Traffic Sign Recognition Project Documentation

Theodoros Milad

May 6, 2025

Helwan University
Faculty of Computer and Artificial Intelligence
Department of Artificial Intelligence

Abstract

This document details a project focused on the recognition of traffic signs. A system was developed capable of image input, identification of traffic sign class, and visualization of underlying computer vision processes. This report discusses the problem domain, the dataset utilized (German Traffic Sign Recognition Benchmark), and the primary image processing methodologies implemented, including classical feature-based approaches and a Convolutional Neural Network (CNN). Comparative analysis of these methods is also presented.

Contents

1 Introduction: The Challenge of Traffic Sign Recognition

Automated traffic sign recognition (TSR) is a critical component for Advanced Driver Assistance Systems (ADAS) and autonomous vehicle navigation. These systems rely on accurately interpreting road signs to make informed decisions, such as adhering to speed limits, recognizing stop or yield conditions, and understanding lane directives.

However, robust TSR in real-world scenarios presents numerous challenges:

- **Illumination Variability:** Signs may appear under diverse lighting conditions, from bright daylight to shadowed areas or nighttime, affecting their visual appearance.
- **Adverse Weather Conditions:** Rain, fog, snow, or hail can obscure signs, reduce visibility, and degrade image quality.
- **Occlusion and Damage:** Signs can be partially occluded by objects (e.g., foliage, other vehicles) or may be physically damaged, faded, or defaced.
- **Viewpoint and Scale Variations:** Signs are observed from various distances and angles, leading to significant changes in their apparent size and perspective distortion.
- **Intra-class Variability:** Even within the same sign class, variations in design, material, and age can exist.

This project addresses these challenges by developing and evaluating systems for classifying traffic signs from images.

2 Dataset: German Traffic Sign Recognition Benchmark (GTSRB)

The primary dataset employed for training and evaluating the TSR systems is the German Traffic Sign Recognition Benchmark (GTSRB). This dataset is a widely used standard in the field, originally introduced as part of the IJCNN 2011 competition.

2.1 Dataset Composition

The GTSRB training set is structured as follows:

- **Class-Specific Directories:** The dataset comprises 43 distinct classes of traffic signs, with each class housed in a separate directory (named `00000` through `00042`).
- **Training Images:** Each directory contains numerous images corresponding to its specific traffic sign class.
- **Annotation Files:** Accompanying each class directory is a CSV (Comma Separated Value) file providing annotations for the images (e.g., `GT-00000.csv`).

2.2 Image Characteristics

- **Format:** Images are provided in the PPM (Portable PixMap) format, representing RGB color.
- **Naming Convention:** Image filenames follow the pattern `XXXXX_YYYYY.ppm`.
  - `XXXXX`: Denotes the track number. Images sharing the same track number originate from the same physical traffic sign, potentially captured under slightly different conditions or viewpoints in sequence.
  - `YYYYY`: A sequential running number for images within a given track, preserving temporal order.

2.3 Annotation File Structure

Annotations are stored in CSV files, using a semicolon (`;`) as the field delimiter. Each record provides metadata for an image:

- `Filename`: The name of the image file.
- `Width`, `Height`: The dimensions of the image in pixels.
- `Roi.x1`, `Roi.y1`, `Roi.x2`, `Roi.y2`: Coordinates defining the Region of Interest (ROI) that bounds the traffic sign within the image. The dataset specifies that these ROIs include a border around the sign (approximately 10% of the sign's size, with a minimum of 5 pixels).
- `ClassId`: The numerical identifier for the traffic sign class (0-42).

2.4 Further Information

Additional details regarding the GTSRB can be found on the official benchmark website: http://benchmark.ini.rub.de/.

3 Methodology 1: Classical Computer Vision Techniques

The initial approach involved traditional computer vision methods for feature extraction, followed by a Support Vector Machine (SVM) for classification.

3.1 Histogram of Oriented Gradients (HOG)

- **Principle:** HOG features describe local object appearance and shape by the distribution of intensity gradients or edge directions. The image is divided into small connected regions (cells), and for each cell, a histogram of gradient directions for the pixels within the cell is compiled. These histograms are then contrast-normalized by grouping cells into larger blocks.
- **Relevance to TSR:** Traffic signs possess distinct shapes and edge structures that HOG can effectively capture, offering a degree of invariance to illumination and minor geometric variations.
- **Implementation (`skimage.feature.hog`):** Key parameters included:
  - `pixels_per_cell=(8, 8)`: Defines the size of individual cells for gradient orientation histograms.
  - `cells_per_block=(2, 2)`: Specifies the number of cells to group into a block for normalization.
  - `block_norm='L2-Hys'`: The normalization method applied to blocks (L2-Hysteresis).
  - `visualize=True`: Enables generation of a visual representation of the HOG features.
  - `feature_vector=True`: Returns the HOG features as a flattened 1D array suitable for classifiers.
- **Mechanism:** The HOG descriptor is formed by concatenating the normalized histograms from all blocks in the detection window.

3.2 Scale-Invariant Feature Transform (SIFT)

- **Principle:** SIFT is an algorithm to detect and describe local features (keypoints) in images. SIFT keypoints are designed to be invariant to image scale, rotation, and partially invariant to changes in illumination and 3D viewpoint. Each keypoint is described by a high-dimensional vector representing the local image structure.
- **Relevance to TSR:** While not the primary classification feature in this SVM pipeline, SIFT keypoints are valuable for tasks like image matching or object recognition under varying conditions. They were visualized to illustrate robust local feature detection.
- **Implementation (`cv2.SIFT_create` and `detectAndCompute`):** The process involves:
  1. Keypoint Detection: Identifying stable interest points across different scales of the image (using Difference of Gaussians).
  2. Keypoint Localization: Refining keypoint locations and eliminating unstable ones.
  3. Orientation Assignment: Assigning one or more orientations to each keypoint based on local image gradient directions.
  4. Descriptor Generation: Computing a local image descriptor for the region around each keypoint, based on oriented gradient histograms.

3.3 Sobel Edge Detection

- **Principle:** The Sobel operator performs a 2D spatial gradient measurement on an image, emphasizing regions of high spatial frequency that correspond to edges. It uses two convolution kernels (one for horizontal changes, one for vertical) to approximate the gradient components.
- **Relevance to TSR:** Edge information is fundamental for defining the shape and boundaries of traffic signs. Sobel edges were visualized to show this structural information.
- **Implementation (`cv2.Sobel`):**
  - Applied to a grayscale image.
  - `cv2.CV_64F`: Output image depth to handle potential negative gradient values.
  - Derivatives calculated separately for x (`dx=1, dy=0`) and y (`dx=0, dy=1`) directions.
  - Kernel size typically 3 (`ksize=3`).
  - The gradient magnitudes from `sobelx` and `sobely` are combined using `cv2.magnitude(sobelx, sobely)`.
  - The resulting magnitude image is normalized to the 0-255 range for visualization.

3.4 Support Vector Machine (SVM) Classifier

The HOG features extracted from the training images were used to train an SVM classifier. SVMs are supervised learning models that find an optimal hyperplane to separate data points of different classes in a high-dimensional space.

- **Training Hardware:** CPU.
- **Training Time:** Approximately 1.30 seconds.
- **Test Accuracy (Classification):** Achieved 80.1%.

4 Methodology 2: Deep Learning with Convolutional Neural Networks (CNNs)

A second approach utilized a CNN, which can learn hierarchical features directly from raw pixel data, integrating feature extraction and classification.

4.1 CNN Architecture: `SimpleResCNN`

A custom CNN architecture, named `SimpleResCNN` (as per the provided Python code), was implemented. This architecture includes convolutional layers, batch normalization, ReLU activations, max-pooling layers, and a residual connection. The network (based on the provided Python snippet) comprises:

1. **Input Layer:** Accepts images (e.g., $64 \times 64 \times 3$).
2. **Initial Convolutional Block (`conv1`):** Two convolutional layers (32 filters, kernel 3×3) with batch normalization and ReLU, followed by max pooling. (Output e.g., $32 \times 32 \times 32$)
3. **Residual Convolutional Block (`conv2_main`, `conv2_skip`):** A main path with two convolutional layers (64 filters, kernel 3×3) and a skip connection (`Conv2d` with 1×1 kernel) to enable residual learning. Followed by ReLU and max pooling. (Output e.g., $16 \times 16 \times 64$)
4. **Final Convolutional Block (`conv3`):** Two convolutional layers (128 filters, kernel 3×3) with batch normalization and ReLU, followed by max pooling. (Output e.g., $8 \times 8 \times 128$)
5. **Global Average Pooling (`gap`):** Reduces spatial dimensions to 1×1 per feature map.
6. **Output Heads (Fully Connected Layers):**
   - **Classification Head (`fc_class`):** Linear layer mapping pooled features to `num_classes` (43 for GTSRB).
   - **Bounding Box Regression Head (`fc_bbox`):** Linear layer mapping pooled features to 4 outputs (for normalized bounding box coordinates, e.g., $x1, y1, x2, y2$).

This multi-head architecture allows the network to perform both classification and localization (bounding box prediction) tasks simultaneously.

4.2 Dataset Handling: `GTSRBDataset` Class

A custom PyTorch `Dataset` class (`GTSRBDataset`) was implemented to load and preprocess images from the GTSRB dataset. Key functionalities:

- Loads image paths and annotations (labels, bounding boxes) for training or testing splits.
- Processes each image: reads, resizes (e.g., to 64×64), normalizes pixel values (to [0, 1]), and converts to grayscale if needed for specific feature extractors (though the CNN typically uses color images).
- Converts images to PyTorch tensors in CHW (Channels, Height, Width) format.
- Normalizes bounding box coordinates to a [0, 1] range relative to the image dimensions.
- Optionally, it can be configured to extract and return classical features like HOG or SIFT, though for the CNN, raw (or transformed) image tensors are primary.

PyTorch `DataLoader` was used to create batches of data for training and evaluation.

4.3 Loss Functions and Optimizer

- **Classification Loss:** Cross-Entropy Loss (`nn.CrossEntropyLoss`).
- **Bounding Box Regression Loss:** Smooth L1 Loss (`nn.SmoothL1Loss`).
- **Combined Loss:** $L_{total} = L_{cls} + L_{box}$ (assuming equal weighting).
- **Optimizer:** AdamW (`optim.AdamW`) with a learning rate of 1×10^{-3}.

4.4 Experimental Setup and Results (CNN)

- **Hardware:** Training performed on a GPU (NVIDIA CUDA).
- **Framework:** PyTorch.
- **Training Epochs:** 5.
- **Batch Size:** 128.
- **Training Time:** Approximately 2 minutes 30 seconds for 5 epochs.
- **Performance:**
  - **Training Accuracy (Classification):** 99.9%.
  - **Test Accuracy (Classification):** 96.84%.
  - **Bounding Box Prediction:** The model concurrently predicted bounding boxes.

5 Supplementary Visualization: K-Means Clustering for Image Segmentation

K-Means clustering was applied as an unsupervised method for image segmentation, primarily for visualization purposes.

5.1 Theory of K-Means Clustering

K-Means aims to partition n data points into k distinct clusters. Each data point is assigned to the cluster with the nearest mean (centroid). The algorithm iteratively:

1. Initializes k centroids.
2. Assigns each data point to its closest centroid.
3. Recalculates each centroid as the mean of all points assigned to it.

This process repeats until convergence. For image segmentation, pixels (represented by their color vectors, e.g., RGB) are the data points.

5.2 Application and Visualization

The `segment_image_kmeans` function (using `sklearn.cluster.KMeans`) reshapes the image pixels into a 2D array, applies K-Means to group pixels by color similarity into k clusters (e.g., $k = 4$ or $k = 10$ in experiments), and then reconstructs the image where each pixel is colored by its assigned cluster's centroid. This produces a segmented version of the image, highlighting dominant color regions.

6 Comparative Analysis: Classical SVM vs. Deep Learning CNN

Table 1 summarizes the comparison between the two approaches.

Table 1: Comparison of Classical SVM and Deep Learning CNN Approaches.

Aspect	**Classical SVM with HOG**	**Deep Learning CNN (`SimpleResCNN`)**
Feature Engineering	Manual (HOG)	Automatic (Learned by network)
Model Complexity	Relatively Lower	Higher
Training Hardware	CPU	GPU (CUDA)
Training Time (approx.)	1.30 seconds	2 min 30 sec (5 epochs)
Test Accuracy (Class.)	80.1%	**96.84%**
Bounding Box Prediction	No (Separate algorithm needed)	**Yes (Integrated)**
Interpretability	HOG features somewhat interpretable	"Black box" (generally)
Data Requirement	Effective with moderate data	Benefits from larger datasets

Discussion: The CNN (`SimpleResCNN`) significantly outperformed the SVM in classification accuracy (96.84% vs. 80.1%). This highlights the ability of deep networks to learn highly discriminative features. The CNN also integrated bounding box prediction, a crucial advantage. While the SVM trained faster on CPU for its specific task, the CNN's training on GPU, though longer per epoch, achieved a much more comprehensive solution. The CNN automates feature learning, whereas the classical method relies on hand-crafted HOG features.

7 Conclusion

This project demonstrated two approaches to GTSRB traffic sign recognition. The classical SVM with HOG features provided a baseline, achieving 80.1% test accuracy. The `SimpleResCNN` deep learning model significantly improved upon this, reaching 96.84% test accuracy and concurrently performing bounding box regression. This underscores the power of CNNs for complex visual recognition tasks. K-Means clustering was also explored for unsupervised segmentation visualization. The CNN approach, despite requiring GPU resources, offers a more robust and functionally complete solution. Future work could involve more advanced CNN architectures, data augmentation, and optimization for real-time deployment.