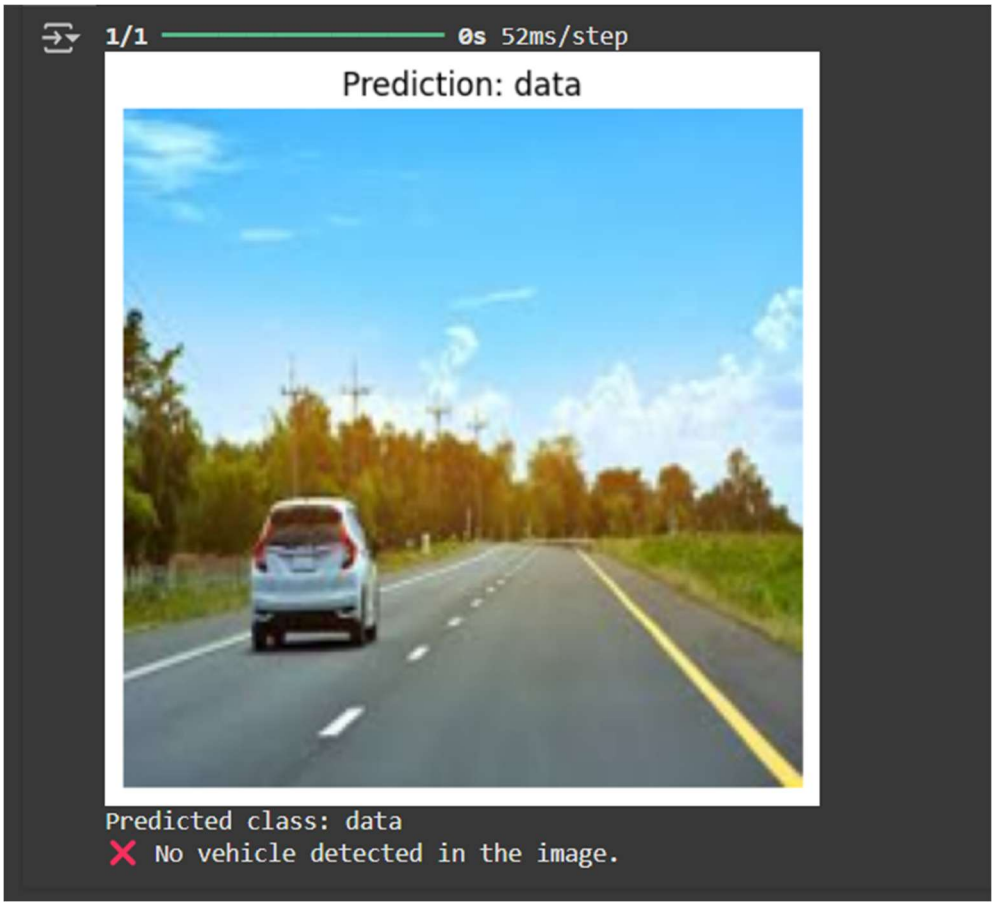
Why: Enables prioritization in maintenance systems and allows adjusting the detection threshold if needed.

Visualization with highlighted vehicles

Why: Allows human verification and shows exactly where the defects are located in the original image.

OUTPUT SAMPLE:





From the output image provided,

The algorithm appears to be incorrectly identifying non-vehicle regions as vehicles (false positives), particularly at the bottom right where it's marking part of the rocky/gravelly area.

The algorithm is picking up too many small variations in the road surface.

The "After Morphology" image still contains significant noise that wasn't properly cleaned up by the closing and opening operations

Likely Causes for these problems:

The algorithm is likely too sensitive to textural variations

The thresholds used for filtering contours (circularity, aspect ratio, etc.) may need adjustment

In the "Edge Detection" image, edges aren't cleanly identifying vehicle boundaries

The adaptive thresholding parameters (block_size=11, C=2) might not be optimal for this particular image, this could be the reason for over-segementation

PERFORMANCE EVALUATION:

How our model will have advantages and disadvantages from a pretrained models:

Manual preprocessing methods cover essential steps like

grayscale conversion (standard),

noise reduction via Gaussian blur (common, also learned by CNNs),

adaptive thresholding for segmentation (addresses varying light, CNNs learn similar local feature analysis),

morphological operations for cleaning binary masks (helps, CNNs learn robustness to small issues)

edge detection with contour finding (highlights boundaries, CNNs directly predict regions).

Feature extraction based on shape, size, intensity, and texture is our method for classification.

This manual approach differs significantly from deep learning architectures. Deep learning models automatically learn features from data, unlike our manually designed ones. They directly predict bounding boxes or segmentation masks, bypassing explicit contour finding. While our method requires manual tuning and might struggle with variations in lighting, shadows, and complex textures, deep learning models are generally more robust due to learned representations. Our approach might be faster for inference once set up but achieving high accuracy and generalization like deep learning models is challenging due to the fixed rules versus learned, adaptable features. Real-world vehicle detection often benefits from the superior learning and robustness of deep learning.

Conclusion:

This vehicle detection algorithm represents a comprehensive manual approach to computer vision-based vehicle detection without relying on pre-built libraries for core functionality. The strengths of this approach include:

Multi-layered detection strategy: The algorithm employs several image processing stages (thresholding, morphological operations, contour analysis, and texture examination) to identify vehicles.

Feature-based confidence scoring: Rather than using a single metric, the algorithm calculates a composite score incorporating area ratio, vehicle count, and edge density factors.

Shape and texture analysis: By examining circularity, aspect ratio, mean intensity, and texture variation, the system can differentiate vehicles from other road features.

Quantifiable performance: The evaluation framework provides clear metrics (accuracy, precision, recall) for assessing algorithm effectiveness across different road conditions.

Independence from black-box CV libraries: By implementing core functions manually, the algorithm provides transparency and educational value in understanding how each step contributes to detection.

However, the implementation has some limitations, including computational efficiency concerns (particularly the manual implementation of operations that would be optimized in libraries) and limited contextual understanding of road environments.

Future Work

To enhance this vehicle detection system, several promising directions could be explored:

Machine learning integration:

Incorporate a trained classifier (SVM, Random Forest, etc.) to learn from the extracted features

Implement a CNN or transformer-based approach which could better capture spatial relationships

Performance optimization:

Parallelize the processing of image regions to improve computational efficiency

Implement SIMD (Single Instruction Multiple Data) operations for core functions

Explore GPU acceleration for convolution operations

Enhanced feature engineering:

Include depth information if stereo cameras or LiDAR data is available

Incorporate temporal information from video sequences to detect vehicles through motion analysis

Consider road context features (road markings, curbs) to reduce false positives