PEDESTRIAN DETECTION USING HOG DESCRIPTOR, NMS AND SKIN DETECTION

A Thesis Submitted

In Partial Fulfillment of the Requirements

For the Degree, of

Bachelor of Technology

Contributors:

DRISHTI AGARWAL – 13/CS/42

RANJEY INDRAJEET JHA – 13/CS/03

Under the valuable guidance of

DR. GOUTAM SANYAL

Professor and Dean (Faculty Welfare)

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY DURGAPUR

`

DECLARATION

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degrees or diplomas of the university or other institutes of higher learning, except which due acknowledgment has been made in this text.

Drishti Agarwal
(13/CS/42)

Ranjey Indrajeet Jha
(13/CS/03)

`

NATIONAL INSTITUTE OF TECHNOLOGY DURGAPUR

CERTIFICATE

It is certified that the work contained in the thesis entitled "Pedestrian Detection using HOG, NMS and Skin Detection" has been carried out by “DRISHTI AGARWAL (Roll No. 13/CS/42)” and “RANJEY INDRAJEET JHA (Roll No. 13/CS/03)” under the guidance of “Dr. GOUTAM SANYAL”, the data reported herein is original and that this work has not been submitted elsewhere for any other Degree or Diploma.

This is to certify that the above declaration is true.

Ravi Kant Kumar
PhD Scholar, Department of
Computer Science and Engineering
NIT Durgapur

Dr. Goutam Sanyal
Professor, Department of
Computer Science and Engineering
NIT Durgapur

`

Acknowledgement

I hereby wish to express my sincere gratitude and respect to **Prof. Goutam Sanyal**, Department of Computer Science and Engineering, NIT Durgapur, under whom I had the privilege to work. His valuable guidance and encouragement have really led me to the path of completion of this project.

I am also highly indebted to **Mr. Ravi Kant Kumar**, Department of Computer Science and Engineering, NIT Durgapur, for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project. Any amount of thanks would not be enough for the valuable guidance of my supervisors. It was a rewarding experience and privilege to work under their expertise.

I would also like to thank all the faculty members of Computer Science and Engineering Department and my batch-mates for their help. Finally, I would like to pen down my gratitude towards my family members for their continuous support and encouragement. It would have not been possible to complete my work without their support.

Drishti Agarwal
(13/CS/42)

Ranjey Indrajeet Jha
(13/CS/03)

`

Table of Content

`

`

List of Figures

`

Abstract

Pedestrian detection has been an important problem for decades, given its relevance to a number of applications in robotics, including driver assistance systems, road scene understanding or surveillance systems. The two main practical requirements for fielding such systems are very high accuracy and real-time speed: we need pedestrian detectors that are accurate enough to be relied on and are fast enough to run on systems with limited compute power.

We present an approach to pedestrian detection in images, which can be easily extended to videos. We apply a HOG descriptor and employ a linear SVM to detect pedestrians on a structural level. To further advance our detection, we apply skin segmentation to the cropped area of the detected pedestrian to distinguish a human from any human-like statue or sign boards.

`

Chapter 1

Introduction

1.1 Pedestrian Detection

A **Pedestrian Detection system** is a computer application capable of identifying or verifying a person from a digital image or a video frame from a video source.

Pedestrian detection is an essential and significant task in any intelligent video surveillance system, as it provides the fundamental information for semantic understanding of the video footages. It has an obvious extension to automotive applications due to the potential for improving safety systems.

1.2 Motivation

All humans have the same basic structure regardless of gender, race, or ethnicity. **At the most structural level we all have a head, two arms, a torso, and two legs.**

We can use computer vision to exploit this semi-rigid structure and extract features to quantify the human body. These features can be passed on to machine learning models that when trained can be used to **detect** and **track** humans in images and video streams. This is especially

`

useful for the task of **pedestrian detection**. Furthermore, we can use Skin segmentation feature on the detected pedestrian like figures to confirm the presence of a human body.

1.3 Techniques Used

1. **HOG (Histogram of Oriented Gradients):** The **histogram of oriented gradients (HOG)** is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image.

2. **NMS (Non-Maximum Suppression):** Non-maximum suppression is used as an intermediate step in many computer vision algorithms. Non-maximum suppression is often used along with edge detection algorithms. The image is scanned along the image gradient direction, and if pixels are not part of the local maxima they are set to zero. This has the effect of suppressing all image information that is not part of local maxima.

3. **Skin Segmentation:** Skin detection is the process of finding skin-colored pixels and regions in an image or a video. This process is typically used as a preprocessing step to find regions that potentially have human faces and limbs in images.
   A skin detector typically transforms a given pixel into an appropriate color space and then use a skin classifier to label the pixel whether it is a skin or a non-skin pixel.

`

Chapter 2

Pedestrian Detection using HOG and NMS

2.1 Introduction to HOG

The essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing.

The HOG descriptor has a few key advantages over other descriptors. Since it operates on local cells, it is invariant to geometric and photometric transformations, except for object orientation. Such changes would only appear in larger spatial regions. Moreover, as Dalal and Triggs discovered, coarse spatial sampling, fine orientation sampling, and strong local photometric normalization permits the individual body movement of pedestrians to be ignored so long as they maintain a roughly upright position. The HOG descriptor is thus particularly suited for human detection in images.

`

The Histogram of Oriented Gradients (HOG) image descriptor and a Linear Support Vector Machine (SVM) could be used to train highly accurate object classifiers — or in their particular study, human detectors.

OpenCV ships with a pre-trained HOG + Linear SVM model that can be used to perform pedestrian detection in both images and video streams. Even though the Histogram of Oriented Gradients descriptor for object recognition is nearly a decade old, it is still heavily used today and with fantastic results.

2.2 HOG Algorithm

Step 1:

Sample *P* positive samples from your training data of the object(s) you want to detect and extract HOG descriptors from these samples.

Step 2:

Sample *N* negative samples from a *negative training set* that ***does not contain*** any of the objects you want to detect and extract HOG descriptors from these samples as well. In practice, *N >> P.*

Step 3:

Train a Linear Support Vector Machine on your positive and negative samples.

`

Step 4:

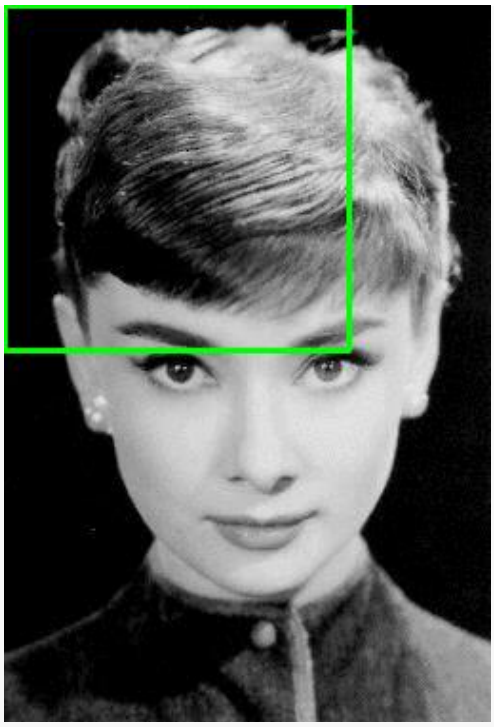

Figure 2.1: Example of the sliding a window approach, where we slide a window from left-to-right and top-to-bottom. *Note: Only a single scale is sown. In practice this window would be applied to multiple scales of the image.*

Apply hard-negative mining. For each image and each possible scale of each image in your negative training set, apply the sliding window technique and slide your window across the image. At each window compute your HOG descriptors and apply your classifier. If your classifier (incorrectly) classifies a given window as an object (and it will, there will absolutely be false-positives), record the feature vector associated with the false-positive patch along with the probability of the classification. **This approach is called *hard-negative mining.***

Step 5:

Take the false-positive samples found during the hard-negative mining stage, sort them by their confidence (i.e. probability) and re-train your classifier using these hard-negative samples. *(Note: You can iteratively apply steps 4-5, but in practice one stage of hard-negative mining usually [not always] tends to be enough. The gains in accuracy on subsequent runs of hard-negative mining tend to be minimal.)*

`

Step 6:

Your classifier is now trained and can be applied to your test dataset. Again, just like in Step 4, for each image in your test set, and for each scale of the image, apply the sliding window technique. At each window extract HOG descriptors and apply your classifier. If your classifier detects an object with sufficiently large probability, record the bounding box of the window. After you have finished scanning the image, apply non-maximum suppression to remove redundant and overlapping bounding boxes.

These are the bare minimum steps required, but by using this 6-step process you can train and build object detection classifiers of your own! Extensions to this approach include a deformable parts model and Exemplar SVMs, where you train a classifier for *each positive instance* rather than a *collection of them.*
However, if you've ever worked with object detection in images you've likely ran into the problem of *detecting multiple bounding boxes around the object you want to detect in the image.*

Here's an example of this overlapping bounding box problem:

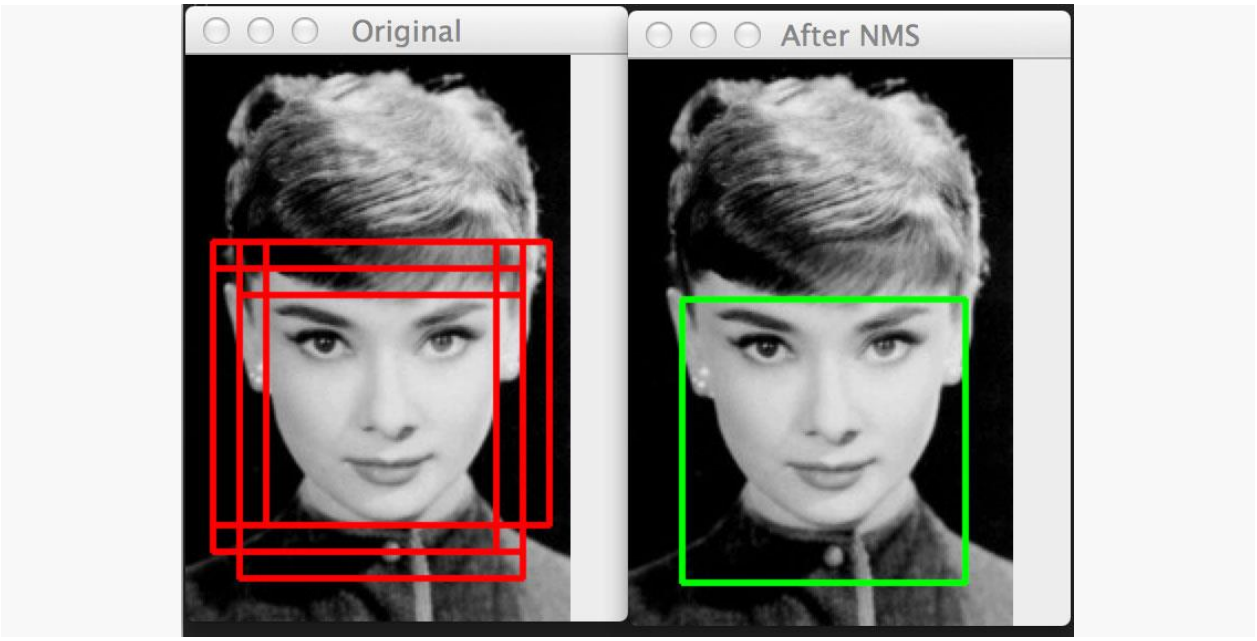

Figure 2.2: *(Left)* Detecting multiple overlapping bounding boxes around the face we want to detect. *(Right)*Applying non-maximum suppression to remove the redundant bounding boxes.

`

Notice on the *left* we have 6 overlapping bounding boxes that have correctly detected Audrey Hepburn's face.

2.3 Problems with using HOG

In Figure 3 we observe that there are 6 bounding boxes all refer to the same face — we need a method to suppress the 5 smallest bounding boxes in the region, keeping only the largest one, as seen on the *right*.
This is a common problem, no matter if you are using the Viola-Jones based method or following the Dalal-Triggs paper.

2.4 Solution to the problem: Non-Maximum Suppression(NMS)

NMS (non-maximum suppression) is a very popular post-processing method for eliminating redundant object detection windows. Non-maximum suppression is an edge thinning technique.

Non-Maximum suppression is applied to "thin" the edge. After applying gradient calculation, the edge extracted from the gradient value is still quite blurred. Thus, non-maximum suppression can help to suppress all the gradient values to 0 except the local maximal, which indicates location with the sharpest change of intensity value. The algorithm for each pixel in the gradient image is:

`

1. Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient directions.
2. If the edge strength of the current pixel is the largest compared to the other pixels in the mask with the same direction (i.e., the pixel that is pointing in the y direction, it will be compared to the pixel above and below it in the vertical axis), the value will be preserved. Otherwise, the value will be suppressed.

We use the above algorithm in our HOG code to basically suppress all the small bounding boxes and finally produce the desired result with a single bounding box.

`

Chapter 3

Modifying Algorithm by applying Skin Detection

3.1 Introduction to Skin Detection

The objective of the skin detection is to find out skin regions in an image. Skin color detection is the process of separation between skin and non-skin pixels. It is the initial step to find the regions that potentially have human faces and limbs in an image. It is difficult to develop uniform method for the segmentation or detection of human skin detection because of color tone of human skin is drastically varied for people from one region to another. For example, the skin color tone of Europeans is completely different from Africans or Asians. The detection and segmentation of skin regions in an image is widely used in many applications such as classification and retrieval of color images in multimedia applications, video surveillance, human motion monitoring, human computer interaction, digital cameras, face detection and recognition, teleconference, hand detection, gesture detection. There are two types of skin detection, either pixel or region based. In the pixel based skin detection, each pixel is classified as either skin or non-skin individually from its neighbor. The skin detection based on color fall in this category. In the region based skin detection, the skin pixels are spatially arranged to enhance the performance. This method requires additional information such as intensity, texture is required.

`

3.2 Use of Skin Detection in Pedestrian Detection

The pedestrian detection algorithm, even after applying NMS still have a few drawbacks.

Consider a Human statue, now obviously, it has similar body structure as that of a human. Our Pedestrian detection algorithm will classify the statue as human as well, which is not the output we expect.

To overcome this drawback, we can use Skin Color Detection such that for statues or any other human like structure, after applying HOG and NMS, if we apply Skin Color detection on it as well, we might get a more accurate result and hence making our code more efficient.

Though this also won't make the code 100% correct for all test cases.

For example, there's a possibility that there's a human who has his whole body covered by some cloth material, or there's a human like statue which is painted with the same color as the human skin, then we won't obtain the desired output from our algorithm, but then even a human eye can get confused in such situations.

3.3 HSV Color Space

HSV is so named for three values—**Hue, Saturation and Value**. This color space describes colors(hue or tint) in terms of their shade (saturation or amount of gray) and their brightness value.

Hue is expressed as a number from 0 to 360 degrees representing hues of red (which start at 0), yellow (starting at 60), green (starting at 120), cyan (starting at 180), blue (starting at 240) and magenta (starting at 300).

Saturation is the amount of gray from zero percent to 100 percent in the color.

`

Value (or brightness) works in conjunction with saturation and describes the brightness or intensity of the color from zero percent to 100 percent.

Advantage of HSV over RGB is that HSV separates *luma*, or the image intensity, from *chroma* or the color information. This is very useful in many applications. For example, if you want to do histogram equalization of a color image, you probably want to do that only on the intensity component, and leave the color components alone. Otherwise you will get very strange colors. In computer vision, you often want to separate color components from intensity for various reasons, such as robustness to lighting changes, or removing shadows.

Note, however, that HSV is one of many color spaces that separate color from intensity (See YCbCr, Lab, etc.). HSV is often used simply because the code for converting between RGB and HSV is widely available and can also be easily implemented.

`

Chapter 4

Evaluation Methodology

After installing necessary libraries for OpenCV Python and collecting a valid dataset of images, we will be implementing the code to efficiently detect a human from a set of images.

4.1 Procedure

During the initial runs of the code, we tried to extract any human figure from a given image based on the structural disposition of humans. We used HOG descriptors to achieve that purpose. The problem faced by us after the first run was of multiple bounding boxes detected for a single pedestrian body.

To fix the above complication, we introduced non-maximum suppression to reduce the multiple bounding boxes to an all-enclosing single bounding box.

To further enhance the efficiency of our code, we cropped our image with respect to the above bounding box and applied skin tone segmentation to it to confirm the presence of skin-like regions on the cropped area. This gives us a better evidence in the favor of the human figures' presence/absence in the given image.

4.2 Implementation of Code using OpenCV-Python

`

Now we will implement the code using OpenCV Python. We'll import print_function to ensure our code is compatible with both Python 2.7 and Python 3 (this code will also work for OpenCV 2.4.X and OpenCV 3). From there, we'll import the non_max_suppression function from my imutils package.

The following is our final Code implementation:

```
# import the necessary packages
from __future__ import print_function
from imutils.object_detection import non_max_suppression
from imutils import paths
import numpy as np
import argparse
import imutils
import cv2
# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--images", required=True, help="path to images directory")
args = vars(ap.parse_args())
# initialize the HOG descriptor/person detector
hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
# loop over the image paths
imagePaths = list(paths.list_images(args["images"]))
lower = np.array([0, 48, 80], dtype = "uint8")
upper = np.array([20, 255, 255], dtype = "uint8")
for imagePath in imagePaths:
        # load the image and resize it to (1) reduce detection time
        # and (2) improve detection accuracy
```

`

```
23.         image = cv2.imread(imagePath)
24.         image = imutils.resize(image, width=min(400, image.shape[1]))
25.         orig = image.copy()
26.         # detect people in the image
27.         (rects, weights) = hog.detectMultiScale(image, winStride=(4, 4),
28.         padding=(8, 8), scale=1.05)
29.         # draw the original bounding boxes
30.         for (x, y, w, h) in rects:
31.                 cv2.rectangle(orig, (x, y), (x + w, y + h), (0, 0, 255), 2)
32.         # apply non-maxima suppression to the bounding boxes using a
33.         # fairly large overlap threshold to try to maintain overlapping
34.         # boxes that are still people
35.         rects = np.array([[x, y, x + w, y + h] for (x, y, w, h) in rects])
36.         pick = non_max_suppression(rects, probs=None, overlapThresh=0.65)
37.         # draw the final bounding boxes
38.         for (xA, yA, xB, yB) in pick:
39.                 cv2.rectangle(image, (xA, yA), (xB, yB), (0, 255, 0), 2)
40.         frame=image[yA:yB,xA:xB]
41.         frame = imutils.resize(frame, width=max(400, frame.shape[1]))
42.         # show some information on the number of bounding boxes
43.         #filename = imagePath[imagePath.rfind("/") + 1:]
44.         #print("[INFO] {}: {} original boxes, {} after suppression".format(
45.         #filename, len(rects), len(pick)))
46.         converted = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
47.         skinMask = cv2.inRange(converted, lower, upper)
48.         # apply a series of erosions and dilations to the mask
49.         # using an elliptical kernel
50.         kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))
51.         skinMask = cv2.erode(skinMask, kernel, iterations = 2)
```

`

```
52.         skinMask = cv2.dilate(skinMask, kernel, iterations = 2)
53.         # blur the mask to help remove noise, then apply the
54.         # mask to the frame
55.         skinMask = cv2.GaussianBlur(skinMask, (3, 3), 0)
56.         skin = cv2.bitwise_and(frame, frame, mask = skinMask)
57.         # show the skin in the image along with the mask
58.         cv2.imshow("images", np.hstack([frame, skin]))
59.         # if the 'q' key is pressed, stop the loop
60.         if (cv2.waitKey(0)&0xff) == 27: break
```

4.3 Explaining key sections of the code

Lines 14 and 15 initialize our pedestrian detector. First, we make a call to hog =cv2.HOGDescriptor() which initializes the Histogram of Oriented Gradients descriptor. Then, we call the setSVMDetector to set the Support Vector Machine to be pre-trained pedestrian detector, loaded via the cv2.HOGDescriptor_getDefaultPeopleDetector() function.

We define the *lower* and *upper* boundaries for pixel intensities to be considered skin on **Lines 18 and 19**. It's important to note that these boundaries are for the HSV color space, **NOT** the RGB color space**.**

On Line 20 we start looping over the images in our --images directory.

Lines 23-25 handle loading our image off disk and resizing it to have a maximum width of 400 pixels. The reason we attempt to reduce our image dimensions is two-fold:

`

1.Reducing image size ensures that less sliding windows in the image pyramid need to be evaluated (i.e., have HOG features extracted from and then passed on to the Linear SVM), thus reducing detection time (and increasing overall detection throughput).

2. Resizing our image also improves the overall accuracy of our pedestrian detection (i.e., less false-positives).

Detecting pedestrians in images is handled by **Lines 27 and 28** by making a call to thedetectMultiScale method of the hog descriptor. The detectMultiScale method constructs an image pyramid with scale=1.05 and a sliding window step size of (4, 4) pixels in both the x and y direction, respectively.

The size of the sliding window is fixed at 64 x 128 pixels, as suggested by the seminal Dalal and Triggs paper, Histograms of Oriented Gradients for Human
Detection. ThedetectMultiScale function returns a 2-tuple of rects , or the bounding box (x, y)-coordinates of each person in the image, and weights , the confidence value returned by the SVM for each detection.

Lines 30 and 31 take our initial bounding boxes and draw them on our image.
However, for some images you'll notice that there are multiple, overlapping bounding boxes detected for each person.

In this case, we have two options. We can detect if one bounding box is fully contained within another (as one of the OpenCV examples implements). Or we can apply non-maxima suppression and suppress bounding boxes that overlap with a significant threshold — and that's exactly what **Lines 35 and 36** do.

`

After applying non-maxima suppression, we draw the finalized bounding boxes on **Lines 38 and 39**.

On Line 40, we are cropping the final bounding box and storing it separately for applying skin color segmentation on the detected pedestrian figure.

The actual skin detection takes place on **Line 46 and 47**.
First, we convert the image from the RGB color space to the HSV color space. Then, we apply the cv2.inRange function, supplying our HSV frame, and our lower and upper boundaries as arguments, respectively.
The output of the cv2.inRange function is our mask This mask is a single channel image, has the same width and height as the frame, and is of the 8-bit unsigned integer data type.
Pixels that are *white* (255) in the mask represent areas of the frame ***that are skin***. Pixels that are *black* (0) in the mask represent areas ***that are not skin***.

However, we may detect many small false-positive skin regions in the image. To remove these small regions, take a look at **Lines 50-52**. First, we create an elliptical structuring kernel. Then, we use this kernel to perform two iterations of erosions and dilations, respectively. These erosions and dilations will help remove the small false-positive skin regions in the image.
From there, we smooth the mask slightly using a Gaussian blur on **Line 55**. This smoothing step, while not critical to the skin detection process, produces a much cleaner mask.
We then apply the skin mask to our frame on **Line 56**.

Line 59 shows us a side-by-side view of the original frame along with the frame with skin detected in it.
We wait for a keypress on **Line 60**, and if it's the q key, then we break from the loop.

`

Chapter 5

Results Obtained

Since OpenCV ships with a pre-trained descriptor, we tested our implemented code on over 30 images. We shall display a few sample image-runs which shall comprehensively cover all sorts of cases encountered during our project completion.

5.1 Result 1

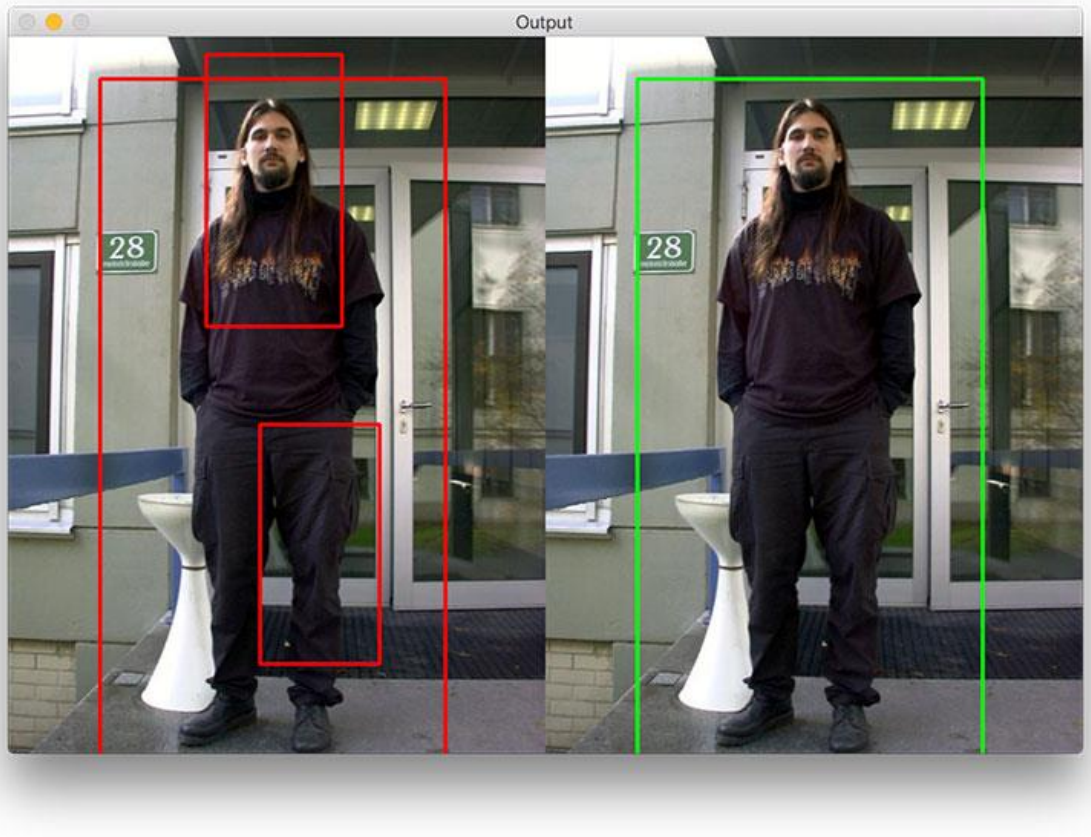

Figure 5.1: Result image after applying HOG and NMS.

`

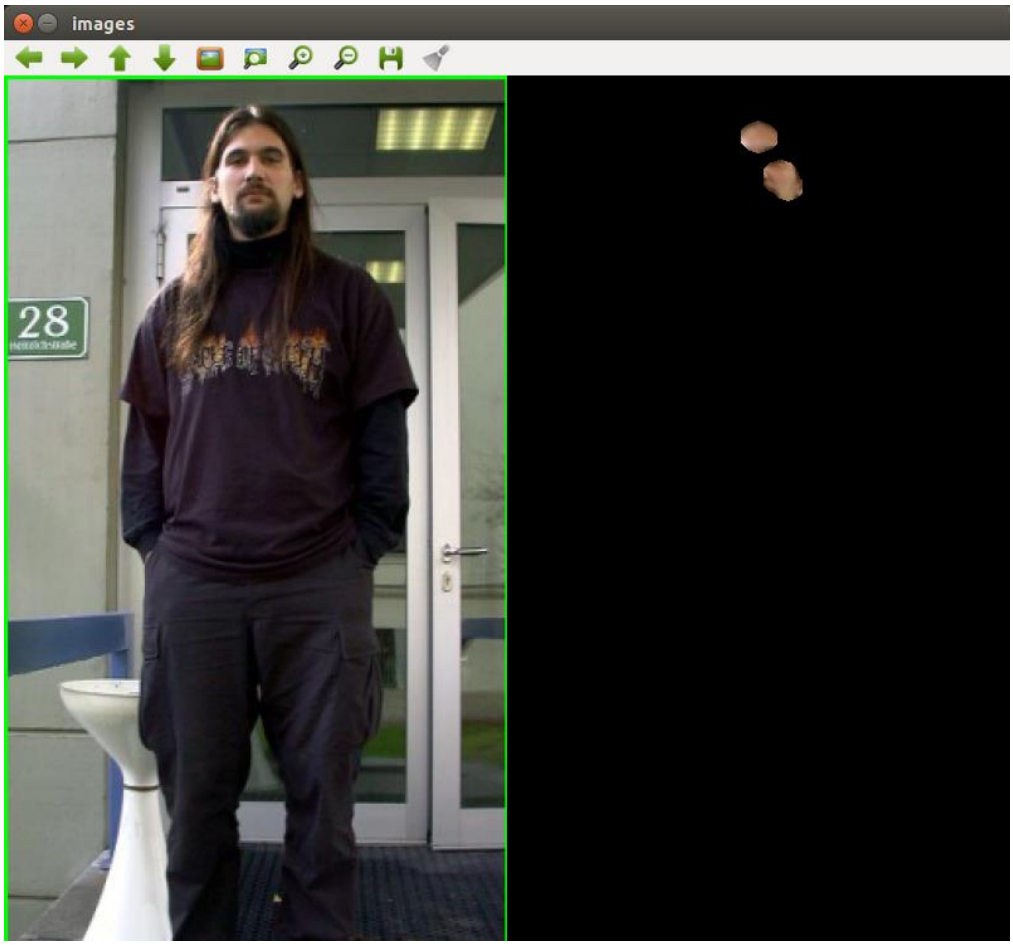

Figure 5.2: Final output of image 1 showing skin segmentation

Figure 5.1 shows the result before and after applying NMS. We initially obtain an output which has multiple bounding box. After applying NMS, we obtain a single bounding box. After applying skin detection, we find that the algorithm is able to detect skin from the given input. Hence, we can conclude that the input image consists of a Human.

`

5.2 Result 2

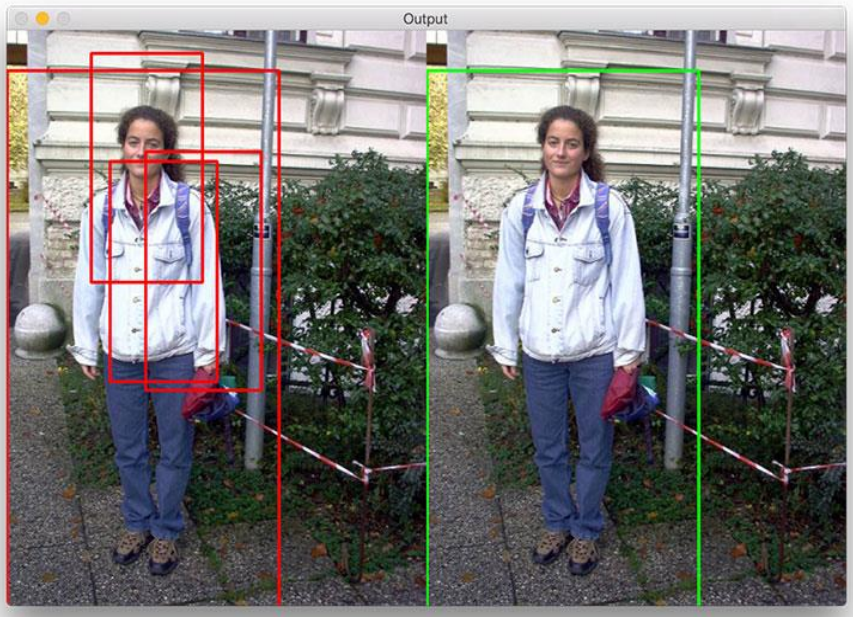

Figure 5.3: Result image after applying HOG and NMS

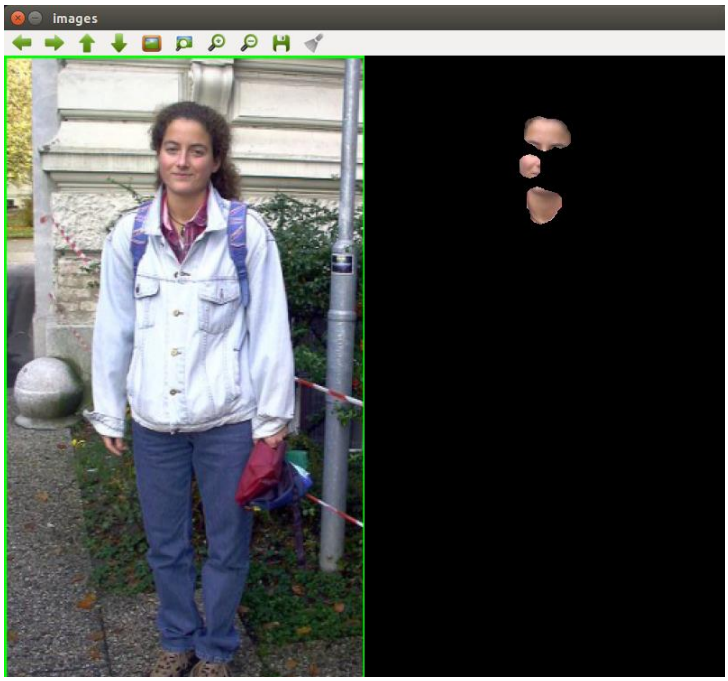

Figure 5.4: Final output of image 2 showing image segmentation

`

In result 2, After applying HOG, we obtain an output which consists of multiple bounding box. After applying NMS, we reduce the bounding box to just one. Output in figure 5.4 is the result obtained after skin detection. We observe that in our result, i.e. the final output, skin is detected by our code, hence it confirms that in the input image, human is present.

5.3 Result 3

Figure 5.5: Result image after applying HOG and NMS on image 3

`

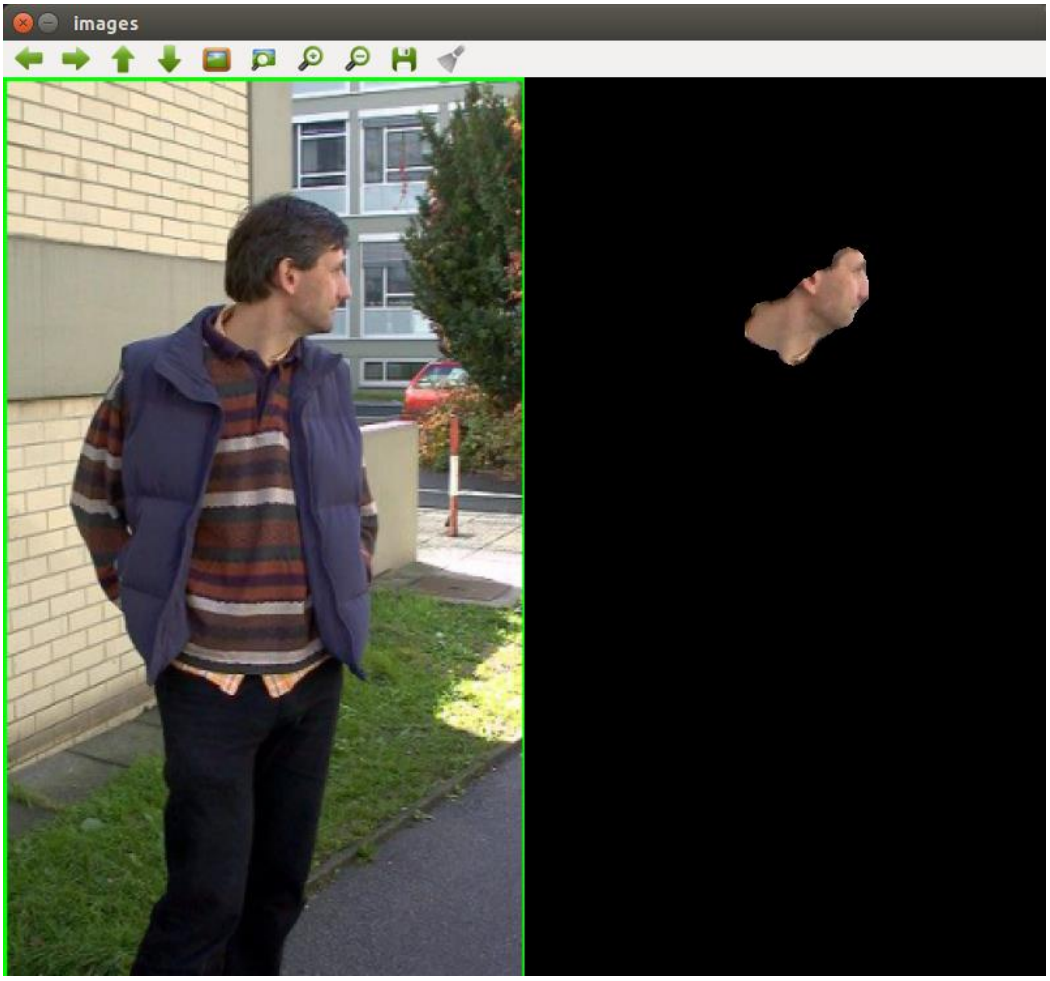

Figure 5.6: Final output of image 3 showing skin detection

Here we detect a person, and initially get multiple bounding boxes for the same person in the frame. After applying NMS, our results get better and the entire person is confined within a single bounding box. To confirm the presence, we apply skin color segmentation and we are able to efficiently segment the skin areas from the person's body thus confirming him as a human pedestrian.

`

5.4 Result 4

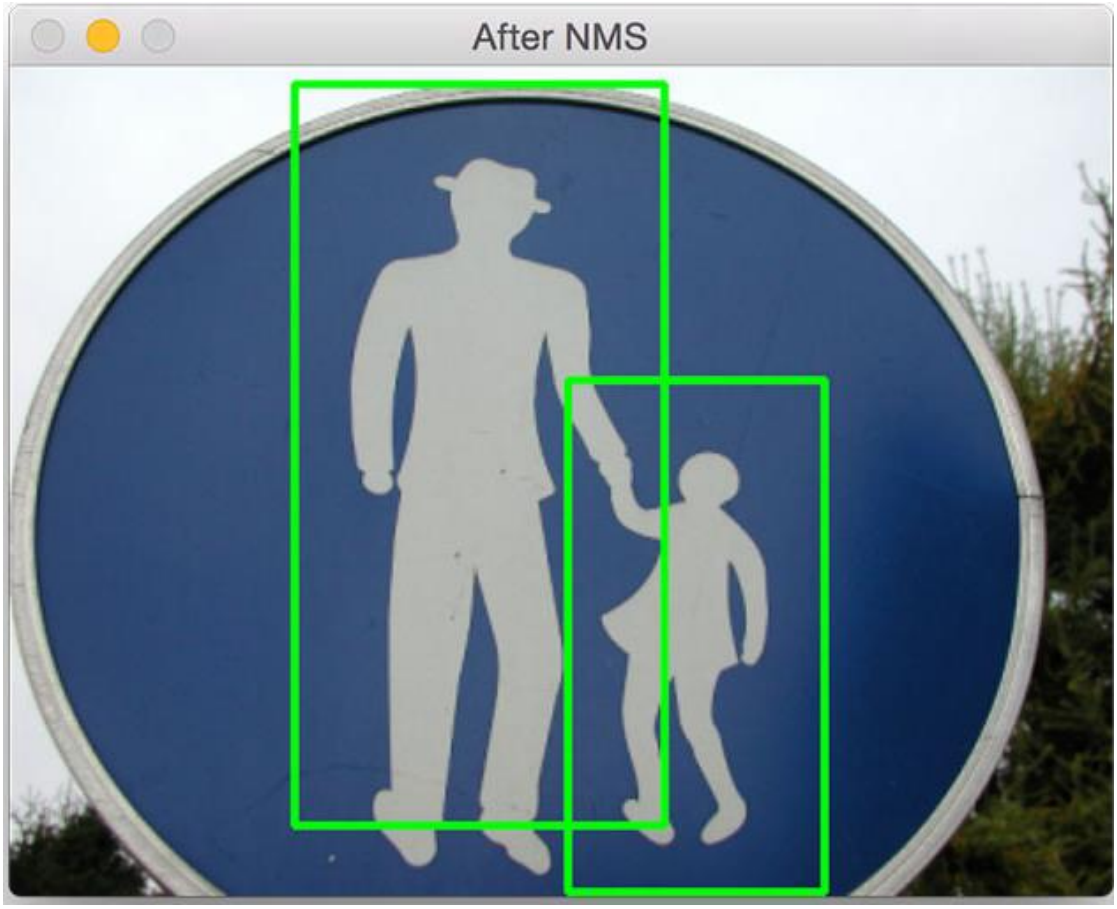

Figure 5.9: Output after applying HOG and NMS on image 5

`

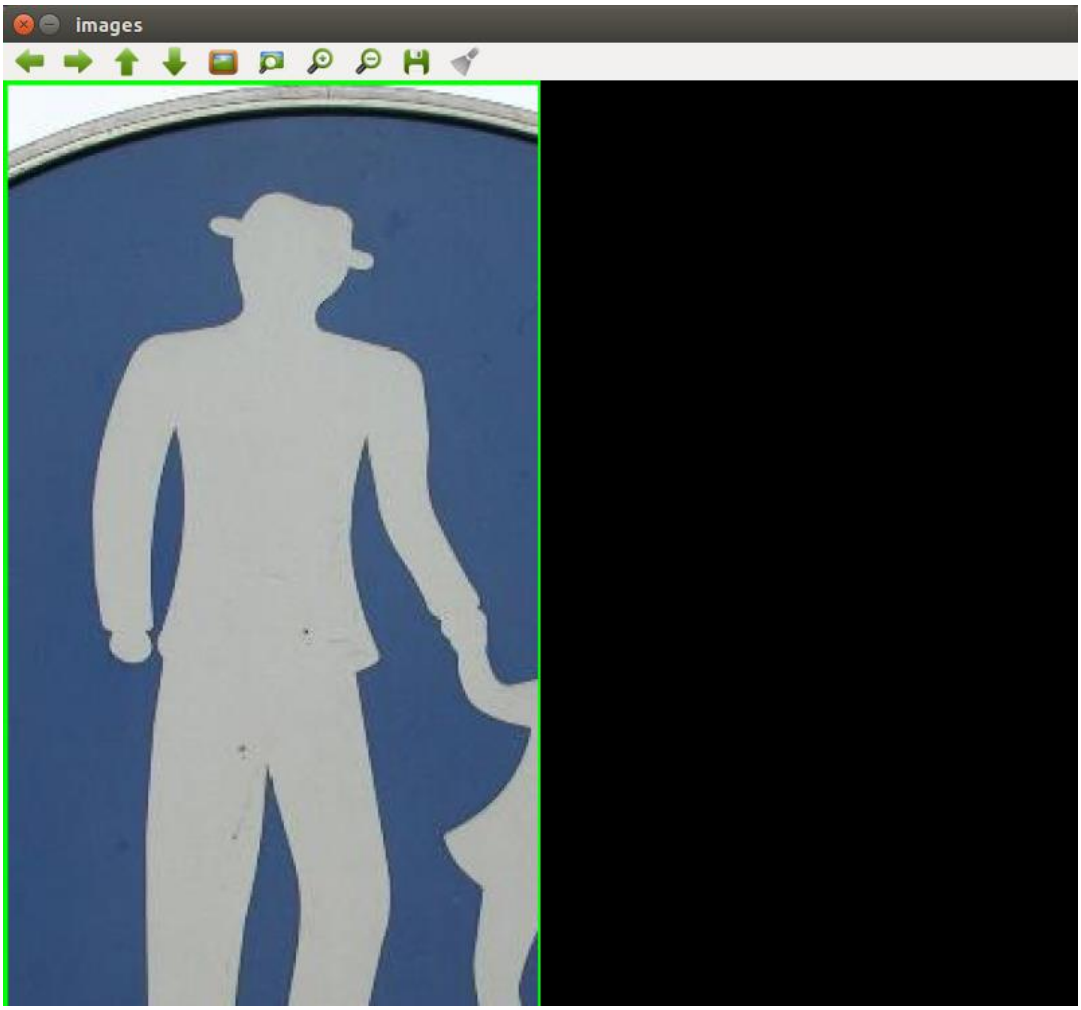

Figure 5.10: Final output after skin detection

This particular result is the one that most clearly distinguishes between an actual human body and a pedestrian – like figure. Here, our HOG descriptor detected it as a human figure and we extracted that information and stored it in a bounding box. But, after applying skin segmentation, we can now be assured that although it appeared to be a pedestrian at first, it is a false detect, as there was no skin found inside the bounding box. Hence, it won't be detected as a pedestrian, making our code efficient.

`

5.5 Result 5

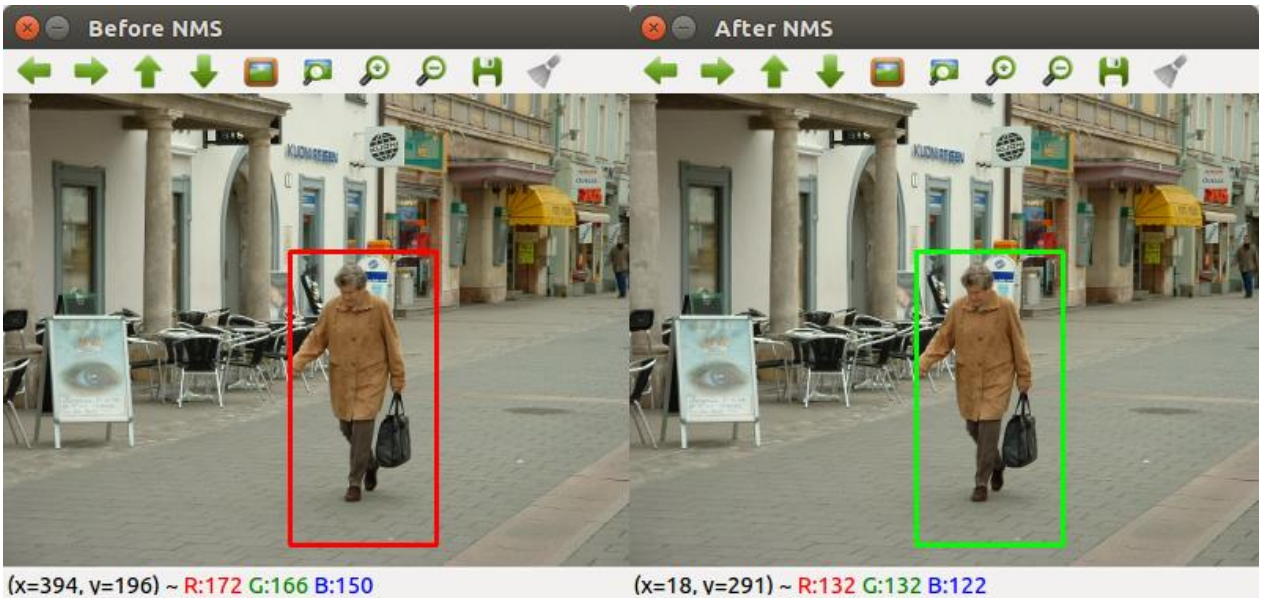

Figure 5.7: Result after Applying HOG and NMS in image 4

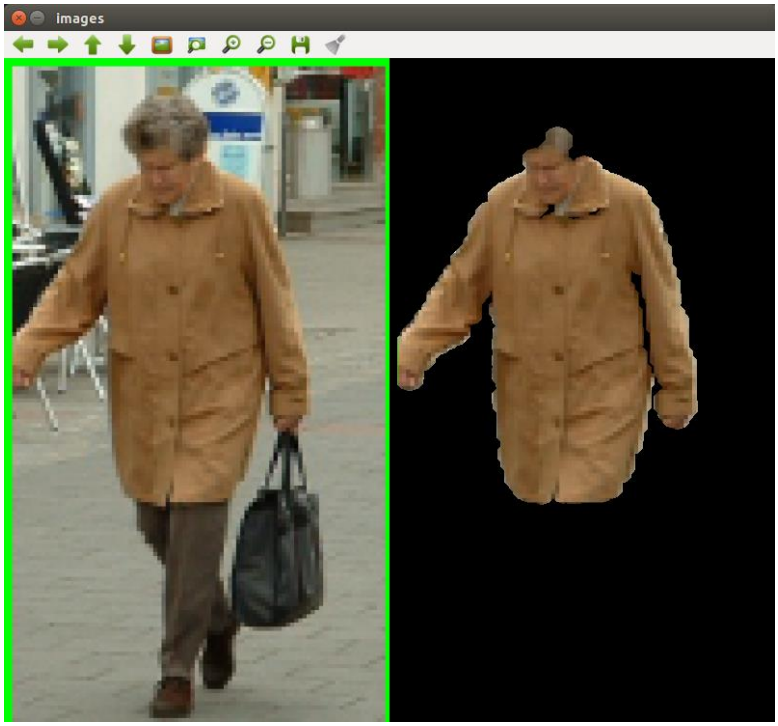

Figure 5.8: Result after applying skin detection algorithm on image 4

`

In this run, we successfully detect a pedestrian, i.e. the old woman walking. Then, on applying skin segmentation on the bounding box so received, we get a whole lot more area of skin than what is actually present in the image. The algorithm considers the skin-like colored jacket as skin and although it is an anomaly, the code correctly classifies and detects a pedestrian. This is an area of flaw where even the human eyes, being the best detector ever, can fail. Even our eye might confuse a skin colored apparel to be actual human skin from a distance.

`

Chapter 6

Conclusion

We found the basic feature among human's that can be used to differentiate them from other things. We also had a sneak peek into a Python framework for object detection in images. From there we had a quick review of how the Histogram of Oriented Gradients method is used in conjunction with a Linear SVM to train a robust object detector.

However, no matter what method of object detection you use, you will likely end up with multiple bounding boxes surrounding the object you want to detect. In order to remove these redundant boxes, you'll need to apply Non-Maximum Suppression.

Non-Maximum Suppression is absolutely critical to obtaining an accurate and robust object detection system using HOG.

Next, we showed you how to detect skin in images using Python and OpenCV.

To accomplish our skin detection, we framed skin detection as an extension to color detection

We were able to supply upper and lower ranges of pixel intensities in the HSV color space to detect skin in images.

While this is not a perfect or robust approach, the simplicity of our skin detection algorithm makes it a very good starting point to build more robust solutions.

We conclude that with HOG and Linear SVM, followed by skin color segmentation, we can detect pedestrian with very great accuracies. This simple pedestrian detector can act as a resource for many other ideas that somedays back were thought to be inexecutable.

`

Chapter 6

References

1. Learning OpenCV: Computer Vision with the OpenCV

library – Gary Bradsky, Adrian Kaehler.

2. The seminal Dalal and Triggs paper, Histograms of Oriented

Gradients for Human Detection.

3. Image Search Engine Resource Guide by Adrian Rosebrock.

4. Tombone's Computer Vision Blog – really fast nms.m

5. A performance evaluation of gradient field HOG descriptor for sketch based image retrieval - by Rui Hu and John Collomosse

6. B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, 5th Annual ACM Workshop on COLT, pages 144-152, Pittsburgh, PA, 1992. ACM Press.

7.J. Brand, J. Mason, "A Comparative Assessment of Three Approaches to Pixel-Level Human Skin Detection", Proc. IEEE Int'l Conf. Pattern Recognition, vol. 1, pp. 1056-1059, Sept. 2000.

8.Skin Detection Using Color Pixel Classification with Application to Face Detection: A Comparative Study by Krishnan Nallaperumal,Subban Ravi,C. Nelson Kennedy Babu

`