**Pedestrian Detection Using Python and OpenCV**

Name: Debashish Shina

Department-School of computer science

Branch-IT-01

Rollno-2206020

Email-2206020@kiit.ac.in

**Abstract**

Pedestrian detection plays a significant role in applications such as autonomous driving, surveillance, and robotics. This project implements a pedestrian detection system using Python and OpenCV. The system utilizes the Histogram of Oriented Gradients (HOG) feature descriptor combined with a Support Vector Machine (SVM) classifier to detect pedestrians in real-time video streams. The approach was tested on publicly available datasets, and its performance was evaluated using metrics like precision, recall, and F1-score. The results show the effectiveness of the method in detecting pedestrians with good accuracy, even in challenging environments.

1. **Introduction**
   1. Background and Motivation

Pedestrian detection is an essential task in computer vision, especially in the context of autonomous vehicles, intelligent transportation systems, and security monitoring. With the rise in pedestrian-related accidents and the development of autonomous vehicles, the need for robust and reliable pedestrian detection systems has grown. Pedestrian detection systems must be able to operate in various real-world scenarios, including diverse lighting conditions, different environments, and varying poses of pedestrians.

* 1. Objective

This project aims to implement a pedestrian detection system using Python and OpenCV. The system is designed to detect pedestrians in real-time video footage using the Histogram of Oriented Gradients (HOG) for feature extraction, followed by classification using Support Vector Machines (SVM).

* 1. Scope of the Project

The scope of this project involves:

Implementing pedestrian detection using HOG feature extraction and SVM classification.

Evaluating the system using a publicly available dataset.

Comparing the performance of the system under different conditions, such as lighting and occlusion.

1. **Literature Review**

Pedestrian detection has been widely studied, with a variety of methods developed over the years. The most notable approaches include:

2.1 Traditional Methods

HOG + SVM: One of the earliest methods used for pedestrian detection is the combination of Histogram of Oriented Gradients (HOG) and Support Vector Machines (SVM). The HOG descriptor captures the gradient structure of images, which is particularly effective for detecting human shapes.

Viola-Jones Algorithm: This method uses Haar-like features and a cascade of classifiers to detect objects such as faces and pedestrians.

2.2 Deep Learning Approaches

Convolutional Neural Networks (CNNs): More recently, deep learning methods have been employed, particularly CNNs, which can learn hierarchical features and perform better in complex environments.

YOLO (You Only Look Once) and SSD (Single Shot Multibox Detector) are some examples of deep learning-based methods that offer real-time pedestrian detection.

2.3 Comparative Studies

Studies have shown that while deep learning approaches tend to outperform traditional methods in complex scenarios, the HOG + SVM approach is still efficient and effective, especially when computational resources are limited.

1. **Methodology**

3.1 Tools and Libraries

Programming Language: Python

Libraries: OpenCV (for image processing), Numpy (for numerical operations), Matplotlib (for plotting), and Scikit-learn (for machine learning).

3.2 Dataset

This project utilizes the INRIA Person Dataset, a well-known dataset for pedestrian detection tasks. The dataset includes images of pedestrians under various conditions and is commonly used to evaluate detection algorithms.

3.3 Pedestrian Detection Method

The system consists of two main stages:

Feature Extraction (HOG): The Histogram of Oriented Gradients (HOG) technique is used to extract features from images. It involves:

Converting the image to grayscale.

Dividing the image into small cells and computing gradient orientations.

Aggregating the gradients into histograms.

These histograms are then used as features to detect pedestrians.

Classification (SVM): The Support Vector Machine (SVM) classifier is trained using the HOG features to classify image regions as pedestrian or non-pedestrian. The SVM finds the optimal hyperplane that separates the two classes.

3.4 Implementation Steps

Preprocessing:

Convert the input image to grayscale.

Resize and normalize the image for consistent input size.

Feature Extraction:

Extract HOG features from each image frame.

Detection:

Use the trained SVM classifier to classify image regions based on the extracted features.

Post-processing:

Apply techniques like non-maximum suppression (NMS) to refine the results by eliminating redundant detections.

1. **Results and Discussion**

4.1 Evaluation Metrics

The following metrics are used to evaluate the performance of the pedestrian detection system:

Precision: The ratio of true positives to the total predicted positives.

Recall: The ratio of true positives to the total actual positives.

F1 Score: The harmonic mean of precision and recall, providing a balance between the two.

4.2 Performance Comparison

The system was evaluated on the INRIA dataset, and the results show that the HOG + SVM approach achieved a detection rate of [X%] with a false positive rate of [Y%].

Precision: [X%]

Recall: [Y%]

F1 Score: [Z%]

The system performed well under normal conditions but struggled with heavily occluded pedestrians and varied lighting conditions.

4.3 Challenges and Limitations

Lighting Variations: The system’s performance decreased in low-light scenarios, leading to false negatives.

Occlusions: Partial occlusion of pedestrians caused detection failures.

Real-time Performance: Although the system works in real-time on a desktop setup, performance may be slower on embedded systems.

1. **Conclusion**

5.1 Summary of Findings

This project successfully implemented a pedestrian detection system using Python and OpenCV. The HOG + SVM method demonstrated its effectiveness in detecting pedestrians in various video frames. The system achieved a good balance between precision and recall but showed limitations in detecting occluded or partially obscured pedestrians.

5.2 Future Work

Future work can explore the following:

Deep Learning Approaches: Implementing deep learning methods like CNN or YOLO to improve detection accuracy, especially under difficult conditions.

Tracking: Integrating object tracking algorithms to maintain the detection of pedestrians over time in video streams.

Real-time Performance Optimization: Improving the system’s speed for real-time applications, possibly by optimizing the feature extraction or using hardware acceleration.

**VI. References**

[1] T. Ojala, M. Pietikainen, and D. Harwood, "A comparative study of texture measures with classification based on feature distributions," Pattern Recognition, vol. 29, no. 1, pp. 51-59, 1996.

[2] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in Proc. of CVPR, 2009.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," IEEE CVPR, pp. 511-518, 2001.

**VII. Appendices**

Appendix A: Code Snippets

Below is a basic example of how to use HOG and SVM with OpenCV in Python:

python

Copy

import cv2

from sklearn import svm

from skimage.feature import hog

# Load dataset (INRIA Dataset for example)

# Load images and labels

# Extract HOG features

features = hog(image, pixels\_per\_cell=(8, 8), cells\_per\_block=(2, 2), visualize=False)

# Train SVM classifier

svm\_classifier = svm.SVC(kernel='linear', C=1.0)

svm\_classifier.fit(features, labels)

# Use the trained classifier for detection

pedestrian\_detection = svm\_classifier.predict(hog\_features)

Appendix B: Figures and Tables

Figure 1: Example image from the dataset.

Figure 2: Pedestrian detection results on sample images.

Table 1: Performance metrics comparison (Precision, Recall, F1-Score).

This template follows the IEEE format, ensuring clarity and structure. You can tailor the content to match your specific project details and findings. If you'd like any additional help, such as code or further details, feel free to ask!