

A  
*Dissertation*  
on

# **SVM and HOG based sculpture recognition**

*Submitted in partial fulfillment of the requirement*

*for the award of the degree of*

**Master of Technology**

*in*

**VLSI Design and Embedded System**

*Submitted*

*by*

**Vishesh Nasir**

**University Roll No. 2K13/VLS/24**

*Under the Guidance of*

**Dr. S. Indu**

**Assistant Professor,**

**&**

**Mr. A.K. Singh**

**Assistant Professor**

**Department of Electronics and Communication Engineering**

**Delhi Technological University**



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING  
DELHI TECHNOLOGICAL UNIVERSITY  
2013-2015**



**Department of Electronics and Communication Engineering**  
**Delhi Technological University**  
**Delhi-110042**  
**www.dce.edu**

## CERTIFICATE

This is to certify that the dissertation titled “**SVM and HOG based sculpture recognition**” is a bonafide record of work done by **Vishesh Nasir, Roll No. 2K13/VLS/24** at **Delhi Technological University** for partial fulfilment of the requirements for the degree of Master of Technology in VLSI and Embedded System Design. This project was carried out under my supervision and has not been submitted anywhere else, either in part or full, for the award of any other degree or diploma to the best of my knowledge and belief.

Date: \_\_\_\_\_

**(Dr. S. indu)**  
**Assistant Professor**  
**Department of Electronics and Communication Engineering**  
**Delhi Technological University**

**Mr. A.K. Singh**  
**Assistant professor**  
**Department of Electronics and Communication Engineering**  
**Delhi Technological University**

## ACKNOWLEDGEMENT

This work would have not been possible without the support of many. First, it is my duty to thank God Almighty for making the completion of this research possible. Next, I would like to express my gratitude to my parents and my family for their continued support.

I would like to express my deep sense of respect and gratitude to my project supervisor **Dr S. Indu**, Assistant Professor, Electronics and Communication Engineering Department, DTU and **Mr. A.K. Singh**, Assistant Professor, Electronics and Communication Engineering Department, DTU for providing the opportunity of carrying out this project and being the guiding force behind this work. I am deeply indebted to them for the support, advice and encouragement they provided without which the project could not have been a success.

I would also like to acknowledge Delhi Technological University for providing the right academic resources and environment for this work to be carried out.

Last but not the least I would like to express sincere gratitude to my friends and my colleagues for constantly encouraging me during the completion of work.

**Vishesh Nasir**  
**University Roll no: 2K13/VLSI/24**  
**M.TECH. (VLSI Design and Embedded System)**  
**Department of Electronics & Communication Engineering**  
**Delhi Technological University**  
**Delhi – 110042**

## ABSTRACT

Historical monuments and their sculptures hold the story of the past attached to the monument. They give an idea of how the ancient people, the kings etc lived. We see a lot of ancient temples, tombs and other monuments. They have sculptures and other architectural designs on them to signify the era to which they belong. We need to preserve and pass these to our upcoming generations so that they can understand and can contribute their part in maintaining the heritage. Sculptures found in these monuments are also very helpful in understanding the culture. It has been observed that the ancient people (in India, for e.g.) many times worship the same god, but with a different name. If using a classifier we can know the name of the god or sculpture in the language of our choice, and then it is much likely that we'll extract other useful information related to them too.

In our proposed work we have worked on to build a classifier based on the features of sculptures (of god/goddesses) to classify the images of gods and goddesses. The images have been taken from various sources such as temples, forts, ancient monuments, sculptures and other images from INTERNET. The sculptures classification can prove useful as it can help us classify the god (i.e. label them as god A or god B), extract information about the god/goddess, the era or history of that particular sculpture. This is a thoughtful step towards preserving the rich history India has and make future generations learn and remember the past efficiently.

The proposed method also helps in solving a 2-class classification problem which works on the basis of training the classifier with the HOG features of the images. The proposed method classifies the query image (of God/Goddesses) as belonging to either of the 4 classes on which it has been trained. We have taken 3 SVM classifiers here and used them in nesting with each other. The accuracy lies between 56 to 70%. The proposed method can be extended for information retrieval after classification.

## Table of Contents

Certificate.....	i
Acknowledgement.....	ii
Abstract.....	iii
Table of contents.....	iv
List of figures.....	vi
List of tables.....	vii
Chapter 1: Introduction.....	1
1.1 History of Indian sculpture making .....	1
1.2 contribution of the project towards preserving history.....	2
1.3 Related work.....	3
1.4 motivation.....	5
1.5 scope of work.....	5
Chapter 2: Image processing using MATLAB.....	7
2.1 Introduction.....	7
2.1 MATLAB's Development Environment .....	8
2.3 Image Representation.....	10
2.3.1 Image Format.....	10
2.3.2 Basic image operations .....	10
2.3.3 Image Information .....	11
2.4. Quantization of the image.....	12
2.4.1 Grey Level Ranges .....	12
2.4.2 Number of Pixels.....	14
2.5 Point Processing.....	15
2.5.1 Value Manipulation.....	15
Chapter 3: HOG based classification.....	18
3.1 SIFT .....	19
3.2 SURF.....	20

3.3 HOG.....	21
Chapter 4: Support vector machine.....	26
4.1 History of SVM: .....	26
4.2 Basic Definition of SVM: .....	26
4.3 Calculating the optimal Hyperplane .....	29
4.4 Biased and Unbiased Hyperplanes: .....	30
4.5 Extensions in SVM: .....	30
4.6 Strengths of SVM: .....	31
4.7 Issues regarding SVM:.....	31
Chapter 5: The MATLAB code .....	33
Chapter 6: The working of the project .....	37
Chapter 7: Result and conclusion .....	43
7.1 Results:.....	43
7.2 Conclusions:.....	43
7.3 Scope of improvement .....	43
7.4 Practical use of the system.....	44
REFERENCES: .....	45

## List of figures

Fig 2.1: working window showing all parts of MATLAB.....	9
Fig 2.2: list of information displayed by imfinfo command.....	12
Fig 3.1: original grayscale image.....	22
Fig 3.2: Y - gradient image.....	23
Fig 3.3: X – gradient image.....	23
Fig 3.4: the magnitude matrix.....	24
Fig 4: the angle matrix.....	24
Fig 4.1: examples of different planes as an example of a classifier.....	27
Fig 4.2: an optimal decision boundary.....	28
Fig 4.3: a bad decision boundary (not equally spaced).....	28
Fig 4.4: a bad decision boundary (not giving maximum separation).....	29
Fig 5.1: code in working image (1).....	37
Fig 5.2: code in working image 2.....	38
Fig 5.3: code in working image 3.....	38
Fig 5.4: code in working image 4.....	39
Fig 5.5: code in working image (4).....	39
Fig 5.6: example run for lord Ganesha	
a) Output of the program.....	40
b) The time taken to calculate output.....	40
Fig 5.7: example run for lord hanuman	
a) output of the program.....	41
b) b) the time taken to calculate output.....	41

Fig 5.8: example run for Natraj: output of the program.....	41
Fig 5.9: example run for lord Natraj: the time taken to calculate output.....	42
Fig 5.10: example run for Shivling	
a) Output of the program.....	42
b) The time taken to calculate output.....	42



## List of tables

Table 2.1: data types supported by MATLAB.....	13
Table 2.2: functions to convert one image type to other.....	14
Table 5.1: workspace after implementation of getfile.m.....	33
Table 5.2: workspace after implementation of trainer.m.....	34

## Chapter 1: Introduction

India has one of the richest and most ancient cultural heritage in the world, rivalled only by Chinese art. The art of sculpture making was a highly respected medium for artists and was practiced throughout the subcontinent. Buildings were beautifully decorated with it. The main subject matter of Indian sculpture making was almost abstracted human forms. These were portrayed to guide people in the founding of the Hindu, Buddhist or Jain religions.

### 1.1 History of Indian sculpture making

India is a treasure house of sculptural art. Among a wide variety of cultural heritage that is spread throughout the country are memorials for the brave, cave temples carved out of stone, amazingly complex designs (pictures), and other artifacts. For a normal person, they might look similar, but a careful research of these sculptures provides deep insight into India's glorious past and her lost history.

The oldest known sculpture in the India is believed to be from the great Indus Valley civilization and is extracted from the sites of Mohenjo-Daro and Harappa which presently fall under the territory of modern-day Pakistan. The sculptures of Indus valley include Natraj which is the small bronze male dancer also worshipped as dance god. However such figures belonging to that time which are made of bronze and stone are found very rare. More easily found are pottery figurines and seals made of stone. After Indus Valley civilization, the work done on sculptures went into a passive phase until start of the Buddhist era, although we have some figures made of metal of around 1500 BCE from Fatimabad. Hence the tradition of Indian sculpture making in stone began relatively late as compared to sculpture making in metal, the estimated time of stone sculpture started with the Ashoka reign around 250 BCE. The most important work of the time were the stone Pillars of Ashoka that were made around India. These pillars carried his edicts that had on their top, famous sculptures of animals. Of all the sculptures that were made in that era, five survive today. Large amounts of figurative sculpture, also exist to the date that are believed to be from

Early Buddhist pilgrimage stupas, above all of them is the stupa of Sanchi; these stupas were made out of wood that also embraced Hinduism. Indeed in southern part of India, especially Kerala and Tamilnadu, wood has always been the main medium for sculptures and other architectural pieces throughout all historic periods until recent decades.

In northern India, during the 2nd to 1st century BCE, in the Greco-Buddhist art of Kandhar, which at present is covered into southern Afghanistan and parts of northern Pakistan, the art of sculpture making became more detailed, the sculptures of the time represents essentially episodes of the Buddha's life and his teachings. Despite India's long sculptural tradition and a mastery of rich history of iconography, it was the first time in this era that Buddha was represented in human form. Before this period, his only representations were through some of his symbols. One of the probable reasons of this representation might be that because Kandharan Buddhist sculpture makers in modern Afghanistan shows signs of Greek and Persian artistic influence.

The sculptures made of pink sandstone in Mathura that date back from the 1st to 3rd centuries CE reflects both native Indian traditions and the Western influences on it. It became the major factor in establishing the basis for subsequent Indian religious sculpture making.

Hence preserving the rich art of sculpture is a great step towards preserving the cultural heritage of India. Digitalization of sculptures can help in making a centralized database of various types of sculptures found across the length and breadth of the country. Such information can then be used for teaching children or for letting visitors from other countries to know more about our culture and rich past heritage. Also it might prove of equal interest to scholars.

## 1.2 contribution of the project towards preserving history

This project is a step towards complete digitalization of sculptural heritage of India and deals with recognizing a particular sculpture and telling to which god it belongs. The images of these sculptures have same problems and are very difficult to work with. They have very minimal background- foreground difference. Also a lot of the sculptures are pretty deteriorated and are difficult to be recognized by human as well. For training a classifier for such images of the sculptures, it becomes mandatory to preprocess the images such that they don't have any of the problems as discussed above.

Currently, there exist no methods or techniques to classify the images Indian gods/goddesses. While thinking of such a 2-class classifier, we need images with similarity.

In our work, we have proposed a nested 2-class classifier that takes an image and classifies it to be from one of the four pre-defined classes (lord Ganesha, lord Hanuman, Natraj or Shivling). The HOG features are calculated over these images only after resizing them. After feature extraction, the classifier is trained and it classifies the query images as either belonging to one of the data classes. We have used predefined MATLAB classifier of SVM for classification. This gives an accuracy of 60% to 100%, depending on the dataset.

### 1.3 Related work

Regarding the sculptures' classification, there is not much work done. According to [17], a recognition system called Thai Buddhist Sculpture Recognition System (TBuSRS) is built which is trained and tested on around 50 types of different Buddhist sculptures with around 500 images in total. The features used for training are 14 in number namely, edge detection feature, eye, nose detection feature, bottom neck feature etc. In our case this method [17] is not suited because it considered only the face and recognizing it as a positive or a negative example of Buddhist sculpture. But, ours is a classification problem where we have two different kinds of datasets with variation in the poses in each dataset. Thus, the features as used in [17] were not feasible to be used in our case.

Features have always played an essential role in object detection and classification problems. Haar wavelet [17] is an effective feature for general object detection, especially when combined with the boosting-based learning framework. The SIFT feature [18] has become one of the most popular features for object recognition and image retrieval/matching due to scale/rotation invariant property.

We have used HOG features in our proposed method to extract the features of sculptures. HOG was used as feature descriptor Dalal and Triggs [16] for person detection. Talking of other features, the unique SIFT descriptor [18] was processed from the picture intensities around intriguing areas

in the picture space which can be alluded to as interest points, or keypoints. These interest focuses are achieved from scale-space-extrema of difference-of-Gaussians (DOG) inside of a difference-of-Gaussians pyramid. HOG descriptor [16] is popular for object recognition for many reasons. HOG uses vector-space model and the Euclidean distance between two HOG vectors is used to calculate the perceptual similarity. This means that many off-the-shelf learning and database algorithms can work directly on HOG representations. Another reason is that it uses intensity gradients rather than intensity directly which means that the responses of edges are localized.

HOG is similar to sift as it is also scale and rotation invariant but better and different because HOG computes descriptors without features as it computes them on a dense and isotropic grid. In [19], Feng et al used two-stage approach to detect people and vehicles in static images using extended histogram of oriented gradient (HOG) and SVM for classification. In the first stage, people and vehicle locations are hypothesized. This step takes the prior knowledge about what people and vehicle may look like in the depth map of the whole scene. The second stage is the verification of hypothesis using extended HOG and SVM.

For our purpose and the dataset, calculating SIFT features was not efficient as the aim of our problem was to classify the images as either class A or class B. For this HOG proved successful as the images of god/goddesses were human poses and in upright positions. HOG was better because the vector-space model of HOG made it easy to approximate the perceptual similarity between two HOG vectors using Euclidean (or cosine) distance. HOG focuses on local shape by capturing edge or gradient structure of that shape. In HOG the responses of edges are localized as HOG uses intensity gradients rather than using intensity directly. The response of the edges is sensitive to local but not global contrast due to normalization scheme. HOG is invariant to local geometric and photometric variations as compared to other descriptors. Rotations and translations within the cell do not affect the HOG values. Another important reason was time. The time to calculate HOG features over an image is about than 2 seconds. This made the use of HOG more suitable.

## 1.4 motivation

The work proposed here is a step towards preservation of our history. Digitalization helps in a better and efficient way of working. Taking the e.g. of inscriptions on the walls of monuments, digitalization can help in shaping the work of preservation in a better way. If the digital form of data is with us, then we can preprocess it, apply number of techniques for preserving them, extracting information out of them etc.

The main reason for tourism at ancient monuments is exploring past, learning the methods and techniques used by ancient people. After digitization, authorities looking after these monuments can build up their own databases where they can store the images, provide translation for the inscriptions in the user-desired language, provide an abstract about each sculpture and many such useful things that can help the past reach maximum people in the form convenient to them. For e.g. a blind person might be able to read inscriptions using Braille if there are some hand-held pads in front of these inscriptions, another tourist might like to know the history of a sculpture using ear phones or head phones.

All this can also be available online. That is, a user might just select his choice of sculpture or inscription from the dataset and he can translate it or read about it.

These are few practical applications and a major motivation behind this work.

## 1.5 scope of work

For the classification of sculptures, we have considered the images from INTERNET. The images were preprocessed by cropping and resizing them and converting them to binary images. Then using HOG, the features were extracted. We have used the classifier available in image processing toolbox of MATLAB in conjunction with their HOG descriptors to train the classifier and classify the sculptures as belonging to either of the four classes (on which the classifier has been trained). The classification results in the testing images being classified as either of the classes used for training. The advantages of classification of sculptures are listed below:

- After classification the information retrieval becomes easy.

- One might use this to know more about the god/goddesses; about the era they belonged to, the people who worshipped them.
- It helps in better categorization based on the temples they are worshipped at and other relevant details about them

These advantages are few areas of applications. There are many more such advantages.

## Chapter 2: Image processing using MATLAB

### 2.1 Introduction

MATLAB is a powerful tool for research practices in various fields of engineering. Nearly 50% of the applications of image processing and Computer Graphics are implemented using MATLAB. This chapter describes the MATLAB development environment and also in particular the image processing toolbox of MATLAB.

MATLAB is a data analysis tool basically designed to make handling and control of matrices easier. The basic operational capabilities of MATLAB can be enhanced by adding to it a wide set of toolboxes available for MATLAB. Each of such toolboxes is designed for a particular type of application. Like signal processing, image processing, neural networks etc. The fundamental set of the software clubbed with any additionally purchased toolboxes designed for particular purposes can provide the user a vast variety of functions that can be invoked by writing a script or by using a command window

MATLAB's basic set of operations are designed to work with matrix. A matrix may be understood as a multi-dimensional array of objects of the same data type. An  $r \times c$  matrix is a matrix having  $r$  number of rows and  $c$  number of columns, an element of a matrix  $M$  is referred as  $M [i, j]$ ,  $M_{i,j}$   $M_{ij}$  etc. In MATLAB a single variable may be considered as a  $1 \times 1$  matrix. In the same way, a string may be thought of as a  $1 \times n$  matrix of characters. An image, similarly may be taken as an  $n \times m$  matrix of pixel intensities.

This chapter summarizes the image processing operations and also introduces the practical implementation of some basic image processing operations. Further help for every MATLAB toolbox is available online. Help can also be availed by either selecting the "Help" menu item, or typing "help" at the command prompt.



## 2.1 MATLAB's Development Environment

MATLAB IDE can be initiated from within the Windows environment by clicking the MATLAB icon.

MATLAB's IDE has five primary parts. These are as listed below:

1. The Command Window
2. The current directory window
3. The workspace browser
4. The Command History Window and
5. Zero or more Figure Windows

1. **Command window:** The Command window is the space in the environment where all individual commands and expressions are typed. Also the result of every command if in the form of numeric or alphabetic, is displayed here.
2. **Workspace browser:** The name 'workspace' represents the set of variables that gets created during an active session of MATLAB. These are stored in the RAM memory and are automatically cleared once the MATLAB is closed. The workspace may also be cleared by using the command:

```
>>clear all;
```

We also have an option of saving the current workspace permanently for future use. So that we do not have to provide data for each and every run. It can be done with the help of available GUI options or with the following command:

```
>>save <file name>
```

The workspace is saved as a **.mat** file. Note that if no file name is given then the default file name '**matlab**' is used. The saved data can be loaded to the workspace anytime again by using the following command:

```
>>load <file name>
```

Along with the name of variables, the browser also displayed some additional information about every variable depending upon its type. Some variables also support the feature of editing in the workspace window only. Also the variables can be deleted from here as well.

3. **Current directory:** The current directory is the place where all new files are saved and picked from. Current directory window displays the path of current working directory along with its contents. The working directory of MATLAB can be changed at any time.
4. **Command history:** The command history window displays the list of all the previously executed commands in the present session as well as previous sessions. Commands appearing here can be re-executed. Note that the contents of command history window are not erased even after closing of current working session of MATLAB.

MATLAB provides an editor for writing scripts. It is invoked by typing edit in the command window. Scripts are stored with the extension .m and are therefore also known as m-files.

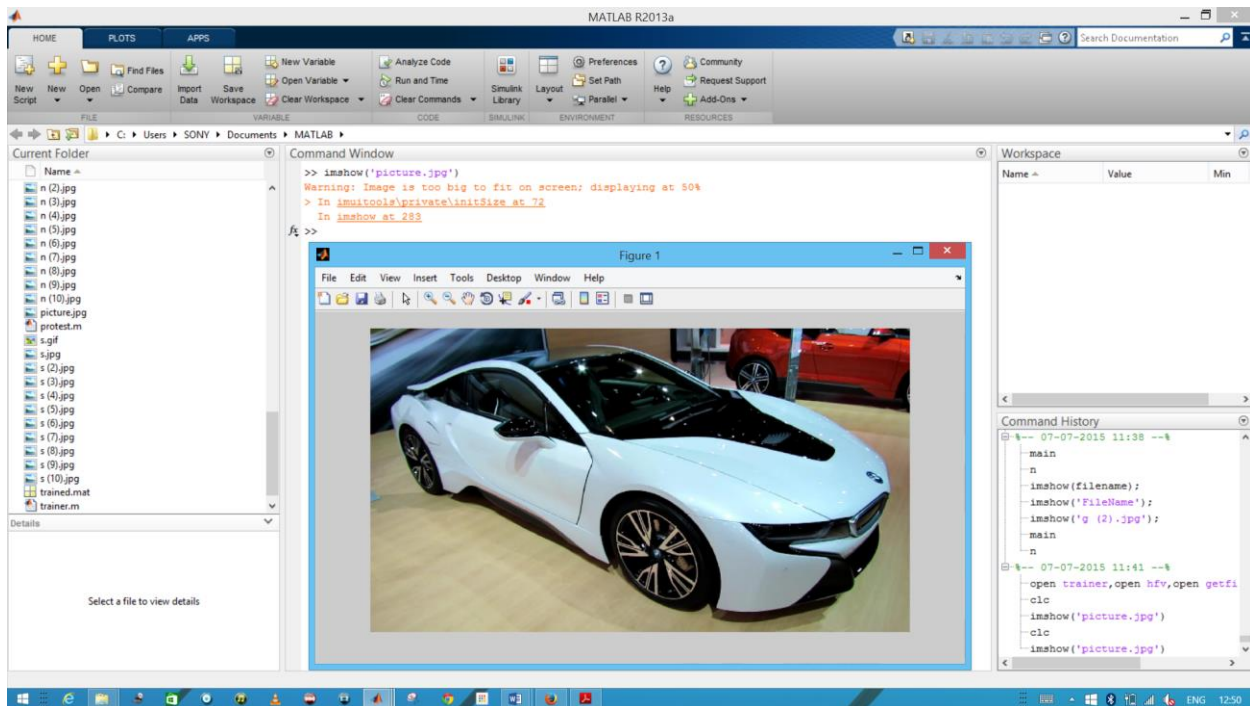


Fig 2.1: working window showing all parts of MATLAB

## 2.3 Image Representation

### 2.3.1 Image Format

An image may be thought of as an array of the values (pixels) of some visual characteristics of a scene. The characteristic could be one among many things associated with a scene, usually in case of a greyscale image we take the average brightness (one value) of the pixel. In case of an RGB image however, the individual brightnesses of the image for red, green and blue colors are taken.

The brightness value of each pixel is generally represented by an eight bit integer. Hence for such an 8 bit representation system, a total of 256 ( $2^8$ ) levels of brightness intensity are possible.

Another important property of an image is its *resolution*. It may be defined as the total number of pixels in an image along with the number of possible brightness levels for each pixel.

For efficient use of memory space, there are many methods available in MATLAB to compress the image. MATLAB supports working with all basic image formats.

### 2.3.2 Basic image operations

An image can be loaded into the workspace using the following command:

```
>> img = imread ('<filename>.<extension>');
```

Note that a semicolon is used at the end of every command in MATLAB. This is to suppress the display of output after each command. If we remove the semicolon then MATLAB will execute the command in the same way but also echo the results of the command onto the screen. In this case, screen will be loaded with the values of all elements of the just formed array *img*. Here we have assigned the image to an array *img*. Note that we also have to give the path where the image resides. If no path for the file is given, MATLAB will take the default path as current working directory and will display error if it is not found there.

The image can be displayed on the screen using the following command:

```
>> imshow(img, r)
```

Where *img* is the image that the user wants to be displayed, *r* defines the range of intensity levels used to display it. If no value for *r* is given, the default value 256 is used. We can also use the syntax [low, high] in place of *r*. this would define the upper and lower threshold values for displaying images. In other words, the pixel values less than *low* will be displayed as black (the intensity being set to 0), and the pixel values which are greater than *high* will be displayed as white (intensity set to maximum) .

Images are usually displayed in a separate figure window. But if a second image is displayed using the same command as discussed above, it will automatically overwrite the first image. To avoid this problem, we use the following command:

```
>> figure, imshow (img2)
```

This will open a new separate blank figure window and display the second image in that window. Note that we can give multiple commands in a single line by separating them with commas.

An image array may be permanently saved as an image file by using the following command:

```
>> imwrite(<array name>,<'file name'>.<extension>)
```

Note that some file formats also have additional arguments.

### 2.3.3 Image Information

Information about an image contains all the characteristics of the image file. Like date of creation, coding method, size, format, bit depth etc. the image information can be retrieved by using the command:

```
>> imfinfo filename;
```

This will create a structure containing the information about the image and will also display the information of the image in command window.

```
Command Window
>> imfinfo 'aa.jpg'

ans =

    Filename: 'C:\Users\SONY\Documents\MATLAB\aa.jpg'
  FileModDate: '09-Oct-2014 16:13:20'
   FileSize: 65605
    Format: 'jpg'
 FormatVersion: ''
    Width: 480
    Height: 640
   BitDepth: 24
   ColorType: 'truecolor'
 FormatSignature: ''
  NumberOfSamples: 3
   CodingMethod: 'Huffman'
  CodingProcess: 'Sequential'
      Comment: {}
         Make: 'SAMSUNG'
        Model: 'SM-N900'
   Orientation: 1
  XResolution: 72
  YResolution: 72
 ResolutionUnit: 'Inch'
      Software: 'N900XXUDNB2'
      DateTime: '2014:04:20 13:32:18'
 YCbCrPositioning: 'Centered'
   DigitalCamera: [1x1 struct]
      GPSInfo: [1x1 struct]

fx >> |
```

Fig 2.2: list of information displayed by imfinfo command

Also for any image array we can use the following command:

```
>>whos <arrayname>;
```

The above command work for any matrix.

## 2.4. Quantization of the image

### 2.4.1 Grey Level Ranges

In general, information of every pixel in an image is represented by 8 bits. (The reason for particularly choosing 8 bits is its compatibility with memory devices. Also due to optical limitations, many devices are not able to differentiate intensity levels more than 256). For display of color pixels, its 8 bits for each of the three colors and hence a total of 24 bits per pixel. But this

is not a hard limit on the number of bits and some devices even use higher number of bits per pixel for greater resolution. MATLAB supports the following image data types:

<b>Table 2.1: data types supported by MATLAB</b>			
<b>Sr no.</b>	<b>Type</b>	<b>Interpretation</b>	<b>Range</b>
1.	single	single precision floating point number	$[-10^{38}, 10^{38}]$
2.	double	double precision floating point number	$[-10308, 10308]$
3.	char	character	(2 bytes per element)
4.	int8	signed 8-bit integer	$[-128, 127]$
5.	int16	signed 16-bit integer	$[-32768, 32767]$
6.	<u>Int</u> 32	Signed 32-bit integer	$[-2147483648, 2147483647]$
7.	uint8	unsigned 8-bit integer	$[0, 255]$
8.	uint16	unsigned 16-bit integer	$[0, 65535]$
9.	uint32	unsigned 32-bit integer	$[0, 4294967295]$

An image is usually represented as being one of the following types:

- Intensity
- binary
- indexed
- RGB.

The pixel value of an intensity image represents its brightness. Pixel of a binary image have just two possible values. i.e. 0 or 1. Indexed image's pixel values are given as the index of a look-up table from which the "true" value of the pixel is read. RGB images have three characteristics corresponding to each pixel, representing intensities corresponding to red, green and blue illumination.

MATLAB also provides support for changing one type of images to another. This can be done by using the following command:

```
>> img2 = new_image_class_name(img);
```

Where

`new_image_class_name` is one of the allowed data types in MATLAB as given in the above table, e.g.

```
>> img2 = uint8(img)
```

This notation will create an image *img2* looking the same as *img* but having a data type **uint8**. Where *img* could be of any other data type. Note that the conversion process may be accompanied with a possible loss of resolution.

MATLAB provides functions for converting between different image types. In which case the data is scaled to fit the new data type's range. These functions are as following:

Table 2.2: functions to convert one image type to other

Function	Input Type	Output Type
im2uint8	Logical, uint8, uint16, double	Uuint8
im2uint16	Logical, uint8, uint16, double	Uuint16
mat2gray	double	Double in range [0, 1]
im2double	Logical, uint8, uint16, double	Double
im2bw	Uuint8, uint16, double	logical

#### 2.4.2 Number of Pixels

Images come in a number of different sizes and aspect ratios, but they are almost always rectangular in shape. There several methods of accessing the elements of an array, which the pixels of an image.

An element of an array can be accessed directly. Typing the array name at the command prompt will return all the array elements in the command window. Typing the array name followed by element indices in round brackets, will return the value of that pixel only.

Ranges of array elements can also be accessed using colons. Like following:

```
>> A(starting_index:finishing_index)
```

This will return all the elements between the starting and the finishing index from the one dimensional array A. Note that the indices of a matrix start with 1 and not 0.

```
>> A(start : step_size : end)
```

This will return elements after a jump of size '*step*' starting from the first value being *start* and finishing when the value of *end* is reached or exceeded.

This command is particularly helpful for accessing a particular portions of interest within an image. An image, *img*, could also be flipped by using the following command:

```
>> img_flipped = img(end : -1 : 1, :);
```

The keyword *end* represents the last element of an array. In case of an image, it represents the last pixel index of the image. Using the colon alone implies that all index values are traversed in sequential order.

To extract a portion of interest from image, following command is used:

```
>> img2 = img(top : bottom, left : right);
```

An image could also be subsampled (subsampling means picking out pixels after fix intervals and leaving intermediate pixels) using the following command:

```
>> fs = f(1 : 2 : end, 1 : 2 : end);
```

## 2.5 Point Processing

Point processing operations are the operations that manipulate individual pixel values. They may be divided into two categories depending on what property of the pixel they are changing i.e. its value or its position.

### 2.5.1 Value Manipulation

The most basic characteristic of a pixel in most of the cases is its brightness (in case of a grayscale image) or color (in case of an RGB image).



### 2.5.1.1 Pixel Scaling

Scaling of a pixel values may be defined as multiplication with a constant. MATLAB provides one single function that can be used to achieve. The function is as given below:

```
>> img2 = imadjust(img, [in_low, in_high], [out_low, out_high]);
```

This command takes the input range of values as specified along with and maps them to the output range that's also specified. Values that lie outside of the given input range are clamped to the extreme values of the output range i.e. values which are below the value `in_low` are all mapped to `out_low`).

If any of the above parameter is not given in the command then the default values will be assumed.

### 2.5.1.2 Histogram

The histogram of an image may be defined as the number of pixels with a given intensity of grey (in greyscale image) or color (in RGB image). Histograms are not usually used with RGB images. The histogram of an image containing  $L$  number of distinct intensity levels in the range of  $[0, G]$  may be defined by the mathematical relation:

$$Hs ( r_i ) = n_i$$

$R_i$  is the  $i^{th}$  intensity level in the image, and  $n_i$  will be the number of pixels with the intensity of grey equal to  $r_i$ . the default value of  $G$  is 255 for an image of type `uint8`, 65536 for image type `uint16` and 1.0 for a double image.

We frequently work with normalized histograms rather than simple histogram. A normalized histogram may be obtained by dividing each element of  $h(r_k)$  by the total number of pixels in the image. A normalized histogram is also known as the probability density function (pdf) as it gives the probability of occurrence of a given intensity level.

$$p(r_i) = n_i/N$$

where  $N$  is the total number of pixels of the image.

The histogram and normalised histogram in MATLAB may be displayed by the following commands:

```
>> h = imhist(img, b);
```

```
>> p=imhist(img,b)/numel(img);
```

Here **b** is the total number of bins in the histogram. If no value of **b** is given, then the default value of 256 is assumed. The function **numel(img)** would give the total number of elements in **img**, in this case this would represent the total number of pixels in the given image.

### 2.5.1.3 *Histogram Equalization*

It is possible to use different type of effects on a greyscale to achieve different outcomes. One of these effects is histogram equalization. The aim is to transform the grey scale such that the probability density function of the output image is uniform. The command is as following:

```
>> h = histeq(img, nlev);
```

Where **nlev** is the number of levels in the output image, **h**. Its default value is 64.

## Chapter 3: HOG based classification

Recognizing objects in images is one of the most important problems in computer vision. In a general approach, we first extract the feature descriptions of the objects to be recognized from reference images, and store such features to make a database. After that when we are given with a new image, its feature descriptions are extracted in the same way as done for reference images and compared to the object descriptions stored in the database created earlier. To see if the image matches with one of the categories we are looking for or is completely different. In practical applications of a classifier, there may be several differences between the query image and the reference database of images. Some such differences are:

- **Scale:** the two images can have same objects in them but with different sizes.
- **Orientation:** the sculptures may not be 100% vertically oriented in all the images.
- **Viewpoint:** images of the same sculpture taken from two different viewpoint may not seem exactly similar and may cause failure in the classification
- **Illumination:** as most of the classifiers use gradient of intensity, the images taken under different illumination may show different results.
- **Partially covered:** images of partially covered sculptures or taken from a place where it's not captured properly may cause problems for the classifier.

Sculpture images have a lot of the problems as discussed above. Apart from these problems, they also suffer from the problem of less differentiating colors between sculpture and its background. Also not all of the sculptures are well preserved. So the degradation of the sculptures make the recognizable features less prominent. So, as with any other image processing applications, the images of sculptures also need to undergo through some amount of preprocessing. Although here the amount of preprocessing requirement is higher.

The HOG based classification of sculptures deals with building a classifier that takes up the query image of a sculpture of a deity and gives its descriptive feature vector. In our work, we have taken 4 types of pictures of deities namely, lord *Ganesh*, *Hanuman*, *Shivling* and *Natraj*. Most images of the sculptures for the dataset are taken from INTERNET. These were the 2D colored images with color backgrounds. The multi colored background again posed a problem in calculating features

vectors. So, the database is made in such a way that most of the images have uniform and clear backgrounds.

Dataset was divided into train data set and test data set. Numerically 10 images of each deity (total 40 images) were used as training data. HOG features were calculated and training was done using SVM which is available as a command in MATLAB 2013A. The testing was done later and the accuracy varied between 60-100% on different images.

Below is explanation of few terms and algorithms which are already present for image recognition along with their advantages and disadvantages.

### 3.1 SIFT

**SIFT** stands for **Scale-invariant feature transform**. It is an algorithm in to detect and describe local features in images. It was published by David Lowe in 1999.

Applications of SIFT include object recognition, robotic navigation, stitching of image, 3D modeling, gesture recognition, identification of wildlife, match moving etc.

For any object in an image, we can extract interesting points on the object to provide what is known as a “feature description” of the object. This feature description, extracted from a training image, can be used to identify the object in a test image containing many other objects. To perform reliable recognition, however it is important that the feature description of the object extracted from the training image should be detectable even when there are changes in image scale, noise and illumination to perform reliable recognition. Such points of interest lie usually on high contrast regions of the image, for example object edges.

Another main requirement of these features is that their relative positions with respect to each other in the original scene shouldn't change between two images. For example, if we use only the corners of a window as interest points, they would work fine no matter what the window's position is; but if we use points in the frame as well then the recognition would not work if the window is opened or closed. In the Similar manner features located in articulated or flexible objects would not work if any change in their shape or internal geometry occurs between two images in the set being processed. However, in practical use, SIFT uses a larger number of features of the images. Hence

the contribution of the errors caused by such type of local variations in the overall average error of feature matching errors is reduced.

SIFT<sup>[4]</sup> can accurately identify objects even under partial occlusion and among clutter, because the SIFT feature descriptor is invariant to uniform scaling, orientation, and partially invariant to affine distortion and illumination changes.<sup>[5]</sup> below here is the summary of Lowe's object recognition method.

For an object, SIFT keypoints are first extracted from a set of reference set of images<sup>[5]</sup> and in this way a database is created. An object afterwards is recognized in a test image by comparing each feature of the new image individually to this database and hence finding out the matching features on the basis of Euclidean distance of their feature vectors. From this complete set of matches, those subsets of keypoints which agree upon the location, scale and orientation of the object in the test image are identified to filter out good matches. The consistent clusters are determined rapidly by the use of a hash table implementation of the Hough transform (known as generalized Hough transform). Each and every cluster having 3 or more features agreeing on an object and its orientation is subjected to a further detailed model verification where in following steps subsequently outliers are discarded. Ultimately the probability that the given set of features indicates the presence of an object or not is computed, by using the accuracy of fit and total number of probable false matches. Object matches that pass all of the above verification stages are then identified to be correct with high confidence.

### 3.2 SURF

The process of finding features using SURF have 2 steps:

1. First, we find the interest points in the image which promise to contain meaningful structures; this is done by comparing the Difference of Gaussian (DoG) in every location in the given image under different scales. We also calculate a major orientation if a point is considered to be a feature point.

2. The next step is to make the scale invariant descriptor for every interest point found in step 1. For achieving rotation invariance, we align a rectangle to the major orientation calculated above. The size of this rectangle is proportional to the scale at the place where the interest point is detected. Then the rectangle is cropped into a grid of 4 by 4. Different informations of the image

such as gradient or absolute value of gradient are subtracted from each of these rectangular regions and are composed into the interest point descriptor.

The SURF feature is a faster version of SIFT. It uses an approximated DoG and the trick of integral image. The integral image method is similar to the one used in Viola and Jones' face detector using adaboost. An integral image, is nothing but just an image in which each of its pixel value is the sum of all the original pixel values which lie left and above it. The advantage of using an integral image is that it can compute block subtraction between any 2 blocks with just 6 calculations. Which means that finding SURF features of an image could be several order faster than the finding the traditional SIFT features.

### 3.3 HOG

The idea behind the HOG (Histogram of Oriented Gradient) descriptors is that appearance of a local object and its shape in an image can be easily described by calculating the distribution of intensity gradients of the object and its edge directions. The implementation of these descriptors can be achieved in the following way:

1. First, the image is divided into small connected regions, known as cells.
2. Then for each cell, a histogram of gradient directions or edge orientations for all the pixels within the given cell is compiled. The collection of all these histograms then represents the HOG descriptor.
3. For better accuracy, the contrast-normalization of the local histograms can be done by calculating the average intensity across a larger region of the image (known as a block), and then normalizing all cells within that block using this value.

The HOG feature vectors are commonly used for image recognition. This algorithm divides the image into small cells, computes the gradient of intensity magnitude and their orientation, and then normalises the resulting histogram over small blocks. Such given descriptor may then be used to train any classifier such as SVM and hence classify other images.

#### **A detailed theoretical explanation of the algorithm given below:**

Histogram of Oriented Gradients (HOG) contains the following steps:

- 1. Gradient computation: HOG takes a grayscale image and calculates two intensity gradient matrices (X-gradient matrix and Y-gradient matrix) from the given image matrix. The X-gradient of a pixel may be given as the difference of intensity of its two neighboring pixels in horizontal direction. The gradient of each and every pixel is calculated in similar way and collected in a matrix. Similarly the Y-gradient of a pixel may be given as the difference of intensity of its two neighboring pixels in vertical direction.

The method of calculating X and Y gradients is as follows:

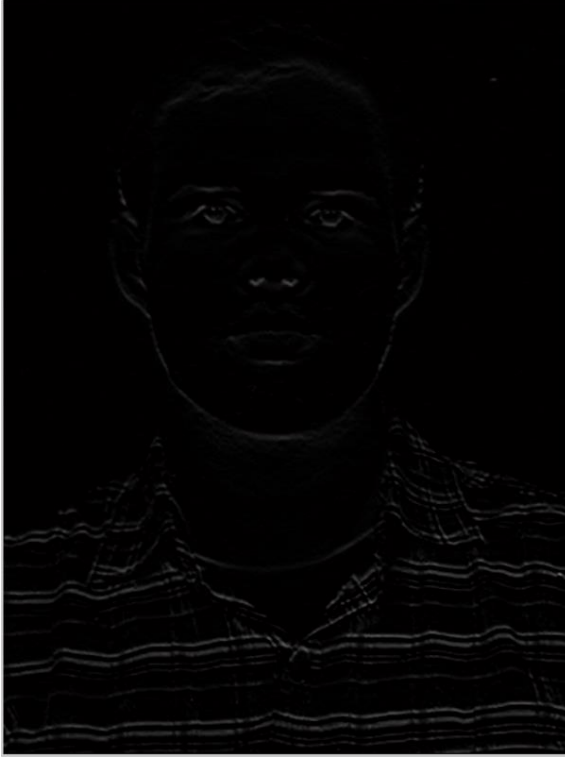
$$G_x(i,j) = \text{img}(i+1,j) - \text{img}(i-1,j)$$

$$G_y(i,j) = \text{img}(i,j+1) - \text{img}(i,j-1)$$

Where 'img' is the given grayscale image and Gx and Gy are the X and Y gradients respectively. The process when repeated for the each pixel, gives the whole gradient matrices. One test image and its X and Y gradient images are as given below:



**Fig 3.1: original grayscale image**



**Fig 3.2: Y - gradient image**



**Fig 3.3: X – gradient image**

As it can be seen that the Y-gradient of the image shows the difference of intensities in vertical direction. Similarly X-gradient shows the difference of intensities in horizontal direction. The gradients are taken instead of original intensities because the information lies in change of intensities and not in the intensities itself. Also they provide considerable immunity to local intensity variations.

**2. Gradient magnitude and edge direction calculation:** the total gradient magnitude and directions are calculated by vector norms.

The magnitude is given as:

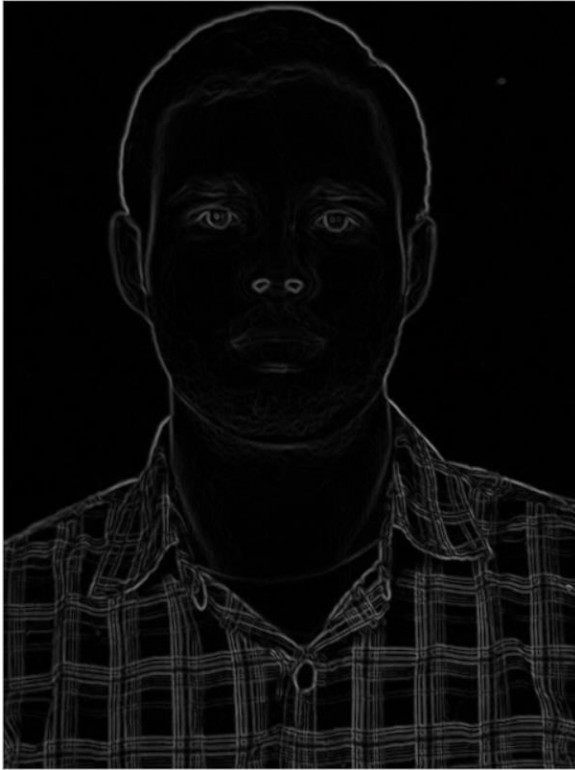
$$\text{Mag} ( I , j ) = \text{sqrt} ( ( G_x ( I , j ) ) ^ 2 + ( G_y ( I , j ) ) ^ 2 )$$

The angle can be given as:



$$\text{Theta} ( I , j ) = \text{atan} ( G_x ( I , j ) / G_y ( I , j ) )$$

The magnitude and angle matrices can be viewed as images. These would look as shown below:



**Fig 3.4: the magnitude matrix**



**Fig 4: the angle matrix**

as can be seen by the images, the magnitude matrix contains information about intensity change in both X and Y direction. The angle matrix ( or the direction matrix) does not contain much useful visual information. And its just a measure of the arctangent of ratio of intensity variations in both directions. these matrices are further used to make histograms.

- 3. Division into cells:** the whole magnitude and image matrices are divided into small interlaced cells. Each cell containing 1/3 number of elements common with previous cell and 1/3 number of pixels common with the next cell both in horizontal and vertical direction. The number of cells may be chosen according to the requirement of the HOG into a particular application. Larger number of cells means more information about the image and hence greater differentiation among them. However it increases the calculation time significantly.

**4. Histogram formation:** for each cell, a histogram is calculated as below:

1). The whole set of values of the angle matrix (  $-180^\circ$  to  $180^\circ$ ) is divided into several (say  $n$ ) bins of equal range on angle. Each having a range of  $360^\circ/n$ .

2). If the angle of a pixel from a cell lies in the range of a bin then the magnitude of the matrix is added to the magnitude of the bin. Mathematically it is given as:

**If**      angles (  $k$  ) < ang\_limit ( bin )

**Then**   Magnitude ( bin ) = magnitude ( bin ) + magnit (  $k$  )

**5. Histogram assembling:** the histograms calculated above for each cells are then concatenated into one big histogram. It is a 1- dimensional column matrix containing only one column and  $n*m*b$  nuber of elements. Where  $n$ ,  $m$ ,  $b$  are number of cell divisions horizontally, number of cell divisions vertically and number of cell divisions respectively.  $n*m$  also gives the total number of cells.

**6. Histogram normalisation:** the histogram can be normalised for contrast immunity towards local contrast variation as following:

$$H2=H2/ (norm (H2) +0.01);$$

The histogram thus calculated is the desired HOG descriptor and can be used to train any classifier.

## Chapter 4: Support vector machine

It is a kind of machine which is used as supervised learning models with the help of associated algorithms to recognize the data and for the pattern analysis. An SVM model can be represented as points in space, mapped in such a way that the examples of the different categories are separated by a clear gap which can be as wide as possible. It is related to statistical learning theory. SVM algorithms are considered to be the best algorithms for supervised learning.

### 4.1 History of SVM:

SVM was invented by Alexey Ya. Chervonenkis and Vladimir N. Vapnik in 1963. But its actual implementation was first introduced in 1992 when Isabelle M. Guyon, Bernhard E. Boser, and Vladimir N. Vapnik suggested that non-linear classifiers can be created by using KERNEL TRICK for maximum margin hyperplanes. The soft margins technique also termed as current standard incarnation was suggested by Vapnik and Corinna Cortes in 1993 and it was published in 1995. Due to its handwritten digit recognition feature it was one of the most fascinating techniques. The test error rate for SVM is nearly 1.1%. This is similar to the error rates of a carefully constructed neural network termed as LeNet 4.

### 4.2 Basic Definition of SVM:

A Support Vector Machine (SVM) is a discriminative classifier which can be defined by a separating hyperplane. Also it can be stated as given labelled training data (*supervised learning*), the algorithm provide an output of optimal hyperplane which categorizes the new examples.

Let's consider a simple problem to determine in which sense is the hyperplane that we obtained optimal?

For a set of 2D-points which are linearly separated and belong to one of two classes, let us find a separating straight line. The figure for this is depicted on the next page.

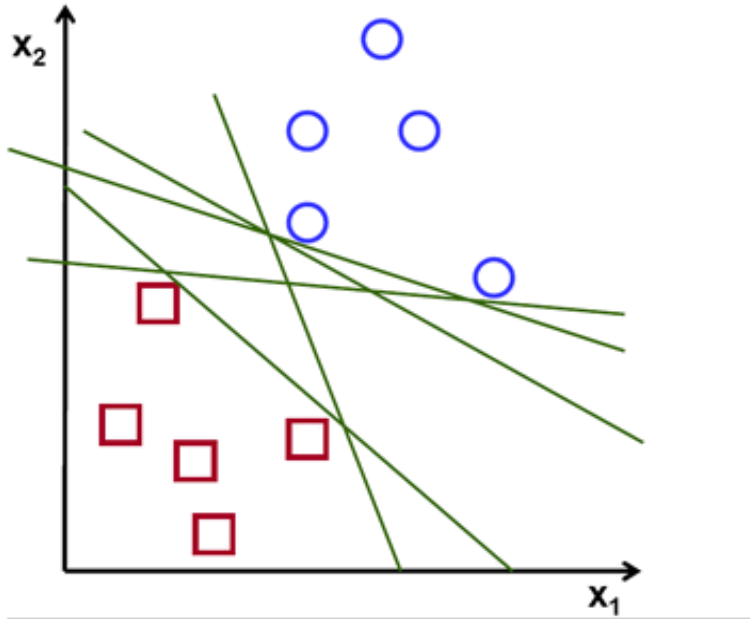


Fig 4.1: examples of different planes as an example of a classifier

In the above picture we find that there are multiple lines that provide a solution to the problem. Are any of them better than the other ones? We can define a criterion to estimate the worthiness of these lines:

**Bad line:** A line is said to be bad if it passes the points too closely. In such a case it will be considered as noise sensitive and it cannot generalize correctly. Therefore, we should focus on finding the line which is passing as far as possible from all the points.

After that, the operation of SVM algorithm is based upon finding the hyperplane which gives the largest (best) minimum distance to the training examples. So, the optimal (best) separating hyperplane will maximize the margin of the data.

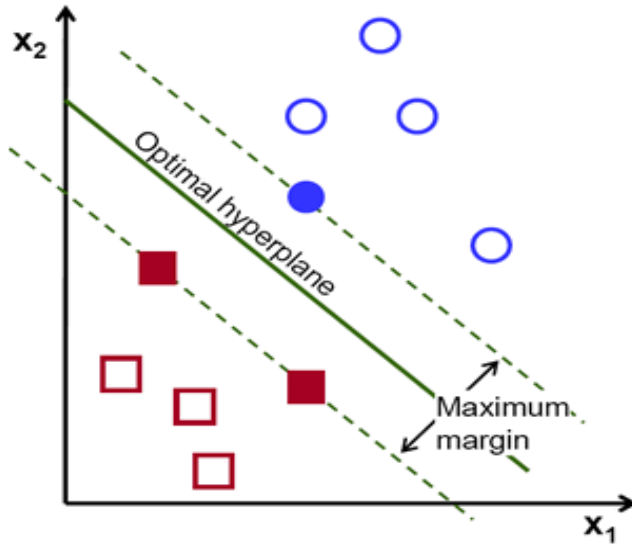


Fig 4.2: an optimal decision boundary

Examples for bad decision boundary:

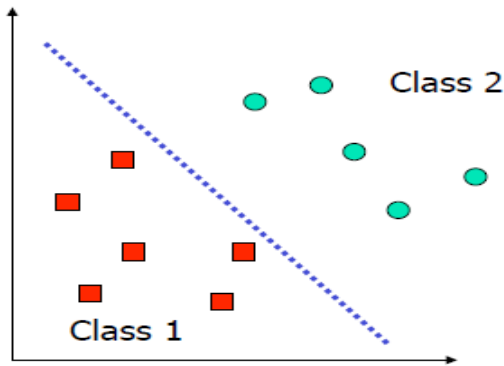


Fig 4.3: a bad decision boundary (not equally spaced)

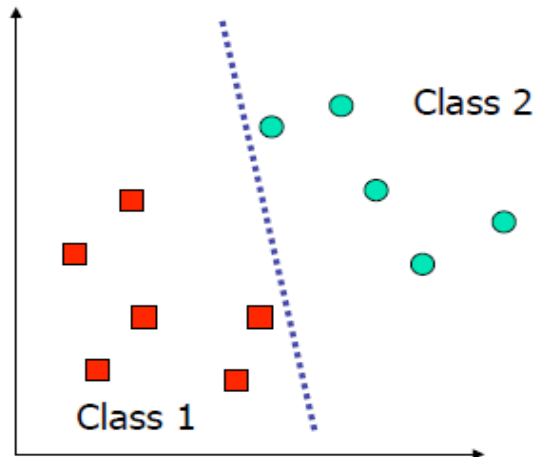


Fig 4.4: a bad decision boundary (not giving maximum separation)

### 4.3 Calculating the optimal Hyperplane

The notation that is used to define the hyperplane is defined as

$$f(x) = \beta_0 + \beta^T x$$

Where  $\beta$  is defined as the weight vector and  $\beta_0$  is called bias.

The optimal hyperplane is represented as an infinite number of distinct ways by scaling  $\beta$  and  $\beta_0$ . From all of the possible representations for the hyperplane, the best one chosen is

$$|\beta_0 + \beta^T x| = 1$$

Here  $x$  is defined as the examples considered for training which are closest to the Hyperplane. Such kinds of examples are termed as support vectors and the representation that is used for this is called as canonical hyperplane.

Now by using the formula from co-ordinate geometry for calculating the distance between  $x$  and the hyperplane selected as  $(\beta, \beta_0)$ .

$$\text{Distance} = \frac{|\beta_0 + \beta^T x|}{\|\beta\|}$$

For canonical hyperplane the value for the numerator is considered to be 1. So the above equation can be written as:

$$\text{Distance} = \frac{1}{\|\beta\|}$$

Also we know that the maximum margin is considered to be the twice of the distance between the closest examples. So we can write

$$M = \frac{2}{\|\beta\|}$$

Where M is the symbol used for margin.

In the end we have to maximize the margin which is equivalent to minimize a function  $L(\beta)$  with respect to some constraints. This process is a kind of Lagrangian optimization that is used to solve these calculations by using Lagrangian multipliers for calculation of the coefficients weight vector and bias.

#### 4.4 Biased and Unbiased Hyperplanes:

When the hyperplane is assumed to be passing through the origin of the coordinate system, such hyperplanes are termed as unbiased hyperplanes. These hyperplanes are considered only for the aspects of simplification. In case of general hyperplanes it is not necessary that the hyperplane pass through the origin. Such hyperplanes are called biased hyperplane. An unbiased hyperplane can be realized by setting  $\beta = 0$  in the primal optimization.

#### 4.5 Extensions in SVM:

##### 1. Multiclass SVM:

Multiclass SVM is used for the assignment of labels to instances with the help of support vector machines, in which the labels are drawn using a finite set of various elements. The main approach for this is to decrease the single multiclass problem into a multiple binary class problem. The Common techniques for such reduction include:

- **Building binary classifiers:** These classifiers distinguish between (i) one label out of all the labels and the rest label that is one-versus-all method is used (ii) between each and every pair of labels that is one-versus-one technique is used. Classification of the instances for situation of one-versus-all case is performed by using a theme of winner takes all strategy. In this scheme the classifier having the highest output function assigns the class whereas for the case of one-versus-one technique, classification is done with the help of max-wins voting strategy where every classifier assign the value of instance to one of two classes. After that the vote for the assigned class get increase by one vote and in the end the class which has the most votes will determine the instance classification.
- Error-correcting output codes
- Directed acyclic graph SVM (DAGSVM)

#### 4.6 Strengths of SVM:

Some of the strengths of SVM are:

- Provides easy training as no local optimum will be needed.
- Can be easily applied to high-dimensional data
- It is easily applicable to trees and strings which act as input to SVM.
- Trade-off between the error and the complexity of classifier can be controlled easily.



#### 4.7 Issues regarding SVM:

The main drawbacks of SVM are stated as:

- Un-calibrated class membership probabilities.
- It can only be directly applied for two-class tasks. So, algorithms which reduce the multi-class task into several binary problems are to be applied.
- It is difficult to calculate the parameters of a solved model.

## Chapter 5: The MATLAB code

The MATLAB code for the project has the following 6 scripts and functions:

1. Main.m
2. Getfile.m
3. Trainer.m
4. Hfv.m
5. Cfy.m

Their brief introduction and purposes are explained below.

1. **Main.m** is the main program that is used first to interact with the user. It asks the user whether he wants to train the classifier first or use a pre-trained classifier. If the user wants to train the classifier first, then the scripts **getfile.m** and **trainer.m** are called. If the user wants to use the pre-trained classifier then a MAT file **trained.mat** is loaded. Which contains all the training data for the classifier that is stored previously to avoid the need of training the classifier before each run.
2. The script **getfile.m** is written to take 10 images of each deity as an input from the user. Converts it to a grayscale image if it's an RGB image and resizes in to (640, 480). Finally it stores them as an array for each deity. Hence there are 4 such array after the execution of the script. Each containing 10 images of a deity in grayscale format with a fix size of (640, 480). At the end of **getfile.m**, we get the following variables in the workspace:

**Table 5.1: workspace after implementation of getfile.m**

Sr. no	Variable name	Type	Description
1	Ganesh	<640x480x10 uint8>	Contains 10 grayscale images of lord Ganesha arranged in one matrix
2	Hanuman	<640x480x10 uint8>	Contains 10 grayscale images of lord Hanuman arranged in one matrix
3	Natraj	<640x480x10 uint8>	Contains 10 grayscale images of Natraj arranged in one matrix

4	Shivling	<640x480x10 uint8>	Contains 10 grayscale images of Shivling arranged in one matrix
---	----------	--------------------	---

3. The script **trainer.m** takes the image arrays created by **getfile.m** and gives it to **hfv.m** for calculating HOG features for each of them. It then stores the feature vector of all 10 images of a deity into one single matrix. Hence creating 4 matrices, one for each deity containing HOG feature vector of 10 images in such a way that each row represents HOG feature vector of one image. After creating the matrices of feature vectors, it gives the matrices to SVM classifiers. In our project three classifiers are used. One classifies the image whether it belongs to lord Ganesha/Hanuman or Natraj/Shivling and gives the result for working of 2<sup>nd</sup> and third classifier. 2<sup>nd</sup> classifier classifies whether the image belongs to lord Ganesha or Hanuman. 3<sup>rd</sup> classifier classifies whether the image belongs to Natraj or Shivling.

After the end of **trainer.m**, the training work is completed and we get the following variables in our workspace.

**Table 5.2: workspace after implementation of trainer.m**

Sr. no	Variable name	Type	Description
1	Ganesh	<640x480x10 uint8>	Contains 10 grayscale images of lord Ganesha arranged in one matrix
2	Hanuman	<640x480x10 uint8>	Contains 10 grayscale images of lord Ganesha arranged in one matrix
3	Natraj	<640x480x10 uint8>	Contains 10 grayscale images of lord Ganesha arranged in one matrix
4	Shivling	<640x480x10 uint8>	Contains 10 grayscale images of lord Ganesha arranged in one matrix
5	Ganesh_HOG	<10x172800 double>	Contains the HOG feature vectors of all the 10 images of lord Ganesha in one matrix

<b>6</b>	Hanuman_HOG	<10x172800 double>	Contains the HOG feature vectors of all the 10 images of lord Hanuman in one matrix
<b>7</b>	Natraj_HOG	<10x172800 double>	Contains the HOG feature vectors of all the 10 images of Natraj in one matrix
<b>8</b>	Shivling_HOG	<10x172800 double>	Contains the HOG feature vectors of all the 10 images of lord Shivling in one matrix
<b>9</b>	T_data	<40x172800 double>	Contains all the 4 HOG matrices concatenated row wise in one matrix
<b>10</b>	T1_data	<20x172800 double>	Contains the HOG matrices of lord Ganesha and lord Hanuman concatenated row wise in one matrix
<b>11</b>	T2_data	<20x172800 double>	Contains the HOG matrices of Natraj and Shivling concatenated row wise in one matrix
<b>12</b>	Svt	<1x1 struct>	Trained classifier for classification between lord Ganesha/Hanuman and Natraj/Shivling
<b>13</b>	Svt1	<1x1 struct>	Trained classifier for classification between lord Ganesha and Hanuman
<b>14</b>	Svt2	<1x1 struct>	Trained classifier for classification between Natraj and Shivling

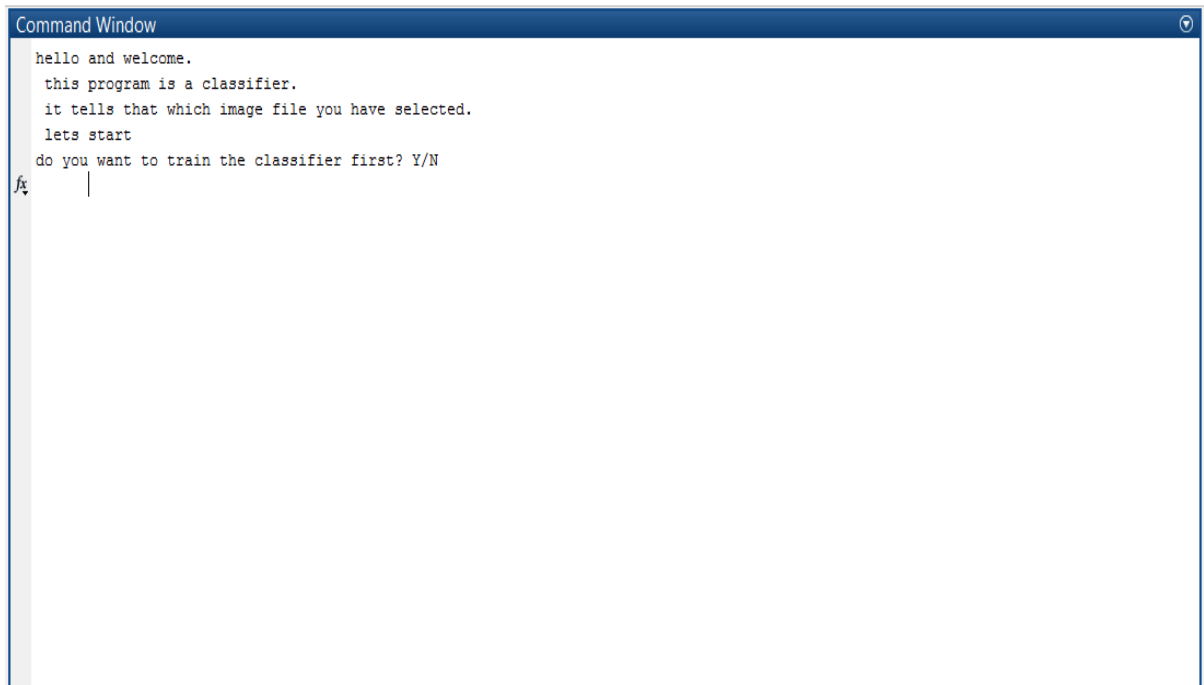
The workspace now contains all the necessary data for classification and can be stored for use in future to avoid training. One such instance of the workspace is already stored under the name of **trained.mat**.

4. The function `hfv.m` is the HOG descriptor used. It takes a grayscale image on size 640 X 480 as its argument and returns its feature vector. It is used by `trainer.m` to find feature vector of training image and by `cfy.m` to find the feature vector of the test image.  
The feature vector given by `hfv.m` is a row vector with  $N \times M \times B$ . where N, M, B represents number of horizontal divisions, number of vertical divisions and number of bins for an image respectively.
5. The script `cfy.m` is the classifier function. It does the following work:
  - a. Takes the test image from the user.
  - b. Converts it to grayscale image if it is an RGB image
  - c. Resizes the image to 640×480.
  - d. Gives the pre-processed image to `hfv.m` for calculating HOG feature vector
  - e. Uses the pre-trained classifier to evaluate the class of the image.
  - f. Displays the image back with the title showing the class of the image.

## Chapter 6: The working of the project

The code working of the code with the windows that appear for UI are shown below in the sequence of their appearance.

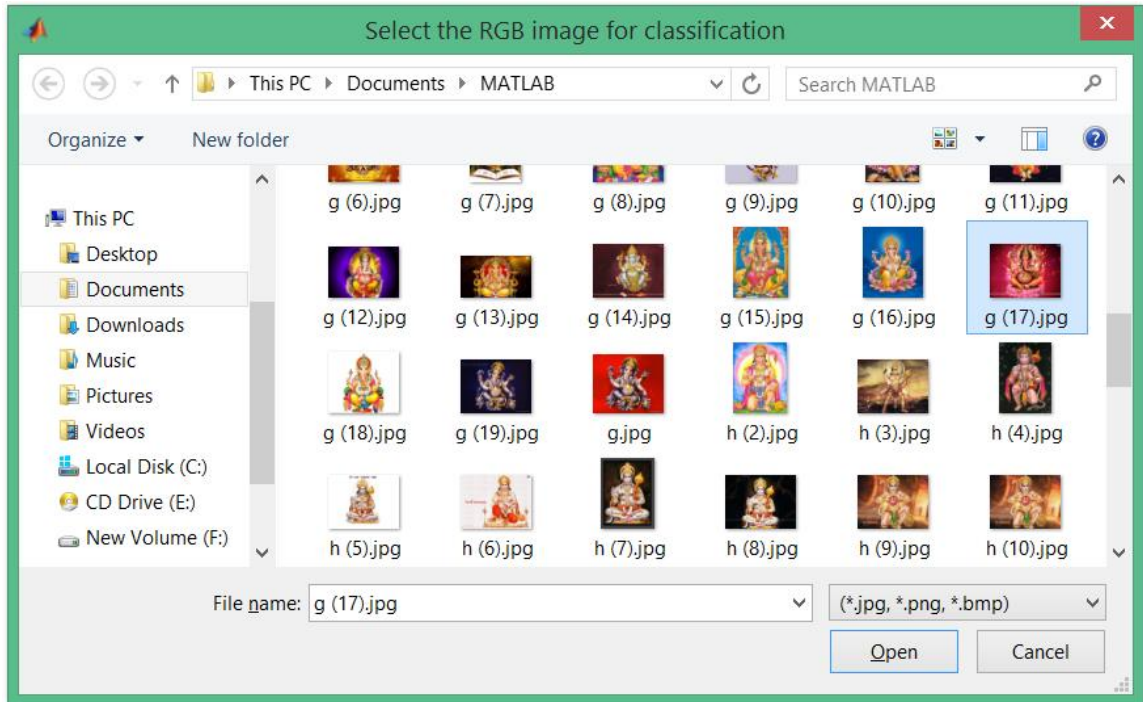
1. The code starts by running the main program.
2. It asks welcomes the user and asks whether he wants to train the classifier first. The interaction is done through command window. Below is the view of the command window.



```
Command Window
hello and welcome.
this program is a classifier.
it tells that which image file you have selected.
lets start
do you want to train the classifier first? Y/N
fx |
```

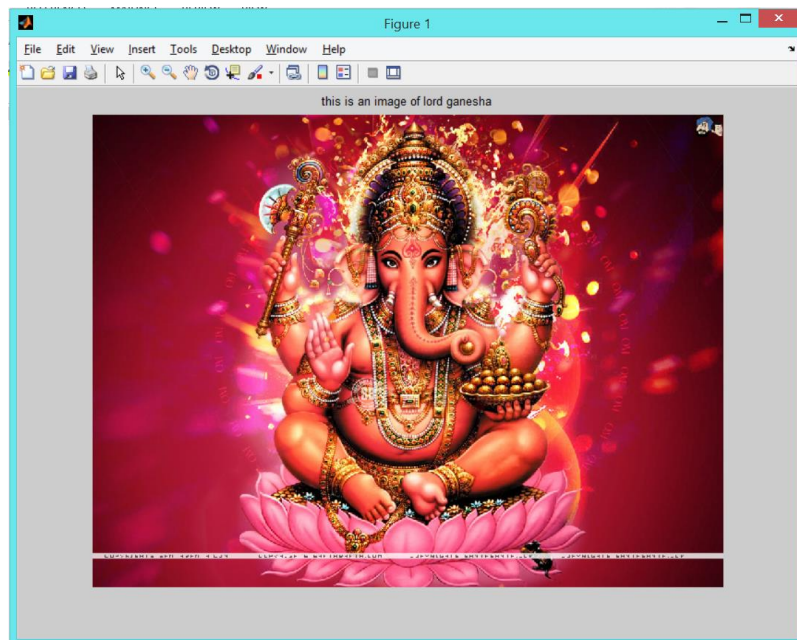
Fig 5.1: code in working image (1)

3. The user gets the choice of choosing whether he wants to train the classifier first or wants to work with the pre- trained classifier. Let's first see the working with the trained classifier. I.e. when we give the response as 'N'.
4. It gives the user a selection window and asks him to select a test image for classification. Let's for example take image "g (17).jpg".



**Fig 5.2: code in working image 2**

- The input image is taken for the processing. The computer takes a little time for feature vector calculation and then classification. After that it shows the image in 'imshow' window. Along with the title showing the name of the deity to which the image belongs.



**Fig 5.3: code in working image 3**

- Along with the image and its title, it shows the time elapsed in classification. The information is displayed in command window.

```

Command Window
hello and welcome.
this program is a classifier.
it tells that which image file you have selected.
lets start
do you want to train the classifier first? Y/N
n
classification starts
Warning: Image is too big to fit on screen; displaying at 67%
> In imuitools\private\initSize at 72
In imshow at 283
In cfy at 27
In main at 13
classification complete
Elapsed time is 5.245248 seconds.
fx >> |

```

Fig 5.4: code in working image 4

- In case if the user choose to train the classifier, he has to choose 40 images one by one for the training. That is 10 images each for each deity. The program asks for then images one by one. It tells which deity's image it's asking for. Also the command prompt displays the number of the image.

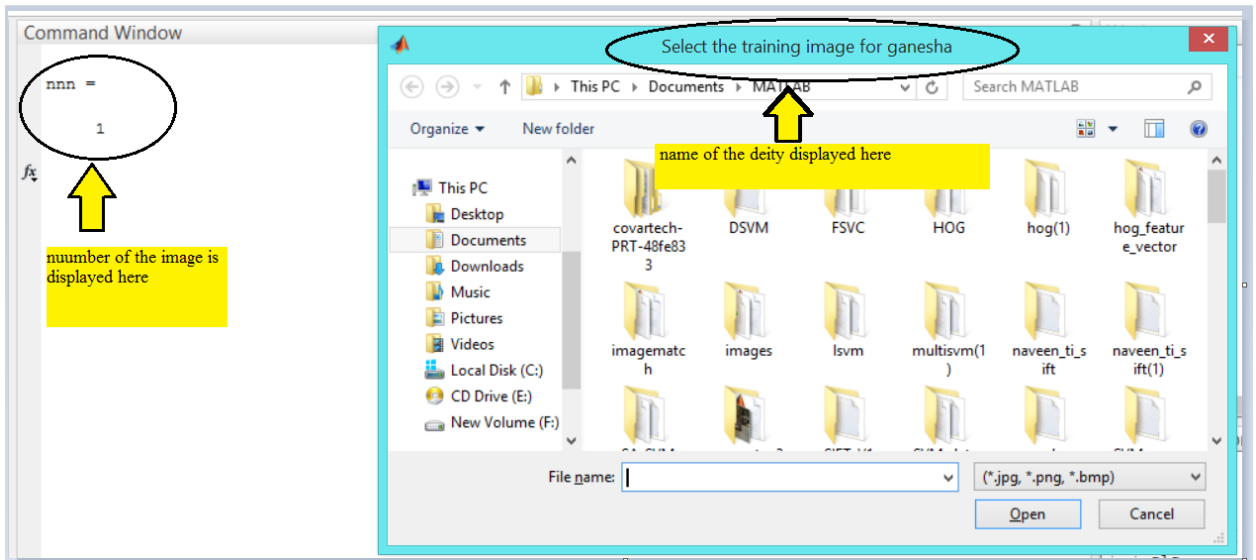


Fig 5.5: code in working image (4)



8. After accepting the 40 images, a message is displayed on command window saying ‘ *get file terminates*’.
9. After a wait of 2 seconds, another message flashes on the command window saying ‘ *SVM training starts*’.
10. the environment shows a sign of ‘ **busy**’ as the training of the SVM is going on in the background.
11. After the training is over another message is displayed saying ‘ *SVM training ends*’ along with the message, the time taken in the training is also displayed.
12. After step 11 the working is same as that for a pre-trained classifier. And the steps 3 to 6 are followed.

Some other examples of the working of the project which followed the same process are shown below along with the time taken for calculation:

1. Example run for lord ganesha:

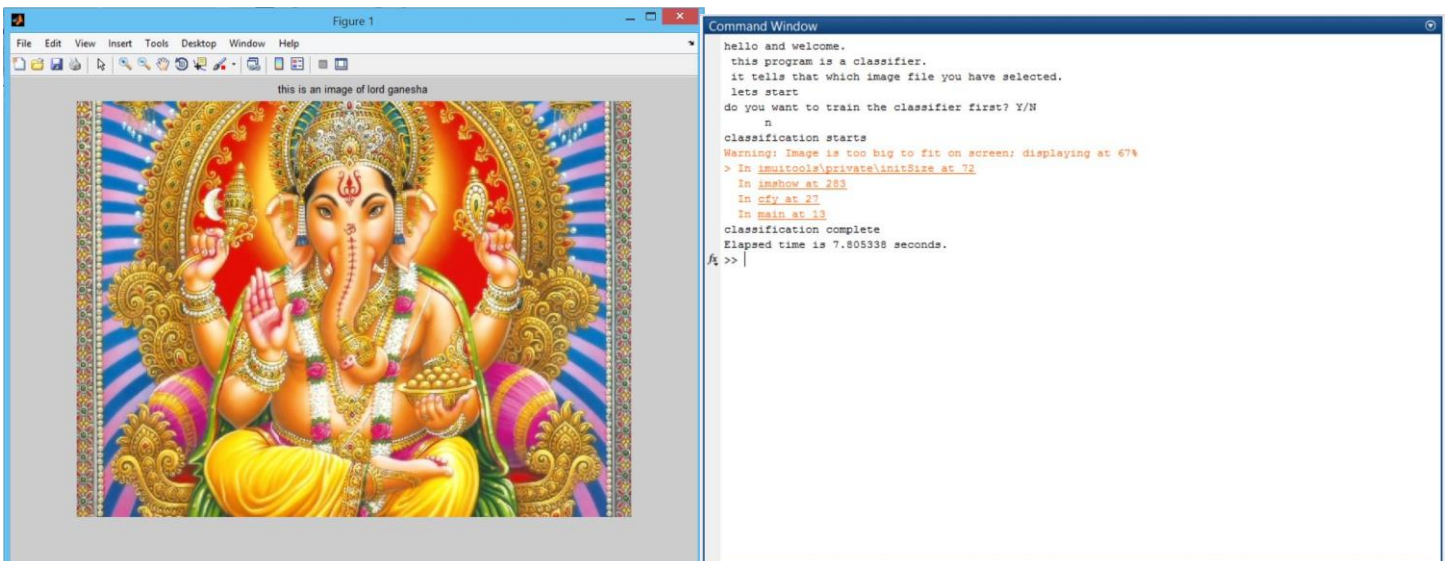


Fig 5.6: example run for lord Ganesha  
a)output of the program b)the time taken to calculate output

## 2. Example run for lord hanuman

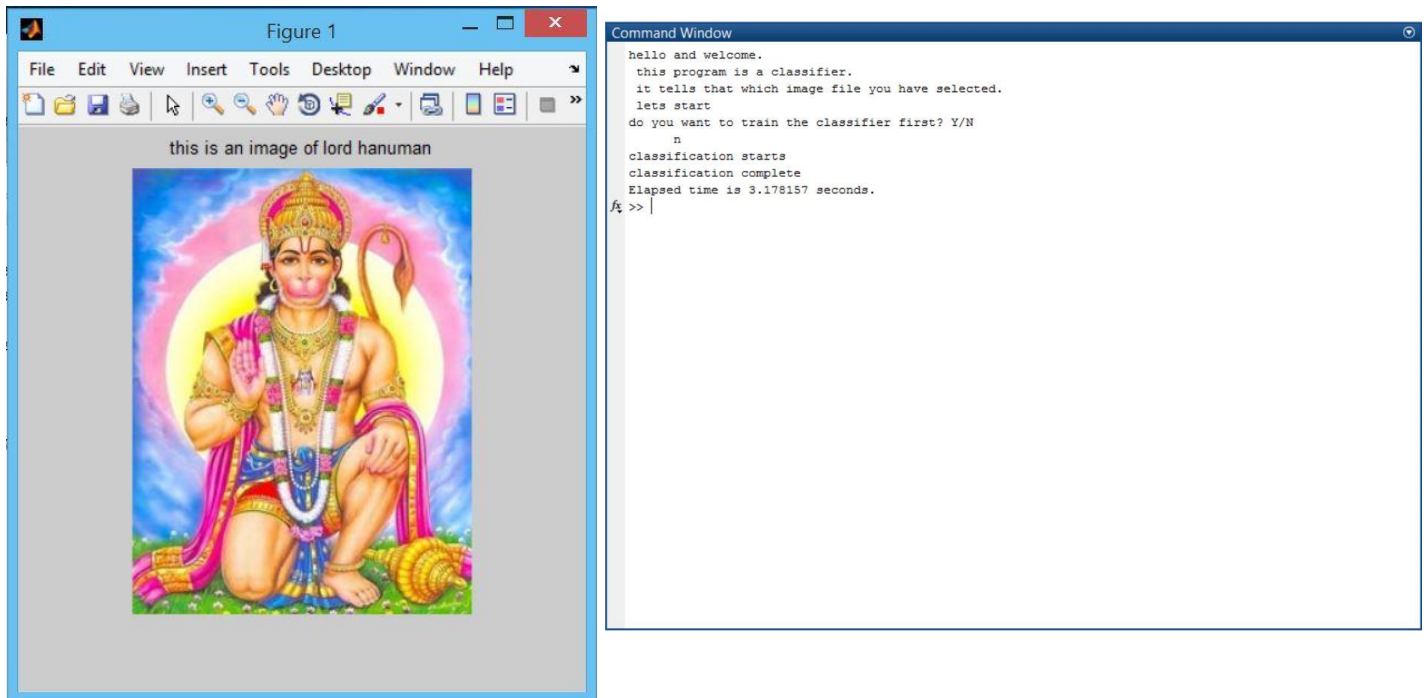


Fig 5.7: example run for lord hanuman a)output of the program  
b)the time taken to calculate output

## 3. Example run for Natraj

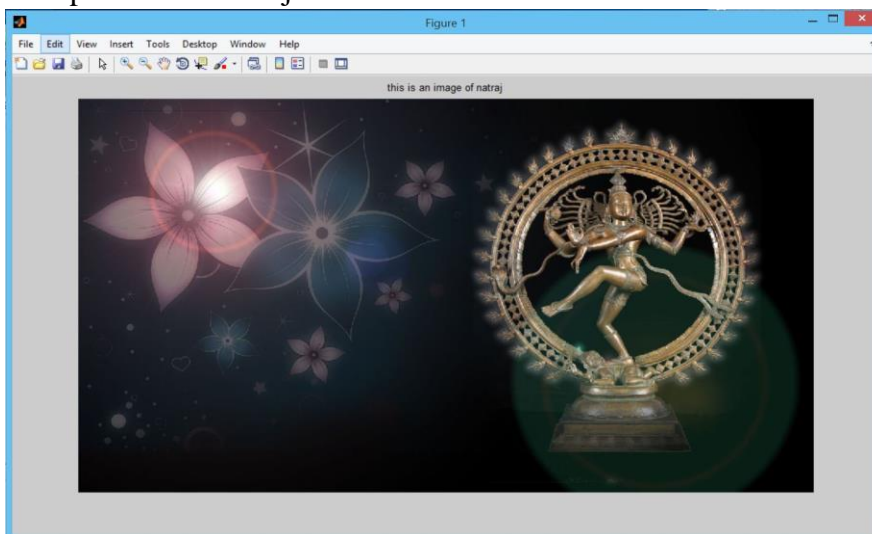
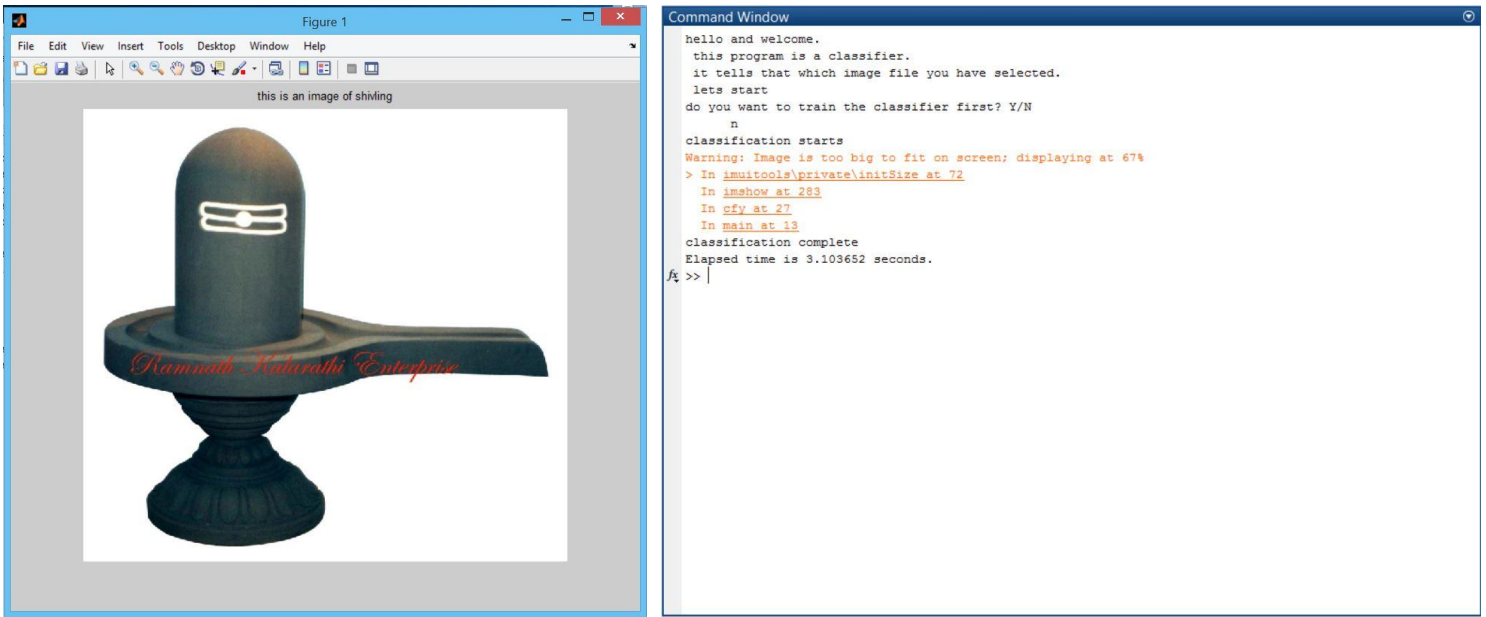


Fig 5.8: example run for natraj  
:output of the program

```
Command Window
hello and welcome.
this program is a classifier.
it tells that which image file you have selected.
lets start
do you want to train the classifier first? Y/N
n
classification starts
Warning: Image is too big to fit on screen; displaying at 67%
> In imuitools\private\initSize at 72
In imshow at 283
In cfy at 27
In main at 13
classification complete
Elapsed time is 3.266717 seconds.
fx >> |
```

Fig 5.9: example run for lord natraj :the time taken to calculate output

#### 4. Example run for shivling



The figure consists of two side-by-side windows. The left window, titled 'Figure 1', displays an image of a shivling (a black, cylindrical stone with a white 'U' shaped mark on top) on a pedestal. Below the image, the text 'this is an image of shivling' is visible. The right window, titled 'Command Window', shows the program's execution output, including a warning about the image size and the elapsed time of 3.103652 seconds.

```
Figure 1
File Edit View Insert Tools Desktop Window Help
this is an image of shivling
Warning: Image is too big to fit on screen; displaying at 67%
> In imuitools\private\initSize at 72
In imshow at 283
In cfy at 27
In main at 13
classification complete
Elapsed time is 3.103652 seconds.
fx >> |
```

Fig 5.10: example run for shivling a)output of the program b)the time taken to calculate output

## Chapter 7: Result and conclusion

### 7.1 Results:

The database was created by taking images from INTERNET. Nearly 100 images were taken as a whole. The training was done on a data set of 40 images. That is 10 images for each god. Here are the results of the test:

1. A total of 60 images were taken for the test.
2. The first system of trainer + HOG that classifies whether an image belongs to Hanuman\Ganesha or Natraj\Shivling gives an accuracy of 70%.
3. The second system of trainer + HOG that classifies whether an image belongs to Hanuman or Ganesha gives an accuracy of 60%.
4. The third system of trainer + HOG that classifies whether an image belongs to Natraj or Shivling gives an accuracy of 56%.

### 7.2 Conclusions:

Here are the conclusions of the work:

1. The program gives really good accuracy when it has to differentiate between human and non-human shapes.
2. The program accuracy falls a little bit while differentiating between two human shapes.
3. Accuracy is lowest when the differentiation is between two non-human shapes.
4. The images with non-uniform background also presented more error than the images with uniform background.

### 7.3 Scope of improvement

Some areas where the project can be improved for performance are given below:

1. The time taken by the HOG to find feature vector is directly proportional to the number of feature vectors. Which accounts for accuracy. Hence there is a trade off between accuracy and time. So, on a faster system, a higher number of feature vector may be chosen without affecting much on the calculation time.

2. Uniformity of background also affects the classifier a lot. So, a complex pre-process that may remove the background may increase the accuracy a lot.
3. A higher resolution image definitely accounts for more details and hence better accuracy but again it increases the classification time. Hence again a better hardware would increase the performance significantly.

#### 7.4 Practical use of the system

Presently at many places, an audio guide in form of recorded tapes are provided to foreign tourists for helping them understand the heritage and history of the places. If it may be replaced with a sculpture recognition system that captures the images using a smartphone camera and then gives the detail, it will be a great step forward for tourism industry.

Also the system may be used with autonomous robots that moves through a site on its own and captures the images. In this way we can make an online 3d tour of such sites which when connected with the sculpture recognition system and a central database, would also give the information about the site along with its tour.

Present government's digital India scheme can take advantage through our system which eases out the work of digitalisation of heritage sites very easy and hassle free.

## REFERENCES:

- [1] S. Kuo and M. Ranganath, “Real time image enhancement for both text and color photo images,” ICIP, vol. 1, pp. 159–162, 1995.
- [2] Dalal, Triggs “Histograms of Oriented Gradients for Human Detection” Conference on Computer Vision and Pattern Recognition (CVPR), 2005.
- [3] David G. Lowe, “Distinctive image features from scale-invariant keypoints,” International Journal of Computer Vision, 60, 2 (2004), pp. 91-110
- [4].[https://en.wikipedia.org/wiki/Histogram\\_of\\_oriented\\_gradients](https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients)
- [5].D. G. Lowe. Distinctive image features from scale-invariant keypoints. IJCV, 60(2):91–110, 2004.
- [6].N. Dalal, B. triggs: histogram of oriented gradients for human detection, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005.
- [7].U.S. Patent 6,711,293, “Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image”, David Lowe’s patent for the SIFT algorithm, March 23, 2004
- [8]Lowe, David G. (1999). “Object recognition from local scale-invariant features”. Proceedings of the International Conference on Computer Vision 2. pp. 1150–1157. doi:10.1109/ICCV.1999.790410.
- [9]R.C. Gonzalez, R.E. Woods, S.L. Eddins, (2004) Digital Image Processing Using Matlab, Prentice Hall, Upper Saddle River, New Jersey.
- [10]<http://studentnet.cs.manchester.ac.uk/ugt/COMP27112/doc/matlab.pdf>
- [11][https://en.wikipedia.org/wiki/Scale-invariant\\_feature\\_transform](https://en.wikipedia.org/wiki/Scale-invariant_feature_transform)
- [12][https://en.wikipedia.org/wiki/Support\\_vector\\_machine](https://en.wikipedia.org/wiki/Support_vector_machine)
- [13]<http://fr.slideshare.net/minhtaispkt/image-processing-with-matlab>