

Empirical Evaluation of AdaBoost Method in  
Detecting Transparent and Occluded Objects

by

Sujan Tamang

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of

Computing and Information Systems

YOUNGSTOWN STATE UNIVERSITY

May, 2018

Empirical Evaluation of AdaBoost Method in  
Detecting Transparent and Occluded Objects

Sujan Tamang

I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature:

---

*Sujan Tamang*, Student

Date

Approvals:

---

*Dr. Yong Zhang*, Thesis Advisor

Date

---

*Dr. John Sullins*, Committee Member

Date

---

*Dr. Feng George Yu*, Committee Member

Date

---

Dr. Salvatore A. Sanders, Dean of Graduate Studies

Date

## DEDICATION

I would dedicate this thesis to my parents and sisters for their unconditional love and all the pain and sufferings that they willingly endured to secure higher education for me.

## ABSTRACT

Detecting and counting nano-particles in the Transmission Electron Microscopy (TEM) images is a challenging task due to two reasons: (1) The particles are semi-transparent which means that the backgrounds and objects have similar image characteristics. As a result, it is extremely difficult to separate the positive samples and the negative samples with a single or simple image feature; (2) Particles are often severely occluded (overlapped) and hence it is impossible to select a large number of clean positive samples to train a good classifier, which in turn significantly affects the detection outcomes. In this thesis, a series of empirical experiments and data analysis were conducted to compare the performances of two popular image features: Haar feature and Local Binary Pattern (LBP), within the framework of Cascade AdaBoost algorithm. It was found that the two features exhibited complex relationships with respect to several key training parameters and performance metrics, including the training time, sample size, true positive rate, false alarm rate and detection window size, etc. The experimental results and insights gained from this study help build a solid foundation upon which more detailed investigations can be carried out in the future.

## **ACKNOWLEDGEMENTS**

I would like to thank Dr. Yong Zhang for giving me the opportunity to work on this research project and providing tremendous advisements and insights as how to approach and solve a challenging problem. He has been a huge resource for the overall experimental design, testing and thesis writeup.

I would also like to thank the thesis committee members, Dr. John Sullins and Dr. Feng George Yu, for their encouragement throughout my academic coursework at YSU and sparing their invaluable time to serve in the committee.

I am forever indebted to my friends whose continuous guidance and support were instrumental in the completion of this work.

## TABLE OF CONTENTS

ABSTRACT.....	iv
ACKNOWLEDGEMENTS.....	v
LIST OF FIGURES .....	viii
LIST OF TABLES.....	x
CHAPTER 1 INTRODUCTION .....	1
1.1.    TECHNICAL ISSUES .....	1
1.2.    CONTRIBUTIONS .....	2
CHAPTER 2 METHODOLOGY .....	3
2.1.    CASCADE ADABOOST ALGORITHM.....	3
2.2.    FEATURE EXTRACTION.....	5
2.2.1. HAAR FEATURE .....	6
2.2.2. LOCAL BINARY PATTERN.....	8
CHAPTER 3 DATA PROCESSING.....	10
3.1.    DATA SET .....	10
3.2.    SAMPLE PREPARATION .....	11
3.2.1. POSITIVE SAMPLES.....	11
3.2.2. NEGATIVE SAMPLES .....	13
CHAPTER 4 EXPERIMENT DESIGN .....	14
4.1. TRAINING .....	14

STEP-1: GENERATE A LIST OF NEGATIVE SAMPLES: .....	14
STEP-2: GENERATE A LIST OF POSITIVE SAMPLES: .....	15
STEP-3: MERGE THE VEC FILES OF INDIVIDUAL SAMPLES INTO A SINGLE ONE: .....	15
STEP-4: TRAIN A CASCADE CLASSIFIER: .....	15
4.2. TESTING .....	16
CHAPTER 5 RESULTS AND DISCUSSIONS .....	17
5.1. IMPACT OF DETECTION WINDOW SIZE (CUBE-A AS TRAINING) .....	17
5.2. IMPACT OF SAMPLE SIZE (CUBE-A AS TRAINING) .....	20
5.3. IMPACT OF SAMPLE SIZE (CUBE-C AS TRAINING) .....	24
5.4. IMPACT OF ROTATED SAMPLE SETS (CUBE-A AS TRAINING) .....	26
5.5. IMPACT OF KEY TRAINING PARAMETER (CUBE-A AS TRAINING) .....	28
5.6. VISUALIZATION OF SELECTED HAAR FEATURES .....	30
5.7. VISUALIZATION OF SELECTED LBP FEATURES .....	33
CHAPTER 6 CONCLUSIONS .....	36
REFERENCES .....	37

## LIST OF FIGURES

Figure 1: The cascade of classifiers with $N$ stages. ....	3
Figure 2: Illustration of the AdaBoost Algorithm.....	4
Figure 3: Illustration of the Cascade AdaBoost Algorithm. ....	5
Figure 4: Basic Haar features.....	6
Figure 5: The value of the integral image at point $(x,y)$ .....	7
Figure 6: Rectangular feature computation using integral image.....	7
Figure 7: The basic LBP operator. ....	8
Figure 8: The 9x9 MB-LBP operator.....	9
Figure 9: Cube-a image used for selecting training samples. ....	10
Figure 10: Cube-b image used for testing.....	10
Figure 11: Cube-c image used for testing.....	11
Figure 12: Positive samples cropped from the "cube-a" image.....	12
Figure 13: Negative samples.....	13
Figure 14: Impact of window size on training time.....	18
Figure 15: Impact of window size on TPR.....	18
Figure 16: Impact of window size on FPR.....	19
Figure 17: Impact of window size on precision.....	19
Figure 18: Detection results with different window sizes. ....	20
Figure 19: Impact of sample size on training time. ....	21
Figure 20: Impact of sample size on TPR.....	22
Figure 21: Impact of sample size on FPR.....	22
Figure 24: Impact of sample size on training time (cube-c). ....	24



Figure 25: Impact of sample size on TPR (cube-c).....	25
Figure 26: Impact of sample size on FPR (cube-c).....	25
Figure 27: Impact of sample size on precision (cube-c).....	25
Figure 28: Impact of rotated sample sets (20) on TPR.....	26
Figure 29: Impact of rotated sample sets (20) on FPR.....	26
Figure 30: Impact of rotated sample sets (20) on precision.....	27
Figure 31: Impact of rotated sample sets (60) on TPR.....	27
Figure 32: Impact of rotated sample sets (60) on FPR.....	27
Figure 33: Impact of rotated sample sets (60) on precision.....	28
Figure 34: Impact of a key training parameter.....	29
Figure 35: Training parameters for a classifier of good performance.....	30
Figure 36: Training parameters for a classifier of poor performance.....	30
Figure 37: A classifier of good performance and its selected Haar features.....	31
Figure 38: A classifier of poor performance and its selected Haar features.....	32
Figure 39: Training parameters for a classifier of good performance (LBP).....	33
Figure 40: Training parameters for a classifier of poor performance (LBP).....	33
Figure 41: A classifier of good performance and its selected LBP features.....	34
Figure 42: A classifier of poor performance and its selected LBP features.....	35

## LIST OF TABLES

Table 1: Sets of Positive Samples Cropped From Cube-A.....	13
Table 2: Impact of Detection Window Size (Cube-A) .....	18
Table 3: Impact of Sample Size (Cube-A).....	21
Table 4: Impact of Sample Size (Cube-C).....	24

# CHAPTER 1

## INTRODUCTION

With the rapid increase of using TEM imaging technology in both fundamental and applied nano-research, a large amount of image data measured in Gigabyte and Terabyte scale has been generated annually. It is not feasible to process this type of information manually for further investigations. For example, the analysis of complex relationships among object properties (size, shape, area, growth rate, chemical activities) often requires accurate accounting of individual crystals. However, detecting nanoparticles in TEM images is very challenging because of the presence of object occlusion and transparency.

In this study, the Cascade AdaBoost method is used to detect nanoparticle in cube shapes. AdaBoost is a robust ensemble learning algorithm that has a good generalization capability and has been widely used in object detections, especially in face detection [1]. The majority of object detection applications used the Haar feature as a weak learner, mainly because of its simplicity in terms of the feature structure, calculation cost, and parameter fine tuning.

### 1.1. TECHNICAL ISSUES

To achieve a satisfactory performance, a classifier must address two technical issues:

1. The objects are severely overlapped and causes class labeling errors, which in turn complicates the relationships of training and detection parameters.
2. A simple feature is easy to use but a large number of them is needed in training, while a sophisticated feature can reduce the training time significantly but at a cost of less accurate detection rate.

## 1.2. CONTRIBUTIONS

This study focuses on the empirical evaluation of the Cascade AdaBoost method with respect to the performances of two popular image features: Haar and LBP in the context of detecting overlapped nanoparticles. The primary contributions are: (i) Determining the impacts of the kernel window size and sample size on the training time, true positive rate, false positive rate and precision rate with various sample sets; (ii) Understanding whether the use of rotated sample subsets and the change of key training parameter would have a significant impacts on detection results; (iii) Comparing the performances of two image features by the standard metrics under the same or similar experimental setups and running environments; (iv) Visualizing and assessing the possible connections among the selected feature sets in each cascade stage and the overall performances of the classifiers of different weak learners.

## CHAPTER 2

### METHODOLOGY

#### 2.1. CASCADE ADABOOST ALGORITHM

Boosting is an ensemble learning strategy that combines weak classifiers of slightly better than random guessing rates [2,3] into a strong classifier. Since weak classifiers only need to be above-average ones, they are often simple to implement and easy to use. AdaBoost is a boosting algorithm that shows resistance to over-fitting as often observed in other methods [4, 5, 6, 7]. As a practical method, AdaBoost is considered one of the most successful learning algorithms [8, 9]. The original AdaBoost can be modified to solve a specific problem. For example, the Cascade AdaBoost is designed to detect faces in real-time by rejecting non-face patterns early (Fig. 1). The concise descriptions of the original AdaBoost and Cascade AdaBoost algorithms are given in Fig. 2 and Fig. 3.

At each stage, a classifier is trained to achieve a hit rate ( $h$ ) and a false alarm rate ( $f$ ). The overall hit rate and false alarm rate can be expected to be the  $N^{th}$  power of  $h$  and  $f$ . Given a classifier of 15 stages, with each state eliminating 60% negative samples while falsely discarding 0.1% true objects, it would have the overall false alarm rate of about  $0.4^{15} \approx 1.07e-06$  and hit rate of about  $0.999^{15} \approx 0.985$ .

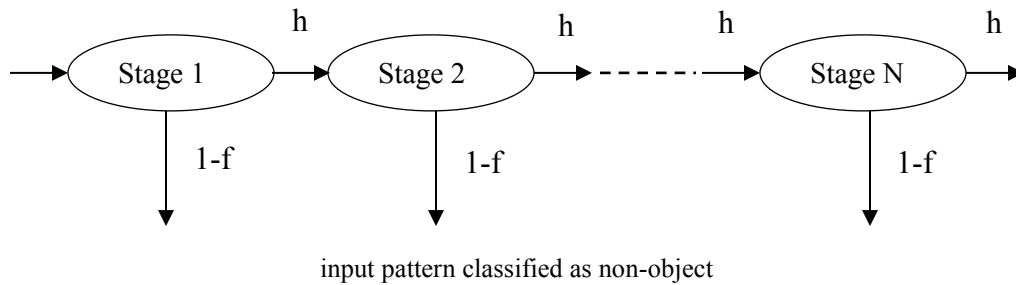


Figure 1: The cascade of classifiers with  $N$  stages.

Terminologies:

- hypothesis = learner = classifier
- weak learner: better than 0.5 prediction rate.
- strong learner: a weighted linear combination of weak learners.
- $D_t(i)$ : distribution of sample weight for all  $x_i$ .
- $\alpha_t$ : learner weight for  $h_t$ .

Input:

- Training set:  $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m), x_i \in X, y_i \in \{-1, +1\}$ .
- Sample weight distribution:  $D_t$  over  $m$  examples.
- Weak learner or hypothesis:  $h_t$ .
- $T$ : number of iterations/training rounds/ weak learners selected.

Procedure:

Initialize sample weights:  $D_t(i) = 1/m$ .

for  $t = 1$  to  $T$

- Try many weak learners using sample weights of  $D_t(i)$ .
  - Get a weak learner  $h_t(x): X \rightarrow Y = \{-1, +1\}$
  - Compute its prediction error:  $\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i]$
  - From all weak learners, select one with the smallest error.
  - Compute learner weight of the selected one:  $\alpha_t = \frac{1}{2} \ln((1 - \epsilon_t)/\epsilon_t)$
- Add the selected weak learner to the final strong learner:  
 $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$

- Update sample weights:

for  $i = 1$  to  $m$ :

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$
$$= \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(x_i) = y_i \text{ decrease } D_{t+1}(i) \\ \exp(\alpha_t), & \text{if } h_t(x_i) \neq y_i \text{ increase } D_{t+1}(i) \end{cases}$$

end for

$Z_t$  is a normalization factor chosen so that  $D_{t+1}$  will be a distribution.

end for

Output:

The final strong learner:  $H_T(x) = \text{sign}(\sum_{t=1}^T \alpha_t h_t(x))$

Figure 2: Illustration of the AdaBoost Algorithm.

Given a data set:  $(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$ , where  $x_i \in X, y_i \in Y = \{-1, +1\}$ , initialize a distribution (sample weights) over  $m$  samples:  $D_1(i) = 1/m$ .  $m$  includes both positive ( $mp$ ) and negative samples ( $mn$ ):  $m = mp + mn$ .

for  $t = 1$  to  $T$

Step1: Find the best learner ( $h_t$ ) that minimizes the error ( $\epsilon_t$ ) on  $D_t$ :

$$h_t = \arg.\min[\epsilon_t]$$

Step1-a: Get a weak learner  $h_t: X \rightarrow Y = \{-1, +1\}$

Step1-b: Compute the error of  $h_t$ :

$$\epsilon_t = \sum_{i=1}^m D_t(i) * I_{[h_t(x), y]}, \text{ where } I = \begin{cases} 1, & \text{if } h_t(x_i) \neq y_i \\ 0, & \text{otherwise} \end{cases}$$

Step2: Compute learner weight:  $\alpha_t = \frac{1}{2} \ln((1 - \epsilon_t) / \epsilon_t)$

Step3: Update sample weights:

for  $i = 1$  to  $m$ :

$$w_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

$$= \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t), & \text{if } h_t(x_i) = y_i \\ \exp(\alpha_t), & \text{if } h_t(x_i) \neq y_i \end{cases}$$

end for

$Z_t$  is a normalization factor chosen so that  $D_{t+1}$  will be a distribution.

Step4: Compute the total error using the strong learner up to current  $t$ :

$$CE_t = H_t(x)$$

If  $CE_t < \text{threshold}$ :  $T = t$ , break.

end for

Output:

The final hypothesis (strong learner):  $H_T(x) = \text{sign}(\sum_{t=1}^T \alpha_t h_t(x))$

Figure 3: Illustration of the Cascade AdaBoost Algorithm.

## 2.2. FEATURE EXTRACTION

The main motivation of extracting features rather than using the raw pixels is that, features can encode domain specific knowledge, which is difficult to learn from raw data. Moreover, the computational cost of using a feature-based model is far less than that of using a pixel-based model.

### 2.2.1. HAAR FEATURE

Haar features compute the intensity difference between adjacent rectangles and have two advantages: they can be computed efficiently with the aid of integral images and they are visually intuitive as they can be view as simple edge descriptors. Four basic types of Haar features are often used (Fig. 4) with an optional set of rotated types [10]. The value of the *two-rectangle feature* is the difference between the sum of pixels within two rectangular regions. In the similar manner, the value of *three-rectangle feature* is the difference between the sum of pixels of two outside rectangle subtracted from the sum in a center rectangle. Finally, the *four-rectangle feature* is the difference between diagonal pairs of rectangles. In Fig. 4, (A) and (B) are two-rectangle feature whereas (C) is a three-rectangle feature and (D) is a four-rectangle feature.

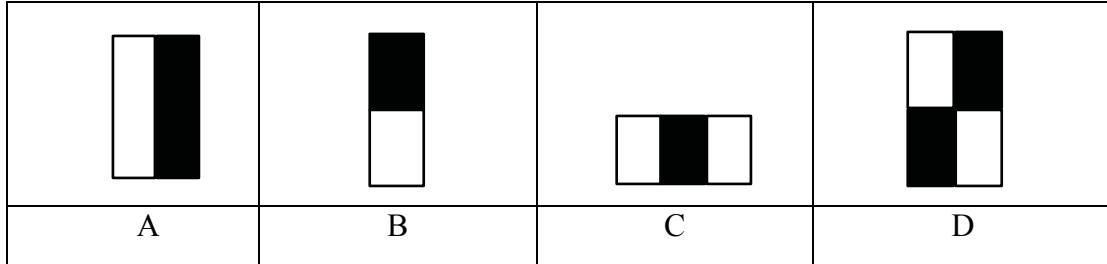


Figure 4: Basic Haar features.

An integral image is a special representation of the original image that is used to facilitate image processing work. The integral image at location  $(x,y)$  (Fig. 5) is the sum of the original pixel values above and to the left of  $(x,y)$ , including the pixel at  $(x,y)$ . Mathematically, the integral image at location  $(x,y)$  can be represented as:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \quad (1)$$



where  $ii(x,y)$  is the integral image and  $i(x,y)$  is the original image. The integral image can be computed over the original image in one pass by:

$$s(x,y) = s(x,y-1) + i(x,y) \quad (2)$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \quad (3)$$

where  $s(x,y)$  is the cumulative row sum,  $s(x,-1)=0$  and  $ii(-1,y) = 0$ .

Given an integral image below (Fig. 6), the sum of the original pixel values in the shaded area can be computed in a linear time:  $ii(4) + ii(1) - ii(2) - ii(3)$

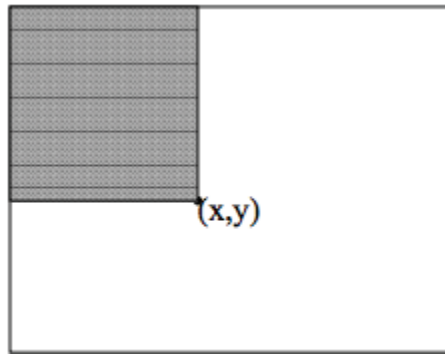


Figure 5: The value of the integral image at point  $(x,y)$

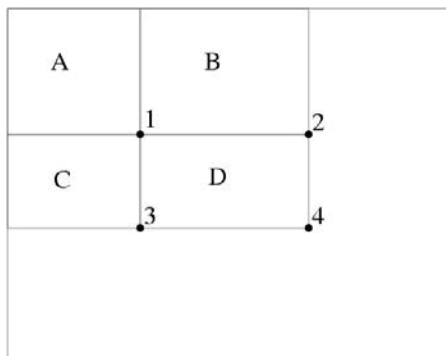


Figure 6: Rectangular feature computation using integral image.

## 2.2.2. LOCAL BINARY PATTERN

Local Binary Pattern (LBP) operator is a local descriptor of image microstructures [11]. LBP describes local properties of an object rather than the whole image as a vector of high dimension. The local features have low dimensions and reduce the computational cost during the training. LBP summarizes the local structure by comparing each pixel with its neighborhood. The operator labels pixels by thresholding the 3x3 neighborhood of each pixel with the center value and considering the result as a binary number[12]. If the central pixel value is greater or equal to its neighbor, it is denoted by 1 or 0 otherwise, resulting a binary number for each pixel. With 8 surrounding pixels, Local Binary Patterns would have  $2^8$  combinations [13]. A LBP example is given in Fig. 7.

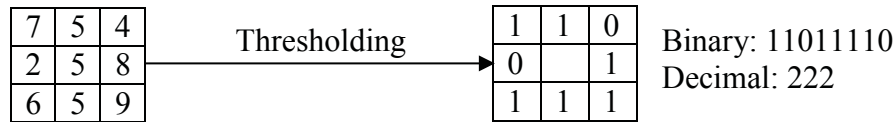


Figure 7: The basic LBP operator.

The LBP operator can be described as below:

$$\text{LBP}(x_c, y_c) = \sum_{p=0}^{P-1} 2^p S(i_n - i_c)$$

where  $(x_c, y_c)$  is the central pixel with intensity  $i_c$  and  $i_n$  is the intensity of the neighbor pixel.  $S$  is the sign function defined as:

$$S(x) = 1 \text{ if } x \geq 0, \text{ and } 0 \text{ otherwise}$$

This description enables us to capture fine-grained details of an object. Although the basic LBP has been proved to be effective for image representation, it is too local to be robust [14]. A number of variants have been developed, including Multi-scale Block

Local Binary Pattern (MB-LBP) that is more robust because it encodes both the microstructures and macrostructures of an image. The MB-LBP is computed based on the average values of block sub-regions instead of individual pixels (Fig. 8). The average sum of image intensity is computed for each sub-region with each sub-region being thresholded against the central block.

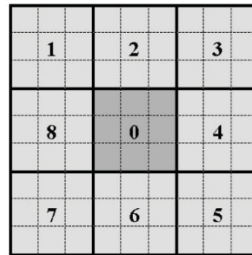


Figure 8: The 9x9 MB-LBP operator.

## CHAPTER 3

### DATA PROCESSING

#### 3.1. DATA SET

Three sets of TEM images were used in the experiments. The training dataset includes the cropped cubes from the “cube-a” image which is of 20nm scale (Fig. 9) and from the “cube-c” image which is of 50nm scale (Fig. 11). On the other hand, “cube-b” was used to carry out the cross- validation tests that are acquired on 50nm scale (Fig. 10).

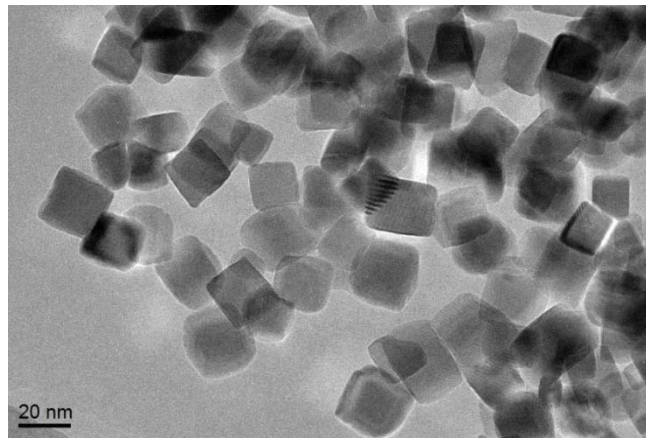


Figure 9: Cube-a image used for selecting training samples.

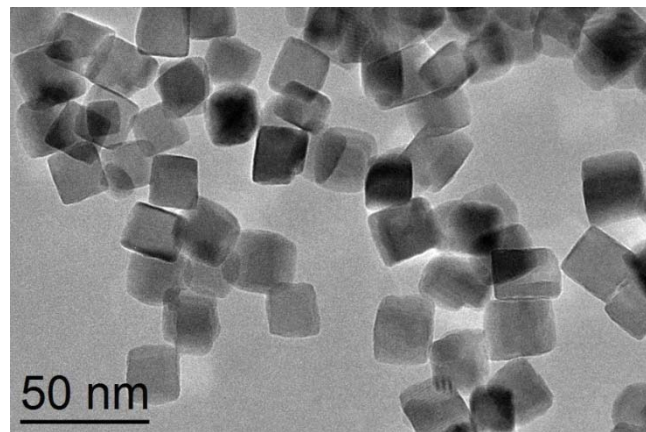


Figure 10: Cube-b image used for testing.

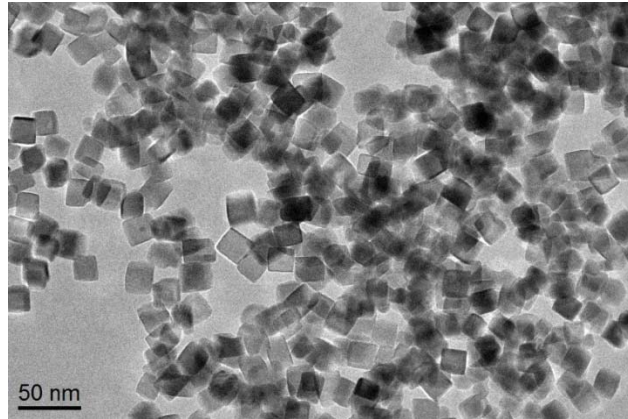


Figure 11: Cube-c image used for testing.

### 3.2.SAMPLE PREPARATION

Two sets of training samples were collected: positive samples and negative samples. Positive samples refer to the cube-shaped particles that we are interested in detecting whereas negative samples are selected from random background images.

#### 3.2.1. POSITIVE SAMPLES

The positive samples used in the training of a cascade classifier were cropped manually from the “cube-a” image, with a total of 80 positive samples being collected. Cropping was done using a GIMP image editing software package. Before cropping, the image was rotated to match the vertical and horizontal edge alignments of each actual particle. For severely overlapped cubes with potentially more than two or three objects involved, only the shape and boundary of the primary one is considered in cropping. A few positive samples are shown in Fig. 12 and Table 1.

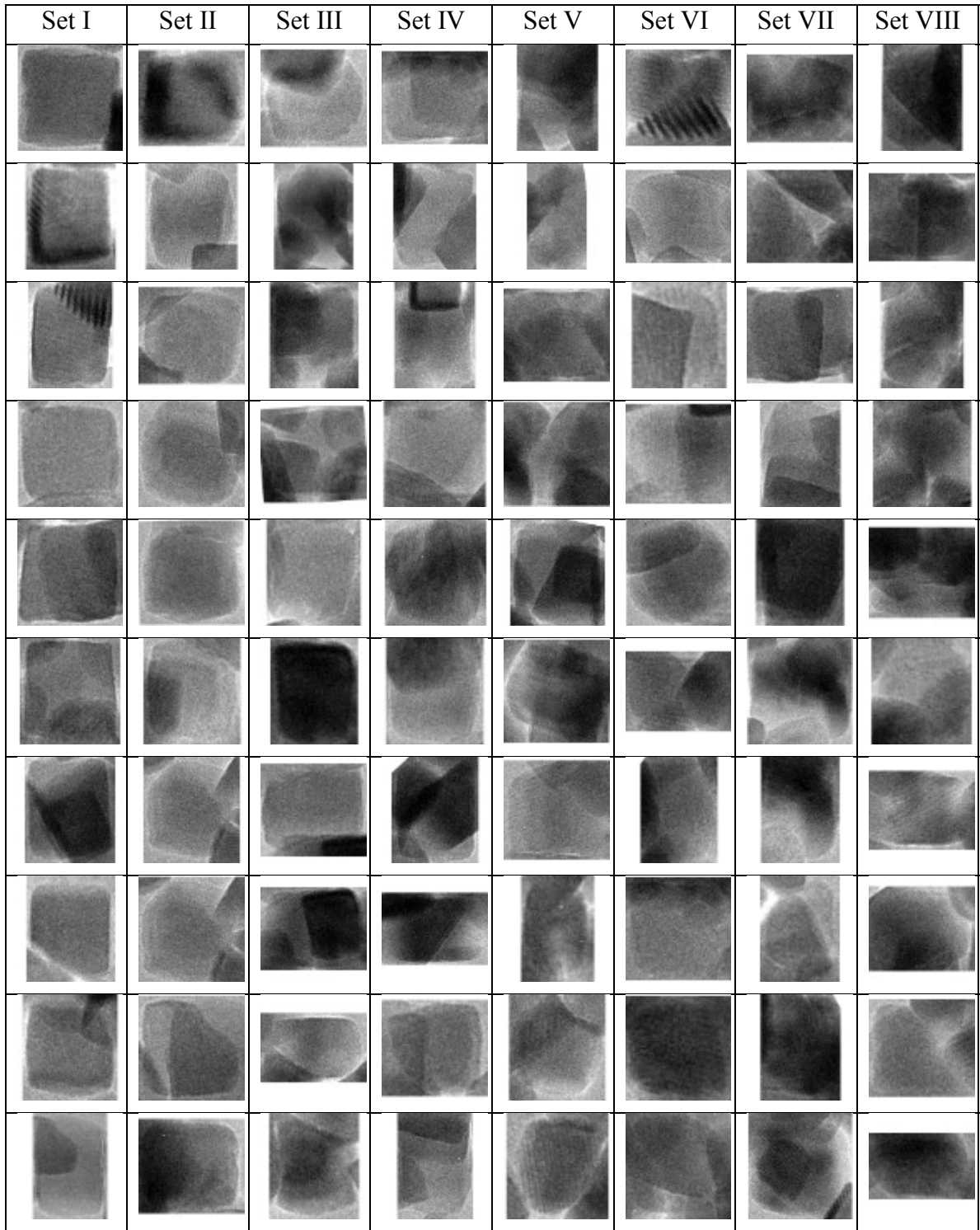


Figure 12: Positive samples cropped from the "cube-a" image.

TABLE 1: SETS OF POSITIVE SAMPLES CROPPED FROM CUBE-A

Set	Number of samples
I	10
II	10
III	10
IV	10
V	10
VI	10
VII	10
VIII	10

### 3.2.2. NEGATIVE SAMPLES

The negative sample consists of 600 background images generated by slicing a few random images including a few with similar intensity distribution as TEM images. The diverse nature of negative samples will enable the classifier to reduce the false alarm rate thereby making the classifier more robust for unseen test images. Several negative samples are shown in Fig. 13.

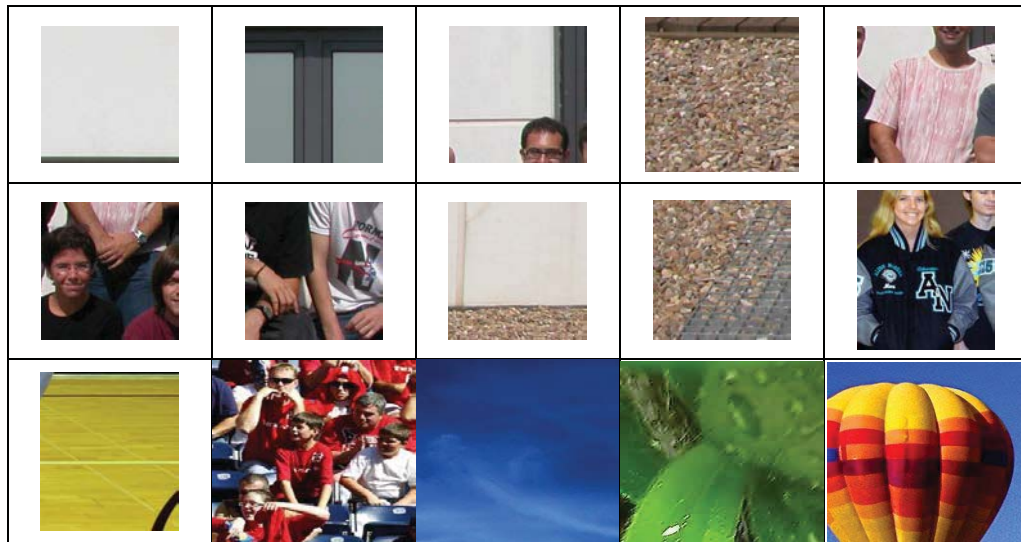


Figure 13: Negative samples.



## CHAPTER 4

### EXPERIMENT DESIGN

This section describes the experimental details during the training/testing phases with various parameter setups. All experiments were carried out using the OpenCV packages installed under the Linux environment on a desktop machine of Intel® Core™ i5-5200U CPU at 2.2GHz and with an 8 GB of RAM. The training phase uses the positive and negative samples to create a classifier in the xml format while the testing phase makes use of the trained model to detect objects in a test image. The following training and testing pairs were conducted: (i) Training and testing by changing the kernel window size; (ii) Training and testing by changing the sample size; (iii) Training and testing with different rotated sample sets of two sample size (20 and 60); (iv) Training and testing with different *maxFalseAlarmRate* values.

#### 4.1. TRAINING

The training phase has four steps:

##### STEP-1: GENERATE A LIST OF NEGATIVE SAMPLES:

```
find ./posImgDir -name "*.jpg" | sort -V -f > posImgList.txt
echo "" >> posImgList.txt
find ./negImgDir -name "*.jpg" | sort -V -f > allNegImgList.txt
echo "" >> allNegImgList.txt

numNeg=300
k=0
while read varLine
do
    echo "$varLine" >> negImgList.txt
    ((k++))
    if [ $k -ge $numNeg ]
    then
        break
    fi
done < allNegImgList.txt
echo "" >> negImgList.txt
```



## STEP-2: GENERATE A LIST OF POSITIVE SAMPLES:

The following script generates positive samples with “opencv\_createsamples” utility. In case that the number of positive samples is not sufficient, more samples can be generated by randomly rotating the samples and adding noise to the original ones. A total of 400 samples was obtained by rotating samples along its z-axis with a maximum rotation angle of 180 degrees.

```
perl ./bin/step2.pl\  
posImgList.txt\  
negImgList.txt\  
vecSampleDir\  
400\  
"opencv_createsamples\  
-bgcolor 0\  
-bgthresh 0\  
-maxxangle 0.00\  
-maxyangle 0.00\  
-maxzangle 3.14\  
-maxidev 0\  
-w 20\  
-h 20"
```

## STEP-3: MERGE THE VEC FILES OF INDIVIDUAL SAMPLES INTO A SINGLE ONE:

```
find ./vecSampleDir/posImgDir -name '*.vec' | sort -V -f >  
./vecSampleDir/vecList.txt  
  
./bin/mergevec ./vecSampleDir/vecList.txt  
./vecSampleDir/allPositiveSamples.vec
```

## STEP-4: TRAIN A CASCADE CLASSIFIER:

A cascade classifier was trained with the merged `vec` data and negative samples. One thing to be noted is that the number of positives (`numPos`) and negatives (`numNeg`) should be more or less the same and should be 80-90% of the number of training samples generated in step-2. Other parameters include the number of cascade stages, type of stages, type of feature, size of detection window, type of boosted classifiers, minimal

desired hit rate for each stage of the classifier, maximal desired false alarm rate for each stage of the classifier, maximal depth of a weak tree, maximal count of weak trees for each stage, type of Haar features used in training, etc. All of these parameters can be fine-tuned to achieve a better performance.

```
opencv_traincascade -data          trainedClassifier\  
                    -vec  
./vecSampleDir/allPositiveSamples.vec\  
                    -bg            negImgList.txt\  
                    -numPos        300\  
                    -numNeg        300\  
                    -numStages     15\  
                    -precalcValBufSize 512\  
                    -precalcIdxBufSize 512\  
                    -stageType     BOOST\  
                    -featureType   HAAR\  
                    -w             20\  
                    -h             20\  
                    -bt            GAB\  
                    -minHitRate    0.985\  
                    -maxFalseAlarmRate 0.500\  
                    -weightTrimRate 0.950\  
                    -maxDepth      1\  
                    -maxWeakCount  100\  
                    -mode          ALL
```

## 4.2. TESTING

A testing was done by running the final classifier (cascade.xml) on a test image with several parameters specifying the sizes of objects to be detected as well as the number of hits in a neighborhood window to confirm a detection.

```
echo "test ..."  
  
# exe classifier image scale min-nbr min-obj-size max-obj-size equalizer  
./cascadeAdaBoost ./cascade.xml ./cube_b.jpg 1.1 3 70 160 0  
  
echo "test is done."
```

## CHAPTER 5

### RESULTS AND DISCUSSIONS

#### 5.1. IMPACT OF DETECTION WINDOW SIZE (CUBE-A AS TRAINING)

The detection window size is one of the most important parameters that could have a significant impact on a classifier's performance. It is generally believed that a larger window size can result in a better performance at the cost of a longer training time. Experimental results of using different window sizes for both Haar and LBP features are given in Table 2 and plotted in Fig. 14 to Fig. 18. It is clear that LBP reduces the training time drastically, by almost 2 to 4 orders of magnitude. The second noticeable outcome is that the three performance metrics (TPR, FPR, Precision) exhibit much more complex variations as the detection window size gradually increases. It is generally believed that the true positive rate would show a positive correlation with the detection window size at the early stage and then level off as window size continues to increase. The explanation is that larger window sizes provide more information to enhance the detection rate, but after certain point, the benefits of large window size drops due to the saturation of new image information. The same logic and reasoning can be applied to the curves of FPR and precision. However, LBP display almost opposite trends as compared to these of Haar. As can be seen in Fig. 18, TPR of Haar improves as window sizes grows at the cost of slightly large false positive rate. But the LBP generate a large number of false positive cases regardless of the window size used. Based on these experimental results, it can be said that the window size is a significant factor for a detection project using the Haar features, whereas the influence of window size on LBP is more complicated and more research is needed.

TABLE 2: IMPACT OF DETECTION WINDOW SIZE (CUBE-A)

Window size	Training Time (s)		TPR		FPR		Precision	
	Haar	LBP	Haar	LBP	Haar	LBP	Haar	LBP
20	492	08	0.49	0.81	0.0003	0.0157	0.97	0.52
24	782	22	0.62	0.76	0.0007	0.0257	0.95	0.38
28	1912	34	0.63	0.57	0.0000	0.0197	1.00	0.38
32	4523	50	0.71	0.33	0.0003	0.0160	0.98	0.30
36	10814	80	0.62	0.43	0.0057	0.0090	0.696	0.5

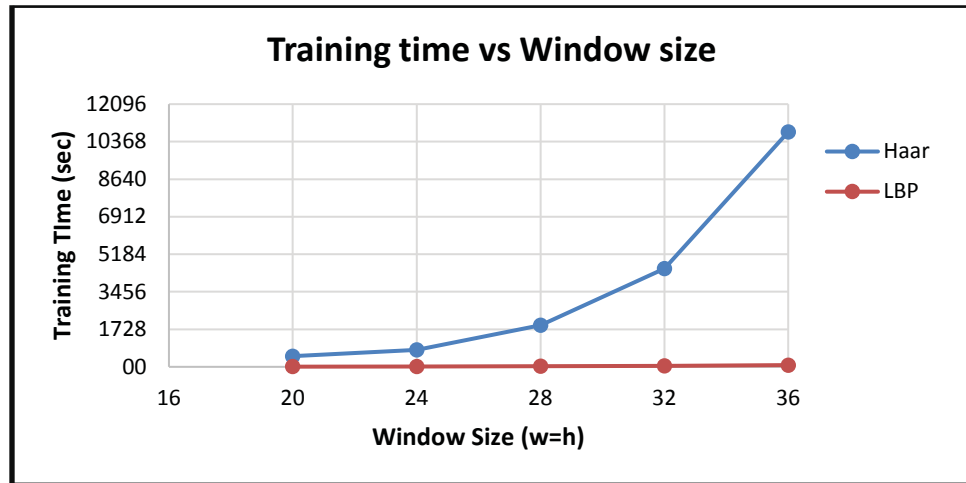


Figure 14: Impact of window size on training time

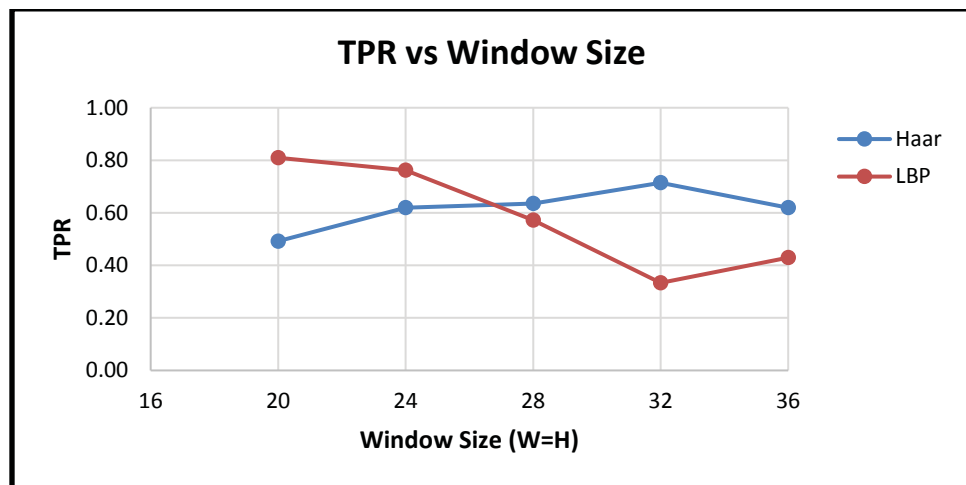


Figure 15: Impact of window size on TPR

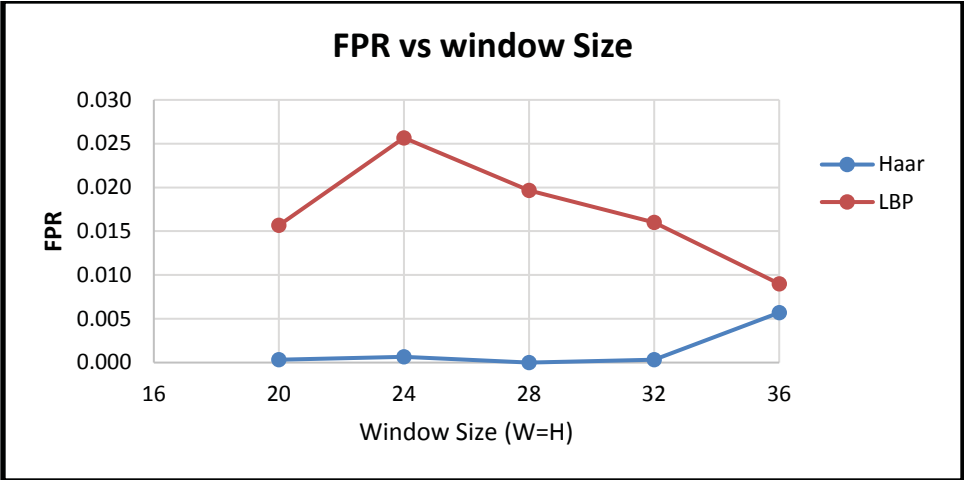


Figure 16: Impact of window size on FPR

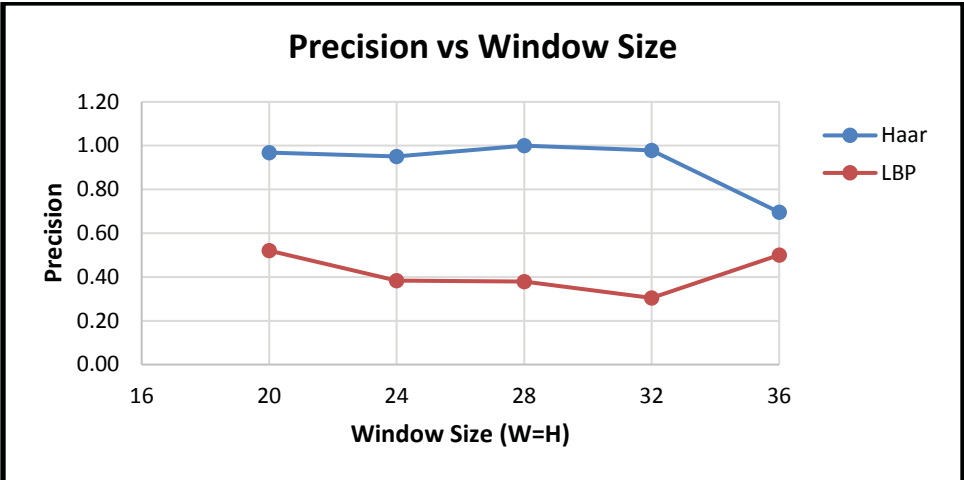


Figure 17: Impact of window size on precision

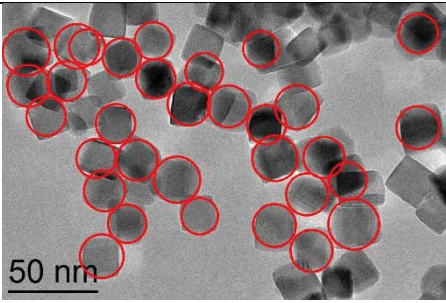
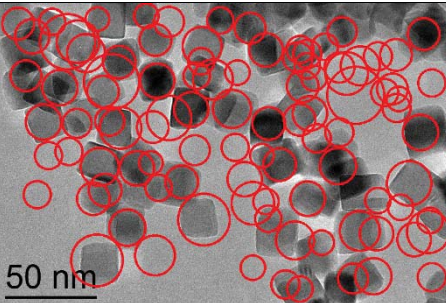
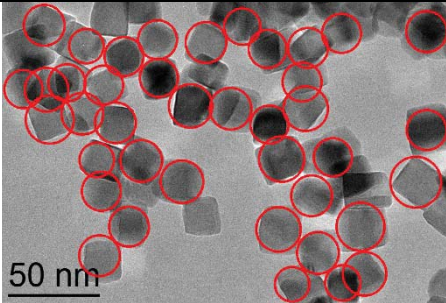
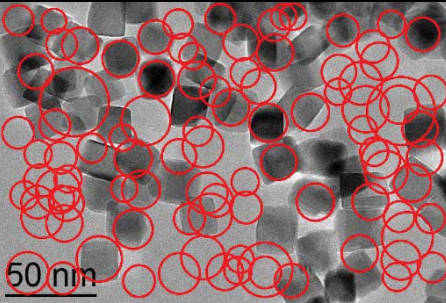
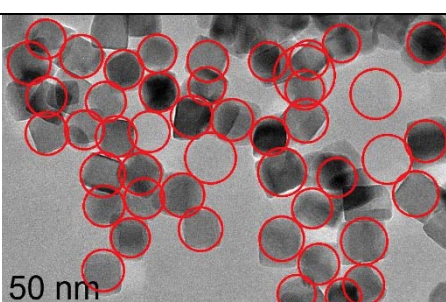
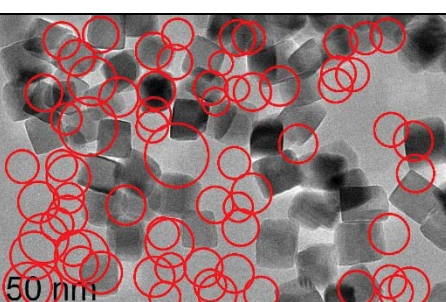
Window Size	Haar	LBP
20		
28		
32		

Figure 18: Detection results with different window sizes.

## 5.2. IMPACT OF SAMPLE SIZE (CUBE-A AS TRAINING)

Depending upon the application domain, the number of samples used in a test varies widely, from less than one hundred to over one million. It is a critical issue to understand the mechanism by which the sample size affects a classifier's performance. In this test, sample size increases from 20 to 160 with a step size of 20. Since the number of actual positive samples is 80 from cube-a image, another 80 samples were generated by adding white noise and random rotation along the z-axis. The test results are listed in

Table 3 and plotted in Fig. 19 to Fig. 23. As expected, the training time of Haar feature grows linearly as the sample size increases whereas the training time of LBP is more stabilized. Although Haar shows better performance than LBP as measured by TPR, FPR and precision, no clear pattern is observed with respect to the change of sample size. In other words, sample size carries more weight on the training time but its influence on the detection outcomes seems unpredictable, which can also be seen in Fig. 23.

TABLE 3: IMPACT OF SAMPLE SIZE (CUBE-A).

Sample Size	Training Time (s)		TPR		FPR		Precision	
	Haar	LBP	Haar	LBP	Haar	LBP	Haar	LBP
20	26	01	0.349	0.032	0.000	0.001	1.000	0.400
40	90	02	0.302	0.746	0.000	0.025	0.950	0.388
60	96	03	0.810	0.079	0.005	0.004	0.785	0.278
80	149	04	0.825	0.159	0.004	0.004	0.800	0.435
100	197	05	0.540	0.063	0.007	0.004	0.607	0.250
120	263	08	0.571	0.079	0.004	0.002	0.766	0.455
140	281	09	0.492	0.016	0.002	0.001	0.816	0.250
160	345	09	0.413	0.079	0.011	0.003	0.448	0.385

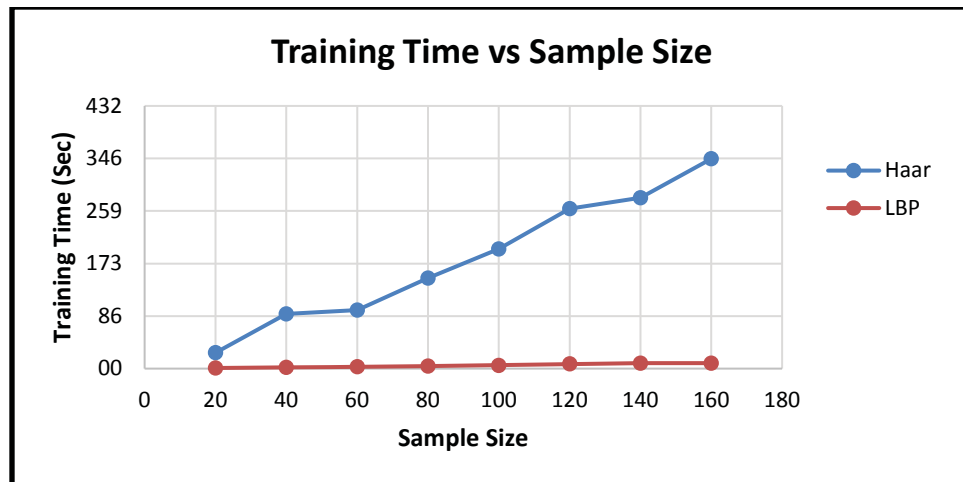


Figure 19: Impact of sample size on training time.

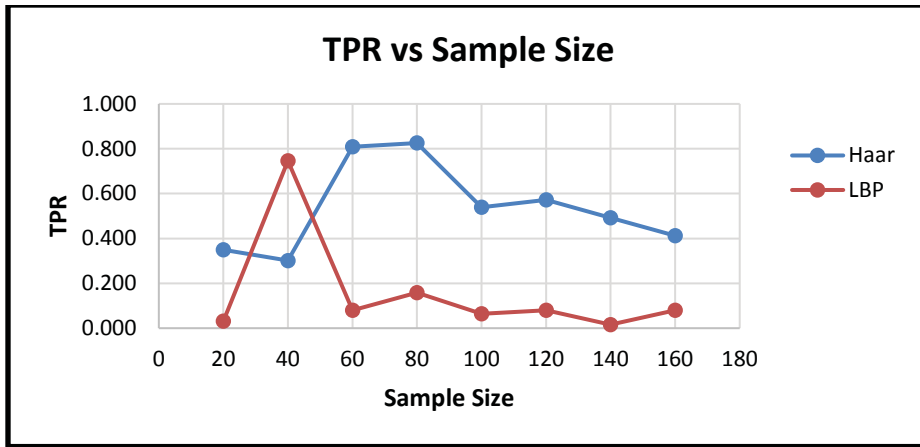


Figure 20: Impact of sample size on TPR.

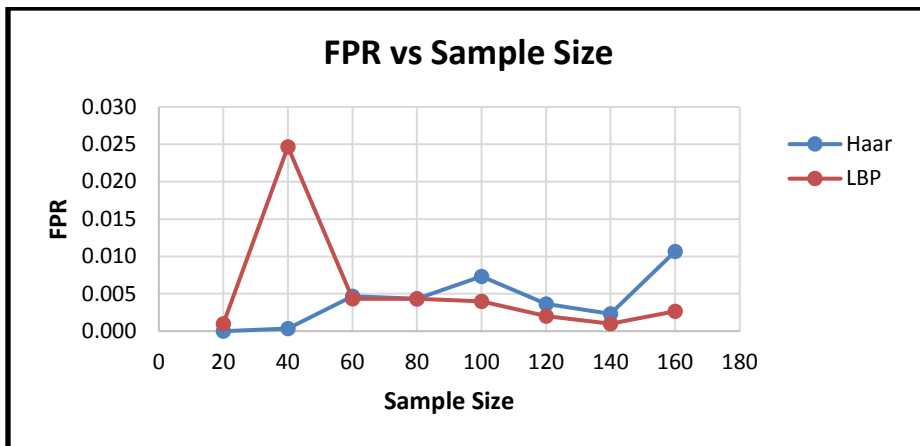


Figure 21: Impact of sample size on FPR.

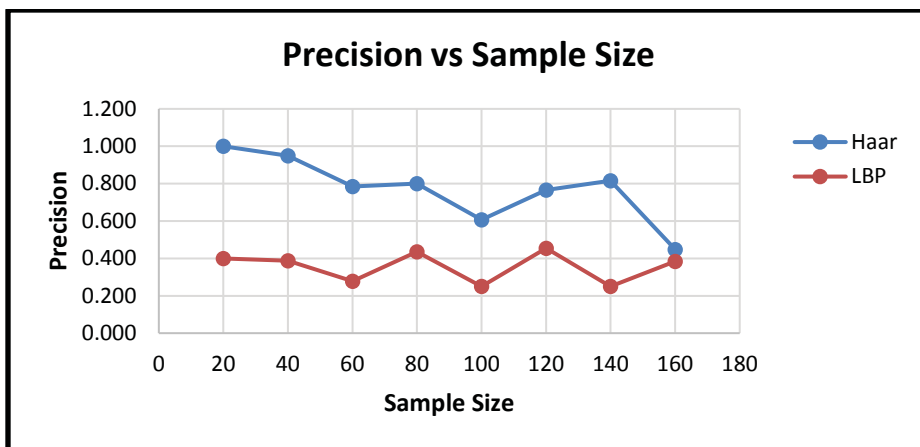


Figure 22: Impact of sample size on precision.



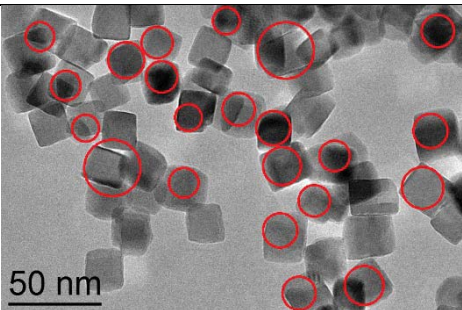
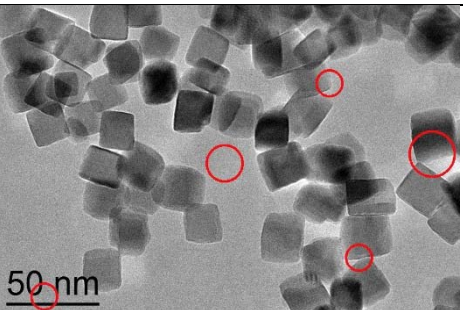
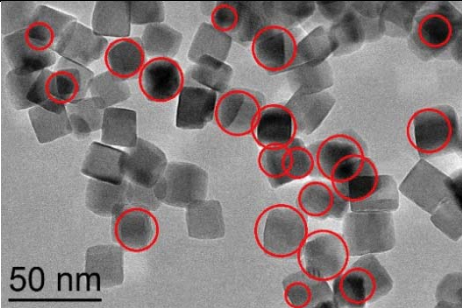
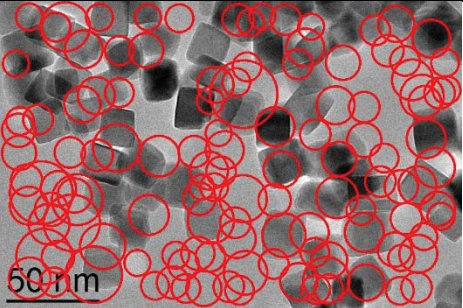
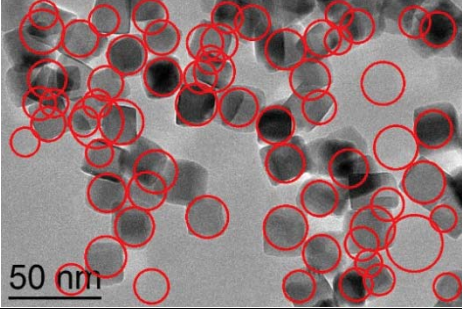
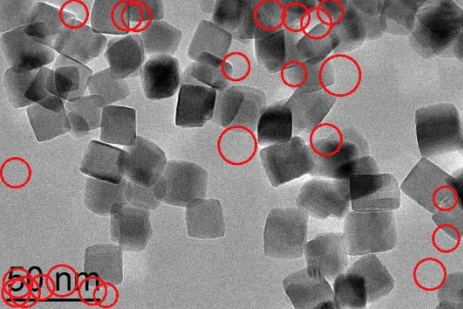
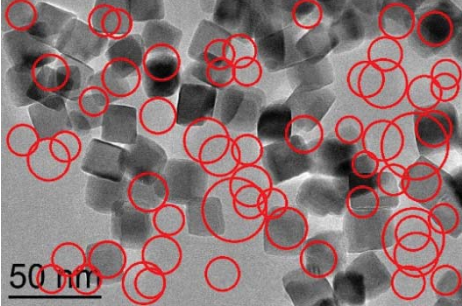
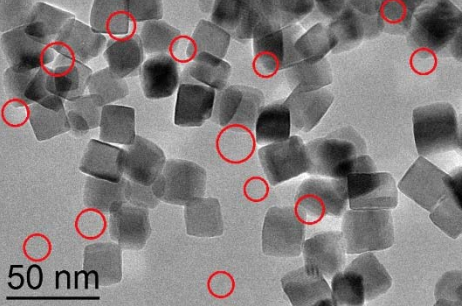
Sample Size	Haar	LBP
20	 <p>50 nm</p>	 <p>50 nm</p>
40	 <p>50 nm</p>	 <p>50 nm</p>
80	 <p>50 nm</p>	 <p>50 nm</p>
160	 <p>50 nm</p>	 <p>50 nm</p>

Figure 23: Detection results with different sample sizes.

### 5.3. IMPACT OF SAMPLE SIZE (CUBE-C AS TRAINING)

To further verify the observations of the previous tests with cube-a, experiments were repeated using positive samples extracted from cub-c image. The test results are listed in Table 4 and plotted in Fig. 24 to Fig. 27. Similar patterns were observed regarding the relationships among samples size, training time, TPR, FPR and precision.

TABLE 4: IMPACT OF SAMPLE SIZE (CUBE-C)

Sample Size	Training Time (s)		TPR		FPR		Precision	
	Haar	LBP	Haar	LBP	Haar	LBP	Haar	LBP
40	62	01	0.4444	0.5079	0.0003	0.0283	0.97	0.27
80	171	04	0.5714	0.0159	0.0000	0.0017	1.00	0.17
120	328	08	0.6508	0.4127	0.0000	0.0193	1.00	0.31
160	534	10	0.7143	0.3651	0.0010	0.0133	0.94	0.37
200	691	16	0.6984	0.6508	0.0027	0.0253	0.85	0.35
240	998	20	0.7143	0.6825	0.0017	0.0117	0.90	0.55
280	1307	23	0.746	0.6508	0.0040	0.0243	0.80	0.36
320	1593	26	0.6984	0.7143	0.0010	0.0240	0.94	0.38
360	2213	32	0.4921	0.7143	0.0017	0.0193	0.86	0.44
400	3744	42	0.6984	0.7778	0.0047	0.0217	0.76	0.43
440	3929	39	0.6984	0.4921	0.0040	0.0253	0.79	0.29
480	6049	40	0.5714	0.6825	0.0023	0.0243	0.84	0.37
520	7697	47	0.7143	0.6032	0.0037	0.0230	0.80	0.36

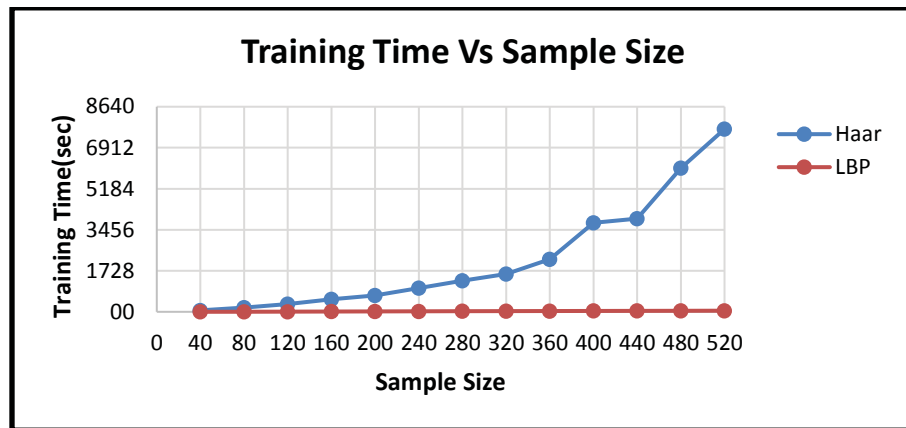


Figure 22: Impact of sample size on training time (cube-c).

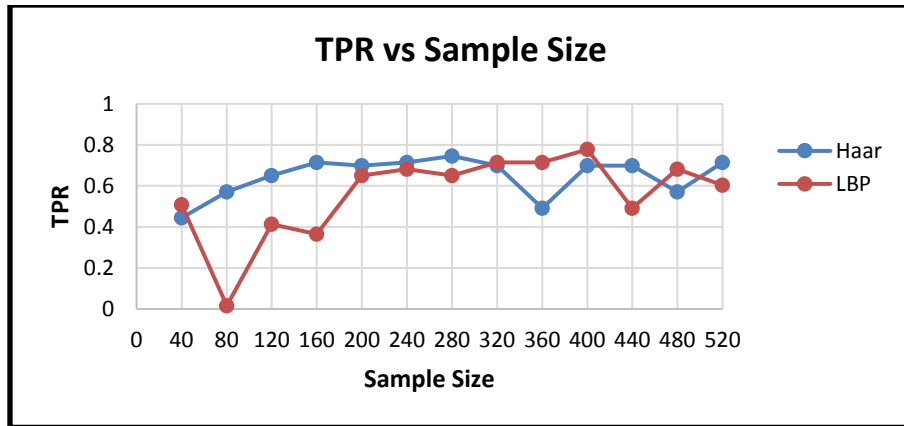


Figure 23: Impact of sample size on TPR (cube-c).

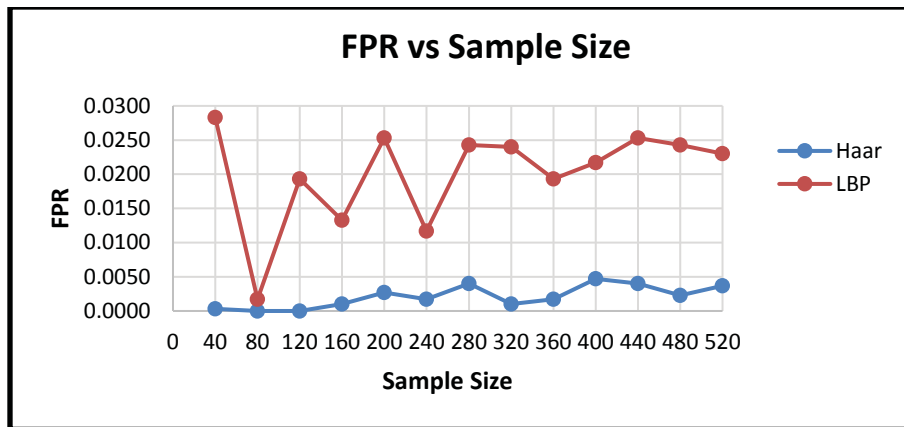


Figure 24: Impact of sample size on FPR (cube-c).

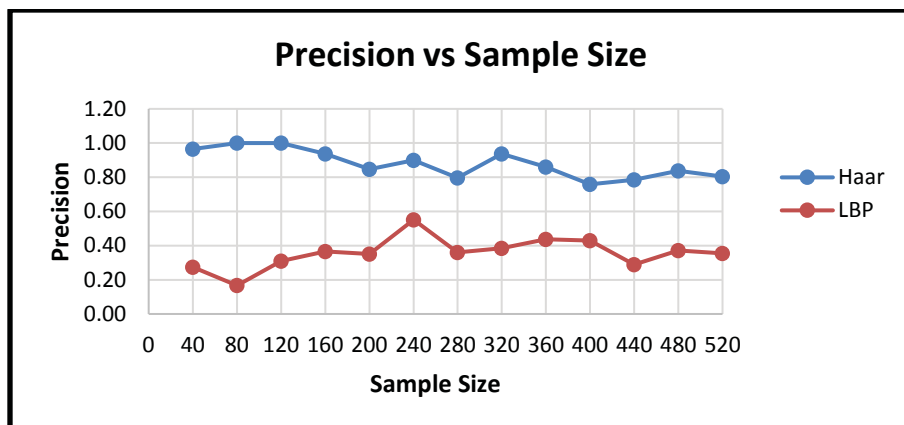


Figure 25: Impact of sample size on precision (cube-c).

#### 5.4. IMPACT OF ROTATED SAMPLE SETS (CUBE-A AS TRAINING)

In testing phase, a sample subset may have more influences on the results than others and hence could skew the prediction. So, a series of tests with different subsets (subset size of 20 and 60) were conducted using Haar feature. The test results are plotted in Fig. 28 to Fig. 33. Although variations were observed with respect to TPR, FPR and precision, they are just random fluctuations. It can be concluded that no single subset has a stronger impact than others. In other words, the distribution of the original data set is relatively uniform.

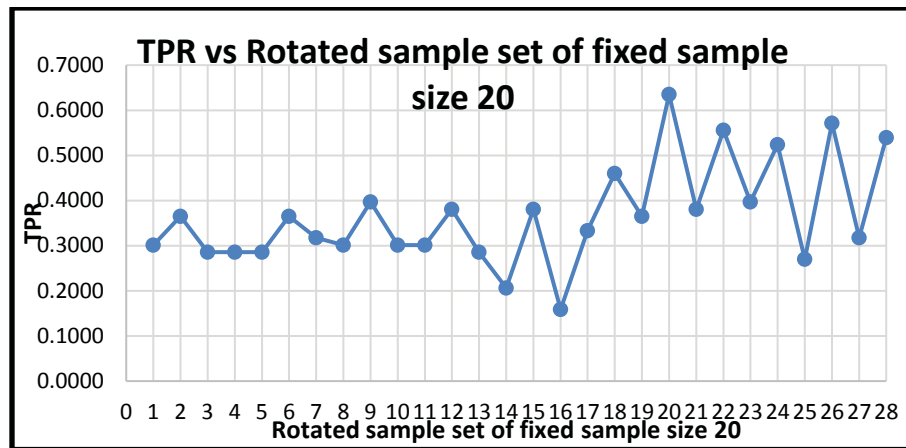


Figure 26: Impact of rotated sample sets (20) on TPR.

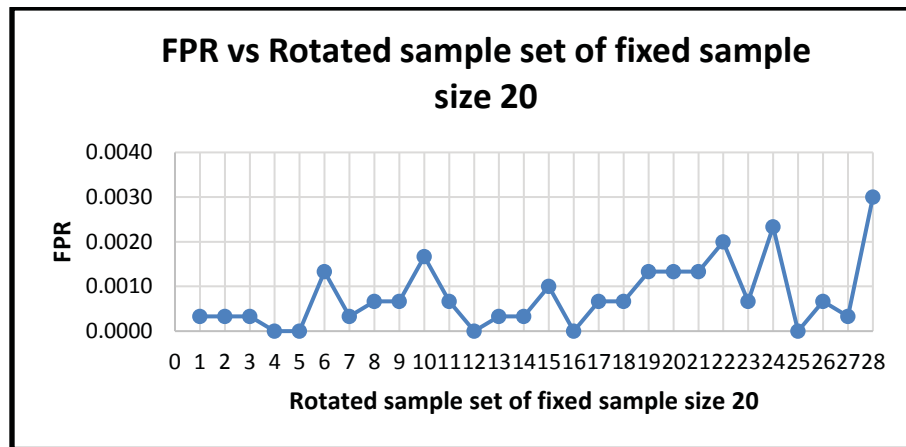


Figure 27: Impact of rotated sample sets (20) on FPR.

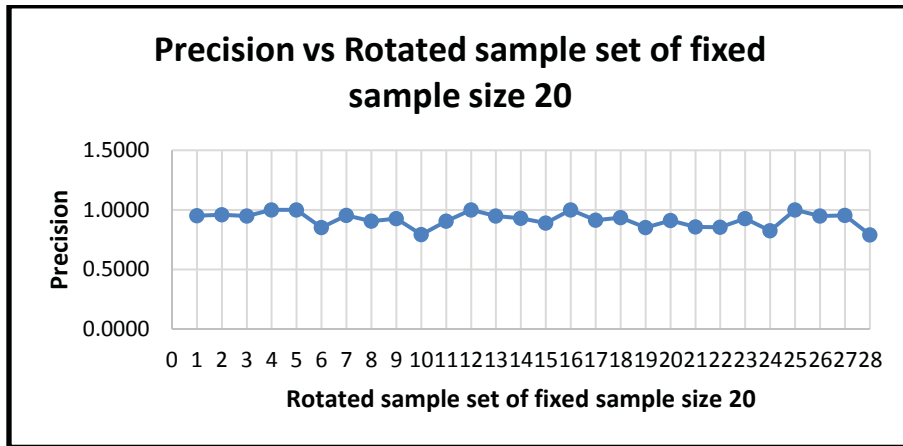


Figure 28: Impact of rotated sample sets (20) on precision.

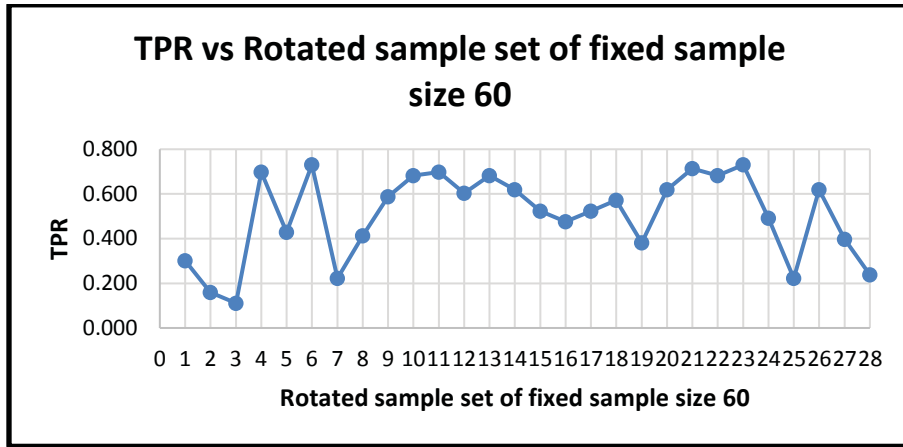


Figure 29: Impact of rotated sample sets (60) on TPR.

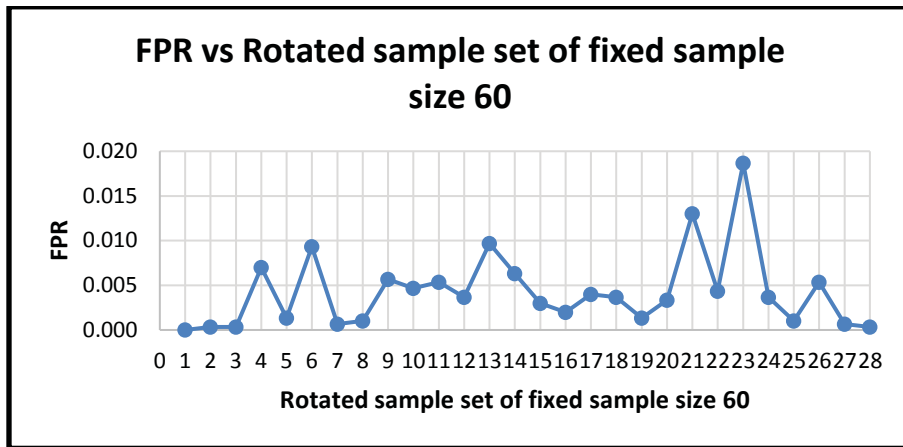


Figure 30: Impact of rotated sample sets (60) on FPR.

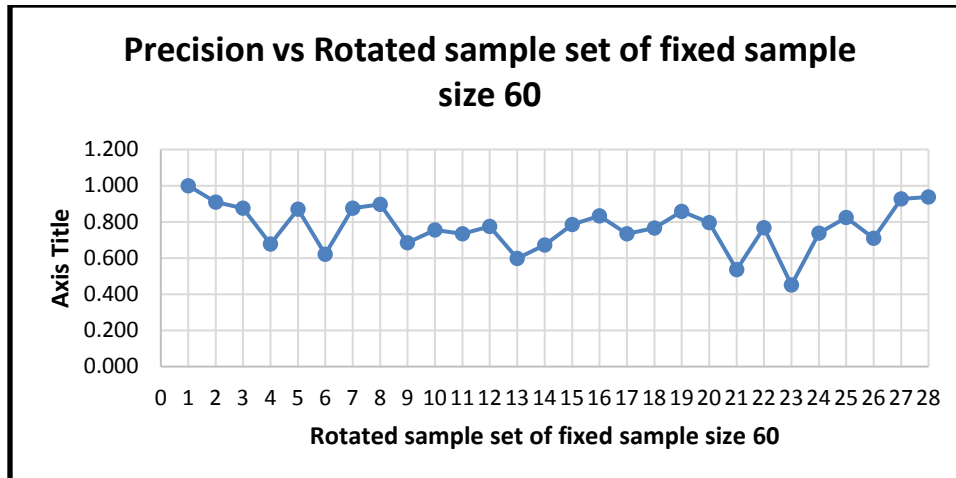


Figure 31: Impact of rotated sample sets (60) on precision.

### 5.5. IMPACT OF KEY TRAINING PARAMETER (CUBE-A AS TRAINING)

Even though AdaBoost algorithm requires much less number of parameter during the training than other algorithm, it still has a few key parameters that could potentially affect the performance of a classifier. One of such parameters is “maxFalseAlarmRate”. Several experiments were carried out by varying the value of “maxFalseAlarmRate” with four different window sizes and the results are presented in Fig. 34. For a given window size, TPR increases as the parameter value grows and then drops as it approaches 0.5. This is probably due to the elimination of “bad” Haar features at the very early stages caused by demanding parameter thresholds. FPR shows opposite trends, at least in the cases of Haar features. FPR results in the tests of using LBP features are more complicated as they displayed an inversed “U-shape” curves, except the case with a window size of 20. In most studies, the value of “maxFalseAlarmRate” is set to 0.5 or slightly larger. The results from this experiment suggests that an optimal parameter value might exist and more investigations on this issue is needed.

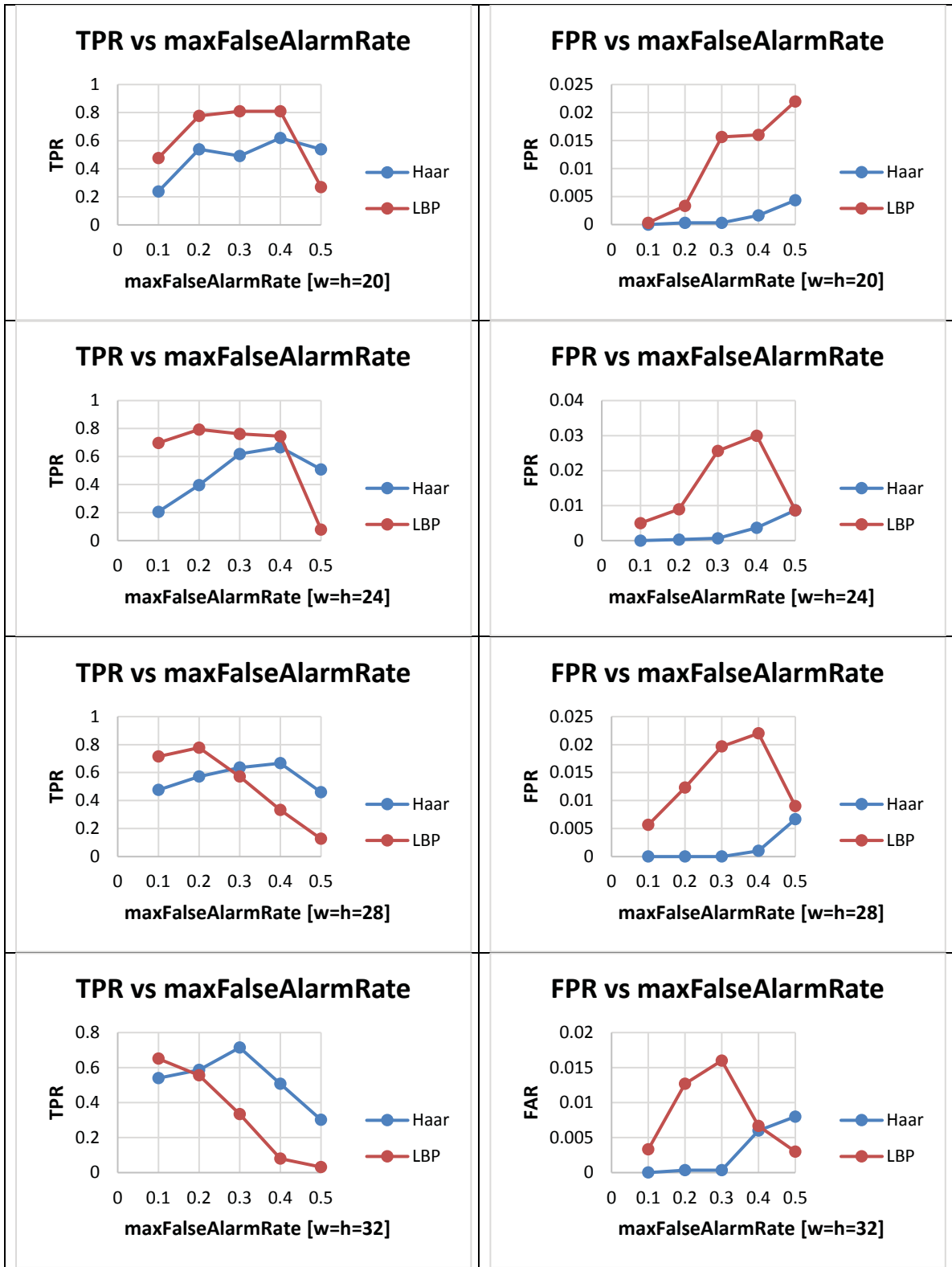


Figure 32: Impact of a key training parameter.

## 5.6. VISUALIZATION OF SELECTED HAAR FEATURES

Sometimes the selected Haar features can provide insights to the performance of a classifier, especially those in early stages. In Fig. 35 to Fig. 38, the training parameters of two classifiers and their selected Haar features in three stages are listed and plotted. It seems that a good model tends to select larger Haar features along the object's boundary while a poor model includes more random and small features.

```
opencv_traincascade -data          trainedClassifier\  
                    -vec          ./vecSampleDir/allPositiveSamples.vec\  
                    -bg          negImgList.txt\  
                    -numPos      300\  
                    -numNeg      300\  
                    -numStages   15\  
                    -precalcValBufSize 1024\  
                    -precalcIdxBufSize 1024\  
                    -stageType   BOOST\  
                    -featureType HAAR\  
                    -w           32\  
                    -h           32\  
                    -bt          GAB\  
                    -minHitRate  0.999\  
                    -maxFalseAlarmRate 0.2\  
                    -weightTrimRate 0.950\  
                    -maxDepth    1\  
                    -maxWeakCount 100\  
                    -mode        ALL
```

Figure 33: Training parameters for a classifier of good performance.

```
opencv_traincascade -data          trainedClassifier\  
                    -vec          ./vecSampleDir/allPositiveSamples.vec\  
                    -bg          negImgList.txt\  
                    -numPos      300\  
                    -numNeg      300\  
                    -numStages   15\  
                    -precalcValBufSize 1024\  
                    -precalcIdxBufSize 1024\  
                    -stageType   BOOST\  
                    -featureType HAAR\  
                    -w           32\  
                    -h           32\  
                    -bt          GAB\  
                    -minHitRate  0.999\  
                    -maxFalseAlarmRate 0.500\  
                    -weightTrimRate 0.950\  
                    -maxDepth    1\  
                    -maxWeakCount 100\  
                    -mode        ALL
```

Figure 34: Training parameters for a classifier of poor performance.



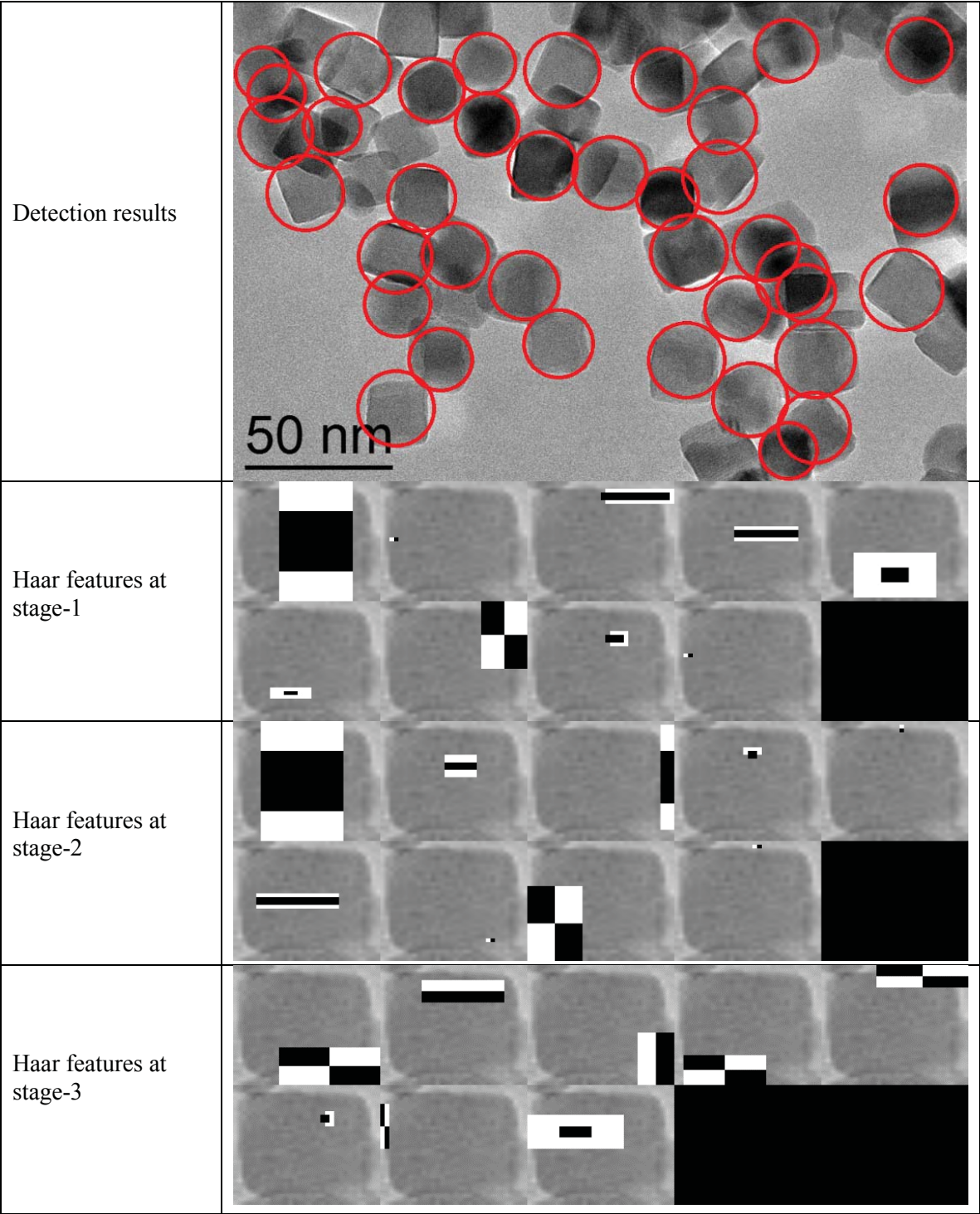


Figure 35: A classifier of good performance and its selected Haar features.

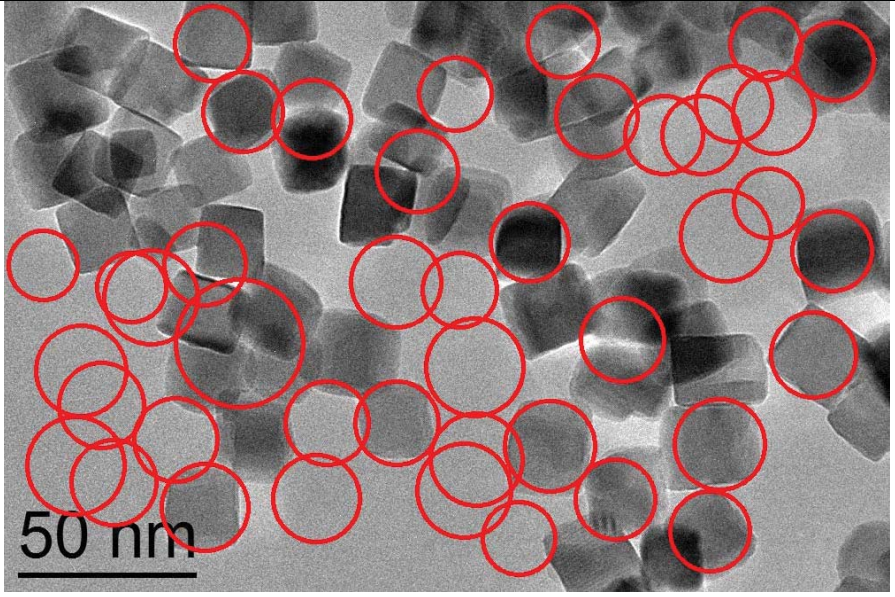
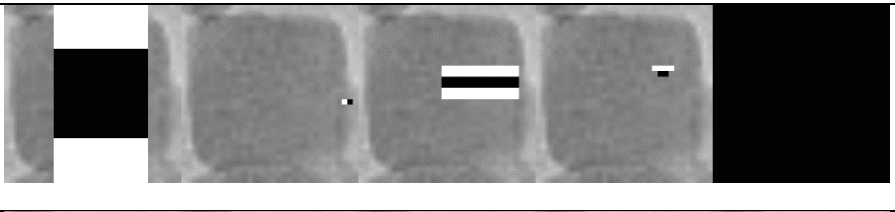
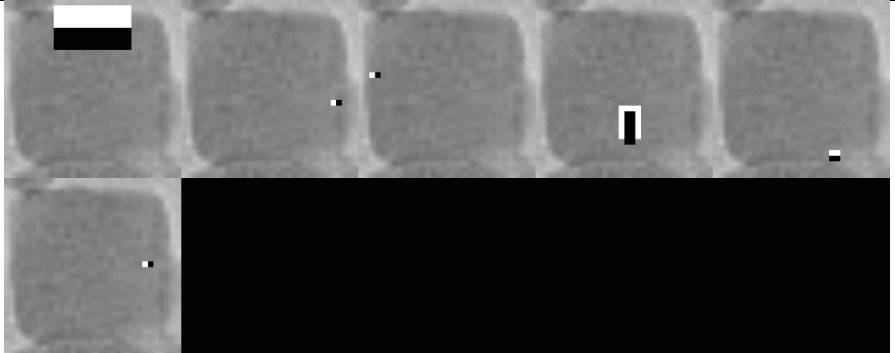
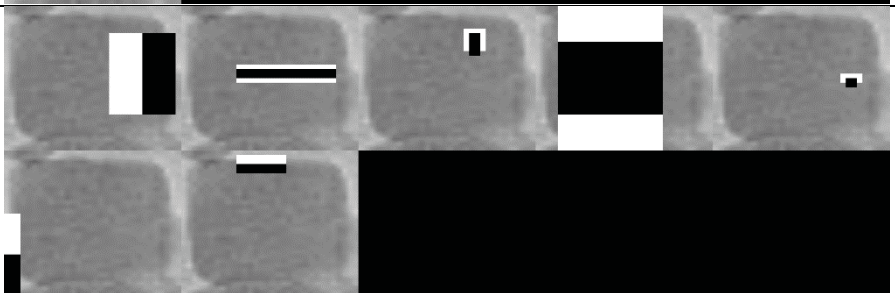
<p>Detection results</p>	
<p>Haar features at stage-1</p>	
<p>Haar features at stage-2</p>	
<p>Haar features at stage-3</p>	

Figure 36: A classifier of poor performance and its selected Haar features.

## 5.7. VISUALIZATION OF SELECTED LBP FEATURES

Similar tests as described in the section of 5.6 were conducted by replacing Haar features with LBP features. The training scripts and test results are given and plotted in Fig. 39 to Fig. 42. The good classifier seemed to have more medium-sized LBP features that are more evenly distributed inside the object, while the poor classifier tends to have LBP features of either very large or very smaller sizes.

```
opencv_traincascade -data          trainedClassifier\  
                    -vec          ./vecSampleDir/allPositiveSamples.vec\  
                    -bg          negImgList.txt\  
                    -numPos      300\  
                    -numNeg      300\  
                    -numStages   15\  
                    -precalcValBufSize 1024\  
                    -precalcIdxBufSize 1024\  
                    -stageType   BOOST\  
                    -featureType HAAR\  
                    -w           20\  
                    -h           20\  
                    -bt          GAB\  
                    -minHitRate  0.999\  
                    -maxFalseAlarmRate 0.100\  
                    -weightTrimRate 0.950\  
                    -maxDepth    1\  
                    -maxWeakCount 100\  
                    -mode        ALL
```

Figure 37: Training parameters for a classifier of good performance (LBP).

```
opencv_traincascade -data          trainedClassifier\  
                    -vec          ./vecSampleDir/allPositiveSamples.vec\  
                    -bg          negImgList.txt\  
                    -numPos      300\  
                    -numNeg      300\  
                    -numStages   15\  
                    -precalcValBufSize 1024\  
                    -precalcIdxBufSize 1024\  
                    -stageType   BOOST\  
                    -featureType HAAR\  
                    -w           24\  
                    -h           24\  
                    -bt          GAB\  
                    -minHitRate  0.999\  
                    -maxFalseAlarmRate 0.400\  
                    -weightTrimRate 0.950\  
                    -maxDepth    1\  
                    -maxWeakCount 100\  
                    -mode        ALL
```

Figure 38: Training parameters for a classifier of poor performance (LBP).



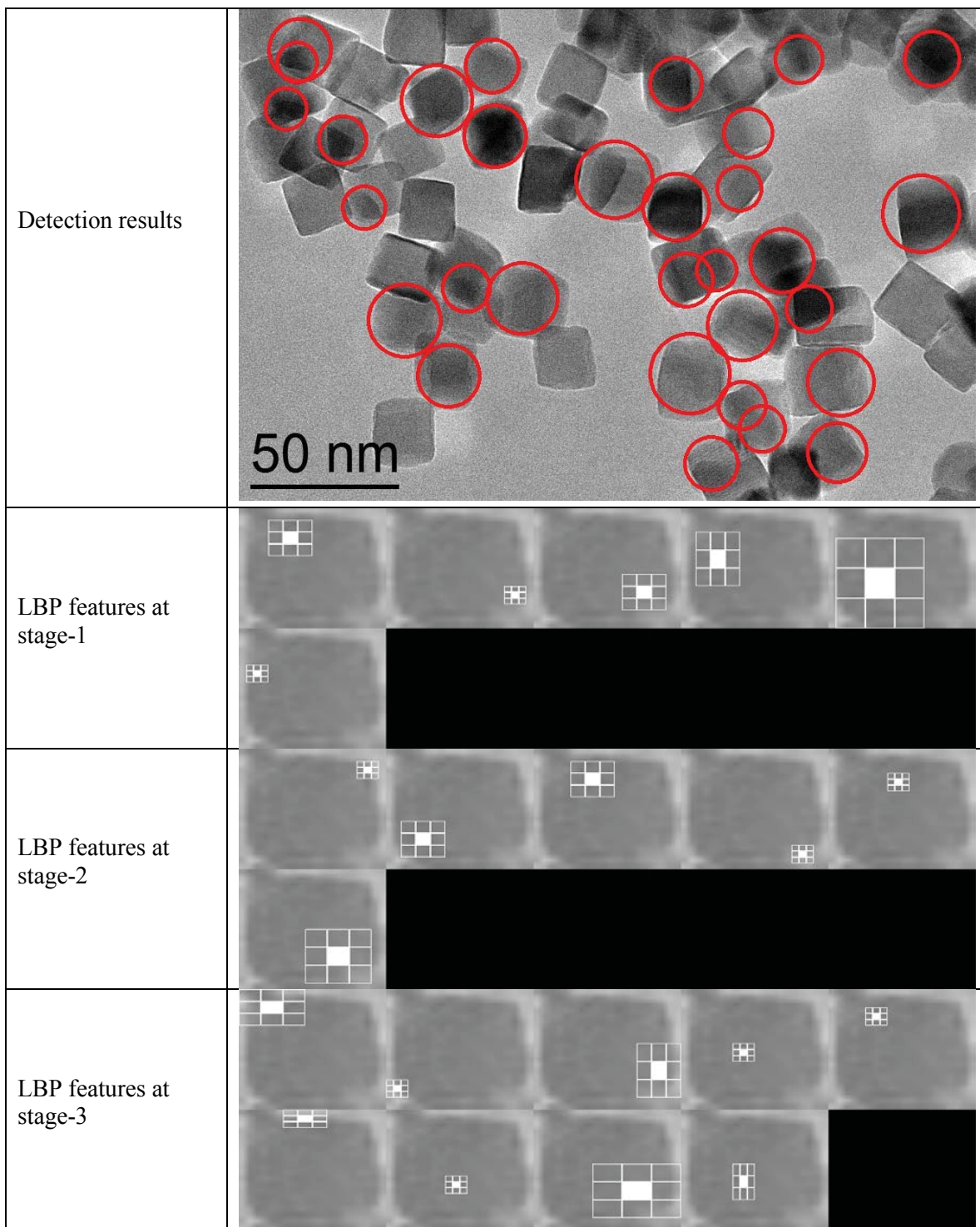


Figure 39: A classifier of good performance and its selected LBP features.

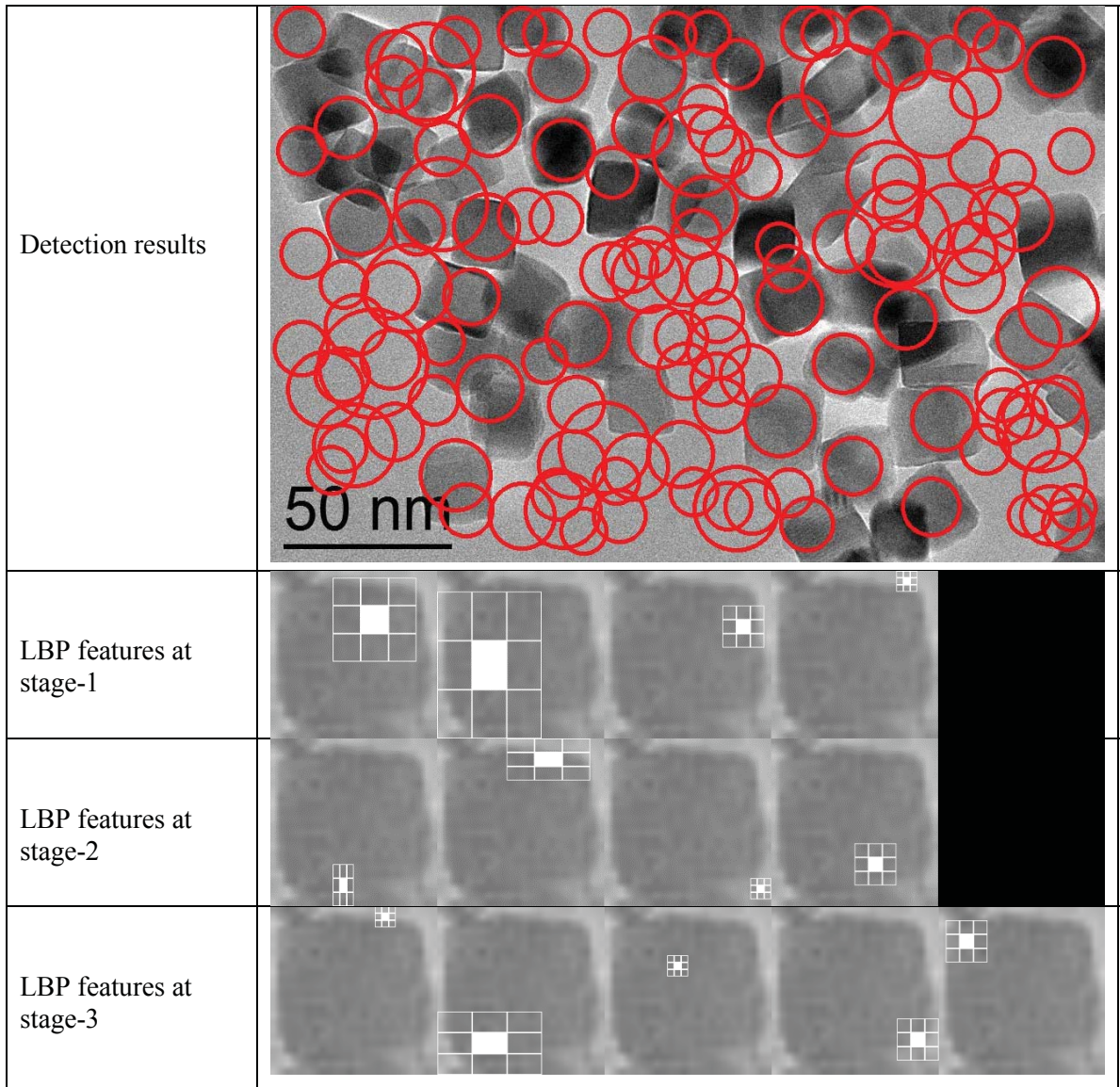


Figure 40: A classifier of poor performance and its selected LBP features.

## CHAPTER 6

### CONCLUSIONS

Due to the presence of object occlusions (overlapping) and the semi-transparent photographic characteristics of nano-particles in TEM images, it is extremely challenging to automatically count the number of particles accurately. There is no clear guideline regarding the optimal training/testing parameters for a Cascade AdaBoost algorithm in detecting nano-particles. This thesis investigates the impacts of several key parameters by conducting a series of empirical evaluation tests using Haar features and LBP features. The primary findings are summarized as below:

1. LBP features is superior to Haar features as they reduce the training time by several orders of magnitude, even though the detection rate of LBP is less competitive than that of Haar features in some cases.
2. Kernel window size has a large impact on the detection outcomes, especially when Haar features are used. Larger window sizes can increase the detection rate, but also comes with a higher training cost and its benefit can level off after a saturation point.
3. No clear pattern was observed as to how the sample size affect the detection results.
4. Sample subset seems not a major factor in determining the detection quality.
5. A smaller value for a key training parameter might lead to a better detection rate as compared to the commonly used default value.
6. The visualization of selected feature might shed lights on the feature selection mechanism and could potentially explain the performance of a classifier and more future work is need in this direction.

## REFERENCES

- [1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 2001, vol. 1, pp. I–I.
- [2] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Icml*, 1996, vol. 96, pp. 148–156.
- [3] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *J.-Jpn. Soc. Artif. Intell.*, vol. 14, no. 771–780, p. 1612, 1999.
- [4] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.
- [5] L. Breiman, "Arcing classifier (with discussion and a rejoinder by the author)," *Ann. Stat.*, vol. 26, no. 3, pp. 801–849, 1998.
- [6] H. Drucker and C. Cortes, "Boosting decision trees," in *Advances in neural information processing systems*, 1996, pp. 479–485.
- [7] J. R. Quinlan, "Bagging, boosting, and C4. 5," in *AAAI/IAAI, Vol. 1*, 1996, pp. 725–730.
- [8] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.
- [9] R. Lienhart, A. Kuranov, and V. Pisarevsky, "Empirical analysis of detection cascades of boosted classifiers for rapid object detection," *Pattern Recognit.*, pp. 297–304, 2003.

- [10] R. Lienhart and J. Maydt, “An extended set of haar-like features for rapid object detection,” in *Image Processing. 2002. Proceedings. 2002 International Conference on*, 2002, vol. 1, pp. I–I.
- [11] T. Ojala, M. Pietikäinen, and D. Harwood, “A comparative study of texture measures with classification based on featured distributions,” *Pattern Recognit.*, vol. 29, no. 1, pp. 51–59, 1996.
- [12] T. Ahonen, A. Hadid, and M. Pietikäinen, “Face recognition with local binary patterns,” in *European conference on computer vision*, 2004, pp. 469–481.
- [13] “Face Recognition with OpenCV — OpenCV 2.4.13.5 documentation.”  
[https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec\\_tutorial.html](https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html)
- [14] S. Liao, X. Zhu, Z. Lei, L. Zhang, and S. Z. Li, “Learning multi-scale block local binary patterns for face recognition,” in *International Conference on Biometrics*, 2007, pp. 828–837.