

COL780 : Computer Vision

Assignment 3

Object Detection & Recognition

Submitted By : Manish Kumar

Entry No. : 2023MCS2497

Table of Content

Introduction	2
Problem Statement	3
Approach utilized	4
Output	9
Problems faced	11
Result	12
References	13

Introduction

This report focuses on the development of an algorithm for detecting and recognizing hands in images using the **Histogram of Oriented Gradients (HOG) descriptor**. The primary objective of this assignment is to create a robust algorithm capable of accurately identifying whether a detected hand is *open* or *closed*. Hand detection and recognition are fundamental tasks in computer vision with numerous applications, ranging from gesture recognition to human-computer interaction. By leveraging the HOG descriptor, which captures the distribution of gradient orientations in an image, we aim to achieve reliable hand gesture classification.

The remainder of this report will delve into the methodology employed in developing the algorithm, present the results obtained and discuss any challenges faced during the assignment. Additionally, references to external sources or materials used in this assignment have been appropriately cited.

Tool used :

Primary : Python 3.10, WSL

Secondary : Webcam for video capture, multiple modules (as listed in requirements.txt)

Problem Statement

We have been given the responsibility of classifying hand images. Our objective is to classify them into open or closed hand gestures using a binary classifier such as Support Vector Machine (SVM). The process involves several key components :

- **Preprocessing:** To prepare dataset images for having ROI bounding boxes around them.
- **Feature Extraction:** Then, to extract meaningful features from the images that can effectively represent the hand gestures. In this context, we will utilize Histogram of Oriented Gradients (HOG) descriptors, which are known for their effectiveness in capturing local gradient information. These descriptors will be computed for regions of interest (ROI) within the images, focusing on the areas containing the hand gestures.
- **Training:** Once the HOG descriptors are extracted, they will serve as the input for training the SVM model. The SVM model will be trained to classify the hand images based on whether the gesture is an open hand or a closed hand. During training, the SVM algorithm will learn to distinguish between the distinctive features associated with open and closed hand gestures. .

Approach used for Feature extraction

Step I : Input Data Handling:

To begin with, we first work on managing the input data by organizing it into appropriate folders. The dataset is divided into separate directories for **training** and **validation sets**, each containing subdirectories for open and closed hand images. This structured arrangement facilitates efficient data handling and preprocessing.

Step II : Region of Interests (ROI) Extraction:

To focus solely on the hand gestures within the images, we employ the **MediaPipe Hands library** for hand detection. By utilizing its pre-trained model, we extract the landmarks corresponding to hand key points from each image using *hands.process()* fn. These landmarks are then used to determine the bounding box enclosing the hand region.

Additionally, we apply a *scaling factor*(= 1.3) to the bounding box to ensure a suitable margin around the hand. Subsequently, we **crop** the region of interest (ROI) from the original image based on the calculated bounding box coordinates. The extracted ROI images are saved in *separate directories* for both training and validation sets.

Step III : Histogram of Oriented Gradients(HOG) Feature Extraction:

Following ROI extraction, we first check if a file containing *precomputed HOG features* exists. If the file is found (*hog_features.pkl*), the script loads the features directly from the pickle dump to avoid recomputing the features if they have already been computed in a previous run.

If the HOG features file does not exist, we then proceed to **compute the features** by iterating through the training folders (**train_folders**). For each folder, the script extracts **HOG features** from the ROI images using the *calculate_hog_features_multiple* fn from the *HoG.py* module. These features are then stored in a list along with their corresponding labels. After computing the features for all training images, the script saves the feature list to a pickle dump for future use.

Brief details about implementation of HOG Algo.

➤ Preprocessing of Image:

Before computing the HOG features, the input image is preprocessed to enhance the quality of the ROI. This involved converting the image to **grayscale** and **resizing** it to a standardized dimension of *128x64 pixels*.

➤ Gradient Calculation:

The **gradient magnitude** and **orientation** are calculated for each pixel within the ROI. This is achieved by convolving the image with derivative filters to determine the intensity gradients in the *horizontal (Gx)* and *vertical (Gy)* directions. Considering a block of 3x3 pixels, Gx & Gy are computed for every pixel. The magnitude and angle of the gradient are then computed using these derivatives.

➤ Histogram Binning:

The gradient orientations are **quantized** into a fixed number of bins, typically 9 bins covering *180 degrees*. Here, each bin has a range of 20 degrees. For each cell within the ROI, a histogram of gradient orientations is constructed by accumulating the gradient magnitudes

weighted by their corresponding orientations. This histogram captures the distribution of gradient directions within the cell.

➤ **Block Normalization:**

To enhance the robustness of the feature representation and account for local variations in illumination and contrast, normalization is applied to *groups of adjacent* cells called blocks. For each block, a *36 point feature vector* is collected. In the horizontal direction there are *7 blocks* and in the vertical direction there are *15 blocks*. So the *total length* of HOG features will be : $7 \times 15 \times 36 = 3780$. The gradient histograms within each block are concatenated to form the final feature vector, which is then **normalized** to ensure scale invariance in scale and illumination conditions.

➤ **Feature Vector Generation:**

Finally, the feature vectors obtained from all the blocks are **concatenated** to form the complete HOG feature descriptor for the ROI image. This feature vector encapsulates the spatial distribution of gradient orientations and serves as input to subsequent stages of the classification pipeline.

Lastly the extracted feature vector is passed back to main fn. For the calls made to *HoG.calculate_hog_features_multiple()* which can now be used for model training.

Step IV : Model Training:

After loading or computing the HOG features, we check if a trained SVM model file exists (`svm_model.sav`). If the file exists, we **load the model** directly from the saved file. This step helps to avoid retraining the model if it has already been trained and saved.

If the SVM model file *does not exist*, we proceed to **split** the extracted HOG features into **training samples** (X_{train}) and **corresponding labels** (y_{train}). The features are reshaped into a flattened format ($X_{train_flattened}$) suitable for training the SVM classifier.

Next, we've defined a parameter grid (`param_grid`) for **hyper-parameter tuning** using *Grid Search Cross-Validation*. Then we initialize an SVM classifier with probability estimation enabled and set up a Grid Search Cross-Validation object (`grid_search`) to search for the best hyper-parameters based on accuracy.

Lastly we **train** the *SVM model* using the training data and the Grid Search object. Once training is complete, the best performing SVM model is selected based on the **cross-validation results** {in our case with params 'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}. The trained model is then saved to a file (`svm_model.sav`) for future use.

Step V : Model Evaluation:

➤ Accuracy:

Accuracy measures the percentage of correctly classified instances out of all instances in the dataset. For each class (closed and open hand gestures), the accuracy is then calculated. For the provided dataset, the accuracy for *closed hand gestures* is approximately 96.578%, and for *open hand gestures*, it is approximately 99.791%. The *overall accuracy* across both classes is computed as 98.652%.

➤ **Precision, Recall, and F1-score:**

Precision measures the proportion of true positive predictions out of all positive predictions, while recall (also known as sensitivity) measures the proportion of true positive predictions out of all actual positive instances. The F1-score is the harmonic mean of precision and recall, providing a balanced assessment of the model's performance.

➤ **Receiver Operating Characteristic (ROC) Curve:**

The ROC curve is the graphical representation of the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. It illustrates the trade-off between sensitivity and specificity for different classification thresholds. The ROC curve we've achieved is almost perfect, hugging the top left corner of the plot, suggesting an area under the curve (AUC) of 1.00, which indicates an excellent model performance.

By evaluating the model using these metrics, we gain insights into its ability to classify hand gestures accurately. The high accuracy, precision, recall, and AUC values demonstrate the effectiveness of the trained SVM model in distinguishing between open and closed hand gestures, validating its suitability for real-world applications in hand gesture recognition.

Extraction of ROIs & cropping images

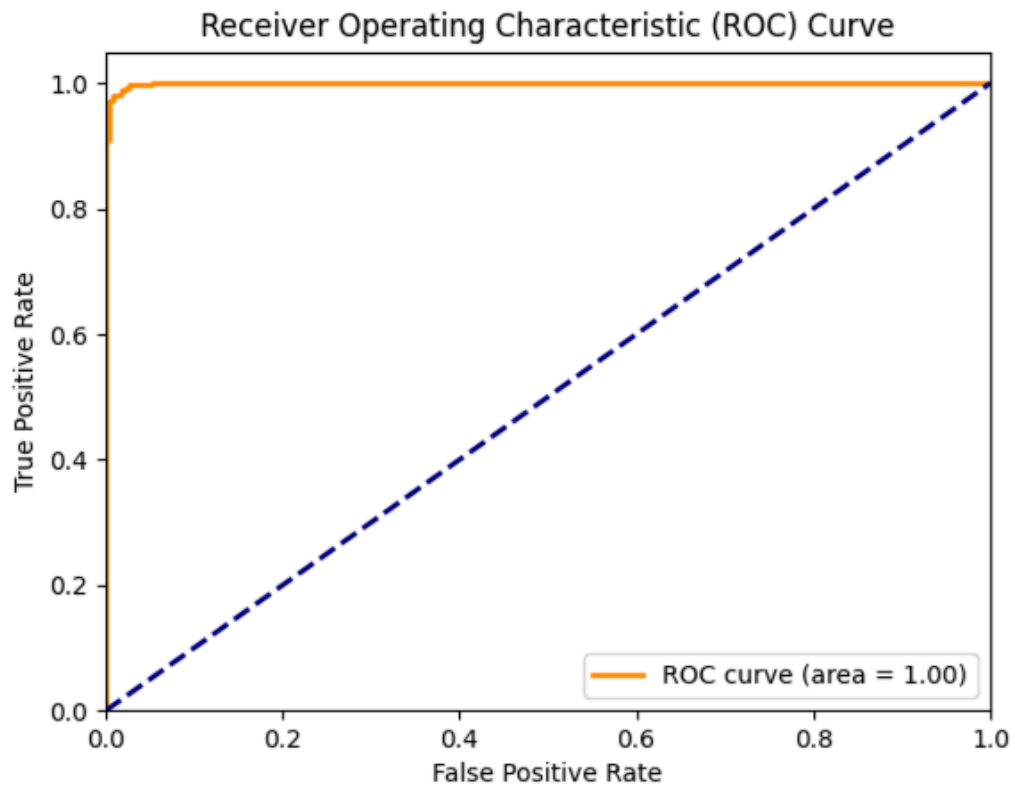
Extraction of HoG from Training set

Training model

Model Evaluation

9

ROC Curve



Problems Faced :

During the development process, we encountered challenges related to the **installation of the Mediapipe module** on the Windows platform. Unfortunately, this limitation restricted us from utilizing Windows for our development environment. As a workaround, we opted to use the Linux operating system, leveraging the *Windows Subsystem for Linux (WSL)* to execute our code.

However, one significant drawback of using WSL is its ***lack of support for graphical user interfaces (GUI)***, that prevented us from directly displaying plots using the `plt.show()` function in Matplotlib. To circumvent this limitation, we resorted to save the plots generated by Matplotlib using the `plt.savefig()` function. After the program execution, we manually viewed these saved plots to analyze the results.

This approach, although effective in overcoming the GUI limitation of WSL, introduced an additional step in the workflow, requiring manual inspection of the saved plots post-execution.

Result

The model's performance was evaluated on the validation datasets for both closed and open hand gestures. The following statistics were obtained:

Closed Hand Gesture Validation:

Accuracy: 96.578%

Precision: 0.98

Recall: 1.00

F1-score: 0.99

Open Hand Gesture Validation:

Accuracy: 99.791%

Precision: 0.98


Recall: 1.00

F1-score: 0.99

These results indicate that the trained model performed exceptionally well in distinguishing between closed and open hand gestures. The **overall accuracy** of the model on the combined validation dataset was found to be **98.652%**. Additionally, the **Receiver Operating Characteristic (ROC) curve**, illustrating the trade-off between true positive rate and false positive rate, demonstrated high performance with an area under the curve (AUC) value of **0.99**.

The *high accuracy, precision, recall, and F1-score values* obtained during validation indicate the effectiveness of the Histogram of Oriented Gradients (HOG) feature extraction technique and the Support Vector Machine (SVM) classifier in accurately classifying hand gestures. These results *validate* the suitability of the proposed approach for the task of hand gesture recognition.

References

 SVM Kernal- Polynomial And RBF Implementation Using Sklearn- Machine Learning

https://www.researchgate.net/publication/3857598_Pose_classification_using_support_vector_machines