

Implementation and Experiments on Face Detection System (FDS) Using Perceptual Quality Aware Features

Amir Khan

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
February 2017
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Mustafa Tümer
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Alexander Chefranov
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Alexander Chefranov
2. Assoc. Prof. Dr. Önsen Toygar
3. Asst. Prof. Dr. Adnan Acan

ABSTRACT

This thesis is motivated by developing a face detection system for detecting faces in distorted images. Interaction between face detection and perceptual image quality is studied and analyzed to develop this robust face detection system. It is observed that accuracy of existing face detection systems are degraded with increase in distortion which is occurred due to many factors like low resolution of cameras, during transmission or storing. These types of distortions are AWGN, G Blur and JPEG. To overcome this problem, a new set of features named QUALHOG (which is a combination of NSS features and HOG features) is proposed for better and accurate face detection which augments Histogram of Oriented Gradients (HOG) features with perceptual quality-aware spatial Natural Scene Statistics (NSS) features.

Face detection system based on QUALHOG features shows a great improvement in detecting faces as compared to face detection system based on HOG features. A large set of images are used for experimentation. To facilitate these experiments, a distorted face database (DFD) which contains face and non-face images by a variety of common distortion types and levels is used. This new dataset is available for download and further experimentation and it contains images at 10 distortion levels. Precision and Recall are calculated, Precision versus distortion level and Recall versus Distortion level curves are obtained to show the comparison between HOG and QUALHOG based face detection systems.

Furthermore obtained results are compared with known results and presented as AUPR versus Distortion level curves to show the feasibility of FDS.

Keywords: Face detection system, Distorted images, Perceptual Quality Aware features, Histogram of Oriented Gradients

ÖZ

Bu tez, bozulmuş görüntülerde yüzleri tespit etmek için bir yüz algılama sistemi geliştirerek motive edilir. Yüz tanıma ve algılamalı görüntü kalitesi arasındaki etkileşim incelenmiş ve sağlam bir yüz tespit sistemi geliştirilmiştir. Mevcut yüz tanıma sistemlerinin doğruluğunun, kameralarda düşük çözünürlük, iletim veya depolama gibi pek çok faktöre bağlı olarak bozulma artışı ile bozulduğu gözlemlenmiştir. Bu tür bozulmaların sebebi AWGN, G Blur ve JPEG'tir. Bu sorunun üstesinden gelmek için, QUALHOG isminde yeni bir öznitelik kümesi önerilmiştir. Bu yöntem, NSS vs HOG özniteliklerini içermektedir.

QUALHOG özniteliklerini temel alan yüz tanıma sistemi, HOG özniteliklerini kullanan yüz tanıma sistemi ile karşılaştırıldığında, HOG özniteliklerine göre yüz tanımada büyük bir gelişme olduğunu gösterir. Deneyler için geniş bir görüntü grubu kullanılmıştır. Bu deneyleri kolaylaştırmak için, çeşitli genel çarpıtma türleri ve seviyeleri ile yüz ve yüz olmayan görüntüler içeren çarpık bir yüz veritabanı (DFD) kullanılmıştır. Bu yeni veri kümesi, indirilebilir, ileriki deneyler için kullanılabilir ve 10 bozulma seviyesinde görüntüler içerir. Deneylerde, Hassas ve Geri Çağırma hesaplanmış, Hassas ve bozulma seviyesi, ve Geri Çağırma ve Çarpışma seviyesi eğrileri, HOG ve QUALHOG tabanlı yüz tanıma sistemleri arasındaki karşılaştırmayı göstermek için elde edilmiştir.

Ayrıca elde edilen sonuçlar, bilinen sonuçlarla karşılaştırıldığında, FDS'nin fizibilitesini göstermek için AUPR ve Distorsiyon seviyesi eğrileri olarak gösterilmiştir.

Anahtar Kelimeler: Yüzalgılamasistemi, Bozulmuş görüntüler, Algısal Kaliteye Duyarlı Öznitelikler, Odaklı Eğim Histogramı

DEDICATION

This thesis dedicated to my mother and my father.

ACKNOWLEDGEMENT

I would like to record my gratitude to Assoc. Prof. Dr. Alexander Chefranov for his supervision, advice, and guidance from the very early stage of this thesis as well as giving me extraordinary experiences throughout the work. Above all and the most needed, he provided me constant encouragement and support in various ways. His ideas, experiences, and passions has truly inspire and enrich my growth as a student. I am indebted to him more than he knows.

I would like to acknowledge Assoc. Prof. Dr. Önsen Toygar for all her advice and encouragement, I am grateful in every possible way. Special thanks go to Asst. Prof. Dr. Adnan Acan and Prof. Dr. Işık Aybay (Chairman of Computer engineering department) for their advice and guidance.

In the end, I would like to thank Almighty Allah for everything I accomplished.

TABLE OF CONTENTS

ABSTRACT.....	iii
ÖZ.....	v
DEDICATION.....	vii
ACKNOWLEDGEMENT.....	viii
LIST OF TABLES	xii
LIST OF FIGURES	xiv
LIST OF ABBREVIATIONS	xvii
1 INTRODUCTION	1
2 LITERATURE REVIEW OF FDS AND PROBLEM DEFINITION.....	3
2.1 Face Detection on Distorted Images Augmented by Perceptual Quality Aware Features	3
2.2 Distortion of images by AWGN, GBlur and JPEG.....	5
2.3 Extraction of QUALHOG features.....	9
2.3.1 Extraction of NSS features	9
2.3.2 Extraction of HOG features.....	15
2.4 SVM Classifiers	18
2.4.1 Liblinear SVM Classifier.....	18
2.4.2 SVM Light Classifier.....	22
2.5 Known experimental results according to [1]	22
2.5.1 Experiment setup	22
2.5.2 Experiment results	23
2.6 Problem Definition.....	26
2.7 Related work.....	27

2.8 Conclusion.....	29
3 DESIGN, IMPLEMENTATION AND TESTING OF FDS USING QUALHOG	30
3.1 Overall Structure of FDS.....	30
3.2 Implementation and testing of NSS features	30
3.2.1 Implementation and testing for image normalization	30
3.2.2 Implementation and testing of GGD parameters	35
3.2.3 Implementation and testing of AGGD parameters	41
3.3 Implementation and testing of extraction of HOG features	49
3.3.1 Implementation and testing of gradient vectors.....	49
3.3.2 Implementation and testing of orientation binning.....	52
3.3.3 Implementation and testing of block normalization	54
3.4 Implementation and testing of Classifiers usage	55
3.4.1 Implementation and testing of Liblinear SVM usage.....	55
3.4.2 Implementation and testing of SVM Light usage	58
3.5 Implementation of FDS for any size of images using sliding window	58
3.6 Conclusion.....	59
4 EXPERIMENTS ON FDS FOR DISTORTED IMAGES.....	60
4.1 Experiment setup	60
4.2 Experiments results	61
4.2.1 Images distorted by AWGN (Additive White Gaussian Noise).....	62
4.2.2 Images distorted by G Blur (Gaussian Blur)	66
4.2.3 Images distorted by JPEG.....	70
4.3 Comparison with known results	74
4.4 Experiments on images of any size using sliding window technique	75
4.4.1 Training.....	75

4.4.2 Prediction	75
4.4.3 Results of experiments	75
4.5 Conclusion	79
5 CONCLUSION	81
REFERENCES	83
APPENDICES	86
Appendix A: Code for training images using Liblinear SVM	87
Appendix B: Code for testing images using Liblinear SVM	88
Appendix C: Code for calculating NSS features	89
Appendix D: Code for calculating GGD parameters	90
Appendix E: Code for calculating AGGD parameters	91
Appendix F: Code for extracting HOG features	91
Appendix G: Code for normalize images features between -1 and +1	93
Appendix H: Code for training of images in test images using SVM light	93
Appendix I: Code for testing images using SVM light through sliding window	95
Appendix J: Code for generating curve QUALHOGvs HOG	96
Appendix K: Code for generating Gaussian filter without using Matlab function	97
Appendix L: Manually created table for saving experiment results	98

LIST OF TABLES

Table 1: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by AWGN with different levels using QUALHOG features, Precision and Recall are calculated.	62
Table 2: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by AWGN with different levels using HOG features, Precision and Recall are calculated.	63
Table 3: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by Gaussian Blur with different levels using QUALHOG features, Precision and Recall are calculated.....	66
Table 4: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by Gaussian Blur with different levels using HOG features, Precision and Recall are calculated.....	67
Table 5: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by JPEG in different levels (Q factor) using QUALHOG features, Precision and Recall are calculated.....	70
Table 6: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by JPEG in different levels (Q factor) using HOG features, Precision and Recall are calculated.	71
Table 7: True positive and false positive are obtained for images distorted by AWGN at different log scale	75
Table 8: True positive and false positive are obtained for images distorted by AWGN at different log scale.....	76

Table 9: True positive and false positive are obtained for images distorted by G Blur at different log scale.....	76
Table 10: True positive and false positive are obtained for images distorted by G Blur at different log scale.....	77
Table 11: True positive and false positive are obtained for images distorted by JPEG at different log scale.....	77
Table 12: True positive and false positive are obtained for images distorted by JPEG at different log scale.....	78

LIST OF FIGURES

Figure 1: Overall structure of FDS according to [1]	5
Figure 2: NIQE vs AWGN curve	23
Figure 3: NIQE vs GBlur curve	24
Figure 4: NIQE vs JPEG curve	24
Figure 5: AUPR vs AWGN curve.....	25
Figure 6: AUPR vs GBlur curve	25
Figure 7: AUPR vs JPEG curve	25
Figure 8: Input Image used for testing for NSS and HOG features [1]	31
Figure 9: Gray scale conversion of an image in Figure 8	31
Figure 10: Pixel intensity values of image (24..29, 54..58) from Figure 9	32
Figure 11: Generated Gaussian Filter for an image (Figure 9)	32
Figure 12: Generated Gaussian Filter for an image (Figure 6) by using code in Appendix K.....	33
Figure 13: Mean values of gray scale image (24...29, 54....58) are shown in Figure 9 ...	34
Figure 14: Variance values (24...29, 54...58) of a given image (Figure 9)	34
Figure 15: Normalize pixel intensity values of gray scale image (24...29, 54...58) (Figure 9)	35
Figure 16: Normalized image	35
Figure 17: Values of $r(\gamma)$ is shown (from position 79 to 91) which is obtained using lookup table.....	36
Figure 18: Determine array position in Gaussian ratio table (5) by using value of ratio.	39
Figure 19: Determine value in look up table corresponds to array position found in Figure 18	40

Figure 20: A totals of 4 parameters are calculated for a given image (Figure 16)	40
Figure 21: Pixel value intensities (1...4, 1...5) of an image (Figure 16)	41
Figure 22: Pixel value intensities (1...4, 1...5) of an image (Figure 16) after horizontal shift.....	42
Figure 23: Values of vector (1...7) obtained by multiplication of shifted image pixel values in horizontal direction (Figure 22) with normalized image pixel values (Figure 21) and stored in pair.....	42
Figure 24: Initial 13 values of $r(\alpha)$ is shown of image (Figure 13) which is obtained using lookup table	43
Figure 25: A total of 4 parameters that are shape parameter, statistical mean, left and right standard deviation for an image (Figure 16) are estimated	48
Figure 26: A total of 16 features in horizontal, vertical and in diagonal direction are calculated for an image (Figure 16)	48
Figure 27: A total of 32 parameters obtained by concatenation of features from normalized image (Figure 16) and down sampled image.	48
Figure 28: Gradient in x direction [2..5, 27..31] for an image (Figure 9).....	51
Figure 29: Gradient in y direction [2..5, 27..31] for an image (Figure 9).....	51
Figure 30: Angles are calculated [2..5, 27..31] for an image (Figure 9).....	51
Figure 31: Magnitudes are calculated [2..5, 27..31] for an image (Figure 9)	52
Figure 32: Grouping of 64 pixels (1..2, 21..28) calculated angles into a cell for Figure 9	53
Figure 33: Grouping of first 64 pixels (1..2, 21..28) calculated magnitudes into a cell for Figure 9	53
Figure 34: Histogram values in 36 bins for one block for Figure 9	54
Figure 35: All features in a block after L2 normalization for Figure 9.....	54

Figure 36: All features in a block after L1 normalization for Figure 9.....	55
Figure 37: All features in a block after L1 sqrt normalization for Figure 9.....	55
Figure 38: First 13 features of total 2268 features for Figure 9	55
Figure 39: Result obtained after training	56
Figure 40: Accuracy (percentage of true positive images) is obtained for tested images	57
Figure 41: Precision vs AWGN curve for QUALHOG (red curve) and HOG (green curve) features	64
Figure 42: Recall vs AWGN curve for QUALHOG (red curve) and HOG (green curve) features	65
Figure 43: Precision vs GBlur curve for QUALHOG (red curve) and HOG (green curve) features	68
Figure 44: Recall vs GBlur curve for QUALHOG (red curve) and HOG (green curve) features	69
Figure 45: Precision vs JPEG curve for QUALHOG (red curve) and HOG (green curve) features	72
Figure 46: Recall vs JPEG curve for QUALHOG (red curve) and HOG (green curve) features	73
Figure 47: A tested example of true positive (left) and false positive (right) detection in two distorted images	78

LIST OF ABBREVIATIONS

AGGD	Asymmetric Generalized Gaussian distribution
AUPR	Area under precision recall curve.
AWGN	Additive White Gaussian Noise
DCT	Discrete Cosine Transformation
DFD	Distorted Face Database
FDS	Face Detection System
G Blur	Gaussian Blur
GGD	Generalized Gaussian distribution
HOG	Histogram of Oriented Gradients
JPEG	Joint Photographic Experts Group
NIQE	Natural Image Quality Evaluator
NSS	Natural Scene Statics in Spatial domain
QUALHOG	Augmentation of Histogram of Gradient features (HOG) with perceptual quality (NSS) features

Chapter 1

INTRODUCTION

The arrival of low cost digital storage device and social networking and photo sharing websites like Instagram, Snapchat, Facebook, Twitter etc leads to rapid growth of sharing visual data like photos and videos across these platforms over internet. Image processing and Computer vision algorithms focus on studying the content based on real life applications like surveillance, image exploring etc. The main task of Computer vision algorithms are object detection and object recognition. But these algorithms work well in certain limits and performance is degraded with the decline in image quality. Automatic face detection which is used commercially these days is one of the examples. Face detection is used for security purpose and also for surveillance but often images are subjected to be distorted due to the weather conditions or due to low quality of camera device and these distortion have direct effect on performance of detector.

In this research, a new set of features QUALHOG [1] are studied and implemented, which shows more tolerance to common distortion like AWGN, GBlur and JPEG in images. This research inspires by the fact that perceptual quality aware feature can be used for modeling face detectors. Widely used and implemented Histogram of gradient features (HOG) based detection algorithm is used as foundation for implementation of this research. Perceptual quality aware features (NSS spatial) [2]

are also explored along with HOG [5] in order to make detector more tolerant to image distortions.

The main contribution of this thesis is as follows

- Images are distorted by using three different types of distortion techniques like Additive White Gaussian noise (AWGN), Gaussian Blur (G Blur) and JPEG at different levels.
- A new set of features name QUALHOG, which augments perceptual quality aware (NSS spatial) features with HOG features are studied and implemented.
- Robustness of QUALHOG is measured and calculated and compare it with HOG in order to show its better performance.
- LIBLINEAR [6] and SVM Light [13] Support vector machine is studied and implemented for classification and prediction of facial images.

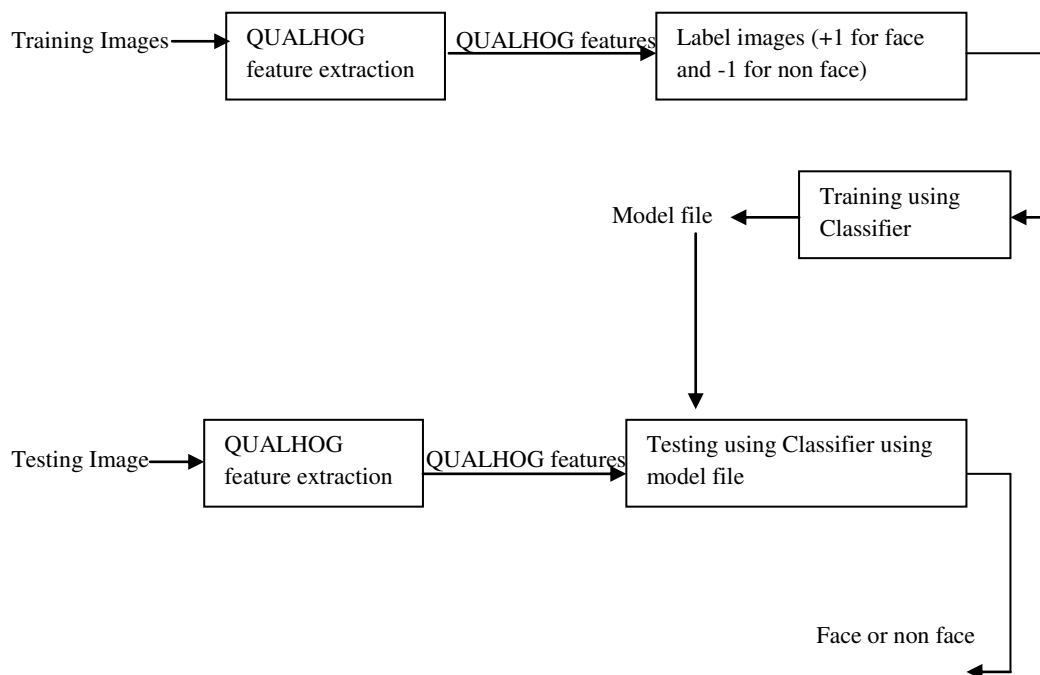
The rest of the thesis is organized as follows. Chapter 2 contains literature review of FDS and problem definition. Chapter 3 contains design, implementation and testing of FDS. Chapter 4 contains experiments and results of FDS. Chapter 5 contains conclusion. Appendices contain code of FDS and raw data on experiments with FDS.

Chapter 2

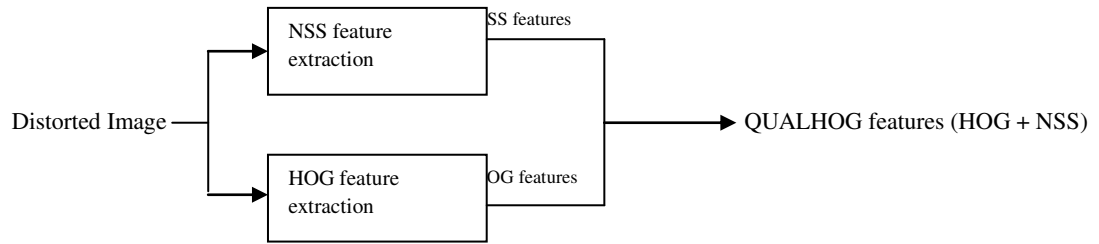
LITERATURE REVIEW OF FDS AND PROBLEM DEFINITION

2.1 Face Detection on Distorted Images Augmented by Perceptual Quality-Aware Features

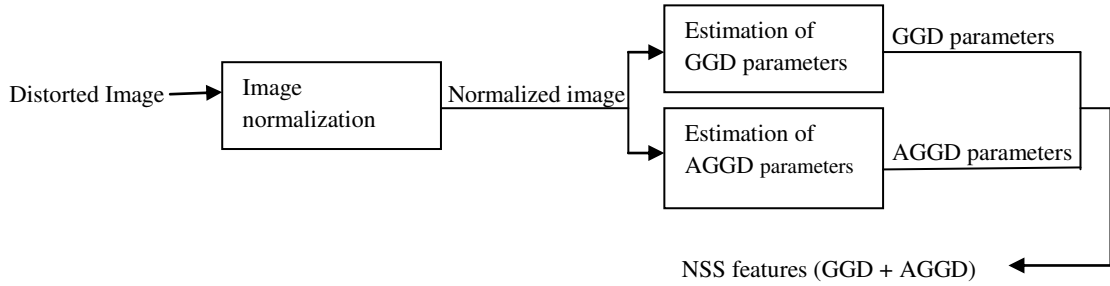
Overall FDS structure



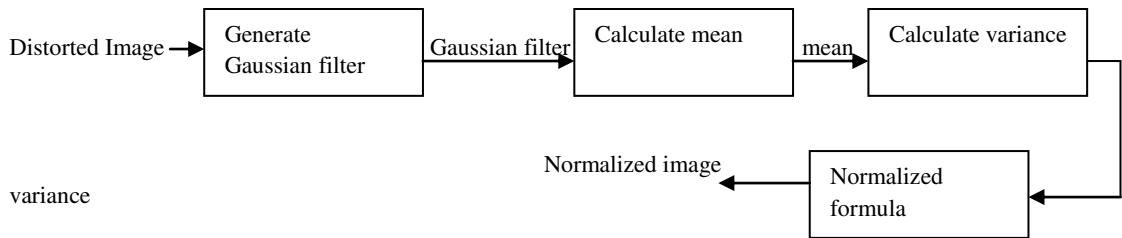
a. Overall FDS structure



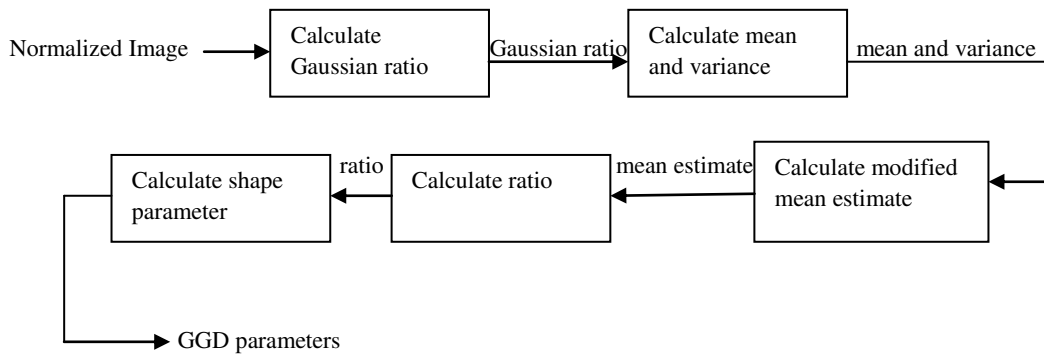
b. Structure for QUALHOG feature extraction



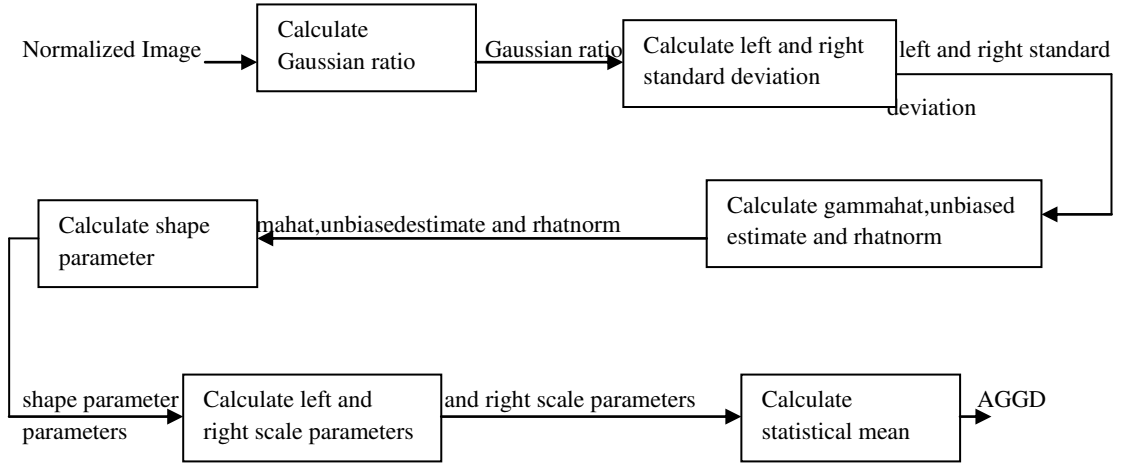
c. Structure for NSS feature extraction



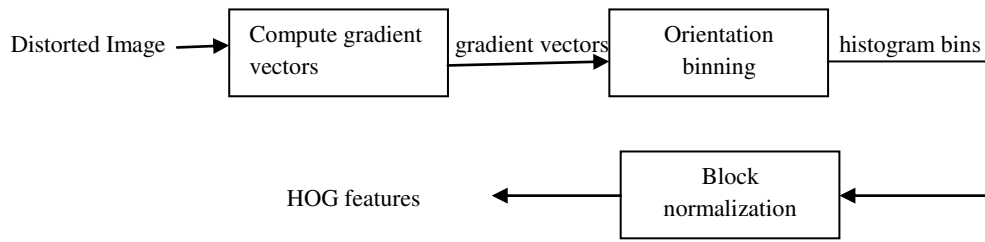
d. Structure for image normalization



e. Structure for estimation of GGD parameters



f. Structure for estimation of AGGD parameters



g. Structure for HOG features extraction

Figure 1: Overall structure of FDS according to [1]

Figure 1 shows overall structure of FDS as it is defined in [1], in which QUALHOG features are extracted by augmentation of NSS and HOG features. NSS features are obtained by augmentation of GGD and AGGD parameters. In the end, LIBLINEAR SVM is used for training and testing of images.

2.2 Distortion of images by AWGN, GBlur and JPEG

In [1] detection of faces on distorted images mainly by Gaussian Blur, Additive white Gaussian noise and JPEG is considered.

Additive White Gaussian Noise (AGWN) - It's a type of distortion which is used in networking as well as in images processing. In image processing, zero mean normal distributed noise which is Gaussian noise is added to every pixel of an image.

AWGN is defined as

$$\tilde{I}(i, j) = I(i, j) + N_{ij} \quad (1)$$

where, $N_{ij} \sim N(\mu, \sigma_N^2)$, μ is mean, σ_N^2 is variance and $I(i, j)$ is original image and $\tilde{I}(i, j)$ is image distorted by AWGN. μ is always zero in AWGN therefore distortion level depend on σ_N^2 .

Gaussian Blur (G Blur) – In Gaussian blur, images are distorted using normal distribution or Gaussian function. Convolution of an image with its Gaussian function used to obtain desired distortion, new value of a pixel is obtained by taking weighted average of neighbor pixels.

The Gaussian function in two dimensions is given by

$$G(x, y) = \frac{1}{2\pi\sigma_B^2} * e^{-\frac{x^2 + y^2}{2\sigma_B^2}} \quad (2)$$

In (2), x refers to rows and y refers to column of Gaussian kernel and σ_B refers to standard deviation.

In images processing, Gaussian function is pruned to $6\sigma_B$ which means

$$-[3\sigma_B] \leq x \leq [3\sigma_B] \text{ and } -[3\sigma_B] \leq y \leq [3\sigma_B] \quad (3)$$

An image distorted by Gaussian blur can be written as

$$\tilde{I} = I * G \quad (4)$$

Where, '*' stands for convolution.

Formula (4) can be rewritten as

$$\tilde{I}(i, j) = \sum_{x=-[3\sigma_B]}^{[3\sigma_B]} \sum_{y=-[3\sigma_B]}^{[3\sigma_B]} I(i+x, j+y) * G(x, y) \quad (5)$$

In (5), i is row and j is the column number of an image.

Joint Picture Experts Group (JPEG) – It is a compression technique to reduce size of an image so that it is easy to store and send them over network medium. It is a lossy-compression technique in which once data is loosed in order to reduce storage space can't be recovered and quality of an image degrades with the increase in compression.

Steps to perform JPEG Compression and decompression

- a. JPEG works on 8*8 blocks, an image is converted into a grayscale image and then is divided into sub images of size dimension 8*8 which is called blocks.
- b. As it is a grayscale image, it ranges from 0 to 255, we need to convert the value between -128 to 127, so we subtract 128 from each pixel of an image block we selected.

- c. In this step DCT is performed on x axis and y axis, that is 2-D DCT, in order change image values into frequency domain.

Mathematical expression for 2 dimensional DCT is

$$G_{u,v} = \frac{1}{4} \alpha(u) \alpha(v) \sum_{x=0}^7 \sum_{y=0}^7 G_{x,y} \cos \left[\frac{(2x+1)u\pi}{16} \right] \cos \left[\frac{(2y+1)v\pi}{16} \right] \quad (6)$$

where, u and v horizontal and vertical spatial frequencies respectively and ranges from $0 \leq u < 8$ and $0 \leq v < 8$

$$\alpha(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } u = 0 \\ 1, & \text{otherwise} \end{cases}$$

$G_{x,y}$ is a pixel value at location (x,y) and $G_{u,v}$ is Discrete Fourier Transformation coefficient at location (u,v).

After applying the above expression, image pixel values will change into frequency domain.

- d. This stage is called Quantization and it is the process which decides the image quality and compression efficiency and it is also called 'Q' factor. Quantization reduces information at higher frequencies as human eye is less susceptible to see minor changes at higher frequency. Mathematical expression for Quantization can be written as

$$B_{j,k} = \text{round} \left(\frac{G_{j,k}}{Q_{j,k}} \right) \text{ for } j = 0 \text{ to } 7 \text{ and } k = 0 \text{ to } 7 \quad (7)$$

Where,

G = dequantized DCT coefficient

Q = quantization matrix for an image

B = quantized Discrete Fourier transform coefficient

Coding – In this stage, components of image coded in zigzag manner which uses RLE (run to length algorithm which picks up same frequencies in image component and group them. Huffman coding is used in later part to code image components.

Decompression – It's the inverse of compression, Its consists of decoding, then de-quantized image , then apply inverse DCT on de-quantized image to convert it from frequency domain to its original form.

Note – since it's a lossy-compression hence, original image can't be obtained after decompression.

2.3 Extraction of QUALHOG features

QUALHOG Features are the augmentation of NSS and HOG features.

2.3.1 Extraction of NSS features

This paper [2] proposed a method of image quality assessment in spatial domain with no references available. This method works in spatial domain and no transformation is needed to another coordinated frame like DCT, wavelet, etc. hence require very little computation. This is done in three stages -

1. Compute locally normalized luminance of an image.

It is done using normalization and local mean subtraction over an image

$$\hat{I}(i, j) = \frac{I(i, j) - \mu(i, j)}{\sigma(i, j) + C} \quad (8)$$

In equation (8), 'I' is the original image in which $i \in 1, 2, 3 \dots M$ and $j \in 1, 2, 3 \dots N$ where M and N are the rows and columns of an image. C is a constant which is always 1, the role of C is to keep image stable when denominator goes to zero. μ and σ are the mean and variance of an image. Mathematical expression to calculate mean and variance of an image.

$$\mu(i, j) = \sum_{k=-K}^K \sum_{l=-L}^L w_{k,l} I_{k,l}(i, j) \quad (9)$$

$$\sigma(i, j) = \sqrt{\sum_{k=-K}^K \sum_{l=-L}^L w_{k,l} (I_{k,l}(i, j) - \mu(i, j))^2} \quad (10)$$

In equation (9) and (10), $w_{k,l}$ is a two dimensional Gaussian filter applied circularly and down sampled to unit volume and three standard deviation, $K = 3$ and $L = 3$. By applying the above equation, locally normalized luminance image is obtained, the performance of the above equation fluctuate according to the window size of an image.

2. Applying Generalized Gaussian distribution (GGD) [3] on normalized image.

Computer histograms over pixels of distorted images befitted by applying Generalized Gaussian distribution (GGD) and digression of that image from its original form comes handy to estimate type of distortion and level of distortion.

3. Applying Asymmetric Generalized Gaussian Distribution (AGGD) [4] on normalized image. AGGD generalizes the Generalized Gaussian Distribution and comprehend it by allowing asymmetry in the distribution. The features are calculated along four orientations horizontal (H), vertical (V), main-diagonal (D1) and secondary diagonal (D2).

$$\begin{aligned}
H_{(i,j)} &= \check{I}_{(i,j)} \check{I}_{(i,j+1)} \\
V_{(i,j)} &= \check{I}_{(i,j)} \check{I}_{(i+1,j)} \\
D1_{(i,j)} &= \check{I}_{(i,j)} \check{I}_{(i+1,j+1)} \\
D2_{(i,j)} &= \check{I}_{(i,j)} \check{I}_{(i+1,j-1)}
\end{aligned} \tag{11}$$

In equations (11), AGGD parameters are calculated in horizontal, vertical, diagonal D1 and diagonal D2 directions.

Estimation of GGD parameters

In [3], two parameters, shape parameter and standard deviation are estimated for animage. The GGD probability density function (pdf) is given by

$$f(x; \alpha, \sigma^2) = \frac{\alpha}{2\beta\tau(1/\alpha)} \exp\left(\frac{-|x|}{\beta}\right)^\alpha \tag{12}$$

$$\beta = \sigma \sqrt{\frac{\tau(1/\alpha)}{\tau(3/\alpha)}}$$

$$\tau(a) = \int_0^{\infty} t^{a-1} * a^{-t} dt > 0$$

Where, τ is gamma function

In equation (12), α is shape parameter and σ is the standard deviation.

The steps used to calculate shape parameter (α) and variance (σ^2) of any image are as follows

- a. Calculate Generalized Gaussian ratio function

$$r(\gamma) = \frac{\sigma^2}{E^2[|X|]} = \frac{\tau(1/\gamma) * \tau(3/\gamma)}{\tau^2(2/\gamma)} \quad (13)$$

To calculate Gaussian ration function, Look up table is defined for parameter (γ).

- b. Calculate mean (μ_x) and variance (σ_x^2) of any image (x_{ij})

$$\mu_x = \frac{1}{M} \sum_{i=1}^M x_{ij} \quad (14)$$

$$\sigma_x^2 = \frac{1}{M} \sum_{i=1}^M (x_{ij} - \mu_x)^2 \quad (15)$$

In equation (14) & (15), M is total number of pixel values in any image.

- c. Calculate estimate ($E[|X|]$) for the absolute values modified mean

$$E[|X|] = \frac{1}{M} \sum_{i=1}^M |x_{ij} - \mu_x| \quad (16)$$

In equation (16), M is total number of pixel values in any image.

d. Determine the ratio (ρ)

$$\rho = \left(\sigma_x^2 / E^2[|X|] \right) \quad (17)$$

e. Solve the equation $\gamma = r^{-1}(\rho)$ (18)

where, r is the generalized Gaussian ratio function, by using a lookup table. The look up can be defined by the user or by giving instructions to machine to choose random variable with a constant difference between two variables.

By applying the above four steps in a image, two important features that is shape parameter (γ) and variance (σ^2) are found. These parameters are used for examining distortion level in images and videos and useful for detection of faces in distorted images.

Estimation of AGGD parameters

In [4], AGGD parameters (4 parameters i.e. shape parameter, left and right standard deviation and statistical mean) are calculated. Probability density function for AGGD is given by

$$f(x; \alpha, \beta) = \begin{cases} \frac{\gamma}{2(\beta_l + \beta_r)\tau(1/\gamma)} \exp\left(\frac{-|x|}{\beta_l}\right)^\alpha, & \text{if } x \leq 0 \\ \frac{\gamma}{2(\beta_l + \beta_r)\tau(1/\gamma)} \exp\left(\frac{-|x|}{\beta_r}\right)^\alpha, & \text{if } x > 0 \end{cases} \quad (19)$$

Steps to calculate parameters

- a. Calculate Generalized Gaussian ratio $\rho(\alpha)$ function

$$\rho(\alpha) = \frac{\tau^2(2/\alpha)}{\tau(1/\alpha)\tau(3/\alpha)} \quad (20)$$

- b. Calculate left standard deviation (β_l) and right standard deviation (β_r)

$$\beta_l = \sqrt{\frac{1}{N_l-1} \sum_{k=1, x_k < 0}^{N_l} x_k^2} \quad (21)$$

$$\beta_r = \sqrt{\frac{1}{N_r-1} \sum_{k=1, x_k \geq 0}^{N_r} x_k^2} \quad (22)$$

In equations (21) & (22), β_l and β_r are left and right standard deviation, N_l and N_r is number of sample of x_k when $x_k < 0$ and $x_k \geq 0$ respectively.

- c. Calculate the value of gamma hat ($\check{\gamma}$), $\check{\gamma}$ (unbiased estimate) and $\check{R}$ using $\check{\gamma}$ and $\check{\gamma}$

$$(\check{\gamma}) = \frac{\beta_l}{\beta_r} \quad (23)$$

$$\check{\gamma} = \frac{(\sum |x_k|)^2}{\sum x_k^2} \quad (24)$$

$$\check{R} = \check{\gamma} \cdot \frac{(\check{\gamma}^3 + 1) \cdot (\check{\gamma} + 1)}{(\check{\gamma}^2 + 1)^2} \quad (25)$$

- d. According to $\check{R}$ ($\hat{R}$) value estimate (α) using the approximation of the inverse generalized Gaussian ratio

$$\alpha = \rho^{-1}(\check{R}) \quad (26)$$

- e. Calculate left scale parameter ($\check{\beta}_l$) and right scale parameter ($\check{\beta}_r$)

$$\check{\beta}_l = \beta_l * \sqrt{\frac{\tau(3/\alpha)}{\tau(1/\alpha)}} \quad (27)$$

$$\check{\beta}_r = \beta_r * \sqrt{\frac{\tau(3/\alpha)}{\tau(1/\alpha)}} \quad (28)$$

- f. Compute statistical mean

$$\eta = (\beta_l - \beta_r) \frac{\tau(2/\alpha)}{\tau(1/\alpha)} \quad (29)$$

2.3.2 Extraction of HOG features

This paper [5] studies the extraction for histogram of oriented gradient features in images. This paper shows the superiority of Histogram of oriented gradient descriptor over SIFT and shape context descriptor.

Gamma/Color normalization – RGB, grayscale, LAB color images were tested, these color were subject to normalization before computing gradients.

Gradient computation – Gradient have been computed in x and y direction. It is computed for every pixel of an image, It is calculated the variance in pixel value in horizontal and vertical direction.

$$\text{Gradient vector in x direction of pixel (hx)} = hx_l - hx_r \quad (30)$$

$$\text{Gradient vector in y direction of pixel (hy)} = hx_t - hx_b \quad (31)$$

In equations (30) and (31), hx is centrepixel, hx_r is a pixel right direction of center pixel, hx_l is a pixel in left direction of center pixel, hx_t is a pixel on top of center pixel and hx_b is a pixel in bottom of center pixel. By appending these two values we get out gradient vector.

Magnitude and angle is also obtained by

$$\text{Magnitude} = \sqrt{hx^2 + hy^2} \quad (32)$$

$$\text{Angle} = \arctan\left(\frac{hx}{hy}\right) \quad (33)$$

Orientation Binning – The next step is orientation binning, pixels of an image are grouped into cells and their magnitude is place into histogram bins, a total of 9 bins are used for the purpose. Histogram is ranges from 0 to 180 degrees and a total of 9 bin so every bin there are 20 degrees. Unsigned gradients changes into signed gradient to fall into range of 0 to 180 degree. For each gradient vector, it's contribution to the histogram is given by the magnitude of the vector (so stronger gradients have a bigger impact on the histogram). Image pixel is grouped into cells,

every cell has 8x8 pixels that's a total of 64 pixels, and each pixel have it angle and magnitude. In histogram calculation if angles of pixel all in the range of any bin angle, then that pixel magnitude added into that bin.

Normalization and descriptor block - Four cells grouped together into one block and normalized based on histograms value in the block and blocks have 50% overlap. This block normalization is executed by appending the histograms of the four cells within the block into a vector with 36 components (4 histograms x 9 bins per histogram). Divide this vector by its magnitude to normalize it. This block normalization is executed by appending the histograms of the four cells within the block into a vector with 36 components (4 histograms x 9 bins per histogram). Divide this vector by its magnitude to normalize it. L2 normalization followed by L1 normalization and it followed by L1- sqrt normalization.

$$\text{L2 - norm} = \frac{v}{\sqrt{\|v\|_2^2 + e^2}} \quad (34)$$

$$\text{L1 - norm} = \frac{v}{(\|v\|_1 + e)} \quad (35)$$

$$\text{L1 sqrt norm} = \sqrt{\frac{v}{(\|v\|_1 + e)}} \quad (36)$$

Where, v is non-normalized vector containing all histogram in a block and $\|v\|_k$ is its k norm for k = 1,2, and e is small value.

2.4 SVM Classifiers

Liblinear and SVM Light are the two classifiers used in this research.

2.4.1 Liblinear SVM Classifier

The paper [6] studies about a linear Support Vector Machine Classification (SVM). This paper studies the feasibility of Liblinear on large scale data for classification. It supports two classifiers Logistic regression and linear SVM.

It solves the following optimization problem

$$\min \frac{1}{2} w^T w + C \sum_{i=1}^l \varepsilon(w; x_i, y_i) \quad (37)$$

Where, (x_i, y_i) are set of instant label pairs, $i = 1, \dots, l$, $x_i \in R^n$, $y_i \in \{-1, +1\}$

$\varepsilon(w; x_i, y_i)$ is a loss function and $C > 0$ is a penalty parameter.

For prediction, the linear predictor with for a sample with feature X is given by

$$\tilde{y} = \text{sign}(W^T X + b) \quad (38)$$

In equation (38), $W = [W^{HOG}, W^{NSS}]$, W^{HOG} are weight corresponds to 2268 HOG features and W^{NSS} corresponds to 36 spatial NSS features.

This optimization problem is modified because the weight correlate with 36 dimensional spatial NSS features (perceptual quality aware features) W^{NSS} can be inequitably penalized in contrast to weight of 2268 dimensional HOG features W^{HOG} .

The modified optimization equation can be written as

$$\min_{w,b,\{\xi_i\}} \frac{1}{2} \|W^{HOG}\|_2^2 + \frac{1}{2} \|\tilde{W}^{NSS}\|_2^2 + \lambda \sum_{i=1}^n \xi_i \quad (39)$$

Such that $y_i = \left((W^{HOG}, X_i^{HOG}) + (\tilde{W}^{NSS}, C_s X_i^{NSS}) + b \right) \geq 1 - \xi_i$

In equation (39) $\tilde{X}_i = [X_i^{HOG}, C_s X_i^{NSS}]$, parameters λ and C_s are selected by cross validation.

LIBLINEAR is an efficient and straightforward package for linear classification on large sets of data. It is proved to be better than other large scale classification packages like LIBSVM. The main goal of LIBLINEAR is to classify large set of data in less time efficiently.

Training using Liblinear SVM

Design for training images of QUALHOG and HOG features is done as shown in Figure 1. A total of 2731 images (1231 facial images and 1500 non-facial images) are used in each level for training. Training is done by extracting QUALHOG features for all images, store them in a vector and liblinear SVM is trained on these vectors to obtain a file named model. HOG features are also extracted in order to compare results of QUALHOG with HOG.

For QUALHOG feature training, NSS features are concatenated with HOG features and store in the form of vector, liblinear SVM is trained on these features. LIBLINEAR [11] which is a linear SVM is used to classification of data. The reason behind choosing LIBLINEAR is because of large data for testing. There are more

than half a million of images to be tested, so LIBLINEAR is used for the job to be done.

It is an open source package and developed by National Taiwan University and it is written in C++ language and can be compiled in Matlab or python through make file easily. Training a Liblinear SVM based on solving an optimization problem given in equation (39).

Steps for training images

- Load all positive (faces) and negative (non faces) training images.
- Calculate 36 NSS feature vector of given images (one by one).
- Calculate HOG features of images (one by one).
- Concatenate both features and store them into a vector.
- Once features for all images are computed, store them in to a matrix.
- Classify it by using LIBLINEAR and store the result into a file.

Detection of face in test images using Liblinear SVM

Design for detection of face in test images is obtained as shown in Figure 1. A total of 17872 images (393 facial images and 17479 non-facial images) are used in each level for detection. Detection is done by extracting QUALHOG features for all

images, store them in a vector and Liblinear SVM along with model file (obtained from training) are used for face detection in images. NSS features are concatenated with HOG features and store in the form of vector, Liblinear SVM is used for testing by using these features.

LIBLINEAR [11] which is a linear SVM is used to classification of data. The reason behind choosing LIBLINEAR is because of large data for testing. There are more than half a million of images to be tested, so LIBLINEAR is used for the job to be done. It is an open source package and developed by National Taiwan University and it is written in C++ language and can be compiled in Matlab or python through make file easily. Images are tested and predicted by using equation (38).

Steps for testing images

- Load all the testing data (images).
- Calculate 36 dimensional feature vector of given image (one by one).
- Calculate HOG features of an image (one by one).
- Concatenate both features and store them into a vector.
- Once features for all images are computed, store them in to a matrix.
- Predict it by using LIBLINEAR by using a file which stored results of training data.

- LIBLINEAR predict the data by showing -1 for non-face and 1 for face.
- Prediction accuracy is depends on variable λ , which is selected carefully by cross validation.

2.4.2 SVM Light Classifier usage

SVM light [13] is an implementation of Support Vector Machines (SVMs) in C. The main features of the program are the following:

It is a fast optimization algorithm which solves classification and regression problems. It solves ranking problems, computes XiAlpha-estimates of the error rate, the precision, and the recall. It can train SVMs with cost models and example dependent costs. It also handles many thousands of support vectors and supports standard kernel functions. It is used for training and testing of images of any size for face detection system.

2.5 Known experiment results according to [1]

2.5.1 Experiment setup

For experiments [1], Matlab is used. In addition to Matlab 2015. For training and testing of images LIBLINEAR SVM [11] is used. A database for 25 GB of image containing images of (80x64) dimension is created by the author of article [1] to conduct experiments. QUALHOG and HOG method is analyzed on distorted image database. This database contains images distorted on 10 levels by AWGN, GBlur and JPEG separately. At every level 2731 images (1231 faces and 1500 non faces) are used for training, and 17872 images (393 faces and 17479 non faces) are used for testing FDS. All the images in database are scaled to a size of 80x60 [14]. A total of 536,160 images are used for testing FDS.

QUALHOG is compared with HOG by calculating True positive, False positive and Total number of detected images distorted by all three type of distortion. True positive is an image that is detected by SVM as facial image and actually is a facial image. False positive is an image detected by SVM as facial image and actually is a non- facial image. Precision and recall are calculated by using QUALHOG features as follows

$$\text{Precision} = \frac{\text{True positives}}{\text{Detected positives}} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}} \quad (40)$$

$$\text{Recall} = \frac{\text{True positives}}{\text{Total number of positives}} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}} \quad (41)$$

It is observed that AUPR and NIQE[12] is claimed to be considered in [1] but actually not defined. Dependence of NIQE on distortion levels and on AUPR are shown but not actually defined.

2.5.2 Experiments results

NIQE vs distortion level is given below as shown in [1]

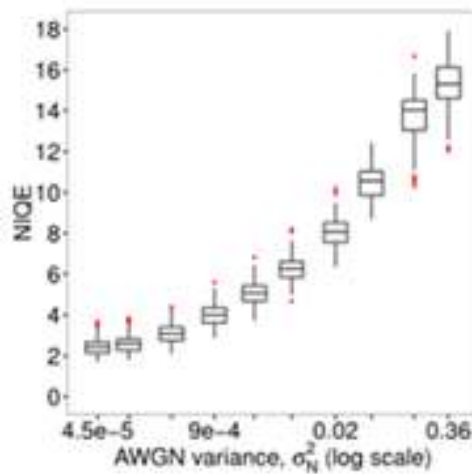


Figure 2: NIQE vs AWGN curve

In Figure 2, NIQE is calculated at every AWGN distortion level and curve is obtained.

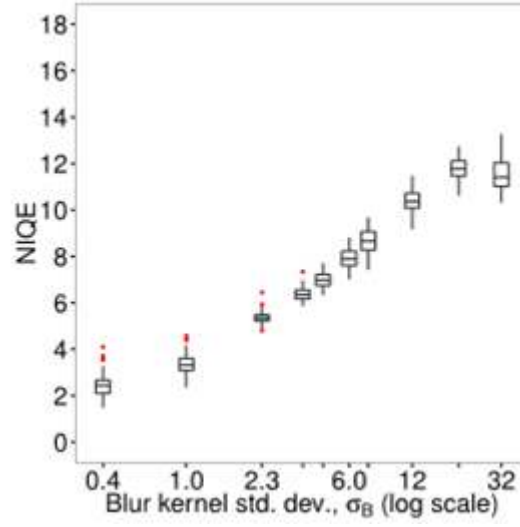


Figure 3:NIQE vsGBlur curve

In Figure 3, NIQE is calculated at every GBlur distortion level and curve is obtained.

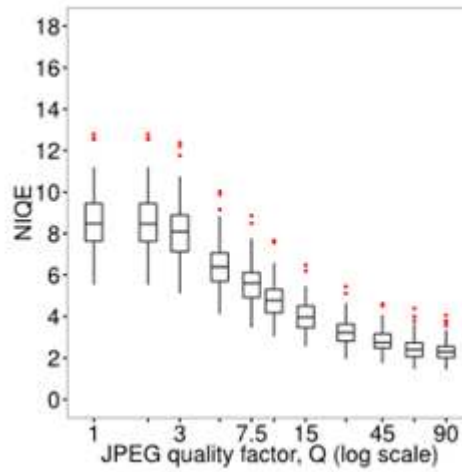


Figure 4:NIQE vs JPEG curve

In Figure 4, NIQE is calculated at every JPEG distortion level and curve is obtained.

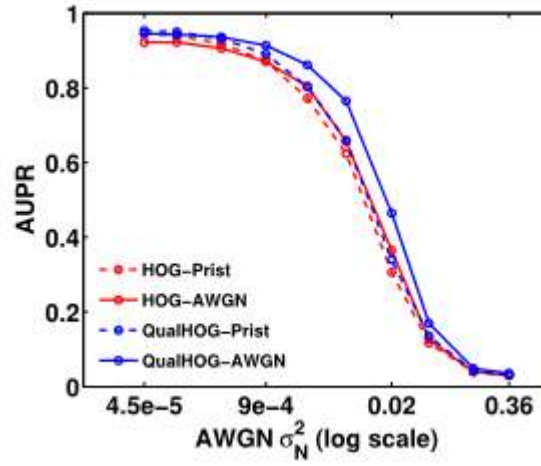


Figure 5:AUPR vs AWGN curve

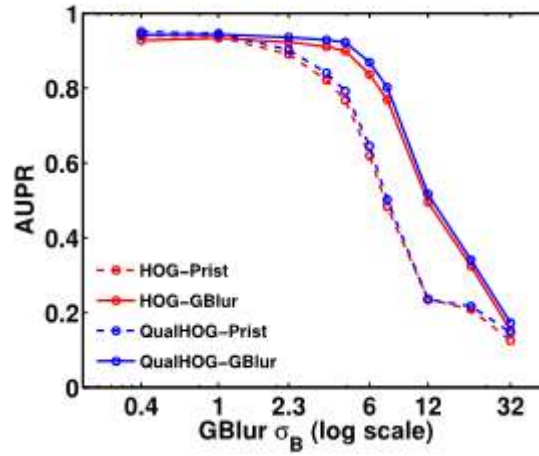


Figure 6:AUPR vsGBlur curve

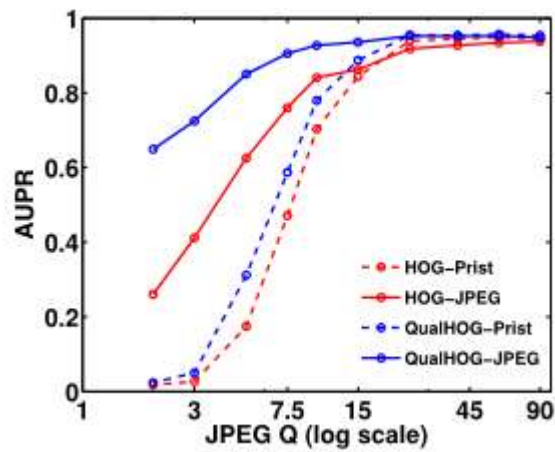


Figure 7:AUPR vs JPEG curve

In Figure 5, 6 and 7, it is observed that with increase in distortion level (AWGN, GBlur and JPEG), AUPR decreases more rapidly for HOG features as compare with QUALHOG features, which shows better performance of FDS based on QUALHOG features.

In [1] NIQE and AUPR are calculated for distorted images at every different level for all the three type of distortion (AWGN, GBlur and JPEG) and NIQE vs distortion level and AUPR vs distortion level curves are obtained.

2.6 Problem Definition

In this thesis, the following problems are considered:

- Implementation of Face Detection System using QUALHOG approach based of HOG and NSS features.
- Test FDS on database of images provided by the author of paper [1].
- Compare experiments on the FDS developed in the conditions of paper [1] and compare it versus HOG.
- Study dependence of accuracy characterized by Precision and Recall on distortion level of images distorted by AWGN, GBlur and JPEG.
- Test on 100 images of any size download from internet randomly is conducted using sliding window.

QUALHOG is compared with HOG by calculating True positive, False positive and Total number of detected images distorted by all three type of distortion.

2.7 Related work

- **AdaBoost based Face Detection for Embedded Systems**

This paper [7] proposed the technique of face detection in still images using AdaBoost which was introduced by Viola and Jones. It is the most popular technique used for detection mainly due to its less complex nature, higher accuracy. The algorithm works in four stages. Selecting Haar like features, taking integral of images, AdaBoost training and Cascading Classifiers.

- **Tolerance for Distorted Faces: Challenges to a Configurational Processing Account of Familiar Face Recognition**

Configurational processing means spatial relation between spatial features and it is widely used in Face recognition. This paper [8] concentrates on familiar faces and check how well configurational processing is able to recognize face. This paper uses three types of configurational processing. Detection of first order relations which define important features of face like eyes above nose, above mouth. Holistic processing, which bind features into perceptual gestalt and sensitivity to second order relation which means recognizing distance among features.

- **Face Gender Classification: A Statistical Study when Neutral and Distorted Faces are combined for Training and Testing purposes**

This paper [9] studies gender identification on distorted faces. Three techniques grey level, Principal Component analysis and Local binary pattern were used to

extract features. Three classifiers (1 –NN, PCA + LDA, SVM) were used to classify images for gender recognition.

Grey level - Images are converted into gray images and stored in a vector.

Principal component analysis (PCA) – It is a statistical procedure which based on orthogonal transformation. It was introduced by Karl Pearson in 1901. It scans down for subspace in original space whose vectors have maximum variance similar to directions in the original space.

Local Binary patterns [10] - It is used to determine image textures and for face detection. Every pixel in an image is described by a binary number, and it is computed for neighborhood of every pixel. Number 1 is assigned to neighborhood pixels if they are brighter than central pixel otherwise 0 value is assigned. Histogram of LBP values of every pixel is calculated and image is characterized according to these histogram values.

Classifiers - Three classifiers are used for gender classification which are nearest neighbor, PCA + LDA and Support vector machine.

Nearest neighbor – It is very simple classifier, and metric used in this Euclidean distance to classify sample data.

- **Neural Network based Face Detection**

This paper [11] proposed neural network based face detection system. A small window of an image inspect by retinally connected neural network system.

Multiple networks are used to improve performance of system. Bootstrap algorithm is used for obtaining negative images for training, Bootstrap algorithm adds face detections into training set as training progresses. Comparisons with several other state-of-the-art face detection systems are presented; showing that our system has comparable performance in terms of detection and false-positive rates.

2.8 Conclusion

In this chapter, articles related to implement methodology are discussed. Metrics and methods used in them such as HOG, NSS, Liblinear, are explained. Problems to be solved in the thesis are defined.

Chapter 3

DESIGN, IMPLEMENTATION AND TESTING OF FDS USING QUALHOG

3.1 Overall Structure of FDS

Overall structure of FDS is we follow is shown in Figure 1. QUALHOG features are calculated for distorted images. These QUALHOG features are contatenation of HOG and NSS features. Liblinear SVM is used for training and testing of images. Images are taken from database provided by the author of article [1]. It contains facial and non-facial images distorted by AWGN, G Blur and JPEG at 10 different levels. Training and testing is done separately for each level.

3.2 Implementation and testing of NSS features

3.2.1 Implementation and testing for image normalization

In this section, image is normalized by using equation (8) as follows. Mean is calculated by using equation (9) and variance is calculated by using equation (10) in section 2.4 in Chapter 2.

This is done by following code in Matlab in Appendix C. In this code, image is converted into grayscale (line 2 to 7), then Gaussian filter is generated and applied (line 9), mean (line 13) and variance (line 13) are calculated. In the end image is normalized by using these mean and variance (line 16).

Testing for image normalization



Figure 8: Input Image used for testing for NSS and HOG features [1]

We test image normalization for image Figure 8. Figure 8 is taken from database provided by author of article [1]. If image in Figure 8 is RGB, convert it into gray-scale image. This is done by the following Code in Matlab in Appendix C. In Appendix C Code, if an image is an RGB image, it is converted to gray scale image.



Figure 9: Gray scale conversion of an image in Figure 8

imdist ✕						
196x174 double						
	54	55	56	57	58	
24	86	41	72	57	41	
25	34	51	54	40	58	
26	54	45	45	24	42	
27	74	93	82	89	63	
28	86	90	63	49	59	
29	70	50	98	103	103	

Figure 10: Pixel intensity values of image (24..29, 54..58) from Figure 9

In Figure 10, Pixel intensity values of image (24..29, 54..58) from Figure 9 are shown.

Test cases for generating Gaussian filter

In Matlab, `fspecial()` function is used to generate random numbers satisfying Gaussian filter (line 9) in Appendix C. `hsize` is an integer specifying size of the square matrix of random numbers to be generated according to Gaussian distribution with zero mean and standard deviation `sigma`. This is done by the following code in Matlab in Appendix C.

1	2	3	4	5	6	7
1.9652e-05	2.3941e-04	0.0011	0.0018	0.0011	2.3941e-04	1.9652e-05
2.3941e-04	0.0029	0.0131	0.0216	0.0131	0.0029	2.3941e-04
0.0011	0.0131	0.0586	0.0966	0.0586	0.0131	0.0011
0.0018	0.0216	0.0966	0.1592	0.0966	0.0216	0.0018
0.0011	0.0131	0.0586	0.0966	0.0586	0.0131	0.0011
2.3941e-04	0.0029	0.0131	0.0216	0.0131	0.0029	2.3941e-04
1.9652e-05	2.3941e-04	0.0011	0.0018	0.0011	2.3941e-04	1.9652e-05

Figure 11: Generated Gaussian Filter for an image (Figure 9)

Kernel ✕							
7x7 double							
	1	2	3	4	5	6	7
1	1.9641e-05	2.3928e-04	0.0011	0.0018	0.0011	2.3928e-04	1.9641e-05
2	2.3928e-04	0.0029	0.0131	0.0215	0.0131	0.0029	2.3928e-04
3	0.0011	0.0131	0.0585	0.0965	0.0585	0.0131	0.0011
4	0.0018	0.0215	0.0965	0.1592	0.0965	0.0215	0.0018
5	0.0011	0.0131	0.0585	0.0965	0.0585	0.0131	0.0011
6	2.3928e-04	0.0029	0.0131	0.0215	0.0131	0.0029	2.3928e-04
7	1.9641e-05	2.3928e-04	0.0011	0.0018	0.0011	2.3928e-04	1.9641e-05

Figure 12: Generated Gaussian Filter for an image (Figure 6) by using code in Appendix K

In Figure 12, at location [2,2] , calculated Gaussian filter is 0.0029 which matches at location [2,2] in Figure 11.

Test cases for Gaussian weighted image mean value calculation using convolution

Now after creating a 7x7 two dimensional Gaussian filter, it is applied on the image to calculate mean using equation (7). filter2 function in Matlab is used to calculate mean in line 13 (Appendix C), filter2 use convolution function to convolve image pixels with 2D Gaussian filter.

$Y = \text{filter2}(h, X)$ filters the data in X with the two-dimensional FIR filter in the matrix h . It computes the result, Y , using two-dimensional correlation, and returns the central part of the correlation that is the same size as X . This is done by the following code in Matlab in Appendix C. It is calculates by using equation (9).

mu					
196x174 double					
	54	55	56	57	58
24	100.7049	71.0634	62.0473	55.3130	46.3969
25	77.2095	58.2026	53.0769	49.1373	45.3339
26	68.8536	60.0682	56.0757	52.1167	50.6888
27	74.2884	71.5918	67.8664	63.3721	62.1678
28	80.8754	77.9972	75.9792	74.3460	76.6977
29	84.3885	80.0995	82.0880	85.5765	91.8864

Figure 13: Mean values of gray scale image (24...29, 54....58) are shown in Figure 9

Test cases for Gaussian weighted image variance calculation using convolution

After calculating mean, next step is to calculate variance using equation (10). This is done by the following code in Matlab in Appendix C, filter2 function is used (line 15) to calculate variance (same like mean).

mu					
sigma					
196x174 double					
	54	55	56	57	58
24	31.5935	24.9799	18.6173	14.7868	12.6494
25	28.2553	21.4029	15.7284	13.7672	12.3920
26	24.7504	18.8497	16.7305	16.8390	14.7262
27	19.4156	16.8924	17.7809	18.5732	16.6910
28	17.2890	16.9886	17.8920	18.4007	17.5421
29	18.4676	17.8669	16.9041	16.5046	17.2567

Figure 14: Variance values (24...29, 54...58) of a given image (Figure 9)

Test cases for image normalization using Gaussian weighted mean and standard deviation according to (8)

Next step is to put the values of mean, standard deviation and C=1 into equation (8).

This is done by the following Code in Matlab in Appendix C in line 16.

structdis					
196x174 double					
	54	55	56	57	58
24	-0.4512	-1.1572	0.5073	0.1069	-0.3954
25	-1.4770	-0.3215	0.0552	-0.6188	0.9458
26	-0.5768	-0.7591	-0.6247	-1.5761	-0.5525
27	-0.0141	1.1965	0.7526	1.3093	0.0470
28	0.2802	0.6672	-0.6870	-1.3064	-0.9545
29	-0.7391	-1.5954	0.8887	0.9954	0.6087

Figure 15: Normalize pixel intensity values of gray scale image (24...29, 54...58)
(Figure 9)

imdist at location [24 54] = 86 (from Figure 9), mean at location [24 54] = 100.7049
(from Figure 13) and variance at location [24 54] = 31.5935 (from Figure 14)

$$\begin{aligned} \text{Normalized pixel at location [24 54]} &= \text{imdist}_{[4 \ 4]} - \mu_{[4 \ 4]} / \sigma_{[4 \ 4]} + 1 \\ &= -0.4512 \end{aligned}$$

And by using equation (8), we get normalized image.



Figure 16: Normalized image

3.2.2 Implementation and testing of GGD parameters

Two parameters, shape parameter and standard deviation are estimated for a normalized image. Steps to calculate the two parameters that are shape parameter (α) and variance (σ^2) of an images using moment matching based approach. It is implemented by Matlab code in Appendix D.

Implementation and testing of Generalized Gaussian ratio

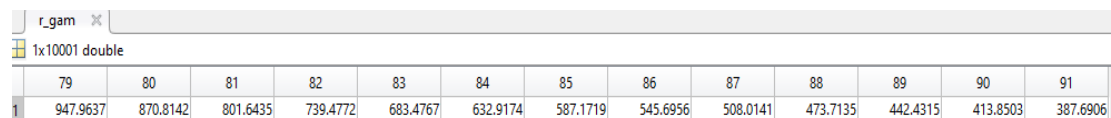
Generalized Gaussian ratio is calculated by equation (13) for a given image. It is implemented by Matlab code in Appendix D (line 7 to 28).

Test cases to calculate Generalized Gaussian ratio function

By using equation (13) Gaussian ration function for an image in Figure 16 is calculated, a look up table is defined for parameter (γ).This is done by the following code in Matlab in Appendix D (line 7 to 28).

To define lookup table in Matlab, two counters counter 1 and counter 2 are initialized in line 7 and 8 and their initial value set to 0.2 and 10 respectively with a difference between of 0.001 between them (line 10) and it iterates 100 times using for loop (line 9), counter 1 decrement by 1 and counter2 increment by 2 after each iteration (line 25 and 26). These values can be adjusted according to the user need after defining look up table. Gaussian ratio has been calculated for each value in look up table (line 11). These values can be adjusted according to the user need.

After defining look up table, Gaussian ratio $r(\gamma)$ which is r_gam (line 11) has been calculated for each value in look up table.



	79	80	81	82	83	84	85	86	87	88	89	90	91
1	947.9637	870.8142	801.6435	739.4772	683.4767	632.9174	587.1719	545.6956	508.0141	473.7135	442.4315	413.8503	387.6906

Figure 17: Values of $r(\gamma)$ is shown (from position 79 to 91) which is obtained using lookup table

At position 80, Gaussian ratio function by using look up(value at position 80 in look up table is 0.0790) table using equation (13)

$$\text{Gaussian ratio} = \frac{\tau(1/0.790)*\tau(3/0.790)}{\tau^2(2/0.790)} = 870.8142$$

which matches with value at position 80 in Figure 17.

Implementation and testing of mean and standard deviation

Mean and standard deviation are calculated using equations (14) and (15). It is implemented by using Matlab code in Appendix D (line 2 to 4).

Test cases to calculate mean μ_x and standard deviation σ_x^2 of an image in Figure 16

Mean and variance is calculated by using equations (14) and (15) for a given image. This is done by the following Code in Matlab in Appendix D in line 2 and 3 respectively. Standard deviation and mean are 0.0483 and 0.6650 respectively using Matlab.

To calculate mean manually,

Number of pixels of given image (Figure 16) = $196 * 174 = 34104$

Sum of all pixels of given image (Figure 16) = $1.648031525168372e+03$

Mean = $1.648031525168372e+03 / 34104 = 0.0483$

After subtracting mean value from every pixel and taking its square of the given image (Figure 16), sum of these pixel are = 15080.41

$$\text{Sigma} = \sqrt{15080.41 / 34104} = \sqrt{0.4422} = 0.6650$$

Implementation and testing of absolute value of modified mean of a given images

It is calculated by using equation (14). It is calculated in Matlab code in Appendix D (line 5).

Test cases to calculate estimate $E[|X|]$ for the absolute values modified mean for a given image (Figure 16)

Estimate of absolute value modified mean is calculated by using equation (16). This is done by the following Code in Matlab in Appendix D in line 5. Estimated of modified mean is 0.4947.

Implementation and testing of ratio

It is calculated by using equation (17) and implements using Matlab code in Appendix D (line 6).

Test cases to determine the ratio ρ

Ratio (ρ) is calculated using equation (15) for Figure 17 as

$$\rho = [0.4422]/[0.4947]^2 = 1.8065$$

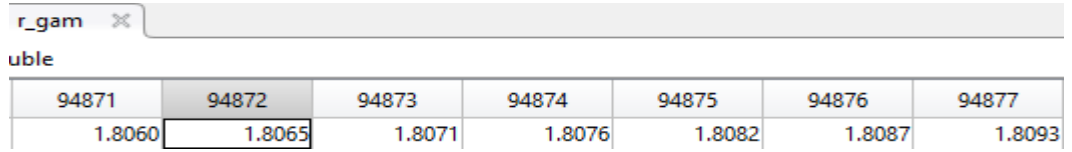
This is done by the following Code in Matlab in Appendix D in line 5.

Implementation and testing for finding shape parameter

This is calculated by using equation (18). This is done by the following code in Matlab in Appendix D. In this code, obtained ratio value is looked in Gaussian ratio vector and once it found on particular position, value is checked for same position in lookup table and that is our shape parameter.

Test case to solve the equation $\gamma = r^{-1}(\rho)$

where, r is the generalized Gaussian ratio function. Look up can be defined by the user or by giving instructions to machine to choose random variable with a constant difference between two variables.

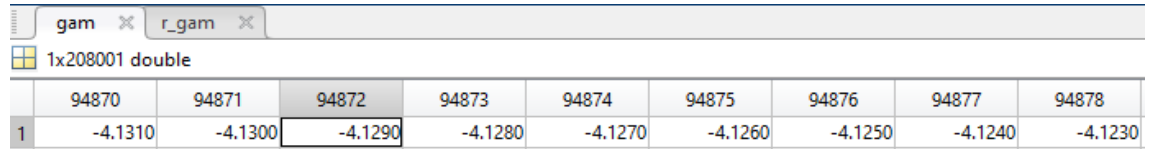


94871	94872	94873	94874	94875	94876	94877
1.8060	1.8065	1.8071	1.8076	1.8082	1.8087	1.8093

Figure 18: Determine array position in Gaussian ratio table (5) by using value of ratio

This is done by the following Code in Matlab in Appendix D. We first calculate at which position, ρ and Gaussian function values are nearest . After determining that array position, we check what is γ value at that position and that will be our shape parameter, γ is calculated using lookup table and shape parameter. γ obtained value is -4.1290.

For example, in our example ρ is 1.8065, we look for same value of Gaussian function in our table for that we check r_gam table and write down the array position. In Figure 15, at array position 94872 in r_gam table (Gaussian function) value is 1.8065 which is equal to ρ . We check the γ at same array position that is at position 94728 is -4.1290.



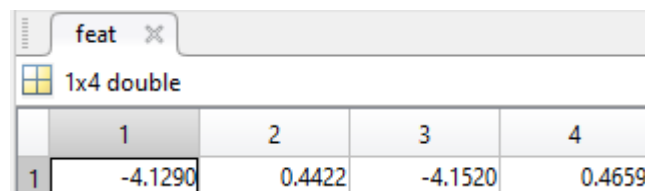
	94870	94871	94872	94873	94874	94875	94876	94877	94878
1	-4.1310	-4.1300	-4.1290	-4.1280	-4.1270	-4.1260	-4.1250	-4.1240	-4.1230

Figure 19: Determine value in look up table corresponds to array position found in Figure 18

So, our shape parameter in this example is -4.1290. By applying the above four steps in an image, two important features that is shape parameter (α) and standard deviation(σ^2) is found, which will prove to very useful later in the experiments.

And the image is down sampled and the same process is applied on down sampled image and a total of 4 features are obtained. This is done by the following code in Matlab in Appendix A3. Two features which are shape parameter (-4.1290) and standard deviation (0.4422) are calculated using Generalized Gaussian distribution on image (Figure 16) are obtained.

The image is down sampled and the same process is applied on down sampled image and a total of 4 features are obtained. This is done by the following Code in Matlab in Appendix A4. Two features which are shape parameter (-4.1520 and standard deviation (0.4659) are calculated for down-sampled image using Generalized Gaussian distribution. We got a total of 4 parameters



	1	2	3	4
1	-4.1290	0.4422	-4.1520	0.4659

Figure 20: A totals of 4 parameters are calculated for a given image (Figure 16)

3.2.3 Implementation and testing of AGGD parameters

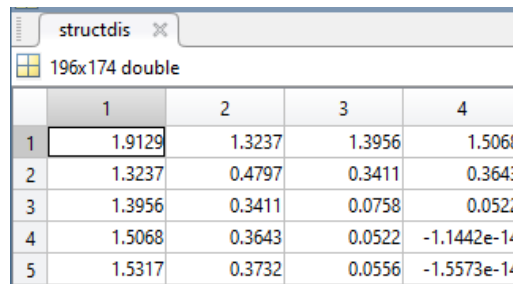
Four parameters that are shape, mean, left variance and right variance are calculated in four orientations – Horizontal (H), vertical (V), main-diagonal (D1) and secondary-diagonal (D2).

Implementation and testing spatial features in horizontal, vertical, diagonal D1 and diagonal D2 directions

It is calculated using equation (11). It is implemented using Matlab code in Appendix B (line 19 to 21).

Test cases to calculate horizontal, vertical, diagonal D1 and diagonal D2 spatial features of given image using equation (11)

This is done by the following Code in Matlab by Appendix C. In this code, circular shift is used to compute pixel value at horizontal, vertical and diagonal levels as defined by shift to get desired value (stored in pair) in horizontal, vertical and diagonal direction (line 19 to 21).



	1	2	3	4
1	1.9129	1.3237	1.3956	1.5068
2	1.3237	0.4797	0.3411	0.3643
3	1.3956	0.3411	0.0758	0.0522
4	1.5068	0.3643	0.0522	-1.1442e-14
5	1.5317	0.3732	0.0556	-1.5573e-14

Figure 21: Pixel value intensities (1...4, 1...5) of an image (Figure 16)

shifted_structdis				
196x174 double				
	1	2	3	4
1	1.9068	1.9129	1.3237	1.3956
2	1.3290	1.3237	0.4797	0.3411
3	1.3968	1.3956	0.3411	0.0758
4	1.5061	1.5068	0.3643	0.0522
5	1.5311	1.5317	0.3732	0.0556

Figure 22: Pixel value intensities (1...4, 1...5) of an image (Figure 16) after horizontal shift

pair	
34104x1 double	
	1
1	3.6475
2	1.7593
3	1.9494
4	2.2695
5	2.3451
6	2.3527
7	2.3545

Figure 23: Values of vector (1...7) obtained by multiplication of shifted image pixel values in horizontal direction (Figure 22) with normalized image pixel values (Figure 21) and stored in pair

For example, using equation (11), pixel at location [1 1]

= pixel [1 1] in normalized image * pixel [1 1] in shifter image (from Figure 21 & 22)

= 1.9129 * 1.9068

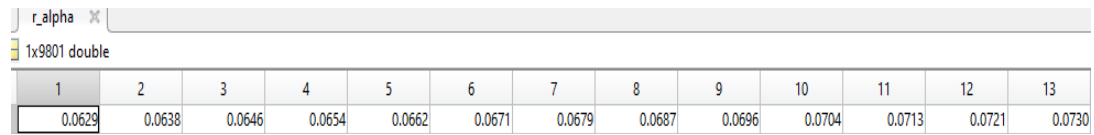
= 3.6475 (from Figure 23)

Implementation and testing of Generalized Gaussian ratio

It is calculated using equation (20). It is implemented using Matlab code in Appendix E (line 7 to 28).

Test cases to calculate Generalized Gaussian ratio function

This is done by the following Code in Matlab in Appendix E. Gaussian ratio function is calculated by using equation (20), a look up table is defined for parameter gam (α). To define lookup table in matlab, two counters counter 1 and counter 2 are initialized (line 7 and 8) and their initial value set to 0.2 and 10 respectively with a difference of 0.001 between them (line 10) and it iterates 100 times using for loop (line 9), counter 1 decrement by 1 and counter2 increment by 2 after each iteration (line 25 and 26). These values can be adjusted according to the user need. After defining look up table, Gaussian ratio has been calculated for each value in look up table (line 11).



1	2	3	4	5	6	7	8	9	10	11	12	13
0.0629	0.0638	0.0646	0.0654	0.0662	0.0671	0.0679	0.0687	0.0696	0.0704	0.0713	0.0721	0.0730

Figure 24: Initial 13 values of $r(\alpha)$ is shown of image (Figure 13) which is obtained using lookup table

To calculate Gaussian ratio at position 1, by using look table value at position 1 which is 0.2 using equation (20)

$$= \frac{\tau^2(2/0.2)}{\tau(1/0.2)\tau(3/0.2)} = 0.0629$$

which matches with the value at position 1 in Figure 24.

Implementation and testing of right standard deviation (β_r) and left standard deviation (β_l)

It is calculated by using equations (21) and (22). It is implemented using Matlab code in Appendix E (line 2 and 3).

Test cases to calculate right standard deviation (β_r) and left standard deviation (β_l)

Left and right standard deviation are calculated by using equations (21) and (22) for an image in Figure (16). This is done by the following Code in Matlab in Appendix E in line 2 and 3. The values of left and right standard deviation for an image (Figure 13) are 0.5554 and 0.5324 respectively.

To calculate left standard deviation manually, all the pixels less than zero value are calculated, there are total of 13322 pixels with value less than zero in an image (Figure 16)

Sum of square of all pixel values less than zero = 4108.92821139148

Left standard deviation = $\sqrt{4108.92/13322}$ = 0.554

Similarly, to calculate right standard deviation manually, all pixels greater than zero values are calculated, there are total of 20782 pixels with value greater than zero in an image (Figure 16)

Sum of square of all pixel values greater than zero = 5889.77500141591

Implementation and testing of gamma hat ($\check{\gamma}$), unbiased estimate ($\check{r}$) and $\check{R}$ using $\check{\gamma}$ and $\check{r}$

They are calculated by using equations (23), (24) and (25). It is implemented using Matlab code in Appendix E (line 4,5 and 6) respectively.

Test cases to calculate the value of gamma hat ($\tilde{\gamma}$), unbiased estimate ($\tilde{r}$) and $\tilde{R}$ using $\tilde{\gamma}$ and $\tilde{r}$

Gamma hat ($\tilde{\gamma}$), r hat (unbiased estimate) and $\tilde{R}$ are calculated by using equations (23) (24) and (25) for an image in Figure (16). This is done by the following Code in Matlab in Appendix E in line 4, 5 and 6 and the values obtained are 1.0432, 0.3333 and 0.3334 respectively for an image in Figure (16).

For manual calculation,

gamma hat ($\tilde{\gamma}$) = left standard deviation/ right standard deviation

$$= 0.554/0.5324 = 1.04$$

unbiased estimate ($\tilde{r}$) = square of mean of absolute value of all pixels / square of mean values of all pixels

$$= 0.0977/0.2932 = 0.3332$$

$$\tilde{R} \text{ is calculated} = (0.3332 * (1.04^3 + 1) * (1.04 + 1)) / ((1.04^2 + 1)^2) = 0.3333$$

Implementation and testing to calculated shape parameter (α)

It is calculated by using equation (26). Obtained Rhatnorm($\tilde{R}$) value is looked in Gaussian ratio vector and once it found on particular position, value is checked for same position in lookup table and that is our shape parameter. It is implemented using Matlab code in Appendix E (line 16).

Test cases to estimate α according to $\check{R}$ value using the approximation of the inverse generalized Gaussian ratio

Shape parameter (α) is calculated by using equation (26) for an image in Figure (16). This is done by the following Code in Matlab in Appendix E (line 16). We first calculate at which position, rho and Gaussian function values are nearest by using the following code

```
[min_difference, array_position] = min((r_alpha - rhatnorm).^2);
```

After determining that array position, we check what is gam (γ) value at that position and that will be our shape parameter. r_alpha is $\rho(\alpha)$, gam is calculated using lookup table and alpha is shape parameter. Alpha value for an image (Figure 16) is calculated as 0.5570.

For example, in our example rhatnorm is 0.3334, we look for same value of Gaussian function in our table, in Code 5, we check r_alpha table and look for same value and write down the array position.

Implementation and testing of left scale parameter ($\check{\beta}_l$) and right scale parameter ($\check{\beta}_r$)

It is calculated by using equations (27) and (28). It is implemented using Matlab code in Appendix C (line 24).

Test cases to compute left scale parameter ($\check{\beta}_l$) and right scale parameter ($\check{\beta}_r$)

This is done by the following Code in Matlab in Appendix C in line 24 for an image in Figure (16). Obtained value for left scale parameter ($\check{\beta}_l$) and right scale parameter ($\check{\beta}_r$) are 3.8030 and 3.6455 respectively.

Implementation and testing of statistical mean (η)

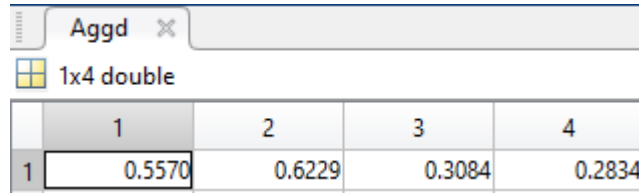
It is calculated by using equation (29). It is implemented using Matlab code in Appendix C (line 25).

Test cases to compute statistical mean (η)

It is calculated by using equation (29) for an image in Figure (16). This is done by the following Code in Matlab in Appendix C in line 25. Obtained value using Matlab is 0.6229.

$$\begin{aligned}\text{Statistical mean} &= (\text{beta_l} - \text{beta_r}) * (\text{gamma}(2/\alpha) / \text{gamma}(1/\alpha)); \\ &= (3.8030 - 3.6455) * \text{gamma}(2/0.5570) / \text{gamma}(1/0.5570) \\ &= 0.6229\end{aligned}$$

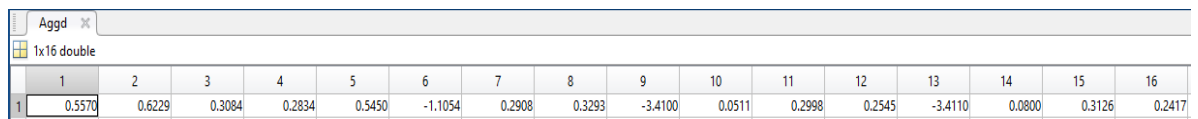
Four features that are shape parameter (α), left standard deviation (β_l), right standard deviation (β_r) and statistical mean (η) are obtained, these features are calculated in horizontal, vertical and diagonal spatial directions in a given image by using equation (31).



	1	2	3	4
1	0.5570	0.6229	0.3084	0.2834

Figure 25: A total of 4 parameters that are shape parameter, statistical mean, left and right standard deviation for an image (Figure 16) are estimated

Similarly features have been calculated in vertical, diagonal D1 and diagonal D2 direction and a total of 16 features have been obtained.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0.5570	0.6229	0.3084	0.2834	0.5450	-1.1054	0.2908	0.3293	-3.4100	0.0511	0.2998	0.2545	-3.4110	0.0800	0.3126	0.2417

Figure 26: A total of 16 features in horizontal, vertical and in diagonal direction are calculated for an image (Figure 16)

A total of 16 features obtained in horizontal, vertical, diagonal 1 and diagonal 2 directions. Circular shift in Matlab is used to compute spatial features at horizontal, vertical and diagonal levels. After calculating 16 features, image is down sampled and again the same process applied and we will get a total of 32 features. In the end features obtain by estimating GGD (4 features) and by estimating AGGD (32 features) are combined together and a total of 36 features are obtained.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0.5570	0.6229	0.3084	0.2834	0.5450	-1.1054	0.2908	0.3293	-3.4100	0.0511	0.2998	0.2545	-3.4110	0.0800	0.3126	0.2417

	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	0.5590	-5.3054	0.2001	0.4207	-3.4330	0.2316	0.1583	0.5450	-3.3960	-0.1310	0.2480	0.3675	-3.3870	-0.0505	0.2963	0.3409

Figure 27: A total of 32 parameters obtained by concatenation of features from normalized image (Figure 16) and down sampled image.

In the end we combine feature obtain by estimating GGD (4 features) and by estimating AGGD (32 features) and we get a total of 36 features.

3.3 Implementation and testing of extraction of HOG features

- Histogram of Oriented Gradients (HOG) - HOG descriptor was introduced Dalal and Triggs at the Conference on Computer Vision and Pattern Recognition(CVPR) back in 2005.
- The purpose of HOG is to describe feature of the given image.
- It includes steps like computation of gradient vectors, then orientation binning and blocks normalization.

3.3.1 Implementation and testing of gradient vectors

Gradient vector in x direction of pixel (h_x) is calculated using equation (30) and gradient vector in y direction of pixel (h_y) is calculated using equation (31).

By appending these two values we get gradient vectors. Magnitude and angle is also obtained by using equations (32) and (33). This is done by the following code in Matlab in Appendix F (line 8 to 13).

Let's take an example, we have pixel (z) and that pixel has four neighbor pixels,

93

56 z 94

55

Gradient vector in x direction of pixel (hx) = pixel value on right– pixel value in left

$$= 94 - 56 = 38$$

Gradient vector in y direction of pixel (hy) = pixel value on top – pixel value in bottom

$$= 93 - 55 = 38$$

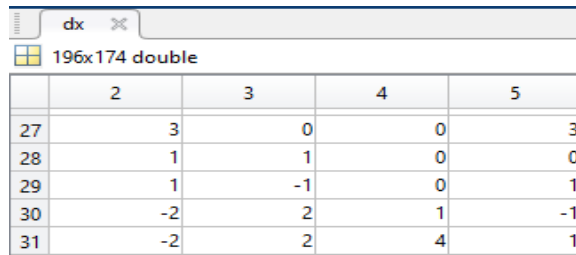
By appending these two values we get out gradient vector $\begin{bmatrix} hx \\ hy \end{bmatrix} = \begin{bmatrix} 38 \\ 38 \end{bmatrix}$

$$\text{Magnitude} = \sqrt{hx^2 + hy^2} = 53.74$$

$$\text{Angle} = \arctan\left(\frac{hx}{hy}\right) = 45 \text{ degrees}$$

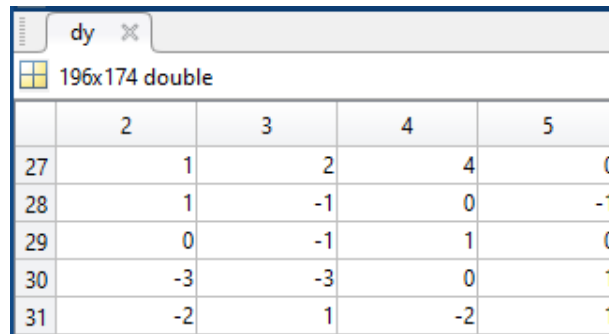
Test for calculation of gradient vectors of an image

Gradients are calculated using equations (30) and (31) for an image in Figure 9. It is computed for every pixel of an image, it calculates the variance in pixel value in horizontal and vertical direction. This is done by the following Code in Matlab in Appendix F (line 8 to 11). In Matlab, derivatives of a pixel created in x and y direction and then compute gradient vector using filter2 function of Matlab in line 10 and 11, then angles and magnitude are calculated in line 12 and 13.



	2	3	4	5
27	3	0	0	3
28	1	1	0	0
29	1	-1	0	1
30	-2	2	1	-1
31	-2	2	4	1

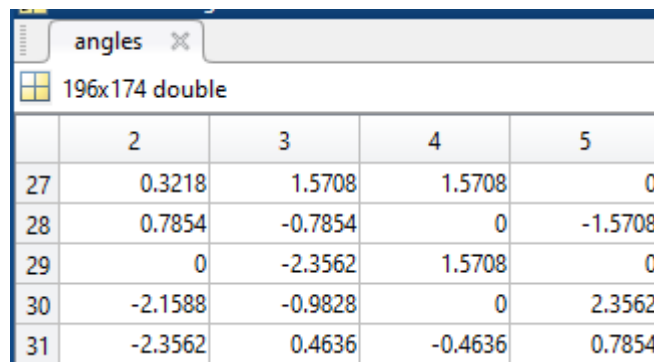
Figure 28: Gradient in x direction [2..5, 27..31] for an image (Figure 9)



	2	3	4	5
27	1	2	4	0
28	1	-1	0	-1
29	0	-1	1	0
30	-3	-3	0	1
31	-2	1	-2	1

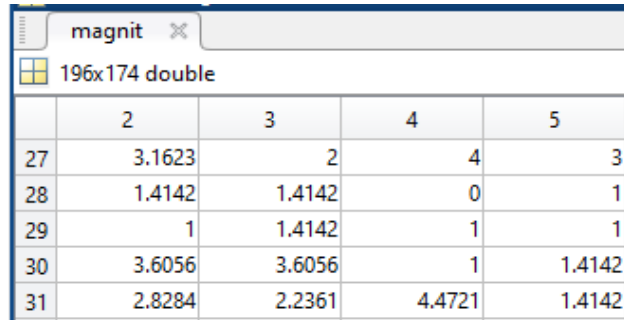
Figure 29: Gradient in y direction [2..5, 27..31] for an image (Figure 9)

Atan2 function in Matlab is used to calculate to angle in line 12, atan2 returns the four-quadrant inverse tangent ($\tan^{-1}$) of dy and dx and magnitude is calculated in line 13.



	2	3	4	5
27	0.3218	1.5708	1.5708	0
28	0.7854	-0.7854	0	-1.5708
29	0	-2.3562	1.5708	0
30	-2.1588	-0.9828	0	2.3562
31	-2.3562	0.4636	-0.4636	0.7854

Figure 30: Angles are calculated [2..5, 27..31] for an image (Figure 9)



	2	3	4	5
27	3.1623	2	4	3
28	1.4142	1.4142	0	1
29	1	1.4142	1	1
30	3.6056	3.6056	1	1.4142
31	2.8284	2.2361	4.4721	1.4142

Figure 31: Magnitudes are calculated [2..5, 27..31] for an image (Figure 9)

3.3.2 Implementation and testing of orientation binning

In our experiment, Number of cells = window size / number of pixels in one cell

Window size = 64 x 80

Number of pixels in one cell = 8x 8

Number of horizontal cells = 64/8 = 8

Number of vertical cells = 80/8 = 10

A cell size of 8×8 is taken which means a total of 64 pixels and we obtain a total of 64 gradient vectors correspond to 64 pixels and place them into total of 9 histogram bins. Unsigned angle changed to signed angle (0 to 180). For each gradient vector, its contribution to the histogram is given by the magnitude of the vector (so stronger gradients have a bigger impact on the histogram). Image pixel is grouped into cells, every cell has 8x8 pixels that's a total of 64 pixels, and each pixel have it angle and magnitude. This is done by the following code in Matlab in Appendix F.

Test for orientation binning

Calculate histogram value for each pixel and in which bin it pixel will fall according to its histogram value. Next magnitude and angle for each 64 pixels in a cell is calculated using equations (32) and (33). This is done by the following Code in Matlab in Appendix F (line 18 to 53).

angles								
196x174 double								
	1	2	3	4	5	6	7	8
21	179.9921	179.0727	179.2146	179.2146	0	0	0	0
22	179.9960	0	179.4120	179.2146	179.2146	0	0	0
23	0	0	178.4292	179.2146	179.2146	0	0	0
24	0.0080	177.6438	0	179.2146	179.4120	179.6782	0	0
25	179.9960	178.4292	177.6438	178.4292	179.7550	179.4120	0	0
26	179.9880	3.1416	3.1416	0.4636	0	0	0	0
27	0	0.3218	1.5708	1.5708	0	179.6782	0	0
28	0.0040	0.7854	179.2146	0	178.4292	179.5364	179.6782	0

Figure 32: Grouping of 64 pixels (1..2, 21..28) calculated angles into a cell for Figure 9

magnit								
196x174 double								
	1	2	3	4	5	6	7	8
21	252.0079	5	4.2426	1.4142	0	0	0	0
22	251.0020	1	3.6056	4.2426	1.4142	0	0	0
23	252	2	2	2.8284	4.2426	1	0	0
24	251.0080	1.4142	1	2.8284	3.6056	3.1623	0	0
25	251.0020	1	1.4142	4	4.1231	3.6056	1	0
26	250.0180	1	2	2.2361	5	4	2	0
27	251	3.1623	2	4	3	3.1623	1	0
28	251.0020	1.4142	1.4142	0	1	4.4721	3.1623	0

Figure 33: Grouping of first 64 pixels (1..2, 21..28) calculated magnitudes into a cell for Figure 9

Next step is to put gradients into histogram, In histogram calculation if angles of pixel all in the range of any bin angle, then that pixel magnitude added into that bin..This is done by the following Code in Matlab in Appendix F (line 18 to 53).

3.3.3 Implementation and testing of block normalization

This block normalization is executed by appending the histograms of the four cells within the block into a vector with 36 components (4 histograms x 9 bins per histogram) and L2 normalization, L1 normalization and L1 sqrt normalization are performed which are calculated by using equation (34), (35) and (36). This is done by the following Code in Matlab in Appendix F in line 58, 59 and 60.

Testing for block normalization

For 1st block, after concatenating 4 histograms (9 bins in each histogram) of cells in a block, 36 features have been obtained for an image in Figure 9.

block_feature													
1x36 double													
1	2	3	4	5	6	7	8	9	10	11	12	13	
3.9306e+03	0	0	0	0	0	0	0	0	2040	0	0	0	
1x36 double													
13	14	15	16	17	18	19	20	21	22	23	24	25	
0	0	0	0	0	0	2040	0	0	0	0	0	0	
1x36 double													
26	27	28	29	30	31	32	33	34	35	36			
0	0	0	0	0	0	0	0	0	0	0			

Figure 34: Histogram values in 36 bins for one block for Figure 9

After obtaining 36 features, L 2, L1 and L1 sqrt normalization is performed over the features using equations (34), (35) and (36).

1	2	3	4	5	6	7	8	9	10	11	12	13	
0.8061	0	0	0	0	0	0	0	0	0.4184	0	0	0	
14	15	16	17	18	19	20	21	22	23	24	25	26	
0	0	0	0	0	0.4184	0	0	0	0	0	0	0	
26	27	28	29	30	31	32	33	34	35	36	37	38	
0	0	0	0	0	0	0	0	0	0	0			

Figure 35: All features in a block after L2 normalization for Figure 9

1	2	3	4	5	6	7	8	9	10	11	12	13
0.8061	0	0	0	0	0	0	0	0	0.4184	0	0	0
14	15	16	17	18	19	20	21	22	23	24	25	26
0	0	0	0	0	0	0.4184	0	0	0	0	0	0
26	27	28	29	30	31	32	33	34	35	36	37	38
0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36: All features in a block after L1 normalization for Figure 9

1	2	3	4	5	6	7	8	9	10	11	12	13
0.8978	0	0	0	0	0	0	0	0	0.6468	0	0	0
13	14	15	16	17	18	19	20	21	22	23	24	25
0	0	0	0	0	0	0.6468	0	0	0	0	0	0
25	26	27	28	29	30	31	32	33	34	35	36	37
0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37: All features in a block after L1 sqrt normalization for Figure 9

In this manner, 2268 features for the entire image are obtained.

1x2268 double	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0.8978	0	0	0	0	0	0	0	0	0.6468	0	0	0

Figure 38: First 13 features of total 2268 features for Figure 9

This conclude the final vector size to 7 blocks across x 9 blocks vertically x 4 cells per block x 9-bins per histogram = 2268 values. These are the final features extracted by using HOG in an image of window size 64 x 80. In the end HOG features and NSS features are concatenated to form QUALHOG features.

3.4 Implementation and testing of Classifiers usage

3.4.1 Implementation and testing of Liblinear SVM usage

For training of Images

Training of images is done by using the code shown in Appendix A. Images are loaded from database in line 9 and 10. HOG features are extracted in line 25, NSS features are extracted in line 28, HOG and NSS features are concatenated in line 31.

Liblinear SVM is trained using these features in line 39. Results are saved as model file in line 40. It is done separately for every level in AWGN, G Blur and JPEG.

It is implemented in Matlab by using the following command.

- For training
`model = train(label_vector, instance_matrix, 's ');`
`save('model.mat');`

Instance_matrix is extracted QUALHOG features of training images, label_vector contains labels (+1 for facial image and -1 for non facial image) and s is a solver and the result is saved in model.mat file. This is done by the Matlab code in Appendix A.

Testing of Liblinearsvm for training of images

For testing, let's take images distorted by level 10 in JPEG. For training on 1231 positive samples and 1500 negative samples in Figure 39.

```
.....**.  
optimization finished, #iter = 140  
Objective value = -13.015946  
nSV = 431
```

Figure 39: Result obtained after training

Figure 39, is a screenshot of result obtained after training using Liblinear.

For detection of faces in test Images

It is done by using the Matlab code in Appendix B. Model file (which is an output of training of images) is loaded in line 4, images are loaded from database in line 6. HOG features are extracted in line 19, NSS features are extracted in line 22, HOG and NSS features are concatenated in line 25.

It is implemented in Matlab by using the following command.

- For testing

```
[predict_label, accuracy, dec_values] = predict(label_vector,  
instance_matrix,model);
```

Instance_matrix is extracted QUALHOG features of tested images, label_vector contains labels (all +1) and model is file which is obtained from training section and the result predict_label which is vector contains +1 and -1. This is done by Matlab code in Appendix B.

Testing of Liblinear SVM for detection of faces in test images

For testing 393 positive images in Figure 40, accuracy is obtained

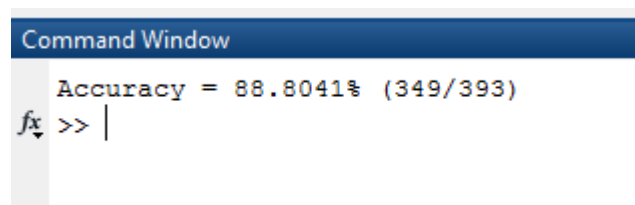


Figure 40: Accuracy (percentage of true positive images) is obtained for tested images

In Figure 40, 349 images out of 393 positive images are detected correctly using Liblinear SVM. Accuracy is obtained automatically by Liblinear SVM.

3.4.2 Implementation and testing of SVM Light usage

For training of images

SVM Lite is used for training of images by using following Matlab command

```
model = svmlearn(X_train, y_train);
```

X_train are the matrix of extracted features and y_train are the labels (+1 for faces and -1 for non faces). This is done by the Matlab code in Appendix H (line 38).

For detection of face in an image

SVM Lite is used for detection of face in an image by using following Matlab command

```
. [predictions] = svmclassify(P,lebel,model);
```

P is the matrix of extracted features, lebel can be anything and model is output of training of images. This is done by the Matlab code in Appendix I (line 52).

3.5 Implementation of FDS for any size of images using sliding window

In this method face detection for any size image (unlikely 80x 64 dimensions) is implemented by using sliding window technique. A window of 80x64 is taken and slide over an image pixel by pixel in order to detect face. This is done by the following code in Matlab in Appendix I. NSS features are extracted for testing and

training images similarly as described in section 3.2. HOG features are extracted for training and training images similarly as described in section 3.2.1.2.

In the end, these QUALHOG features (HOG features augmented with NSS features) are giving to SVM for classification and prediction and result is obtained. This is done by the following code in Matlab in Appendix I.

3.6 Conclusion

In this chapter, design of FDS along with implementation and testing of FDS and its subsystems like HOG, NSS, Liblinear SVM and SVM Light are discussed and explained in detail.

For testing FDS, An image from the database provided [1] is taken and testing is done to show and prove the authenticity of FDS and its subsystems. LIBLINEAR support vector machine is used for training and testing of data. Later sliding window technique is applied to detect face on any size of image by extracting QUALHOG features and using SVM Light.

Results are obtained and screenshot at every step of testing for extraction of QUALHOG features for FDS.

Chapter 4

EXPERIMENTS ON FDS FOR DISTORTED IMAGES

4.1 Experiment setup

To conduct experiments, same experimental setup is used as shown in section 2.5.1. However author of [1] doesn't reveal which operating they used and which version of Matlab were used for conducting experiments. In my case, Matlab 2015a is for Window 10 operating system is used with a memory of 6GB and Intel core i3 processor is used. In addition to Matlab 2015a, Visual studio 2010 is used. For classification LIBLINEAR SVM [6] and SVM light [13] are used. Same database [14] which used by the author is used for getting results and conducting experiments. Lastly 100 images containing face taken randomly from internet, and experiments are conducted on these 100 images using SVM lite and sliding window.

QUALHOG and HOG methods are analyzed on distorted image database. LIBLINEAR [6] is used for training and testing of images. Experiments are done on each level separately and results are noted down manually which is shown in an excel file available in Appendix L. True positive, True negative, False positive and False negative are calculated at every distorted level and based on True positive, True negative, False positive and False negative, Precision and Recall are calculated for 10 levels of distortion. In the end Precision vs Distortion level and Recall vs Distortion level curves for all three types of distortion (AWGN, GBlur and JPEG)

are obtained. Experiments on any size of images are done using SVM light for training and testing of images through sliding window.

Training the face detector

For each of the training set, LIBLINEAR is trained using 1231 positive images and 1500 negative images available in training dataset. Once the data is classified, it is stored in a model.mat file in Matlab. This is done by the following code in Matlab in Appendix A.

Prediction

For prediction, 393 positive images and 17479 negative images were taken from the testing dataset. Since the testing data is highly skewed as compare to training dataset, Precision and Recall is used as evaluation metrics.

4.2 Experiments results

True positive, True negative, False positive and False negative of the testing datasets are obtained for all three type of distortions using HOG features and QUALHOG features and both the detector compare with each other using Precision vsDistortion level (AWGN, GBlur and JPEG) curve.

4.2.1 Images distorted by AWGN (Additive White Gaussian Noise)

Extracting QUALHOG features for images

Table 1: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by AWGN with different levels using QUALHOG features, Precision and Recall are calculated.

AWGN distortion level(σ^2)	True positive (TP)	False negative (FN)	False positive (FP)	True negative (TN)	Detected positive (TP +FP)	Precision $\frac{TP}{TP + FP}$	Recall $\frac{TP}{TP + FN}$
4.5×10^{-5}	376	17	126	17353	502	0.75	0.95
0.0001	370	23	142	17347	512	0.73	0.94
0.0003	367	26	147	17338	514	0.72	0.93
0.0009	363	30	160	17319	523	0.70	0.92
0.0025	359	34	245	17234	604	0.60	0.91
0.0065	354	39	466	17013	820	0.44	0.90
0.02	334	59	1056	16423	1390	0.25	0.85
0.05	322	71	1854	15625	2176	0.15	0.81
0.15	311	52	2698	14781	3009	0.11	0.79
0.36	299	94	3694	13785	3993	0.08	0.760

This is done by the following code in Matlab in Appendix L. Table 1 is created manually by writing down the results of experiment done on images (parts by parts) due to inability of machine to handle large number of images simultaneously.

Extracting HOG features for images

Table 2: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by AWGN with different levels using HOG features, Precision and Recall are calculated.

AWGN distortion level(σ^2)	True positive (TP)	False negative (FN)	False positive (FP)	True negative (TN)	Detected positive (TP + FP)	Precision $\frac{TP}{TP + FP}$	Recall $\frac{TP}{TP + FN}$
4.5×10^{-5}	371	22	142	17337	513	0.73	0.94
0.0001	367	26	148	17331	515	0.71	0.93
0.0003	362	31	154	17325	516	0.70	0.92
0.0009	359	34	192	17287	551	0.65	0.91
0.0025	353	40	286	17193	639	0.55	0.89
0.0065	342	51	589	16890	931	0.37	0.87
0.02	330	59	1330	16149	1641	0.20	0.83
0.05	311	82	2307	15172	2618	0.12	0.79
0.15	301	92	3096	14383	3397	0.09	0.76
0.36	296	97	3805	13674	4101	0.07	0.75

This is done by the following code in Matlab in Appendix L. Table 2 is created manually by writing down the results of experiment done on images (parts by parts) due to inability of machine to handle large number of images simultaneously.

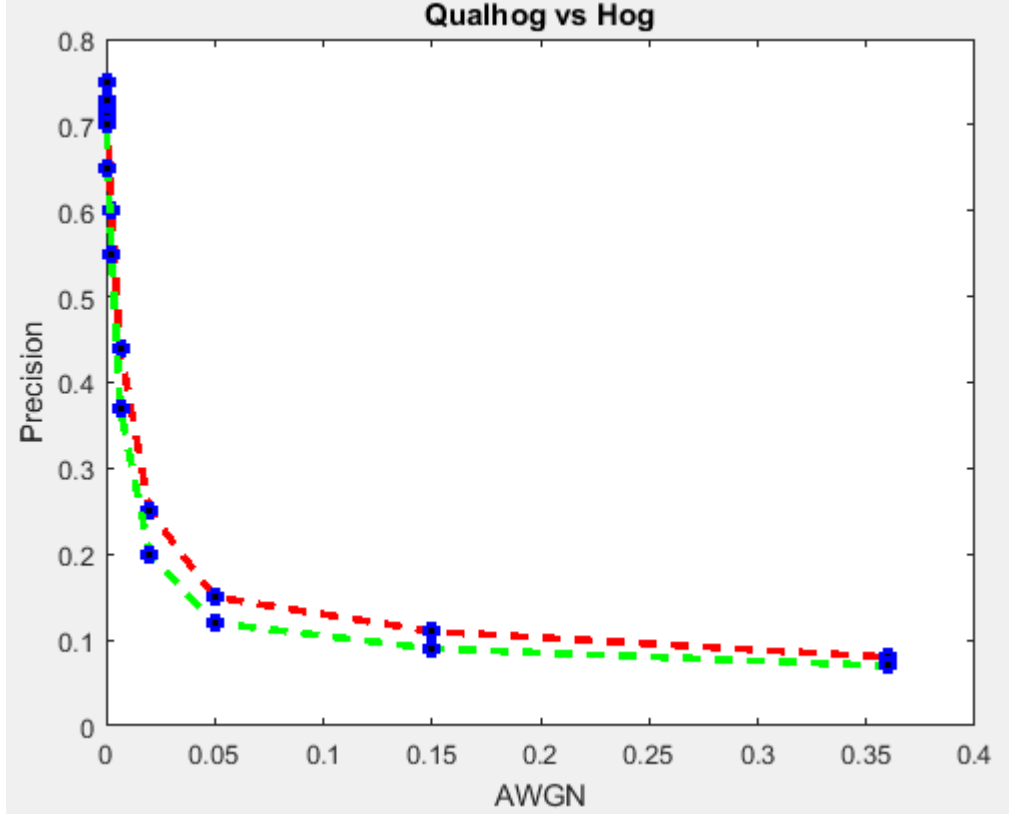


Figure 41: Precision vs AWGN curve for QUALHOG (red curve) and HOG (green curve) features

In Figure 41, red curve is for QUALHOG and green curve is for HOG. It shows that as the distortion (σ^2) increases, Precision decreased. At distortion level 4.5×10^{-5} , Precision is 0.75 for QUALHOG and 0.73 for HOG and as the distortion level increase to 0.08, Precision declines to 0.08 for QUALHOG and 0.07 for HOG which shows that QUALHOG (red curve) face detector performs better than HOG (green curve) face detector. This is done by the following code in Matlab in Appendix J. It is seen that QUALHOG Precision is 1-2% more accurate than HOG Precision for detecting faces in AWGN distorted images.

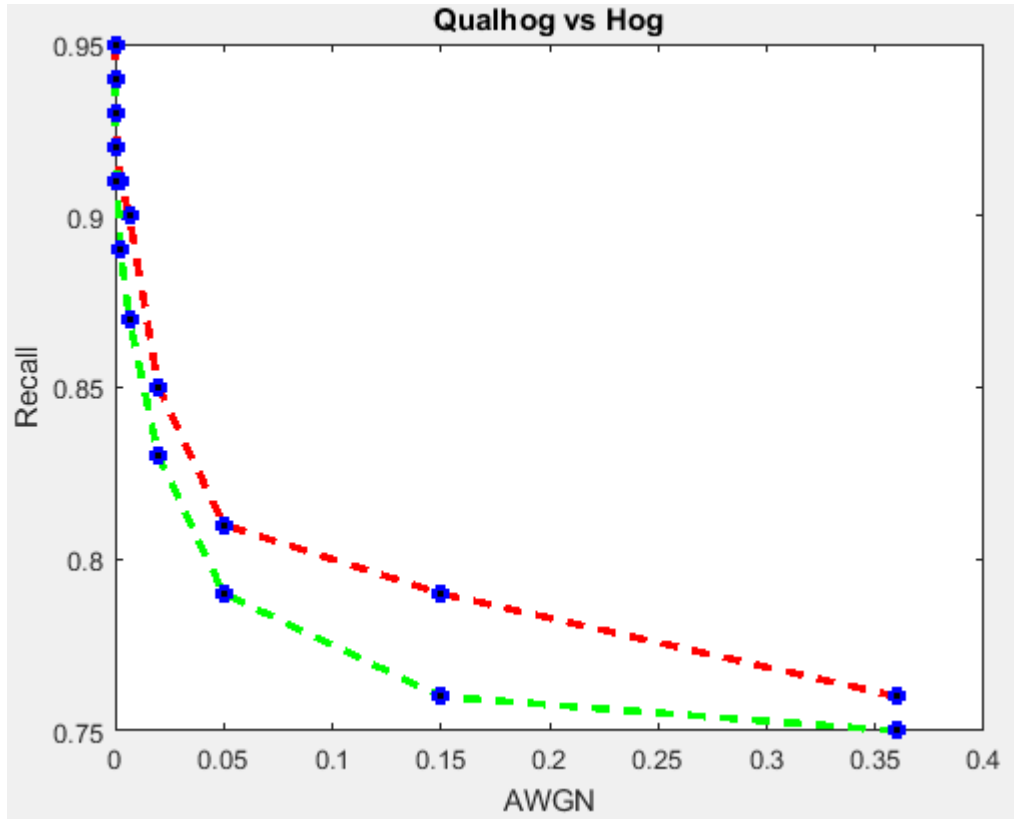


Figure 42: Recall vs AWGN curve for QUALHOG (red curve) and HOG (green curve) features

In Figure 42, red curve is for QUALHOG and green curve is for HOG. It shows that as the distortion (σ^2) increases, Recall decreased. At distortion level 4.5×10^{-5} , Recall is 0.95 for QUALHOG and 0.94 for HOG and as the distortion level increase to 0.08, precision declines to 0.760 for QUALHOG and 0.75 for HOG which shows that QUALHOG (red curve) face detector performs better than HOG (green curve) face detector. This is done by the following code in Matlab in Appendix J. It is seen that QUALHOG Recall is 2-4% more accurate than HOG Recall for detecting faces in AWGN distorted images.

4.2.2 Images distorted by G Blur (Gaussian Blur)

Extracting QUALHOG features of images

Table 3: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by Gaussian Blur with different levels using QUALHOG features, Precision and Recall are calculated.

G Blur distortion level (σ)	True positive (TP)	False negative (FN)	False positive (FP)	True negative (TN)	Detected positive (TP + FP)	Precision $\frac{TP}{TP + FP}$	Recall $\frac{TP}{TP + FN}$
0.4	380	13	133	17346	501	0.76	0.96
1.0	374	19	150	17329	514	0.73	0.95
2.3	373	20	367	17112	740	0.51	0.94
3.6	373	20	553	16926	926	0.41	0.94
4.5	370	23	623	16856	993	0.38	0.94
6.0	364	29	779	16700	1143	0.32	0.92
7.4	361	32	1146	16333	1507	0.24	0.91
12.0	334	59	1806	15673	2104	0.16	0.85
20.0	313	80	2550	14929	2863	0.11	0.79
32.0	282	111	3897	13582	4179	0.07	0.71

This is done by the following code in Matlab in Appendix L. Table 3 is created manually by writing down the results of experiment done on images (parts by parts) due to inability of machine to handle large number of images simultaneously.

Extracting HOG features of images

Table 4: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by Gaussian Blur with different levels using HOG features, Precision and Recall are calculated.

G Blur distortion level (σ)	True positive (TP)	False negative (FN)	False positive (FP)	True negative (TN)	Detected positive (TP +FP)	Precision $\frac{TP}{TP + FP}$	Recall $\frac{TP}{TP + FN}$
0.4	365	28	146	17333	511	0.72	0.93
1.0	362	31	164	17315	526	0.69	0.92
2.3	358	35	402	17077	760	0.47	0.91
3.6	357	36	587	16892	944	0.38	0.91
4.5	354	39	643	16836	997	0.35	0.900
6.0	351	42	805	16674	1156	0.30	0.89
7.4	346	47	1247	16232	1593	0.21	0.88
12.0	324	69	1946	15533	2270	0.14	0.82
20.0	302	91	2895	14584	3197	0.09	0.76
32.0	266	127	4134	13345	4400	0.06	0.67

This is done by the following code in Matlab in Appendix L. Table 4 is created manually by writing down the results of experiment done on images (parts by parts) due to inability of machine to handle large number of images simultaneously.

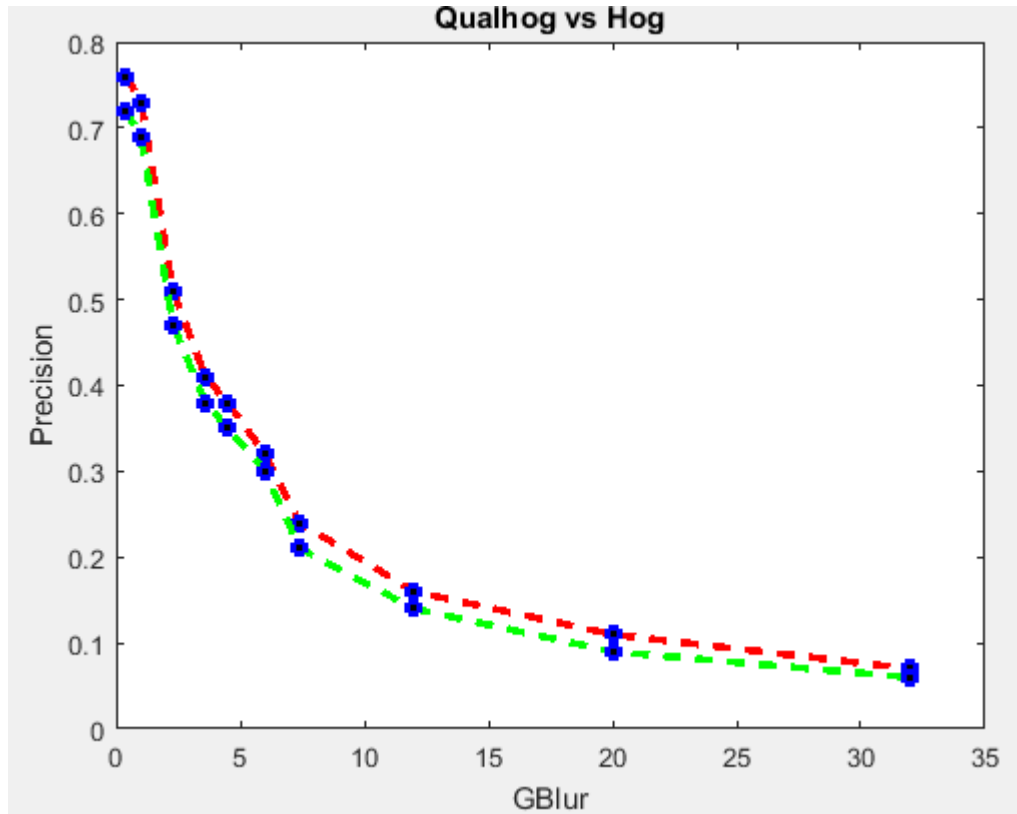


Figure 43: Precision vs GBlur curve for QUALHOG (red curve) and HOG (green curve) features

In Figure 43, red curve is for QUALHOG and green curve is for HOG. It shows that as the distortion (σ) increases, Precision decreased. At distortion level 0.4, Precision is 0.76 for QUALHOG and 0.72 for HOG and as the distortion level increase to 32, Precision declines to 0.07 for QUALHOG and 0.06 for HOG which shows that QUALHOG (red curve) face detector performs better than HOG (green curve) face detector. This is done by the following code in Matlab in Appendix J. It is seen that QUALHOG Precision is 1-2% more accurate than HOG Precision for detecting faces in GBlur distorted images.

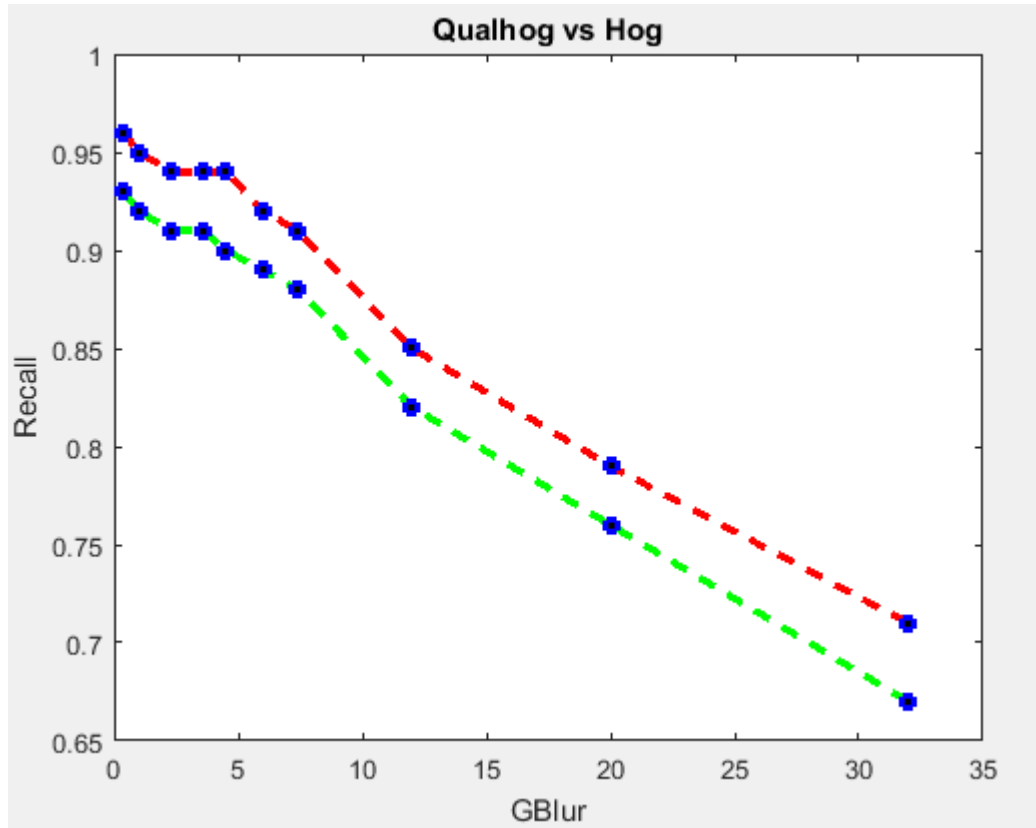


Figure 44: Recall vs GBlur curve for QUALHOG (red curve) and HOG (green curve) features

In Figure 44, red curve is for QUALHOG and green curve is for HOG. It shows that as the distortion (σ) increases, Recall decreased. At distortion level 0.4, Recall is 0.96 for QUALHOG and 0.93 for HOG and as the distortion level increase to 32, Recall declines to 0.71 for QUALHOG and 0.67 for HOG which shows that QUALHOG (red curve) face detector performs better than HOG (green curve) face detector. This is done by the following code in Matlab in Appendix J. It is seen that QUALHOG Recall is 3-4% more accurate than HOG Recall for detecting faces in GBlur distorted images.

4.2.3 Images distorted by JPEG

Extracting QUALHOG features of images

Table 5: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by JPEG in different levels (Q factor) using QUALHOG features, Precision and Recall are calculated.

JPEG distortion level 'Q'	True positive (TP)	False negative (FN)	False positive (FP)	True negative (TN)	Detected positive (TP +FP)	Precision $\frac{TP}{TP + FP}$	Recall $\frac{TP}{TP + FN}$
90	381	12	155	17324	536	0.710	0.970
60	373	20	173	17306	546	0.683	0.950
40	373	20	180	17299	553	0.674	0.950
25	373	20	194	17285	567	0.657	0.950
15	371	22	201	17278	572	0.648	0.944
10	364	29	216	17263	580	0.628	0.926
7.5	358	35	231	17248	589	0.607	0.910
5.0	356	37	405	17074	761	0.468	0.905
3.0	350	43	746	16733	1096	0.320	0.890
2.0	349	44	779	16700	1128	0.309	0.888

This is done by the following code in Matlab in Appendix L. Table 5 is created manually by writing down the results of experiment done on images (parts by parts) due to inability of machine to handle large number of images simultaneously.

Extracting HOG features of images

Table 6: Calculation of True positive, False negative, False positive, True negative, Detected positive, precision and recall for images distorted by JPEG in different levels (Q factor) using HOG features, Precision and Recall are calculated.

JPEG distortion level 'Q'	True positive (TP)	False negative (FN)	False positive (FP)	True negative (TN)	Detected positive (TP +FP)	Precision $\frac{TP}{TP + FP}$	Recall $\frac{TP}{TP + FN}$
90	381	12	158	17321	539	0.70	0.96
60	370	20	197	17282	567	0.65	0.94
40	368	20	199	17280	567	0.64	0.93
25	366	20	205	17274	571	0.64	0.93
15	359	22	215	17264	574	0.62	0.91
10	353	29	223	17256	576	0.61	0.90
7.5	348	35	244	17235	592	0.58	0.88
5.0	344	37	417	17062	761	0.45	0.87
3.0	334	43	805	16674	1139	0.29	0.84
2.0	328	45	846	16633	1174	0.26	0.82

This is done by the following code in Matlab in Appendix L. Table 6 is created manually by writing down the results of experiment done on images (parts by parts) due to inability of machine to handle large number of images simultaneously.

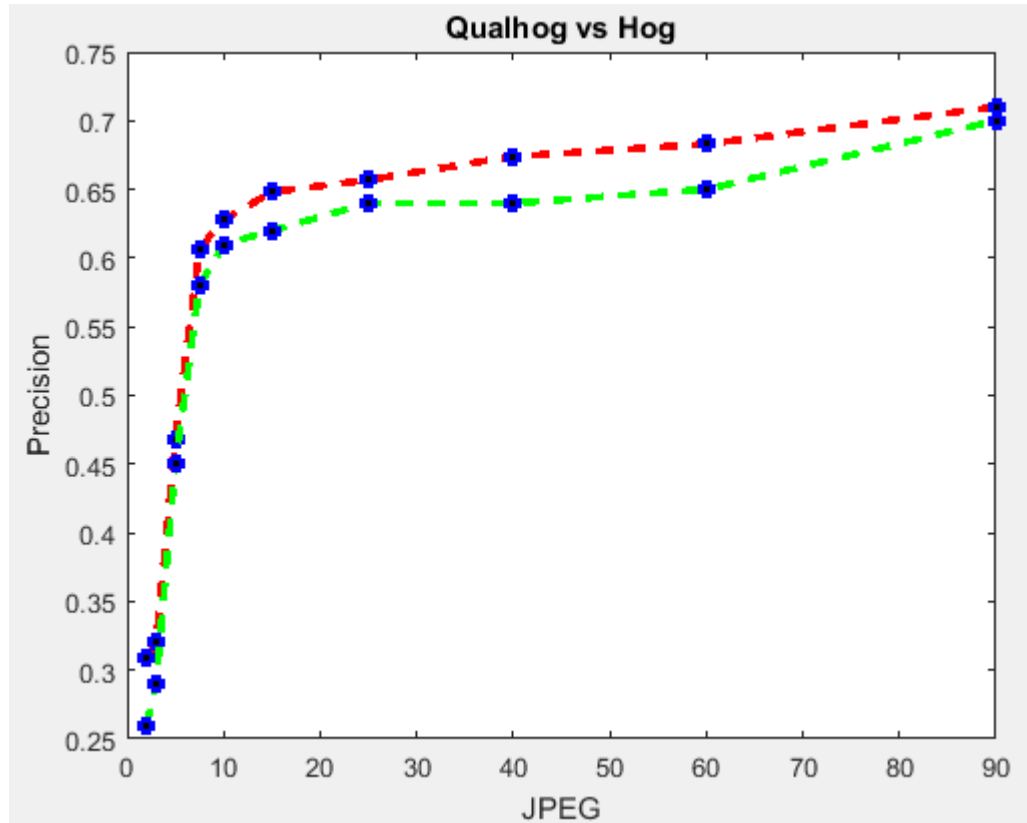


Figure 45: Precision vs JPEG curve for QUALHOG (red curve) and HOG (green curve) features

In Figure 45, red curve is for QUALHOG and green curve is for HOG. It shows that as the distortion ‘Q’ which mean Quality of an image (Quality decreased with increase in compression) increases, Precision increases. At Quality rate 2.0, Precision is 0.309 for QUALHOG and 0.26 for HOG and as the Quality rate increase to 90, Precision increases to 0.71 for QUALHOG and 0.70 for HOG which shows that QUALHOG (red curve) face detector performs better than HOG (green curve) face detector. This is done by the following code in Matlab in Appendix J. It is seen that QUALHOG Precision is 2-3% more accurate than HOG Precision for detecting faces in JPEG distorted images.

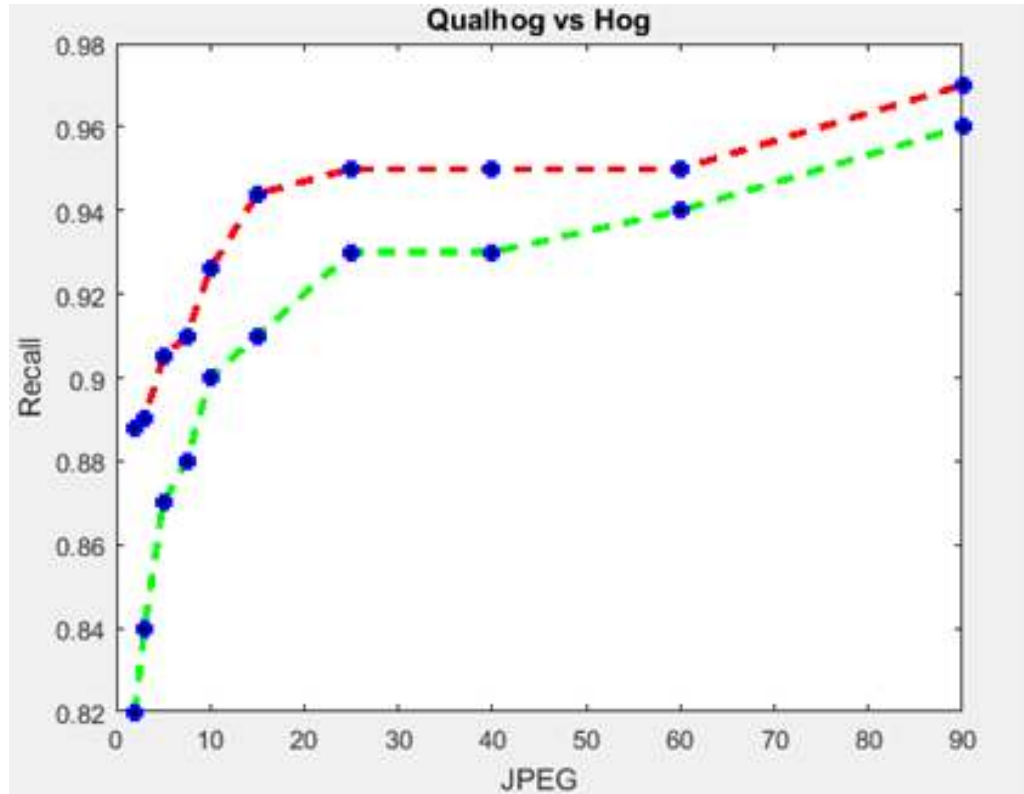


Figure 46: Recall vs JPEG curve for QUALHOG (red curve) and HOG (green curve) features

In Figure 46, red curve is for QUALHOG and green curve is for HOG. It shows that as the distortion 'Q' which mean Quality of an image (Quality decreased with increase in compression) increases, Recall increases. At Quality rate 2.0, Recall is 0.888 for QUALHOG and 0.82 for HOG and as the Quality rate increase to 90, Recall increases to 0.970 for QUALHOG and 0.96 for HOG which shows that QUALHOG (red curve) face detector performs better than HOG (green curve) face detector. This is done by the following code in Matlab in Appendix J. It is seen that QUALHOG Recall is 4-5% more accurate than HOG Recall for detecting faces in JPEG distorted images.

4.3 Comparison with known results

Known results use NIQE vs Distortion level and AUPR vs Distortion level curves (as shown is Figure (2 - 7) to show FDS based on QUALHOG are better than FDS based on HOG for distorted images. However, NIQE and AUPR are not properly defined in paper [1]. It is impossible to obtain the same results for NIQE and AUPR as shown in paper [1] due to the lack of defining NIQE and AUPR.

Hence, Precision vs distortion level (AWGN, GBlur and JPEG) and Recall vs distortion level (AWGN, GBlur and JPEG) curves are obtained for both QUALHOG and HOG to show FDS based on QUALHOG performs better than FDS based on HOG on distorted images.

As shown in Figure 5, For AWGN at distortion level 4.5×10^{-5} AUPR is around 0.96 and it declines with increases in distortion level. While looking at Figure 41 and 42, For AWGN at distortion level 4.5×10^{-5} Precision is 0.75 and Recall is around 0.95 and it declines with increase in distortion level for QUALHOG.

Similarly as shown in Figure 6, For GBlur at distortion level 0.4 AUPR is around 0.96 and it declines with increases in distortion level. While looking at Figure 43 and 44, For GBlur at distortion level 0.4 Precision is 0.76 and Recall is 0.96 and it declines with increase in distortion level for QUALHOG.

Similarly as shown in Figure 7, For JPEG at Quality level 90 AUPR is around 0.97 and it declines with increases in distortion level (decrease in quality level). While looking at Figure 45 and 46, For JPEG at quality level 90 Precision is 0.71 and Recall is 0.97 and it declines with increase in distortion level for QUALHOG.

It is seen that at less distortion level AUPR is more close to Precision and at higher distortion level, AUPR is more close to Recall.

4.4 Experiments on images of any size using sliding window technique

A small patch of 100 images are taken to test the extended method using SVM light. These images are downloaded from internet. True positive, true negative are calculated for QUALHOG and HOG and it is observed that QUALHOG method proves to be effective and works effectively in detecting faces in distorted images.

4.4.1 Training

SVM light is trained with scaled images (80x64) at different level of distortion.

4.4.2 Prediction

Images taken from different cameras with common distortion are taken from internet for testing.

4.4.3 Results of experiments

True positive and false positive are calculated for all three types of distortion at different levels for 100 images.

For images distorted by AWGN

Extracting QUALHOG features of images

Table 7: True positive and false positive are obtained for images distorted by AWGN at different log scale

AWGN distortion level	True positive	False positive
0.0001	94	6
0.02	87	13
0.36	75	25

This is done by the following code in Matlab in Appendix I.

Extracting HOG features of images

Table 8: True positive and false positive are obtained for images distorted by AWGN at different log scale

AWGN distortion level	True positive	False positive
0.0001	94	6
0.02	84	16
0.36	69	31

This is done by the following code in Matlab in Appendix I. It is shown from Table 7 & 8, that accuracy (True positives) for QUALHOG is higher than HOG in all three level of distortions.

For images distorted by G Blur

Extracting QUALHOG features of images

Table 9: True positive and false positive are obtained for images distorted by G Blur at different log scale

G Blur distortion level	True positive	False positive
1.0	95	5
6.0	84	16
20.0	71	29

This is done by the following code in Matlab in Appendix I.

Extracting HOG features of images

Table 10: True positive and false positive are obtained for images distorted by G Blur at different log scale

Gblur distortion level	True positive	False positive
1.0	93	7
6.0	79	21
20.0	73	27

This is done by the following code in Matlab in Appendix I. It is shown from Table 9 & 10, that accuracy (True positives) for QUALHOG is higher than HOG in all three level of distortions.

For images distorted by JPEG

Extracting QUALHOG features of images

Table 11: True positive and false positive are obtained for images distorted by JPEG at different log scale

JPEG distortion level	True positive	False positive
90	97	3
10	91	9
2	80	20

This is done by the following code in Matlab in Appendix I.

Extracting HOG features of images

Table 12: True positive and false positive are obtained for images distorted by JPEG at different log scale

JPEG distortion level	True positive	False positive
90	96	4
10	88	12
2	76	24

This is done by the following code in Matlab in Appendix I. It is shown from Table 11 & 12, that accuracy (True positives) for QUALHOG is higher than HOG in all three levels of distortions.

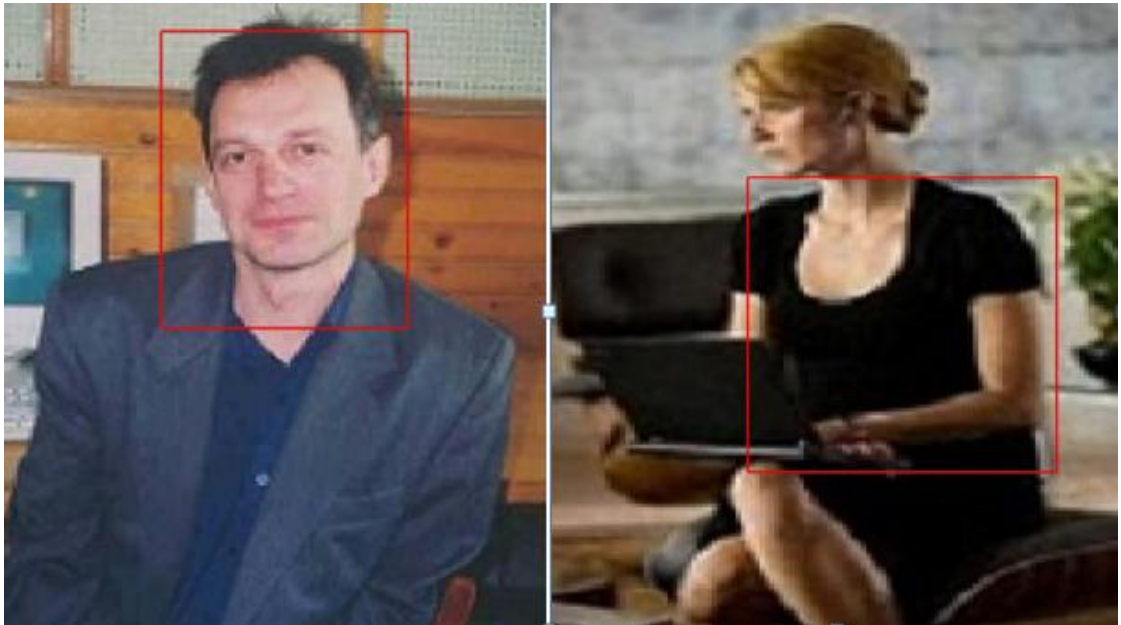


Figure 47: A tested example of true positive (left) and false positive (right) detection in two distorted images

Figure 47 is a result of FDS in which a box is created claiming face to be present inside the box by SVM.

4.5 Conclusion

Comparison of proposed face detector (QUALHOG) and face detector (HOG) is done by training and testing of images. 2731 images (1231 faces and 1500 non faces) are used for training and 17872 images (393 faces and 17479 non faces) are used for testing at 10 levels for all the three types of distortions. It showed that QUALHOG shows higher tolerance to distortion as compared to HOG. Later 100 random images are taken and comparison of proposed face detector (QUALHOG) and face detector (HOG) is done by training and testing of images. Images are distorted at 3 different levels for all three types of distortion.

LIBLINEAR is used for training and testing of data because of the large database for testing and training. It is observed that LIBLINEAR performs better and is faster as compared to other SVM while testing on large database. Later SVM Light is used for training and testing of 100 images.

True positive and false positive are calculated at every level in all three types of distortion and degradation in performance of QUALHOG face detector was seen with increase in distortion level. Precision and Recall are obtained by using true positives, false positives and detected positives numbers as defined paper [1]. Precision vs distortion level (AWGN, G Blur and JPEG) and Recall vs distortion level (AWGN, G Blur and JPEG) curves are plotted and these curves are compared with the known experimental results which are AUPR vs distortion level (AWGN, G Blur and JPEG) curves used by the author of article [1].

QUALHOG face detector shows improved results (1-5% overall) for face detection as compare to HOG face detector when trained on distorted images. Adding

perceptual quality aware features makes QUALHOG face detector more tolerant to images.

Chapter 5

CONCLUSION

In this research, articles related to implemented methodology (Face detection in distorted images augmented by perceptual quality aware features) are discussed. Metrics and methods used in them such as HOG, NSS, Liblinear and SVM Light are explained. Problems to be solved in this thesis are defined. Design of FDS along with implementation and testing of FDS and its subsystems like HOG, NSS, Liblinear SVM and SVM Light are discussed and explained in detail.

For testing FDS, an image from the database is taken and testing is done to show and prove the authenticity of FDS and its subsystems. LIBLINEAR support vector machine is used for training and testing of data. Later sliding window technique is applied to detect face on any size of image by extracting QUALHOG features and using SVM Light. Results are obtained at every step of testing for extraction of QUALHOG features for FDS.

Comparison of proposed face detector (QUALHOG) and face detector (HOG) is done by training and testing of images. 2731 images (1231 faces and 1500 non faces) are used for training and 17872 images (393 faces and 17479 non faces) are used for testing at 10 levels for all the three types of distortions. The results showed that QUALHOG shows higher tolerance to distortion as compared to HOG. Later, 100 random images are taken and comparison of proposed face detector (QUALHOG)

and face detector (HOG) is done by training and testing of images. Images are distorted at 3 different levels for all three types of distortion.

LIBLINEAR is used for training and testing of data because of the large database for testing and training. It is observed that LIBLINEAR performs better and is faster as compared to other SVM while testing on large database. Later SVM Light is used for training and testing of 100 images.

True positive and false positive are calculated at every level in all three types of distortion and degradation in performance of QUALHOG face detector was seen with increase in distortion level. Precision and Recall are obtained by using true positives, false positives and detected positives. Precision versus distortion level (AWGN, G Blur and JPEG) and Recall versus distortion level (AWGN, G Blur and JPEG) curves are plotted and these curves are compared with the known experiment results which are AUPR versus distortion level (AWGN, G Blur and JPEG) curves.

QUALHOG face detector shows improved results (1-5% overall) for face detection as compare to HOG face detector when trained on distorted images. Adding perceptual quality aware features makes QUALHOG face detector more tolerant to images.

REFERENCES

- [1] Gunasekar, S., Ghosh, J., & Bovik, A. C. (2014). Face detection on distorted Images augmented by augmented by perceptual quality - aware features. *IEEE Transactions on Information Forensics and Security*, 9(12), 2119-2131.
- [2] Mittal, A., Moorthy, A. K., & Bovik, A. C. (2012). No-reference image quality assessment in the spatial domain. *IEEE Transactions on Image Processing*, 21(12), 4695-4708.
- [3] Sharifi, K., & Leon -Garia, A. (1995). Estimation of shape parameter generalized gaussian distribution in subband decomposition of video. *IEEE Transaction on Circuits and Systems for Video Techonology*, 5(1), 52-56.
- [4] Lasmar, N.E., Stitou, Y., & Berthoumieu, Y. (2009, November). Multiscale skewed heavy tailed model for texture analysis. In *Image Processing (ICIP) 2009 16th IEEE International Conference on* (pp. 2281-2284). IEEE.
- [5] Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on* (Vol. 1, pp. 886-893). IEEE.
- [6] Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., & Lin, C. J. (2008). LIBLINEAR: A library for large linear classification. *Journal of machine learning research*, 9(Aug), 1871-1874.

- [7] Yang, M., Crenshaw, J., Augustine, B., Mareachen, R., & Wu, Y. (2010). AdaBoost-based face detection for embedded systems. *Computer Vision and Image Understanding*, 114(11), 1116-1125.
- [8] Sandford, A., & Burton, A. M. (2014). Tolerance for distorted faces: Challenges to a configural processing account of familiar face recognition. *Cognition*, 132(3), 262-268.
- [9] Andreu, Y., García-Sevilla, P., & Mollineda, R. A. (2014). Face gender classification: A statistical study when neutral and distorted faces are combined for training and testing purposes. *Image and Vision Computing*, 32(1), 27-36.
- [10] Gudla, Gudla, B., Chalamala, S. R., & Jami, S. K. (2015, December). Local Binary Patterns For Gender Classification. In *Artificial Intelligence, Modelling and Simulation (AIMS), 2015 3rd International Conference on* (pp. 19-22). IEEE.
- [11] Rowley, H. A., Baluja, S., & Kanade, T. (1998). Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1), 23-38.
- [12] Mittal, A., Soundararajan, R., & Bovik, A. C. (2013). Making a “completely blind” image quality analyzer. *IEEE Signal Processing Letters*, 20(3), 209-212.
- [13] SVM Light. (2016, November 20). Retrieved from <http://svmlight.joachims.org/>

- [14] DFD. (2016, November 15). Retrieved from [www.live.ece.utexas.edu/ research / Quality / index.htm](http://www.live.ece.utexas.edu/research/Quality/index.htm).

APPENDICES

Appendix A: Code for training images using Liblinear SVM

```
1. close all
2. clear all
3. clc
4. hog.numBins = 9;
% The number of cells horizontally and vertically.
5. hog.numHorizCells = 8;
6. hog.numVertCells = 10;
% Cell size in pixels (the cells are square).
7. hog.cellSize = 8;
% Compute the expected window size (with 1 pixel border on all sides).
8. hog.winSize = [(hog.numVertCells * hog.cellSize + 2), (hog.numHorizCells *
hog.cellSize + 2)];
% Load all training windows and get their HOG descriptors.
% Get the list of all images in the directory.
9. posFiles = getImagesInDir('./dataset/noise/V9/pos/', true);
10. negFiles = getImagesInDir('./dataset/noise/V9/negS/', true);

% Create the category labels.
11. y_train = [ones(length(posFiles), 1); -ones(length(negFiles), 1)];
%y_train = double(y_train);
% Combine the file lists to get a list of all training images.
12. fileList = [posFiles, negFiles];

% Build a matrix of all of the descriptors, one per row.
13. X_train = zeros(length(fileList), 2304);
%X_train = double(X_train);
14. fprintf('Computing descriptors for %d training windows: ', length(fileList));

% For all training window images...
15. for i = 1 : length(fileList)

    % Get the next filename.
16. imgFile = char(fileList(i));

    % Print the current iteration
17. printIteration(i);

    % Load the image into a matrix.
18. img = imread(imgFile);
19. img = double(img);
20. if(size(img,3)==3)
21. img = uint8(img);
22. img = rgb2gray(img);
23. end
24. img = double(img);
```

```

% Calculate the HOG descriptor for the window.
25. H = HOG(img);
26. H = H';

27. H = normalise(H);
28. M = brisque_feature(img);
29. M = (36/2268)* M;
30. M = normalise(M);
31. L = [H M];

32. L = normalise(L);
% Add the descriptor to the rest.
33. X_train(i, :) = L';
34. end
35. X_train = sparse(X_train);
36. libsvmwrite('data.txt', y_train, X_train) ;
37. [label_vector, instance_matrix] = libsvmread('data.txt');
%Train Liblinear SVM
38. fprintf('\nTraining linear SVM classifier...\n');
39. model = train(label_vector, instance_matrix, 's2 -c 0.25000 ');
40. save('model.mat');
41.end

```

Appendix B: Code for testing images using Liblinear SVM

```

1. close all
2. clear all
3. clc
4. load('model.mat');
5. fcount = 1;
6. testImPath = './test images/noise/V9/V9/pos/';
7. imlist = dir([testImPath '*.bmp']);
8. fprintf('Computing descriptors for %d training windows: ', length(imlist));
9. for j = 1:length(imlist)
% Get the next filename.
10. imgFile = struct(imlist(j));
% Print the current iteration
11. printIteration(j);
12. img = imread([testImPathimlist(j).name]);
13. img = double(img);
14. if(size(img,3)==3)
15. img = uint8(img);
16. img = rgb2gray(img);
17. end
18. img = double(img);
19. M = HOG(img);
20. M = M';

```

```

21. M = normalise( M );
22. T = brisque_feature(img);
23. T =(36/2268)* T;
24. T = normalise( T );
25. featureVector{fcount} = [M T];

26. featureVector{fcount} = normalise(featureVector{fcount});
27. featureVector{fcount} = featureVector{fcount}';
28. fcount = fcount+1;
29. end
30. P = cell2mat(featureVector);
31. P = P';
32. lebel = ones(length(featureVector),1);
33. X = sparse(P);
34. libsvmwrite('data1.txt', lebel,X) ;
35. [label_vector, instance_matrix] = libsvmread('data1.txt');

% Evaluate the liblinear SVM on the descriptor.
36. [predict_label, accuracy, dec_values] = predict(label_vector,
instance_matrix,model);
37. end

```

Appendix C: Code for calculating NSS features

```

1. function feat = brisque_feature(imdist)
2. imdist = double(imdist);
3. if(size(imdist,3)==3)
4. imdist = uint8(imdist);
5. imdist = rgb2gray(imdist);
6. end

7. imdist = double(imdist);
8. scalenum = 2;
9. window = fspecial('gaussian',7,7/6);
10. window = window/sum(sum(window));

11. feat = [];
12. for itr_scale = 1:scalenum

13. mu = filter2(window, imdist, 'same');
14. mu_sq = mu.*mu;
15. sigma = sqrt(abs(filter2(window, imdist.*imdist, 'same') - mu_sq));
16. structdis = (imdist-mu)./(sigma+1);
17. [alpha overallstd] = estimateggdparam(structdis(:));
18. feat = [feat alpha overallstd^2];

19. shifts = [ 0 1;1 0 ; 1 1; -1 1];

```

```

20. for itr_shift =1:4
21. shifted_structdis = circshift(structdis,shifts(itr_shift,:));
22. pair = structdis(:).*shifted_structdis(:);
23. [alpha leftstdrightstd] = estimateaggdparam(pair);
24. const =(sqrt(gamma(1/alpha))/sqrt(gamma(3/alpha)));
25. meanparam
    =(rightstdleftstd)*(gamma(2/alpha)/gamma(1/alpha))*const;
26. feat =[feat alpha meanparam leftstd^2 rightstd^2];
27. end
28. imdist = imresize(imdist,0.5);
29. end

```

Appendix D: Code for calculating GGD parameters

```

1. function [gamparam sigma] = estimateggdparam(vec)
2. mu = mean(vec);
3. sigma_sq = mean((vec - mu).^2)
4. sigma = sqrt(sigma_sq);
5. E = mean(abs(vec - mu));
6. rho = sigma_sq/E^2;

7. counter_1 = 0;
8. counter_2 = 10 ;
9. for v = 1:100
10. gam = counter_1:0.001:counter_2;
11. r_gam = (gamma(1./gam).*gamma(3./gam))./((gamma(2./gam)).^2);
12. [min_difference, array_position] = min(abs(rho - r_gam));
13. previous_array_position = array_position - 1;
14. disp(array_position - 1)
15. disp(array_position)
16. if min_difference == min(abs(rho - r_gam));
17. gamparam = gam(array_position);
18. fprintf('shape_parameter_found');
19. else
20. fprintf('shape_parameter_not_found');
21. end
22. if(gam >= rho)
23. break;
24. end
25. counter_1 = counter_1 - 1;
26. counter_2 = counter_2 + 1;
27. end
28. end

```

Appendix E: Code for calculating AGGD parameters

```
1. function [alpha leftstdrightstd] = estimateaggdparam(vec)
2. leftstd      = sqrt(mean((vec(vec<0)).^2));
3. rightstd     = sqrt(mean((vec(vec>0)).^2));
4. gammahat     = leftstd/rightstd;
5. rhat         = (mean(abs(vec)).^2)/mean((vec).^2);
6. rhatnorm     = (rhat*(gammahat^3 + 1)*(gammahat+1))/((gammahat^2
   + 1)^2);

7. counter_1 = 0.2;
8. counter_2 = 10 ;
9. for v = 1:100

10. gam      = counter_1:0.001:counter_2;
11. r_alpha  = ((gamma(2./gam)).^2)./(gamma(1./gam).*gamma(3./gam));

12. [min_difference, array_position] = min((r_alpha - rhatnorm).^2);
13. previous_array_position = array_position - 1;
14. disp(array_position - 1)
15. disp(array_position)
16. if min_difference == min((r_alpha - rhatnorm).^2);

17. alpha = gam(array_position);
18. fprintf('array_value_found');
19. else
20. fprintf('array_position_not_found');
21. end
22. if(gam >= rhatnorm)
23. break;
24. end
25. counter_1 = counter_1 - 1;
26. counter_2 = counter_2 + 1;
27. end
28. end
```

Appendix F: Code for extracting HOG features

```
1. function HOGv = HOG(Img)
2. I = double(Img);
3. if(size(I,3)==3)
4. I = uint8(I);
5. I = rgb2gray(I);
6. end
7. I = double(I);
```

```

% Compute the gradient vector at every pixel in the image.
% Create the operators for computing image derivative at every pixel.
8.  hx = [-1,0,1];
9.  hy = hx';
% Compute the derivative in the x and y direction for every pixel.
10. dx = filter2(hx, double(I));
11. dy = filter2(hy, double(I));
% Convert the gradient vectors to polar coordinates (angle and magnitude).
12. angles = atan2(dy, dx);
13. magnit = ((dy.^2) + (dx.^2)).^0.5;

% Make the angles unsigned by adding (180 degrees) to all negative angles.
14. angles(angles < 0) = angles(angles < 0) + 180;
15. feature=[]; %initialized the feature vector
16. rows=size(I,1);
17. cols=size(I,2);
% Iterations for Blocks
18. for i = 0: rows/8 - 2
19.  for j= 0: cols/8 - 2
20.   mag_patch = magnit(8*i+1 : 8*i+16 , 8*j+1 : 8*j+16);
21.   ang_patch = angles(8*i+1 : 8*i+16 , 8*j+1 : 8*j+16);
22.   block_feature=[];

      %Iterations for cells in a block
23.   for x= 0:1
24.    for y= 0:1
25.     angleA =angles(8*x+1:8*x+8, 8*y+1:8*y+8);
26.     magA  =magnit(8*x+1:8*x+8, 8*y+1:8*y+8);
27.     histr =zeros(1,9);

        %Iterations for pixels in one cell
28.     for p=1:8
29.      for q=1:8
30.       alpha= angleA(p,q);

          % Binning Process (Bi-Linear Interpolation)
31.       if alpha>=0 && alpha<=20
32.        histr(1)=histr(1)+ magA(p,q);
33.       elseif alpha>20 && alpha<=40
34.        histr(2)=histr(2)+ magA(p,q);
35.       elseif alpha>40 && alpha<=60
36.        histr(3)=histr(3)+ magA(p,q);
37.       elseif alpha>60 && alpha<=80
38.        histr(4)=histr(4)+ magA(p,q);
39.       elseif alpha>80 && alpha<=100
40.        histr(5)=histr(5)+ magA(p,q);
41.       elseif alpha>100 && alpha<=120
42.        histr(6)=histr(6)+ magA(p,q);
43.       elseif alpha>120 && alpha<=140
44.        histr(7)=histr(7)+ magA(p,q);
45.       elseif alpha>140 && alpha<=160
46.        histr(8)=histr(8)+ magA(p,q);
47.       elseif alpha>160 && alpha<=180
48.        histr(9)=histr(9)+ magA(p,q);
49.       else
50.        histr(10)=histr(10)+ magA(p,q);
51.       end
52.     end
53.   end
54. end
55. feature=[feature; block_feature];
56. end

```

```

44. histr(7)=histr(7)+ magA(p,q);
45. elseif alpha>120 && alpha<=140
46. histr(8)=histr(8)+ magA(p,q);
47. elseif alpha>140 && alpha<=160
48. histr(9)=histr(9)+ magA(p,q);
49. elseif alpha>=160 && alpha<=180
50. histr(9)=histr(9)+ magA(p,q);
51.     end
52.     end
53. end
    % Concatenation of Four histograms to form one block feature
54. block_feature=[block_featurehistr];
55. end
56. end
    % Normalize the values in the block using L2-Norm
57. e = 0.0001;
58. block_f=block_feature/sqrt(norm(norm(block_feature))^2+.01);
    %Normalize the values in the block using L1-Norm
59. L_1 = block_f/( norm(block_f)+ 0.0001);
% Normalize the values in the block using L1 sqrt-Norm
60. L1_sqrt = sqrt (L_1/ ( norm(L_1)+ 0.0001));
61. feature=[feature L1_sqrt]; %Features concatenation
62. end
63. end

```

Appendix G: Code for normalize images features between -1 and +1

```

1. function [ H_norm ] = normalise( x )
2. fmin = -1;
3. fmax = 1;
4. minn = min(x);
5. maxx = max(x);
6. H_norm = (x - minn) ./ (maxx - minn);
7. H_normmm = H_norm .* (fmax - fmin) + fmin;
8. end

```

Appendix H: Code for training of images in test images using SVM

light

```

1. close all
2. clear all
3. clc

```

```

4. hog.numBins = 9;
% The number of cells horizontally and vertically.
5. hog.numHorizCells = 8;
6. hog.numVertCells = 10;
% Cell size in pixels (the cells are square).
7. hog.cellSize = 8;
% Compute the expected window size (with 1 pixel border on all sides).
8. hog.winSize = [(hog.numVertCells * hog.cellSize + 2), (hog.numHorizCells *
hog.cellSize + 2)];
% Load all training windows and get their HOG descriptors.
% Get the list of all images in the directory.
9. posFiles = getImagesInDir('./dataset/noise/V9/pos/', true);
10. negFiles = getImagesInDir('./dataset/noise/V9/negS/', true);

% Create the category labels.
11. y_train = [ones(length(posFiles), 1); -ones(length(negFiles), 1)];
%y_train = double(y_train);
% Combine the file lists to get a list of all training images.
12. fileList = [posFiles, negFiles];

% Build a matrix of all of the descriptors, one per row.
13. X_train = zeros(length(fileList), 2304);
%X_train = double(X_train);
14. fprintf('Computing descriptors for %d training windows: ', length(fileList));

% For all training window images...
15. for i = 1 : length(fileList)

    % Get the next filename.
16. imgFile = char(fileList(i));

    % Print the current iteration
17. printIteration(i);

    % Load the image into a matrix.
18. img = imread(imgFile);
19. img = double(img);
20. if(size(img,3)==3)
21. img = uint8(img);
22. img = rgb2gray(img);
23. end
24. img = double(img);
    % Calculate the HOG descriptor for the window.
25. H = HOG(img);
26. H = H';

27. H = normalise(H);
28. M = brisque_feature(img);
29. M = (36/2268)* M;
30. M = normalise(M);

```

```

31. L = [H M];

32. L = normalise(L);
% Add the descriptor to the rest.
33. X_train(i, :) = L';
34. end
35. X_train = sparse(X_train);
36. libsvmwrite('data.txt', y_train, X_train) ;
37. [label_vector, instance_matrix] = libsvmread('data.txt');
%Train SVM light
37. fprintf('\nTraining linear SVM classifier...\n');
38. model = svmlearn(X_train, y_train, '-t -c ');
39. save('model.mat');
40.end

```

Appendix I: Code for testing images using SVM light through sliding window

```

1. clear all
2. close all
3. clc
%% Detection
4. load ('model.mat');
5. tSize = [64, 80];
6. testImPath = './test images/';
7. imlist = dir([testImPath '*.bmp']);
8. for j = 1:length(imlist)
9. img = imread([testImPathimlist(j).name]);
10. img = imresize(img,0.8);
11. imshow(img,[]);
12. img = double(img);
13. if(size(img,3)==3)
14. img = uint8(img);
15. img = rgb2gray(img);
16. end
17. img = double(img);
18. axis equal; axis tight; axis off;
19. hold on;
20. detect(img,model,tSize);
21. saveas(gcf, ['./results (2)/' imlist(j).name], 'jpg');
22. end

23. function detect(im,model,wSize)
24. topLeftRow = 1;
25. topLeftCol = 1;

```

```

26. [bottomRightColbottomRightRow d] = size(im);
27. fcount = 1;
% this for loop scan the entire image and extract features for each sliding window
28. for y = topLeftCol:bottomRightCol-wSize(2)
29.   for x = topLeftRow:bottomRightRow-wSize(1)
30.     p1 = [x,y];
31.     p2 = [x+(wSize(1)-1), y+(wSize(2)-1)];
32.     po = [p1; p2];
33.     img = imcut(po,im);
34.     if size(img,3) > 1
35.       img = rgb2gray(img);
36.     end
37.     H = HOG(double(img));
38.     H = normalise(H);
39.     M = brisque_feature(img);
40.     M = normalise(M);
41.     M = (2268/36)* M;
42.     M = M';
43.     featureVector{fcount} = vertcat(H,M);
44.     boxPoint{fcount} = [x,y];
45.     fcount = fcount+1;
46.   x = x+1;
47. end
48. end
49. lebel = ones(length(featureVector),1);
50. P = cell2mat(featureVector);
51. P = P';
52. [~, predictions] = svmclassify(P,lebel,model); % classifying each window
53. [a,indx]= max(predictions);
54. bBox = cell2mat(boxPoint(indx));
55. rectangle('Position',[bBox(1),bBox(2),64,80],'LineWidth',1, 'EdgeColor','r');
56. end

```

Appendix J: Code for generating curve QUALHOGvs HOG

```

1. (x1,y1) = Qualhog features;
2. (x2,y2) = HOG features;
3. figure,plot(x1,y1,'--rs',x2,y2,'--gs',...
4.   'LineWidth',2,...
5.   'MarkerSize',5,...
6.   'MarkerEdgeColor','b',...
7.   'MarkerFaceColor',[0.0001,0.0001,0.0001]);
8. xlabel('xlabel');
9. ylabel('ylabel');
10. title('Qualhogvs Hog');

```

Appendix K: Code for generating Gaussian filter without using

Matlab function

```
1. clear all
2. close all
3. clc
   % Read an image
4. Img = imread('image_0057.bmp');
5. I = double(Img);
6. If (size(I,3)==3)
7. I = uint8(I);
8. I = rgb2gray(I);
9. End
10. I = double (I);
   % Design the Gaussian kernel
11. Sigma = 1; %standard deviation
12. Sz = 3*sigma; %Window size
13. [x,y] = meshgrid(-sz:sz, -sz:sz);
14. M = size(x,1) -1; % Find the size of each dimension of matrix M
15. N = M;
16. Exp_comp = -(x.^2+y.^2) / (2*sigma*sigma);
17. Kernel = exp (Exp_comp) / (2*pi*sigma*sigma);
18. end
```

Appendix L: Manually created table for saving experiment results

40												
41	awgn	TP	FN	FP	TN	DP	PRECISION	RECALL				
42	v1	371		142								
43	v2	367		148								
44	v3	362		154								
45	v4	359		192								
46	v5	353		286								
47	v6	342		589								
48	v7	330		1330								
49	v8	311		2307								
50	v9	301		3096								
51	v10	296		3805								
52												
53	g blur	TP	TN	FP	DP	PRECISION	RECALL					
54	v1	365		146								
55	v2	362		164								
56	v3	358		402								
57	v4	357		587								
58	v5	354		643								
59	v6	351		805								
60	v7	346		1247								
61	v8	324		1946								
62	v9	302		2895								
63	v10	266		4134								

27	jpeg	TP	FN	FP	TN	DP	PRECISION	RECALL				
28	v1	381		155								
29	v2	373		173								
30	v3	373		180								
31	v4	373		194								
32	v5	371		201								
33	v6	364		216								
34	v7	358		231								
35	v8	356		405								
36	v9	350		746								
37	v10	348		779								
38												
39	For HOG											
40												
41	awgn	TP	FN	FP	TN	DP	PRECISION	RECALL				
42	v1	371		142								
43	v2	367		148								
44	v3	362		154								
45	v4	359		192								
46	v5	353		286								
47	v6	342		589								
48	v7	330		1330								
49	v8	311		2307								
50	v9	301		3096								
51	v10	296		3805								

53	g blur	TP	TN	FP	DP	PRECISION	RECALL			
54	v1	365		146						
55	v2	362		164						
56	v3	358		402						
57	v4	357		587						
58	v5	354		643						
59	v6	351		805						
60	v7	346		1247						
61	v8	324		1946						
62	v9	302		2895						
63	v10	266		4134						
64										
65	jpeg	TP	TN	FP	DP	PRECISION	RECALL			
66	v1	381		158						
67	v2	370		197						
68	v3	368		199						
69	v4	366		205						
70	v5	359		215						
71	v6	353		223						
72	v7	348		244						
73	v8	344		417						
74	v9	334		805						
75	v10	328		846						