159731 Studies in Computer Vision

June 2024

Comparing Performance of Preprocessing Techniques for Traffic Sign Recognition using a HOG-SVM

LUIS VIEIRA

School of Mathematical and Computational Sciences
Massey University – Albany Campus,
Auckland, New Zealand
23012096@massey.ac.nz

TE KUNENGA KI PŪREHUROA
MASSEY UNIVERSITY
UNIVERSITY OF NEW ZEALAND
TE WĀHANGA PŪTAIAO
COLLEGE OF SCIENCES

INTRODUCTION

Context & Motivation

- Traffic Sign Recognition (TSR) is critical for Advanced Driver Assistance Systems (ADAS) and autonomous vehicles, ensuring road safety and efficient navigation

Objective

- Compare the performance of various preprocessing techniques for TSR using Histogram of Oriented Gradients (HOG) and Support Vector Machine (SVM)

Dataset

- The German Traffic Sign Recognition Benchmark (GTSRB)

DATASET

Overview

The GTSRB dataset contains over 50,000 images across 43 classes of traffic signs.

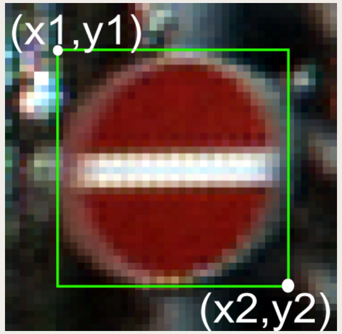
Importance

This dataset provides a comprehensive and diverse set of traffic sign images, ideal for evaluating TSR techniques under varied conditions.

Characteristics

- Images vary in size from 15x15 to 250x250 pixels.
- Each image includes a border of 10% around the traffic sign, aiding edge-based approaches.
- Annotated with class labels and bounding boxes (ROI)

source: dx.doi.org/10.1109/SII.2017.8279326



source: benchmark.ini.rub.de

METHODOLOGY

Compared Preprocessing Techniques

(CLAHE, HUE, YUV, and mix) + Gaussian Blur

Feature Extraction Method:

Histogram of Oriented Gradients (HOG)

Classifier

Support Vector Machine (SVM) with RBF kernel

Tools

C++17, Cmake 3.29.0, VSCode 1.90.0, Python 3.11.2 (for EDA & RandomizedCV)

IMPLEMENTATION

Dataset Handling

- Shuffling to reduce class imbalance

```
void splitDataset(const std::vector<Annotation>& annotations, std::vector<Annotation>& trainSet, std::vector<Annotation>& valSet, float trainRatio) {
std::vector<Annotation> shuffled = annotations;
std::mt19937 g(123);
std::shuffle(shuffled.begin(), shuffled.end(), g);
size_t trainSize = static_cast<size_t>(trainRatio * annotations.size());
trainSet.assign(shuffled.begin(), shuffled.begin() + trainSize);
valSet.assign(shuffled.begin() + trainSize, shuffled.end());
}
```

- Standard resizing images to 32x32 pixels

```
cv::Mat resizedImage = resizeImage(image, 32, 32);
```

- Train-validation split: 80:20 ratio

```
splitDataset(annotations, trainSet, valSet, 0.8);
```

PREPROCESSING IMPLEMENTATION

CLAHE

- Convert and split BGR into L, a, b channels
- Compute mean and std. deviation of L channel to dynamically adjust clip limit:

- Low contrast (std dev < 50): clip limit = 4.0
- Medium contrast (std dev < 100): clip limit = 2.0
- High contrast (std dev >= 100): clip limit = 1.0

- Tile grid size: 8x8

HUE

- Convert and split BGR into H, S, V channels
- Equalise the histogram of the H channel to enhance the contrast of hue values

YUV

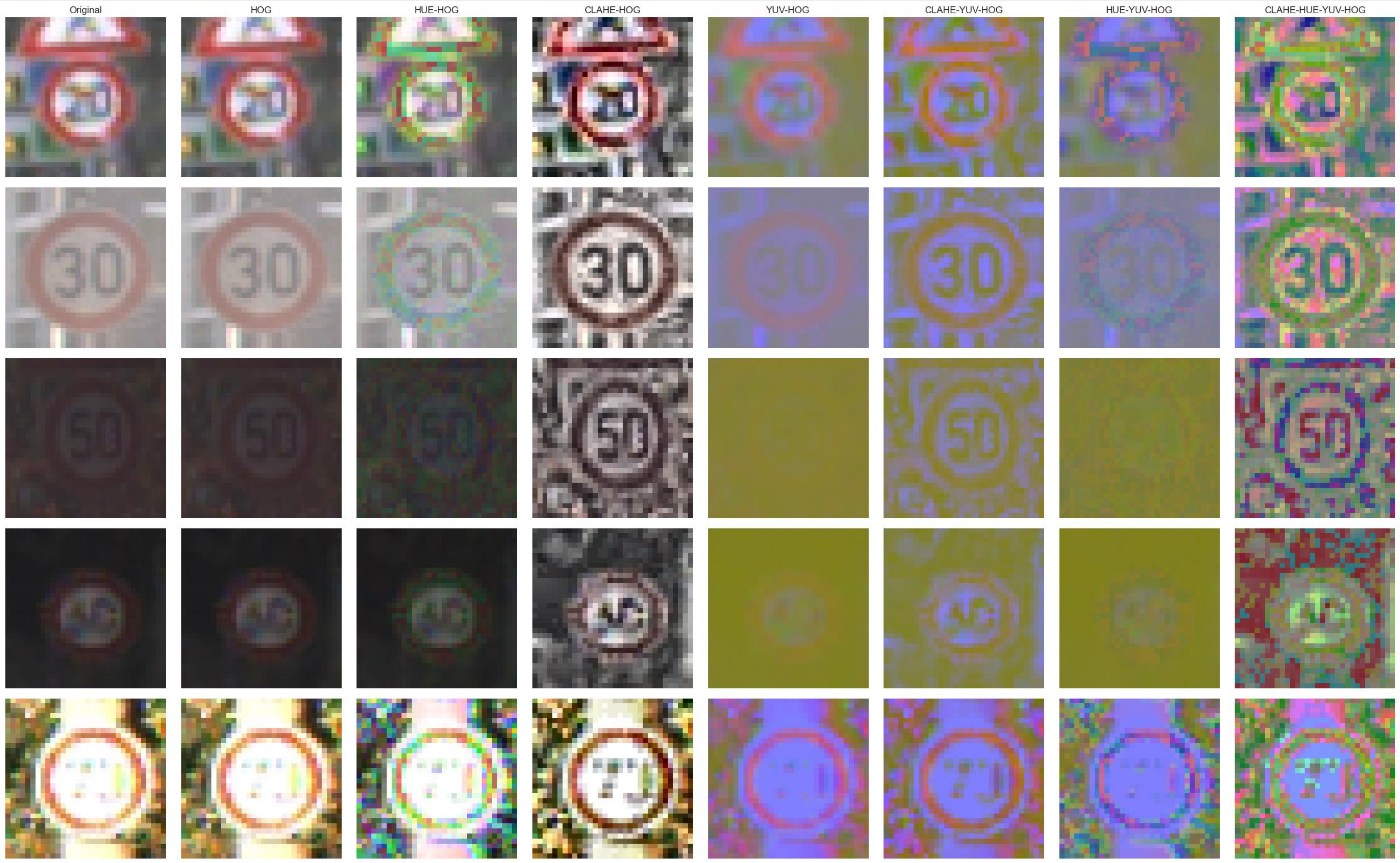
- Convert BGR to YUV

```
void applyCLAHE(cv::Mat& image) {
cv::Mat lab_image;
cv::cvtColor(image, lab_image, cv::COLOR_BGR2Lab);
std::vector<cv::Mat> lab_planes;
cv::split(lab_image, lab_planes);
cv::Scalar mean, stddev;
cv::meanStdDev(lab_planes[0], mean, stddev);
double clipLimit; cv::Size tileGridSize(8, 8);
if (stddev[0] < 50) {clipLimit = 4.0;} else if (stddev[0] < 100) {clipLimit = 2.0;} else {clipLimit = 1.0;}
cv::Ptr<cv::CLAHE> clahe = cv::createCLAHE(clipLimit, tileGridSize);
clahe->apply(lab_planes[0], lab_planes[0]);
cv::merge(lab_planes, lab_image);
cv::cvtColor(lab_image, image, cv::COLOR_Lab2BGR);}

cv::Mat extractHUE(const cv::Mat& image) {
cv::Mat hsv_image, hue_channel;
cv::cvtColor(image, hsv_image, cv::COLOR_BGR2HSV);
std::vector<cv::Mat> hsv_planes;
cv::split(hsv_image, hsv_planes);
cv::Mat hue_channel_equalized;
cv::equalizeHist(hsv_planes[0], hue_channel_equalized);
hsv_planes[0] = hue_channel_equalized;
cv::merge(hsv_planes, hsv_image);
cv::cvtColor(hsv_image, hue_channel, cv::COLOR_HSV2BGR);
return hue_channel;}

cv::Mat yuv_image;
cv::cvtColor(resizedImage, yuv_image, cv::COLOR_BGR2YUV);
```

PREPOSSESSING TECHNIQUES



HOG-SVM IMPLEMENTATION

HOG Parameters

- Gaussian Blur ((3, 3), 0)
- Window size: 32x32 pixels
- Block size: 16x16 pixels
- Stride: 8 pixels
- Cell size: 8x8 pixels
- Bins: 9

```
cv::Mat computeHOG(const cv::Mat& image, int kernelSize = 3, double sigma = 0) {
cv::Mat processedImage;
cv::GaussianBlur(image, processedImage, cv::Size(kernelSize, kernelSize), sigma);
cv::HOGDescriptor hog(cv::Size(32, 32), cv::Size(16, 16), cv::Size(8, 8), cv::Size(8, 8), 9);
std::vector<float> descriptors;
hog.compute(processedImage, descriptors);
return cv::Mat(descriptors).clone().reshape(1, 1);
}
```

SVM Hyperparameters

- $C = 20.5557$
- $\gamma = 0.2167$

```
cv::Ptr<cv::ml::SVM> svmHOG, svmCLAHEHOG, svmYUVHOG, svmHUEHOG, svmCLAHEYUVHOG,
svmHUEYUVHOG, svmCLAHEHUEYUVHOG;
// RandomizedSearchCV 3-fold 10 iteration ('C':(5, 25, 10),'gamma': (0.05, 0.35, 10), ran in python)
// best param: 'C': 9.4445, 11.6667, *20.5557, 22.7778; 'gamma': *0.2167, 0.2833, 0.3166, 0.35
double C = 20.5557;
double gamma = 0.2167;
```

TEST SET METRICS

Method	F1 Score	Accuracy	Precision	Recall
HOG w/ Gaussian Blur (baseline)	0.8793	0.8965	0.9045	0.8648
CLAHE-HOG	0.8741	**0.9033**	0.8859	**0.8681**
YUV-HOG	**0.8909**	**0.9125**	**0.9162**	**0.8765**
HUE-HOG	0.8584	0.8808	0.8835	0.8455
CLAHE-YUV-HOG	0.8732	**0.9049**	0.8877	**0.8651**
HUE-YUV-HOG	0.8676	0.8885	0.8939	0.8529
CLAHE-HUE-YUV-HOG	0.8700	**0.9013**	0.8849	0.8610

Note. Confusion Matrix, Metrics, no. failed images per class, and list of failed images saved to output/ folder.

CONCLUSIONS

Analysis

- Overfitting: some performance drop between validation to test results (around -8.5% in accuracy)
- Class Imbalance: dataset imbalance affecting generalisation.
- Best preprocessing technique: YUV-HOG (91.25% accuracy)

Possible improvements

- Use and compare with PCA/LDA
- Use synthetic data augmentation
- Fine-tunning preprocessing
- Compare with CNN

THANK YOU

LUIS VIEIRA
23012096@massey.ac.nz