Traffic Sign Classification Using Classical Digital Image Processing Techniques

Deadline 9th May, 2025 11:55 pm

General Instructions

- This is a group-based project. Each group must consist of exactly **4 students**.
- All students in the group are required to **individually upload the same project submission**.
- The project submission must be compressed into a single `.zip` file.
- The **filename format** must be: `i211234_i211234_i211234_i211234.zip`
- The `.zip` file must contain the following:
  - `/code/` – a folder containing all implemented Python modules and helper scripts or Jupyter notebook
  - `results.csv` – a CSV file logging predicted and ground-truth labels
  - `metrics.txt` – file summarizing overall accuracy and class-wise metrics
  - `confusion_matrix.png` – heatmap showing the confusion matrix
- Submissions not adhering to the format or missing required files may result in mark deductions.

Project Context and Motivation

Digital image processing (DIP) forms the foundation of many computer vision applications, long before the rise of machine learning. This project revisits classical approaches to image analysis and classification through a real-world use case: **traffic sign classification**.

Rather than relying on pretrained models or learning algorithms, students will implement their own end-to-end image analysis pipeline. Each step—filtering, segmentation, feature extraction, and classification—must be constructed manually using fundamental image processing concepts covered in the course.

This project is an opportunity to deeply understand and apply the core operations of DIP in a meaningful application, testing your problem-solving ability, programming skills, and grasp of computer vision theory.

Project Objective

To develop a complete, rule-based traffic sign classifier that operates entirely using classical image processing techniques. The system must be capable of identifying the class of a sign from a cropped image using color, shape, and geometric features, and evaluate its classification performance using ground-truth labels.

Dataset Description

Dataset Access

The dataset required for this project can be accessed from the following link:
`Download Dataset from Google Drive`

The dataset consists of a pre-cropped image set of traffic signs from multiple classes. The directory structure is as follows:

```
/Train/
 0/       ← Speed Limit 20 km/h
 1/       ← Speed Limit 30 km/h
 ...
 Train.csv
```

Each subfolder contains PNG images corresponding to a single class. The `Train.csv` file includes:

- **Path**: Relative path to the image
- **ClassId**: Integer label representing the sign

Only the `Path` and `ClassId` columns are relevant to this project.

Project Requirements

1. Class Selection

- Select **6 to 8 different traffic sign classes** of your choice.
- From each class, extract approximately **100 images**, resulting in a working dataset of **600–800 images**.

2. Pipeline Implementation

The classification pipeline must include the following stages:

a. Image Reading

- Use either OpenCV (`cv2.imread`) or PIL (`Image.open`) to load images.
- No further operations using these libraries are permitted—image processing must be performed using NumPy only.

b. Preprocessing and Filtering Implement the following filters from scratch:

- Mean Filter (3×3)
- Gaussian Filter (with standard deviation)
- Median Filter
- Adaptive Median Filter
- Unsharp Masking and/or High-Boost Filtering

c. Color Space Conversion and Segmentation

- Convert RGB images to HSV color space manually or using PIL.
- Segment based on:
  - Red signs: Hue in [0–15] or [165–180], Saturation 100, Value 80
  - Blue signs: Hue in [100–130], Saturation 100, Value 80
- Post-processing should include:
  - Binary mask thresholding
  - Morphological operations (erosion, dilation, opening)
  - Noise removal via connected component filtering (area threshold)
  - Hole filling

d. Edge Detection Manually implement the Canny edge detection algorithm, including:

- Gradient computation (Sobel operator)
- Non-maximum suppression
- Double thresholding and edge tracking

e. Geometric Normalization Use affine transformations to:

- Rotate the sign to an upright orientation
- Scale to a fixed size (e.g., 200×200 pixels)
- Optionally, apply a perspective transform

All transformations must be implemented using NumPy.

f. Feature Extraction Extract the following features from the normalized image:

- Corner Count (via Harris Corner Detection)
- Circularity: $C = \frac{4\pi \times \text{Area}}{(\text{Perimeter})^2}$
- Aspect Ratio (width/height of bounding box)
- Extent (ratio of region area to bounding box area)
- Average Hue

g. Rule-Based Classification Construct a set of `if-else` rules that use extracted features to assign a `ClassId`. These rules must:

- Use both color and shape features
- Distinguish visually similar signs (e.g., Stop vs. Yield)
- Be clearly justified and interpretable

3. Evaluation Procedure

Compare predicted labels with ground-truth from `Train.csv`. The following output files are required:

- `results.csv`: One row per image with columns `filename, ground_truth, predicted, correct`
- `metrics.txt`: Contains overall accuracy and class-wise precision, recall, and accuracy
- `confusion_matrix.png`: Heatmap visualizing the confusion matrix