University of Belgrade

School of Electrical Engineering

An efficient SoC FPGA implementation of face detection SVM algorithm

Master's thesis

Supervisor:

doc. dr Jelena Popović Božović

Author:

Aleksa Damljanović, 3064/2016

Belgrade, September 2017.

Abstract

Face detection is an interesting and challenging problem of identifying and locating faces, regardless of their position, scale, in-plane rotation, occlusion, orientation, pose (out-of-plane rotation) and illumination. It is an important step in biometric identification, surveillance systems, etc. With the technological advancements made in hardware development and growing processing power, this has become an important research area. As an noninvasive procedure that can be used almost without interaction, it will be even more exploited in the future. Aim of the thesis is to introduce cascaded SVM (Support Vector Machine) algorithm for face detection together with the efficient hardware implementation of the algorithm achieving satisfying speed while maintaining the reliability and precision of the model obtained using Matlab tool. HLS (High Level Synthesis) will be used for implementing the system on AP SoC XC7Z010-1CLG400C. Xilinx's Vivado tool is used for development and testing.

Keywords: face detection, machine learning, FPGA, SVM, training, algorithm, cascaded

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During last twenty years, computational power of processing units inside electronic devices has increased significantly. Although Moore's law is still valid and the number of transistors per mm^2 is still increasing by utilizing different techniques of packaging, there are research groups focused on using different semiconductor materials as well as utilizing vertical dimension and 3-D space for aligning more levels of transistors. Heterogeneous integration of different technologies in a limited amount of space has revolutionized the industry. Power consumption reduction has replaced improving performance as a main driver for the IC design. This can be proven by the fact that the SoC and SiP products are main semiconductor industry drivers, because the total number of smartphones and tablets during last several years has surpassed production capacities of microprocessors. Due to improved hardware characteristics, executing more demanding tasks is now possible. For example, face detection application exists in the software of digital cameras.

Nowadays, security is a major concern, not only in public areas, like airports, railways stations, government buildings, but also at private properties. Modern cities are adopting the latest models of development including Smart

City goal together with Internet of Things. They utilize distributed systems that process large amounts of data. Identification of people using their face is one of the many subsystems. It has many advantages over standard more intrusive techniques. However, image processing is considered to be both time and resource consuming due to the large amount of managed data. Image processing was exploited on different platforms including usual microprocessors in software and DSPs. In the last decade, processing images is done with FPGAs and parallel computing with GPUs. The main focus of this master thesis is on building a reliable face detection model and its hardware implementation.

This Master's thesis has a following structure: In Chapter 2, face detection is presented as well as its applications and existing (current) solutions to this problem. Different approaches, algorithms are presented and compared. In Chapter 3, main focus is on the Machine Learning. General concepts are described and explained, as well as the mathematical theory behind support vector machine algorithm used for developing a model. Chapter 4 describes the process of obtaining a model using Matlab tool: selecting a database, preparing training and testing sets with preprocessing enhancement, training and building SVM models for different levels of complexity (types of kernels) and testing and verification of the cascaded system. The main purpose of Chapter 5, is to provide insight in both hardware and software implementation of the system on SoC. All processing blocks included in the final design are given with their specifics. Numerical results of the verification that was conducted to make a comparison between the implementation and the golden(reference) model are given. Chapter 6 is focused on concluding, outlining everything that was done for the purpose of writing the thesis and giving main contribution of the thesis.

Chapter 2

Face detection

Face detection is an interesting and challenging problem of identifying and locating faces, regardless of their position, scale, in-plane rotation, occlusion, orientation, pose (out-of-plane rotation) and illumination (Figure 2.1). Moreover, its appearance can vary depending on various appearance features such as facial expression, age, presence of glasses, makeup, facial hair etc. Its importance lies in a fact that face detection represents first step for any face recognition system, and therefore is a first step in biometrics, access control and surveillance systems (Figure 2.2). It has received serious attention during past several years, because all the mentioned systems based on face detection and recognition have several advantages over standard ones using fingerprint and iris. Not only they can be stored for future identification, but are non-invasive and can be obtained without interaction. Face detection has a great number of applications in face tracking, facial expression recognition, gender classification, auto-focus in cameras, human-computer interfaces, etc. Face is a highly dynamic, non-rigid object, so this kind of processing is a step towards generic object recognition (cars, humans, pedestrians, signs, etc.) With hardware development and increase in com-

putational power this problem has gained interest and become an important research area. Face detection is the fundamental technique to the entire facial analysis algorithms, including face alignment, face relighting, face modeling, face recognition, head pose tracking, face verification/authentication, facial expression tracking/recognition, gender/age recognition, etc.

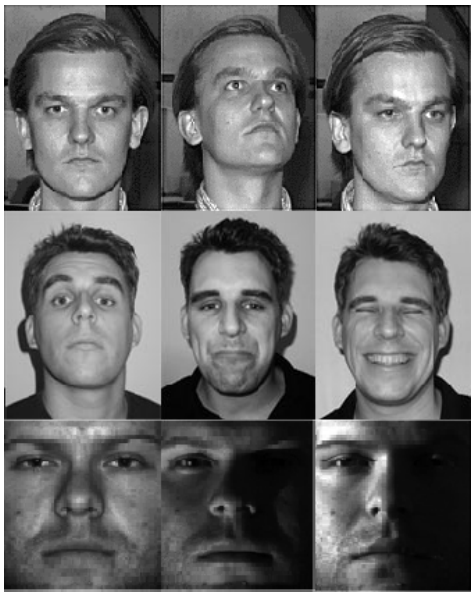


Figure 2.1: Face image under different conditions

Figure 2.2: Biometry application

In the past 20 years, many authors have discussed and evaluated face detection algorithms that are still being used. Nowadays different feature extraction techniques and different algorithms are being combined in order to improve accuracy and robustness of the results. Apart from exploring new methodologies, already existing algorithms are being made more effective.

There are different classifications of algorithms used for face detection, Yang [1] has proposed four major categories: Knowledge-based, Feature invariant, Template matching and Appearance-based.

Knowledge-based methods use human knowledge of what constitutes a typical face. Facial features and their inter-dependencies are being are being used to capture the information. These methods are designed mainly for face localization. For example, a face consists of two eyes, a nose and a mouth, all in certain position to each other.

Feature invariant methods rely on a fact that humans can identify faces regardless of their pose or lightning effects. It is possible because features invariant to these variations exist. Usually certain facial features such as eyebrows, eyes, nose, mouth, and hair-line are extracted using edge detectors and statistical model is being built. Sometimes, due to the illumination, noise, and occlusion these facial features are extremely corrupted, resulting in degrading detector's performance.

Template matching methods corresponds to predefining or parametrizing a standard face pattern with a function. Input image's class is being determined by calculating the correlation between this image and face contour, eyes, nose, mouth separately. Although simple to implement, sometimes the results are not satisfactory because of model's intolerance to different scale, pose and shape.

Appearance based methods use data sets to train models and have shown superior results in comparison to other methods. They rely on statistical analysis and machine learning. Learned characteristics are in the form of distribution models or discriminant functions that are used for face detection. One of the face detection algorithms that are extensively used and have excellent results in face detection is Viola Jones algorithm[2].

2.1 Some of the current solutions

2.1.1 Viola Jones face detection

This is a widely used method for real-time object detection, proposed in 2001 by P. Viola and M. Jones [2]. It is mainly used to detect faces, although it can be used for other objects as well. Although training process is slow, detection is very fast and accurate. Training set consists of 5000 frontal face patterns and 300 million non-face patters obtained from 9400 images. All the faces are normalized (mean-translation and standard deviation-scale) with many variations such as illumination, pose, and across individuals. It is especially successful method for face detection with very low false positive rate. There are 4 main concepts proposed in this method:

- Haar Features – All human faces share some similar properties. These regularities may be matched using Haar Features.

- Using integral image, that enables us to compute the features used by the detectors very quickly. It corresponds to calculating a value at each pixel (x, y) that is the sum of the pixel values above and to the left of (x, y) pixel. Having the values of the integral image at the corners of any rectangle, can be used to calculate the sum of original image vales in only 3 additions.

- Building small and efficient classifier by using Ada Boost algorithm [3]. Relevant features are being distinguished from irrelevant ones. In each round, the optimal visual features are selected based on the previously selected features and exponential loss. The features are called weak classifiers. Ada Boost algorithm builds a strong classifier as a linear combination of the weak ones.

- Classifiers are cascaded in order to quickly reject background regions, while the possible face patterns are being processed through the different stages.

This algorithm is implemented in OpenCV.

2.1.2 Neural network face detection

This face detection system consists of two stages. The first one, applies set of neural network-based filters to image and then decides the filter output, whether input sample is a face or non-face image. Different preprocessing steps are applied to the input window before feeding the network. The filters examine each location in the image at several scales, looking for locations that might contain a face. There are few different types of hidden layers in the network, that have different shapes of input window's subregions. To reduce the number of false detections second stage is introduced. This stage is used for merging detections from individual filters and eliminating overlapping detections. In addition to using large number of images for training, non-face images are collected during the training. The computational complexity is low. This algorithm has detection rate between 78.9% and 90.5% of faces in a test set of 130 images with an acceptable number of false detections.

2.1.3 Face detection using Support Vector Machine

Support Vector Machine algorithm is an appearance based method, which has recently gained certain popularity. Although, the computation is both time and memory intensive, there have been different papers regarding developing fast, efficient and accurate face detection systems using support vectors [6, 9, 10, 11, 15, 21].

Chapter 3

General concepts of Machine Learning techniques for real world applications

3.1 Machine Learning

Nowadays we live in a technologically advanced era. As a result of technological breakthrough and development of embedded system and more and more popular IoT (internet of things), everything we do and every choice we make, is being recorded. Wherever we look we see data. Unimaginably enormous quantities of raw (digital information) data, potentially useful data, which is increasing every day faster than ever. Politics, economics, weather forecasting, health-care and medicine, commerce and industry, not only produce data, but also rely on this data. Data mining is about solving problems by analyzing data already present in databases. It is defined as the process of discovering patterns in data. The process must be automatic or (more usually) semiautomatic. The patterns discovered must be meaningful in that

they lead to some advantage, usually an economic advantage. The data is invariably present in substantial quantities. With more data come more problems and more questions to be answered. Data mining and machine learning are the answer to it. What is Machine Learning? There are multiple definitions: Machine learning represents a way of teaching based on experience, exactly what is given by birth to humans and animals. These algorithms lean on computational methods in order to obtain information directly from the data. The algorithms adaptively improve their performance with increase of the samples used for the learning process, without relying on a predetermined equation as a model.
Subfield of computer science that gives computers the ability to learn without being explicitly programmed (A. Samuel, 1959)

With development of hardware, its resources, speed, reliability, availability, cost and performance, this method has found its place for solving problems in a real-world application. These concepts mean designing a system with some specific abilities that can be more efficient and even more reliable than humans.

- Image processing and computer vision, face recognition, motion detection and object detection
- Computational biology, for tumor detection, drug discovery and DNA sequencing
- Energy production, price and load forecasting
- Automotive, aerospace – manufacturing and predictive maintenance
- Computational finance, for credit scoring and algorithmic trading

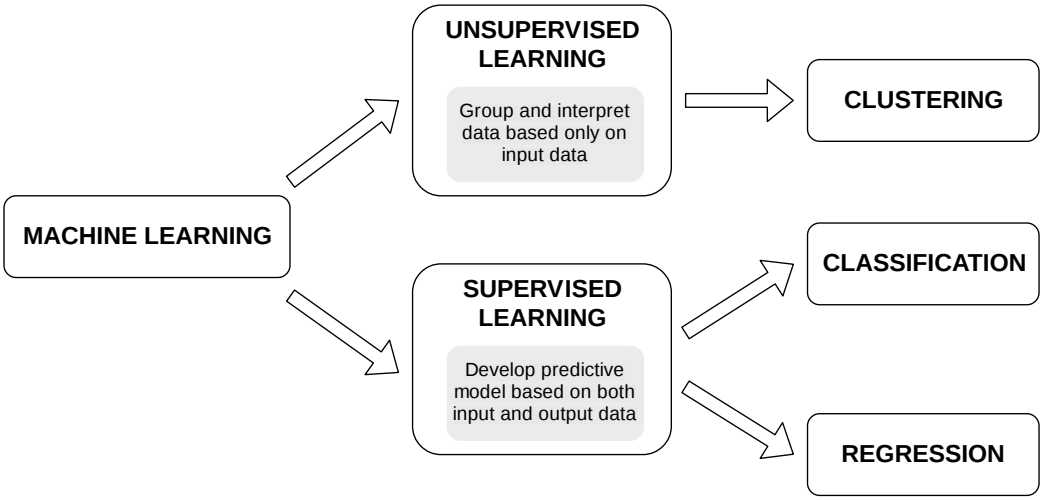


Figure 3.1: Machine learning

As it can be seen from the Figure 3.1 we can identify types and subtypes of machine learning.

Supervised learning represents learning using labeled data sets which consists of input data (value of attributes, usually an array of real numbers) and their corresponding output data (Figure 3.2). Obtained model, after the process of training, has the purpose of calculating the result, thus giving reasonable prediction for the unlabeled data sets. Obtained response or model's output value could have discrete or continuous value. Therefore, we can distinguish two subtypes of supervised machine learning:

- **Classification**
  - Gives discrete output values, which are categorical classes, for example it can decide whether a tumor is benign or malignant, an email is genuine or spam ... Applications include spam filters, advertisement recommendation systems, and image and speech recognition.

- **Regression**
  - Gives continuous output values (the response variables are real numbers), for example changes in temperature or fluctuations in power demand. Applications include forecasting stock prices, energy consumption, or disease incidence.

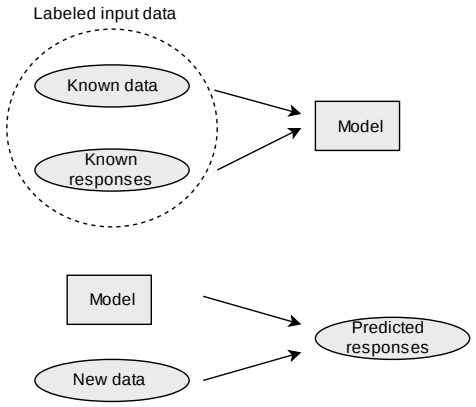


Figure 3.2: Supervised learning and evaluation principle

Unsupervised learning uses data without responses – unlabeled data in order to find certain hidden patterns and to draw conclusions. Most commonly used is Clustering, used for exploratory data analysis. The inputs are being divided into groups and the algorithm finds the common features (Figure 3.3). It found its purpose in bioinformatics with gene sequence analysis, in market research, and computer vision object recognition.

Selecting specific technique depends on the problem that needs to be solved. In this case we have to design a system which would recognize is a certain object (face) present in the image. Therefore, the only two possible answers would be Yes or No. Additionally this feature makes it a classification system, which has discrete responses (1 if object is recognized, and 0 otherwise). To this end we need a labeled set of data for training and obtain-

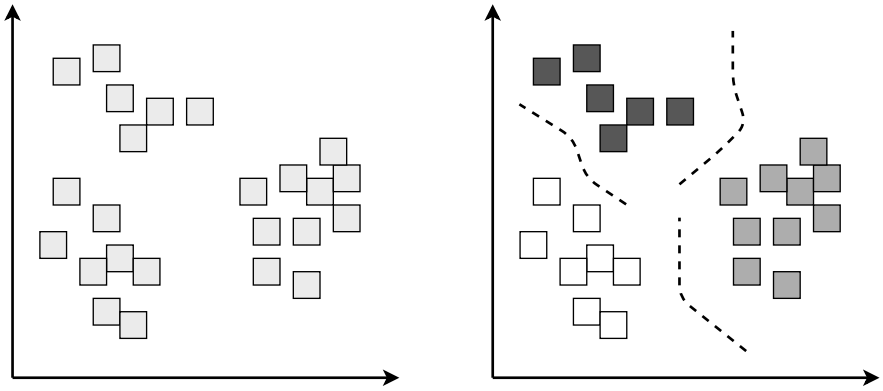


Figure 3.3: Unsupervised learning - Clustering

ing the model. The type of algorithm that has been selected is SVM (*Support Vector Machine*). Although it is a classification algorithm with good performance for state of the art applications, often used for binary classification problems, it requires high computational effort due to the dimensionality of the problem.

3.2 SVM (*Support Vector Machine*) algorithm

SVMs, close to their current form were first introduced with a paper at the COLT 1992 conference. It was presented as a training algorithm that maximizes the margin between the training patterns and the decision boundary (Boser, Guyon, Vapnik, 1992, A Training Algorithm for Optimal Margin Classifier). This algorithm has been developed from the Statistical Learning Theory in the 1960's. (Vapnik, The nature of the Statistical Learning Theory, 2nd edition, Springer, 1999). Excellent results, including outstanding accuracy, were obtained when applying this algorithm to handwritten digit recognition (Bottou, Comparison of classifier methods: a case study in handwritten digit recognition, 1994). Although it needs to be trained, it

runs faster and requires less space than memory-based techniques. Binary classification problem could be solved with neural network approach using perceptrons to define a border between the two classes (Figure 3.4). However, given borders may not be the optimal ones. As there are two classes, goal

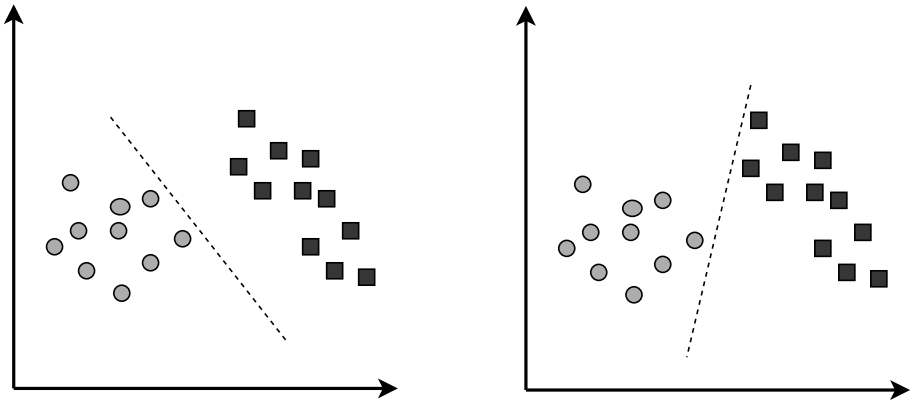


Figure 3.4: Defining a border between classes using perceptrons

of this SVM algorithm is to define a separating hyperplane (for a linearly separable pattern. SVM tend to maximize the width of margin between the support vectors. Support vectors are training samples that lie closest to the separating hyperplane (Figure 3.5). Training input for the system can be represented as a set of r elements: $\{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), (\mathbf{x_3}, y_3), \ldots, (\mathbf{x_r}, y_r)\}$, where $\mathbf{x_i}, i = 1, 2, \ldots, r$ represents a n-dimensional input sample vector with the corresponding response value $y_i, i = 1, 2, \ldots, r$ (3.1).

$$y_i = \begin{cases} 1, & \text{if } \mathbf{x} \in A \\ -1, & \text{if } \mathbf{x} \in B \end{cases} \tag{3.1}$$

In the end, after the training process is done, class of the new input data vector $\mathbf{x}$, will depend on the decision function value, $D(\mathbf{x})$, in such a way that it represents a position below or under the hyperplane that separates two classes.

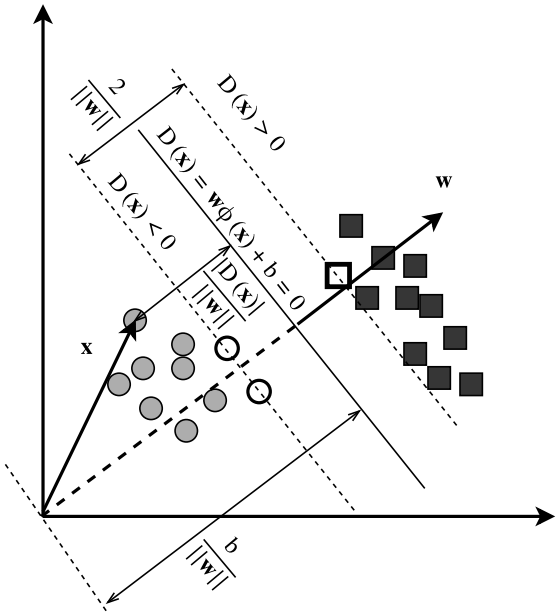


Figure 3.5: Defining a border between classes using SVMs

This function can be expressed as a linear combination of parameters (3.2),

$$D(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b = \mathbf{wx} + b \tag{3.2}$$

where w_j are coefficients, $\mathbf{x}$ is input vector and b is a bias coefficient. Conditions for discrimination between input sample $\mathbf{x_i}$ being on one (3.3) or the other side (3.4) of the hyperplane, can be unified into a single condition (3.5).

$$\mathbf{wx_i} + b \geq 1, y_i = 1 \tag{3.3}$$

$$\mathbf{wx_i} + b \leq -1, y_i = -1 \tag{3.4}$$

$$y_i (\mathbf{wx_i} + b) \geq 1 \tag{3.5}$$

$$\mathbf{wx}_+ + b = 1 \tag{3.6}$$

$$\mathbf{wx}_- + b = -1 \tag{3.7}$$

$$\mathbf{w}\left(\mathbf{x}_{+}-\mathbf{x}_{-}\right)=2 \tag{3.8}$$

$$M=\frac{\mathbf{w}}{\|\mathbf{w}\|}\left(\mathbf{x}_{+}-\mathbf{x}_{-}\right)=\frac{2}{\|\mathbf{w}\|} \tag{3.9}$$

Margin M is defined using a difference (3.8) of two samples $\mathbf{x}_{+}$ (3.6) and $\mathbf{x}_{-}$ (3.7) lying on two boundaries. The objective of this algorithm is finding the coefficient vector $\mathbf{w}$ in order to maximize the margin M (3.9). To summarize, the goal is minimizing $\frac{\|\mathbf{w}\|^2}{2}$ with the condition of correctly classifying all the points $y_i\left(\mathbf{w}\mathbf{x_i}+b\right)\geq 1$. Solution to the problem of minimizing $min_x f(x)$, with respect to $g_i(x)\leq 0$ for $i=1,\ldots,n$ is equivalent to finding the solution to Lagrangian's zero gradient (3.10).

$$\begin{cases}\dfrac{\partial}{\partial x}\left(f\left(\mathbf{x}\right)+\sum_{i=0}\alpha_i g_i\left(\mathbf{x}\right)\right)=0, \quad \exists\alpha_i>0, i=1,\ldots,n \\ g_i\left(\mathbf{x}\right)\geq 0\end{cases} \tag{3.10}$$

Comparing the method with the existing problem $f\left(\mathbf{x}\right)=\frac{\|\mathbf{w}\|}{2}$, $g_i\left(\mathbf{x}\right)=1-y_i\left(\mathbf{w}\mathbf{x_i}+b\right)$. After setting the Lagrangian's gradient to zero, we obtain essential relation between coefficients and the dual problem to be solved. That is $\max\left(W\left(\alpha\right)\right)$ (3.11), which, in the end, is a quadratic optimization problem. Its solution are α coefficients.

$$W\left(\alpha\right)=\sum_{i=1}^{r}\alpha_i-\frac{1}{2}\sum_{i=1,j=1}^{r}\alpha_i\alpha_j y_i y_j \mathbf{x_i}\mathbf{x_j},\ \sum_{i=1}^{r}\alpha_i y_i=0,\ \alpha_i>0 \tag{3.11}$$

After obtaining α_i coefficients, it is trivial to obtain $\mathbf{w}$ as well (3.12). Coefficient vector $\mathbf{w}$ is a linear combination of small number of input sample vectors, and therefore, many of the α_i coefficients are equal to zero. Input vectors $\mathbf{x_i}$ with nonzero α_i coefficients are called *Support Vectors*. From equations (3.2) and (3.12) final form for calculating the score is derived (3.13).

$$\mathbf{w}=\sum_{j=1}^{s}\alpha_{t_j}y_{t_j}\mathbf{x_{t_j}} \tag{3.12}$$

$$D(\mathbf{x}) = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}\mathbf{x}_{\mathbf{t_j}} + b \tag{3.13}$$

Having a non-linearly separable classification problem, ϵ_i deviation can be introduced (3.14).

$$\begin{cases} \mathbf{w}\mathbf{x}_\mathbf{i} + b \geq 1 - \epsilon_i, & y_i = 1 \\ \mathbf{w}\mathbf{x}_\mathbf{i} + b \leq 1 + \epsilon_i, & y_i = -1 \\ \epsilon_i \geq 0, & \forall i \end{cases} \tag{3.14}$$

Although we allow a certain misclassification, we retain the boundary margin (Figure 3.6). In addition, the optimization problem is being modified/altered in such a way that now $\frac{\|\mathbf{w}\|}{2} + C \sum_{i=1}^{n} \epsilon_i$ needs to be minimized, with the condition of classifying all input samples correctly within the modified margin $y_i (\mathbf{w}\mathbf{x}_\mathbf{i} + b) \geq 1 - \epsilon_i$. Furthermore, the conditions regarding dual problem and quadratic optimization are the same, except α_i coefficients have upper boundary $0 \leq \alpha_i \leq C, \ \forall i$. C is a parameter used to adjust the desired error/margin.

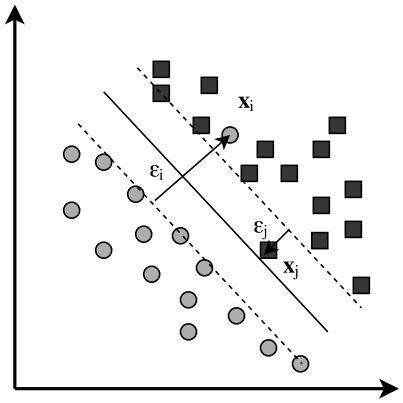


Figure 3.6: Allowing certain degree of missclassification

In many applications, constructing a hyperplane is not possible and will not result in successful classification of input data. Applying so called kernel trick technique is used to solve this particular problem. Linear kernel (3.15)

is a simplest one, corresponding to the dot-product between two vectors.It is being used to construct a hyperplane as we have seen. Those more complex, polynomial (3.16) and gaussian (3.17), map input space into a higher dimensional, linearly separable feature space (Figure 3.7). Linear operation in the feature space is equivalent to nonlinear operation in input space.

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y} \tag{3.15}$$

$$K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^p \tag{3.16}$$

$$K(\mathbf{x}, \mathbf{y}) = \exp \frac{-\|\mathbf{x}^2 - \mathbf{y}^2\|}{2\sigma^2} \tag{3.17}$$

To conclude, depending on the type of kernel function, SVM algorithm can be used for linearly separable, as well as for linearly non-separable data.

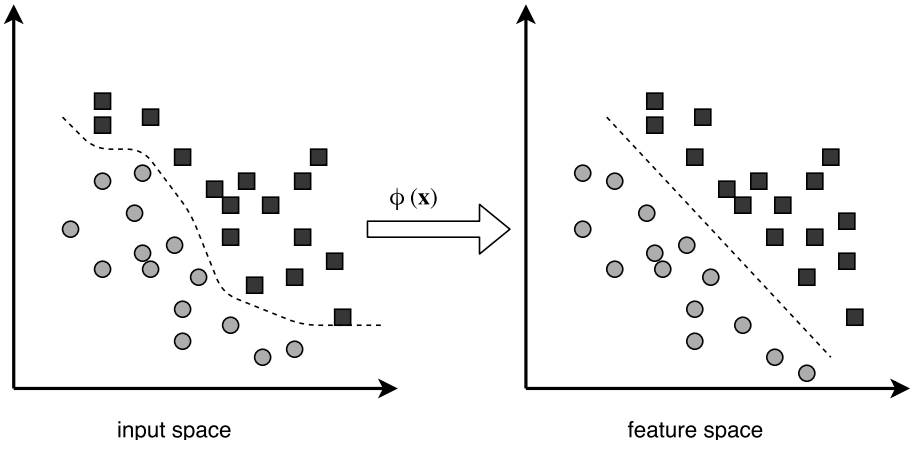


Figure 3.7: Using non-linear kernel functions

$$D(\mathbf{x}) = \sum_{i=1}^{n} w_i \Phi_i(\mathbf{x}) + b = \mathbf{w}\mathbf{\Phi}(\mathbf{x}) + b \tag{3.18}$$

The dual form() where α_i, $i = 1, \ldots, r$ and b are adjustable parameters, $\mathbf{x_i}$ support vectors and $K(\mathbf{x}, \mathbf{x_j})$ is a kernel function that can under certain

conditions be developed into a finite or infinite series expansions (3.19).

$$K\left(\mathbf{x}, \mathbf{x_j}\right) = \sum_i \phi_i\left(\mathbf{x}\right) \phi_i\left(\mathbf{x_j}\right) \tag{3.19}$$

When applying this algorithm, the kernel function in $\phi_i\left(\mathbf{x}\right)$ form is not explicitly stated. Instead, the K form is being used with inner product, that as it happens reflects the degree of similarity between two vectors $\mathbf{x_i}$ and $\mathbf{x_j}$, kernel function $K\left(\mathbf{x_i}, \mathbf{x_j}\right)$ is utilized for mapping input space.

$$W\left(\alpha\right) = \sum_{i=1}^{r} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{r} \alpha_i \alpha_j y_i y_j K\left(\mathbf{x_i}, \mathbf{x_j}\right),\ \sum_{i=1}^{r} \alpha_i y_i = 0,\ C \geq \alpha_i \geq 0 \tag{3.20}$$

$\max\left(W\left(\alpha\right)\right)$ (3.20) is also a quadratic optimization problem. Its solution are α coefficients. $\mathbf{w}$ is expressed as a linear combination of mapped input samples (3.21). *Support Vectors* are input vectors $\mathbf{x_i}$ with nonzero α_i coefficients.

$$\mathbf{w} = \sum_{i=1}^{s} \alpha_i y_i \Phi\left(\mathbf{x_i}\right) \tag{3.21}$$

The decision function value, for calculating the score and classifying the input data has a form (3.22) and it is obtained using equations (3.18),(3.21) and series expansion (3.19).

$$D\left(\mathbf{x}\right) = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} K\left(\mathbf{x}, \mathbf{x_{t_j}}\right) + b \tag{3.22}$$

Chapter 4

Obtaining a face detection model in Matlab

4.1 Training dataset

Important step to consider was finding the appropriate database among the substantial number of available ones. Some of the often-used databases, whose review is given in Table 4.1, are free to use, others require authorization.

Database used in this work contains face and non-face images. It has been used extensively at the Center for Biological and Computational Learning (CBCL) at MIT [15],[16], [17]. This database contains 2 sets, including one training and one testing set. Training set contains 2429 face and 4548 non-face images. Testing set contains 472 face images and 23573 non-face images. Size of the grayscale images is 19x19 pixels. Proposed size of sub-window, 20x20 pixels is used, in order to keep the small dimensionality of feature space and still keep the structural details which make the difference between face and non-face patterns. Therefore, all images in the database are being

Table 4.1: Databases for face detection/identification

Database name	Database description
The Color FERET Database	1654 sets of images for a total of 14126 images that include 1199 individuals and 365 duplicate sets of images
SCface-Surveillance Cameras Face Database	Contains 4160 static images (in visible and infrared spectrum) of 130 subjects. Images obtained from different cameras to mimic real-world conditions.
SCfaceDB Landmarks	21 facial landmarks (from 4160 face images) from 130 users annotated manually
Multi-PIE	Limited number of subjects, a single recording session and only few expressions captured. Contains 337 subjects, captured under 15 view points and 19 illumination conditions in four recording sessions for a total of more than 750000 images.
The Yale Face Database	Contains 165 grayscale images in GIF format of 15 individuals. There are 11 images per subject, one per different facial expression or configuration.
The Yale Face Database B	Contains 5760 single light source images of 10 subjects, each seen under 576 viewing conditions (9 poses x 64 illumination conditions

PIE Database, CMU	A database of 41368 images of 68 people, each person under 13 different poses, 43 different illumination conditions, and with 4 different expressions.
MIT-CBCL Face Recognition Database	Face images of 10 subjects. Contains two training sets: 1.High resolution pictures, including frontal, half-profile and profile view; 2. Synthetic images (324/subject) rendered from 3D head models of 10 subjects.
Image Database of Facial Actions and Expressions-EID	24 subjects are represented in this database, yielding between about 6 to 18 examples of the 150 different requested actions.
M2VTS Multimodal Face Database	37 different faces and provided 5 shots for each person.
AT&T "The Database of Faces"(ORL)	40 subjects with 10 images per each one. The images were taken at different times, varying the lightning, facial expressions and facial details.

rescaled from 19x19 to 20x20. As certain variations exist in images, there are different techniques which are proposed in order to reduce or eliminate these effects: masking, illumination gradient correction and histogram equalization. Masking includes applying binary pattern pixel mask to remove border pixels thus reducing dimensionality and removing background noise. Illumination gradient correction corresponds to subtraction of the brightness plane

from the original image to remove non-uniformities in illumination which are often present. Histogram equalization technique is used to enhance image contrast by using its histogram and adjusting pixel values. After visual inspection of the database it was decided no significant improvement will be achieved by using illumination gradient correction. As we did not want too complex, resource consuming preprocessing with providing just a slight enhancement, histogram equalization was selected as the most efficient and aggressive. Number of face images in training set was virtually increased 3 times by rotating every image 10 degrees clockwise and anti-clockwise (Figure 4.1). This has resulted in the classifier being less sensitive to rotation.

Figure 4.1: Training faces

4.2 Cascaded system structure

For problems with high dimensional input vectors and a rather great number of support vectors, although high accuracy can be achieved, this comes at high computational complexity cost. As has been proposed in [7], [8] and [9], classification system with cascaded stages can significantly improve classification speed, which is extremely important in embedded application. Using simple kernel at first stage will reduce significantly the amount of data to be processed by later stages. Afterwards, more advanced kernels, with higher accuracy and higher computational complexity will be applied for the purpose of classifying the data that was rejected by previous stages. Every

cascade SVM is trained by the images which are not rejected through the previous classifiers (Figure 4.2).

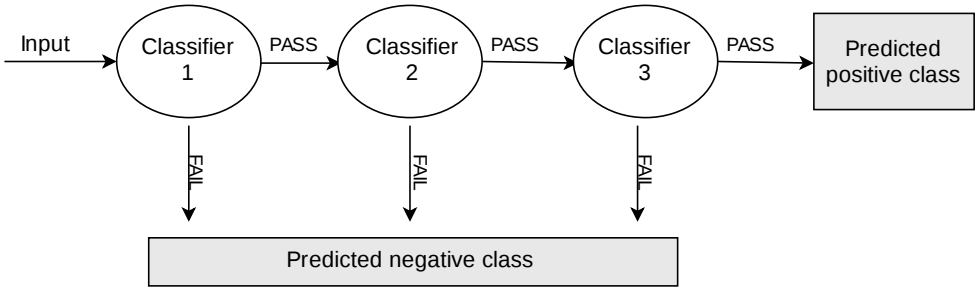


Figure 4.2: Training faces

We perform training in an incremental way, beginning with the first linear classifier and using the whole training set. The input for training first linear SVM stage is a 401-dimensional vector with 400-pixel values and 1 value for sample response. All the images were reshaped by reading image column by column. If image represented a face, value 1 was added, while for the non-face images value 2. Stages with simple kernel and without feature extraction have lower precision and can have a negative impact on cascaded output accuracy. Therefore, more relaxed decision function is used to reject definitely negative samples, and to pass the negative samples with positive score and samples that belong to the positive class but have both positive and slightly negative score [17]. For each stage of the classification ROC (receiver operating characteristic) curves were obtained. Changing the threshold has an influence on four characteristic variables:

- TP *true positive* (face images that were recognized as faces)
- FP *false positive* (non-face images that were recognized as faces)
- TN *true negative* (non-face images that were not recognized as faces)

- FN *false negative* (face images that where not recognized as faces)

Confusion matrix was formed using these results. Sensitivity is the ability to correctly identify the face images (proportion of positives that are correctly classified as such)(4.1). Specificity is ability to correctly identify the non-face images (proportion of negatives that are correctly classified as such).(4.2) ROC curve shows the influence of classification threshold on performance of a binary classifier such as this one. It is created by plotting true positive rate (*sensitivity*) against false positive rate (1-$specificity$).

$$sensitivity = \frac{TP}{TP + FN} \tag{4.1}$$

$$specificity = \frac{TN}{TN + FP} \tag{4.2}$$

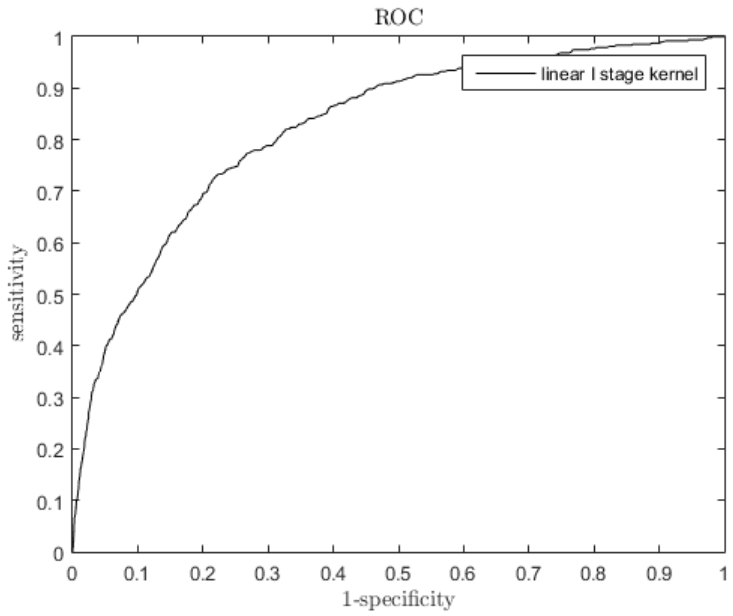


Figure 4.3: ROC curve for the 1st stage

Score of $C_{lin} = -2.5$ was set for the first classifier, given the response of the first stage linear classifier for the data used for training. All the input

samples that have score greater than C_{lin} are used for training the second stage with polynomial kernel that has greater accuracy and therefore, higher discrimination ability. Moreover, all input samples that are to be evaluated, will pass the first stage if their score is higher than C_{lin}.

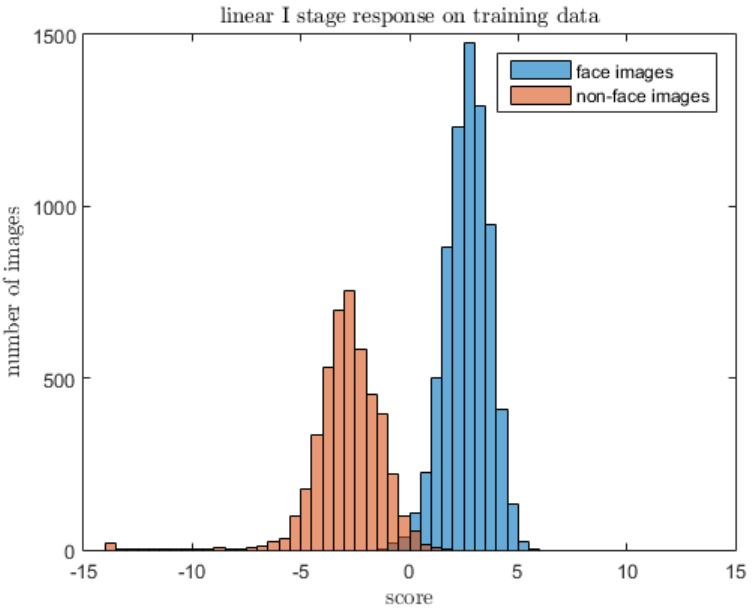


Figure 4.4: Separating training samples

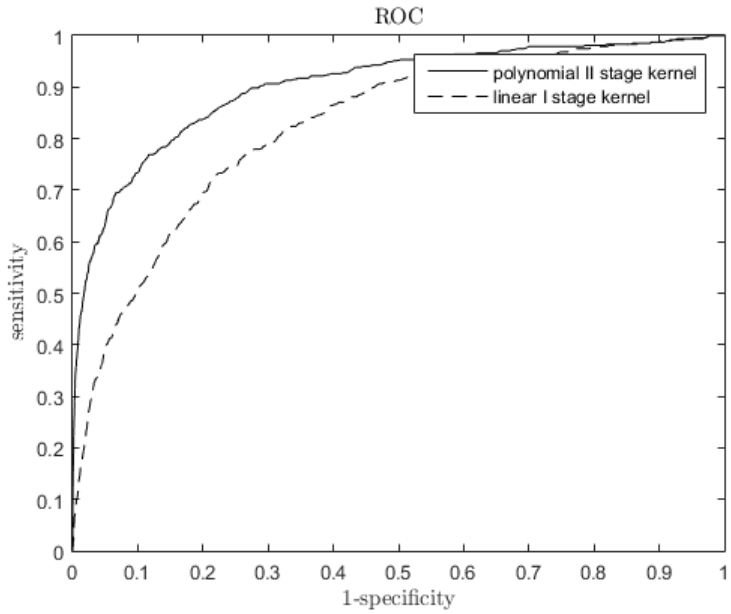


Figure 4.5: ROC curves for 1st and 2nd stage

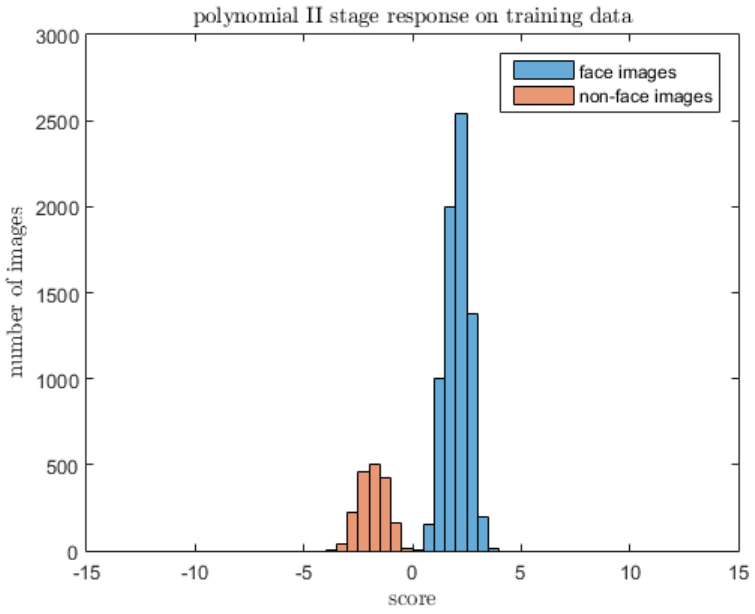


Figure 4.6: Separating training samples

Score of $C_{poly} = -1$ was set for the second classifier, given the response

of the second stage polynomial classifier for the data used for training. All input samples that are to be evaluated, will pass the second stage if their score is higher than C_{poly}.

As a final stage, in order to improve classification results, LBP (Local Binary Pattern) image transformation was used as a feature extraction.

4.3 LBP (*Local Binary Pattern*)

This very efficient texture operator was introduced by [18]. The local binary pattern texture method has been selected as a feature extractor due to its excellent results in many applications [19],[20],[21]. One of its most important features are computational simplicity, robustness and tolerance against monotonic gray-scale changes and discriminative power. The third final stage is a polynomial support vector machine with LBP histogram vector input. This stage is more efficient and more successful in separating face from non-face patterns. The process of extracting the LBP histogram is not computationally demanding although, the process of calculating the score is, due to the high number of support vectors and polynomial kernel complexity. However, only small number of input image sub-windows reach this final stage. The input to this stage is an image LBP histogram. LBP histogram of image is produced by moving 3x3 pixel mask over the image. Values of the pixels inside the mask are being compared to the central pixel value. After thresholding, the resulting binary map is multiplied by the powers of 2. These values are summed in order to obtain LBP code (Figure 4.7). The histogram of these $2^8 = 256$ different labels can then be used as a texture descriptor.

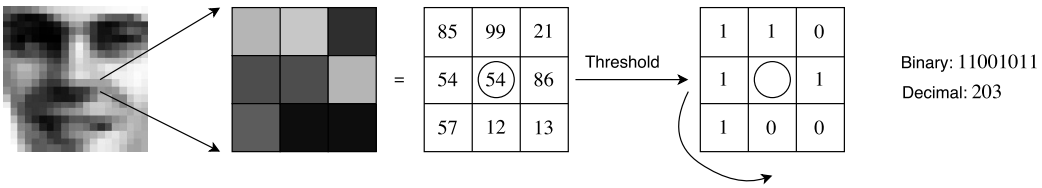


Figure 4.7: Extracting LBP code

$$g_p = I\left(x_p, y_p\right), p = 0, 1, 2, \ldots, P-1 \tag{4.3}$$

$$x_p = x + R\cos\left(\frac{2\pi p}{P}\right) \tag{4.4}$$

$$y_p = y - R\cos\left(\frac{2\pi p}{P}\right) \tag{4.5}$$

Image local texture depends on the distribution of gray values of P+1 pixels. Subtracting the central pixel value will not result in loss of information.

$$t\left(g_c, g_0, g_1, \ldots, g_p\right) \tag{4.6}$$

$$t\left(g_c, g_0 - gc, g_1 - g_c, \ldots, g_p - g_c\right) \tag{4.7}$$

As central pixel is statistically independent on the differences, it does not contain useful information and () is used for modeling the local texture. The differences are invariant to the change of image mean gray value, but are not to local changes. That is why threshold is used to minimize the influence of these differences.

$$t\left(g_0 - gc, g_1 - g_c, \ldots, g_p - g_c\right) \tag{4.8}$$

$$t\left(s\left(g_0 - gc\right), \left(g_1 - g_c\right), \ldots, \left(g_p - g_c\right)\right), \; s\left(z\right) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases} \tag{4.9}$$

$$LBP_{P,R}\left(x_c, y_c\right) = \sum_{p=0}^{P-1} s\left(g_p - g_c\right) 2^p \tag{4.10}$$

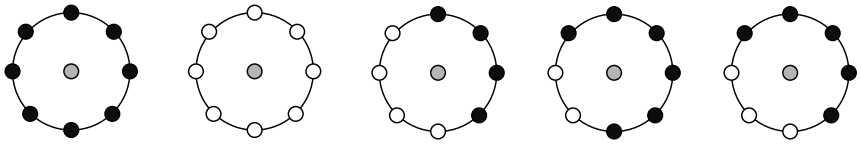


Figure 4.8: Uniform and non-uniform LBP patterns

There are two types of patterns: uniform and non-uniform (Figure 4.8). Uniform patterns contain no more than two bitwise transitions from 0 to 1, or vice versa (for example 11111111, 00000000, 1111000, 10001111...). All other patterns are considered to be non-uniform (10010010, 01000100, 11101101...). In case of 256 different values (8-bit code) 58 values are uniform. Other ones are considered as non-uniform and are grouped into one additional label, all together. Therefore, we have 59 bins in every histogram. There are two main reasons for extracting only uniform patterns. Uniform-patterns are more common, they appear in around 90% of all patterns. When dealing with facial images, it was found that 90.6% of all patters are uniform[21].

Considering that extraction of the LBP histogram for the entire 20x20 pixel image will result in loss of spatial information, coding the texture information should consider keeping the spatial as well. This can be achieved by building several local descriptors and combining them in single global descriptor. The procedure for extracting LBP histogram implies dividing facial image into local neighborhoods. 20x20 pixel image was divided in 18 blocks. Each block has 6 rows and 3 columns (4.9). The local histogram with 59 bins was calculated for every block. Afterwards, these histograms were concatenated to form a single global histogram, represented as an array of length $59 \cdot 18 = 1062$ (4.9). LBP histogram has been extracted from all images in training set for the purpose of training the last cascade part.

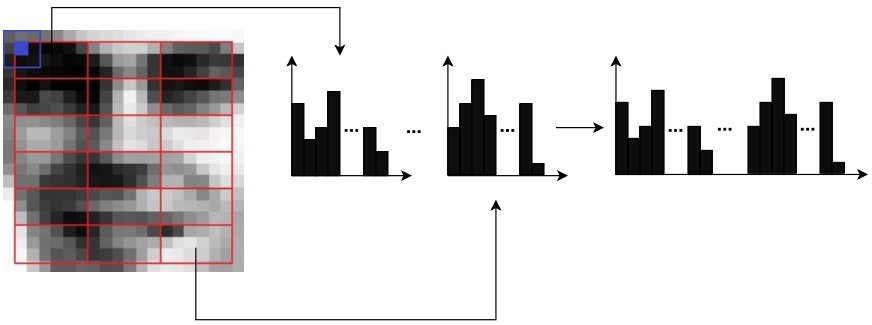


Figure 4.9: Block division and histogram concatenation

The last cascade stage, with polynomial kernel is trained on the whole training set, on which was first applied LBP histogram extraction process (Figures 4.10 and 4.11) .

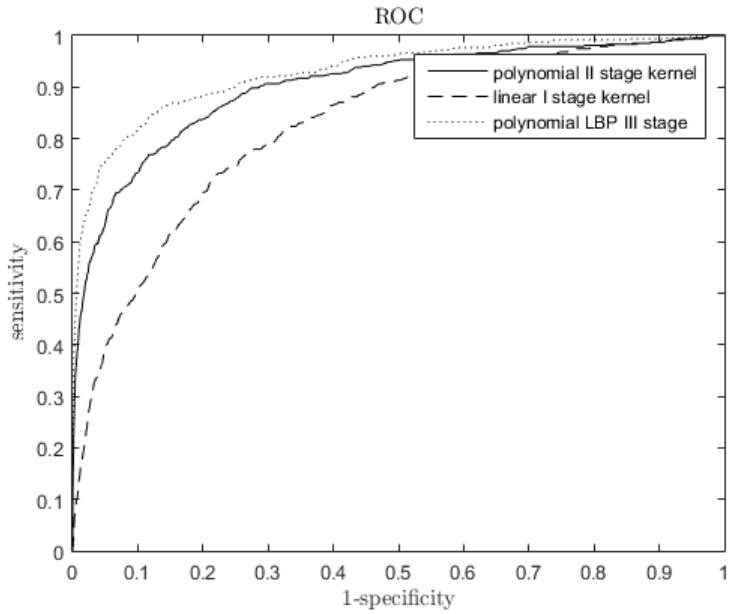


Figure 4.10: Block division and histogram concatenation

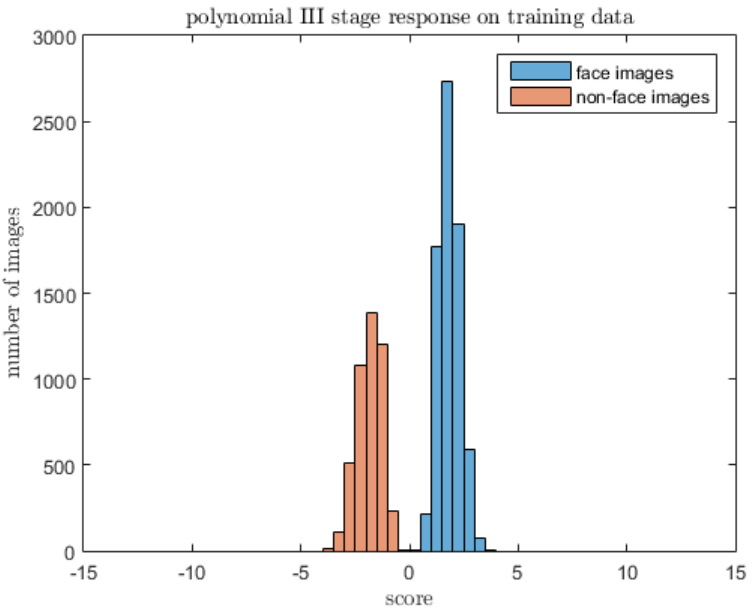


Figure 4.11: Block division and histogram concatenation

4.4 Testing the cascaded SVM model

Testing set contains low-quality images significantly degraded, differing in various attributes from the ones used for training (Figure 4.12). As a result, only 45.6% of face images were recognized as such. On the other hand, 99.9% of non-face images were classified correctly. The results are shown using confusion matrix (Figure 4.13).

Figure 4.12: Degraded face images from testing set

According to equation (3.13), due to dimensionality of input samples, first stage decision function is defined by 400 coefficients. However, second

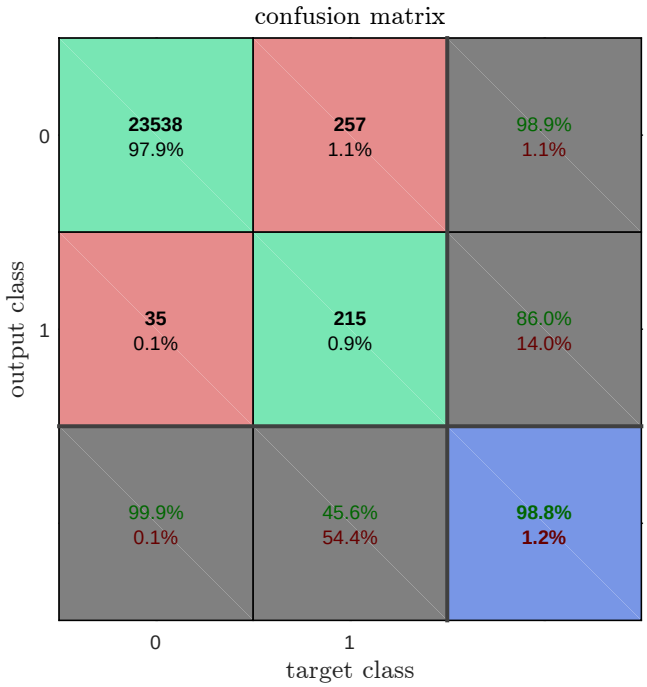


Figure 4.13: Confusion matrix for cascaded system for the testing set

and third stage have different number of support vectors: second stage has 533 SVs, while third stage has 875 SVs. Burges suggested using reduced-set-method [5], to reduce the number of nonlinear kernel SVs in order to further reduce time needed to classify input patterns. In this work, we did not consider using reduction techniques. For this reason, original coefficients and SVs were used for classification, although, the hardware, as it will be shown later, will be designed in such a way that will support the variable number of support vectors if the reduction technique is to be applied in the future.

With fully trained cascaded support vector machine, process of finding face patterns in input image can be decomposed in such a way that along with rescaling the input image, each of these scaled images is being continuously scanned with overlapping fixed-size sub-windows to determine the pattern's

class (face/ non-face)[22]. The system is tested on different images (if portrait resized proportionally to 400 pixels width, if landscape, resized to 400 pixels height). Images are being scanned on 12 different scales (original image scaled down by factor 0.85), while extracting 20x20 pixels sub-windows with 5 pixels window step.

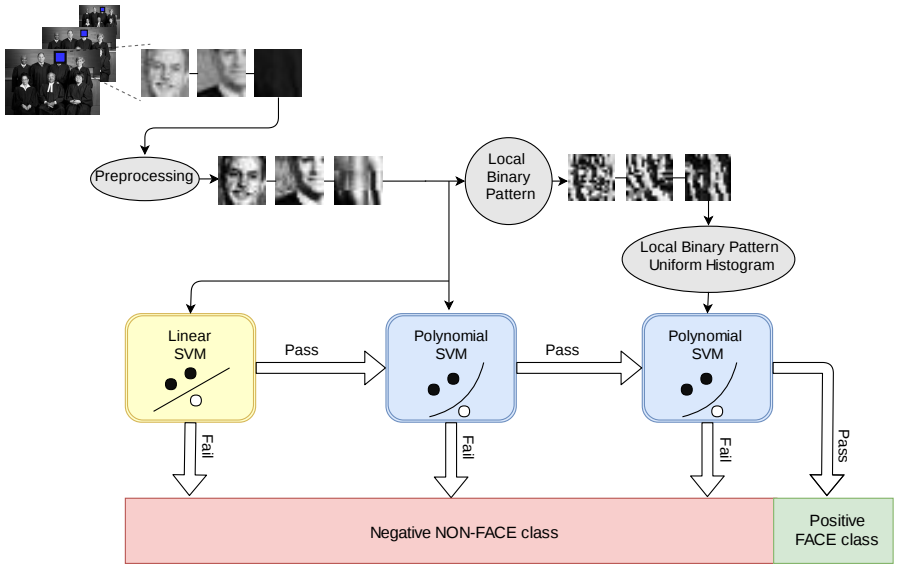
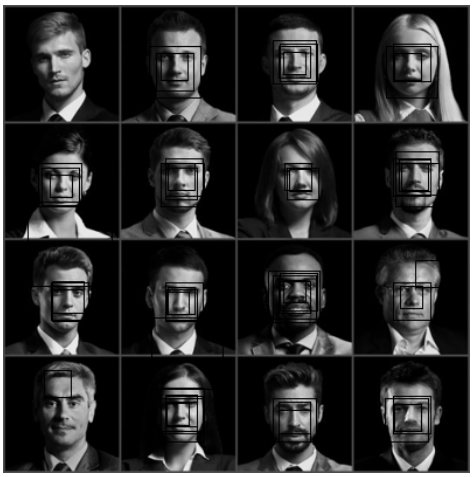
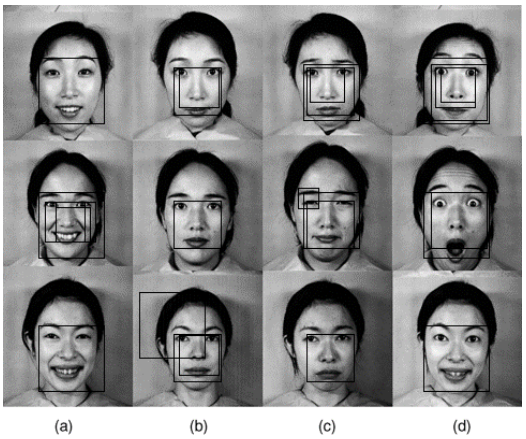


Figure 4.14: Process of detecting faces using cascaded SVM system



(a) Frontal male/female faces



(b) Different face expressions

Figure 4.15: Testing images n1 and n2

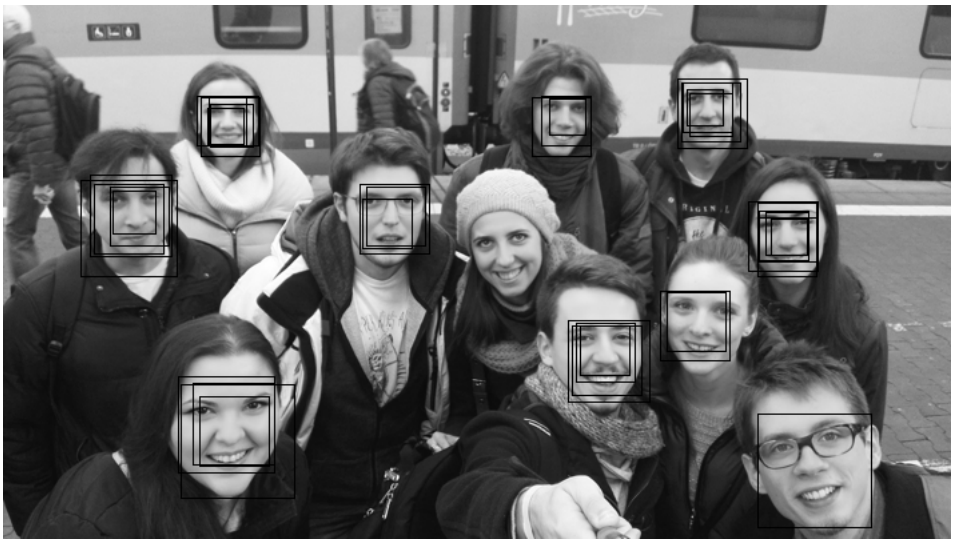


Figure 4.16: Testing image n3

(a) Frontal, large number of faces

(b) Frontal large number of faces

Figure 4.17: Testing images n4 and n5

Test set was manually created. Some photos were randomly selected from personal image collection, others were downloaded from the Internet: 43 images were used, with 288 faces in total. Using the cascaded system has resulted in detecting 87.5% of all the faces. Out of 288 faces, 252 were classified correctly, as a true positive. On the other hand, there were 67 false positive samples. When taken into account, that the total number of sub-

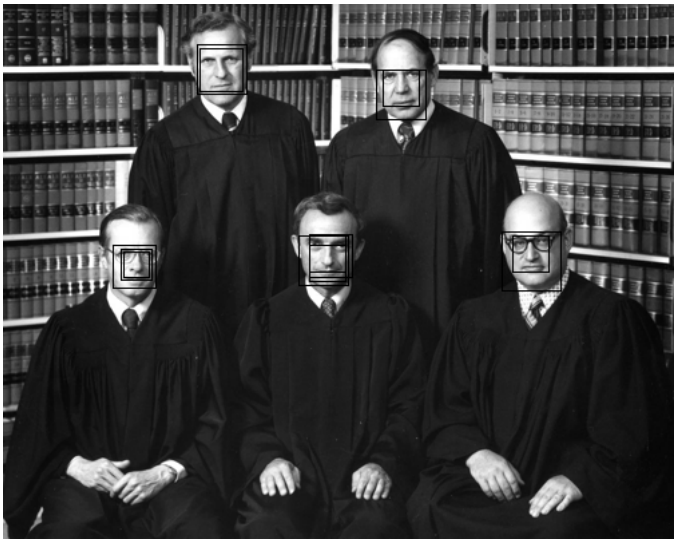


Figure 4.18: Testing image n6

Figure 4.19: Testing image n7

windows processed is equal to 1315000, an amount of false-positive samples of 0.0051% is almost neglectable. All the samples, 1315000 (100%), were processed at the first stage, 167341 samples (12.73%) were processed at the second stage, and finally, only 16248 input samples (1.23%) were process at

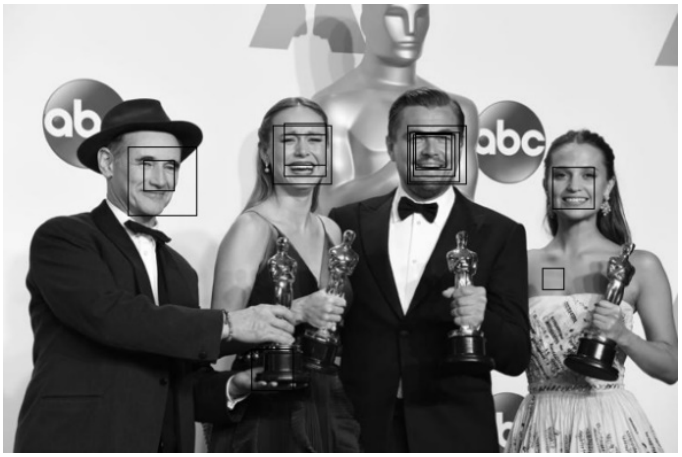
(a) Image containing rotated faces

(b) Image containing faces with closed eyes

Figure 4.20: Testing images n8 and n9

the third, most-complex stage. At the Table 4.2 some more precise data is given for the testing image results on Figures (4.15-4.21).

(a) Different skin types



(b) Out-of-plane face rotation

Figure 4.21: Testing images n10 and n11

Table 4.2: Detailed processing results

Image	Dim.	I	II	III	Time[s]	TP	FP
n1	400x400	19896(100%)	6.06%	0.80%	326.415	14/16	3
n2	400x484	24331(100%)	11.33%	1.34%	419.244	12/12	2
n3	400x710	36293(100%)	12.38%	1.16%	610.381	10/11	0
n4	400x800	41055(100%)	12.82%	1.41%	680.676	10/11	0
n5	400x670	34291(100%)	12.91%	1.20%	584.655	15/19	0
n6	400x502	25179(100%)	16.92%	2.20%	483.430	5/5	0
n7	400x494	24781(100%)	11.97%	0.96%	423.839	6/6	1
n8	400x600	30416(100%)	12.71%	1.06%	494.924	7/7	0
n9	400x712	36412(100%)	15.83%	1.64%	687.831	6/6	2
n10	400x858	44134(100%)	10.84%	0.82%	696.199	5/5	4
n11	400x601	30555(100%)	10.36%	0.86%	445.042	4/5	2

Chapter 5

Hardware implementation

Hardware implementation of the system was developed on Zybo board (Figure 5.1). Zybo board is an embedded software and digital circuit development platform with smallest member of Zynq-7000 family, Z-7010. The Z-7010 is based on the Xilinx All Programmable System-on-Chip (AP SOC) architecture which tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series programmable gate array (FPGA) logic. The review of some of its most important features and possibilities is given in Table 5.1.

Before considering the design of the face-detection system implementation, all data structures obtained in Matlab were converted from floating-point to fixed-point arithmetics.

Table 5.1: Zybo board features

Xilinx Zynq-7000 (XC7Z010-1CLG400C)
240 KB Block RAM
28,000 logic cells
512 MB x32 DDR3 w/ 1050Mbps bandwidth
650 MHz dual-core Cortex™-A9 processor
80 DSP slices
DDR3 memory controller with 8 DMA channels
High-bandwith peripheral controllers: 1G Ethernet, USB 2.0, SDIO
Low-bandwidth peripheral controller: SPI, UART, I2C
On-board JTAG programming and UART to USB converter
microSD slot (supports Linux file system)
GPIO: 6 pushbuttons, 4 slide switches, 5 LEDs

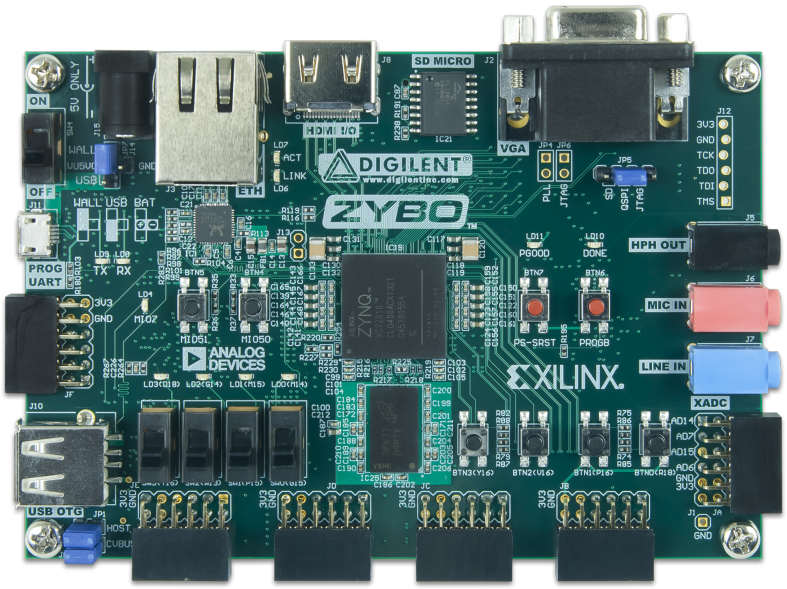


Figure 5.1: Zybo development board

5.1 Floating point to fixed point conversion

Although Xilinx All Programmable devices and tools support a wide range of data types, designs implemented in fixed-point will be more efficient than their equivalent in floating-point. In addition to higher power consumption, floating-point implementations require larger amounts of FPGA resources than an equivalent fixed-point solution. Using fixed-point model has several different advantages:

- Reduction in FPGA resources
  - Fewer DSP48E2s, look-up tables (LUTs), and flip-flops are needed when working with fixed-point data types.
  - Smaller amounts of memory are required to store fixed-point numbers.
- Lower power consumption
  - Reduction in FPGA resource usage inherently leads to lower power consumption.
- Latency improvements
  - Again, reducing the resources, in particular the DSP48E2 slices, can bring a latency improvement in the fixed-point design.
- Comparable performance and accuracy
  - For designs and applications that do not require the dynamic range achievable with floating point, fixed-point implementations can provide comparable results and accuracy. In some cases, the results can even be improved.

The benefits of performing this conversion are so great that it merits serious consideration where applicable—in particular, designs where the dynamic range and precision available with floating-point are not required and the small expected loss of precision will not lead to inefficiencies in the deployed application. All the coefficients which are used to calculate the scores of different SVM stages are converted from floating-point to signed 32-bit fixed-point data. Out of 32 bits, 12 bits are used for integer part and 20 bits for decimal part. Aforementioned division of bits and data width was defined after reviewing the data range dynamics. As a result of high number of bits representing the decimal part, the high-level of precission is maintained. Score deviation is neglectable, especially because no changes were detected in position or number of faces detected, in comparison to the original model where floating-point type was used.

5.2 High Level Synthesis (HLS)

Due to accelerated design time and time-to-market pressures, algorithmic-based approaches gained in popularity over the years. Moreover, design and hardware verification can be demanding. Hardware acceleration has become important for industry in such a way that is used for increasing productivity and enhancing performance. Although hardware accelerators can require a lot of time to understand and design, more and more CPU intensive tasks are being implemented and run on hardware. Vivado HLS tool is used to translate/transform/convert algorithmic description written in C-based design flow (C, C++, System C) into hardware description RTL. Control and datapath can be extracted from C code at the top level. Hardware is created from dataflow using two different processes: scheduling and binding. Scheduling maps operations into cycles, while binding assigns hardware resources to operations. Constraints and directives are used to override default behavior and optimize implementation. In addition, coding style has a significant impact on hardware realization.

Library support includes floating-point support, fixed-point support, OpenCV video function support and DSP function support. However, there are certain unsupported constructs:

- Dynamic memory allocation (`malloc()`, `free()`, etc.)
- Standard I/O and file I/O operations (`fprintf()`, `fscanf()`, etc.)
- System calls (`time()`, `sleep()`, etc.)
- Pointer casting between natural integer types

Macro __SYNTHESIS__ is used to remove unsynthesizable code from the design.

Synthesis flow includes 5 different steps (Figure 5.2):

- **C Simulation** – Validating the functionality of C algorithm
- **C Synthesis** – Synthetizing C to RTL
- **Multiple Solutions** – Exploring different solutions with area/performance comparison
- **C/RTL Co-Simulation** – Verifying that RTL gives same results as the C code using the same stimuli
- **Exporting RTL** – Exporting the RTL as IP to use with Vivado

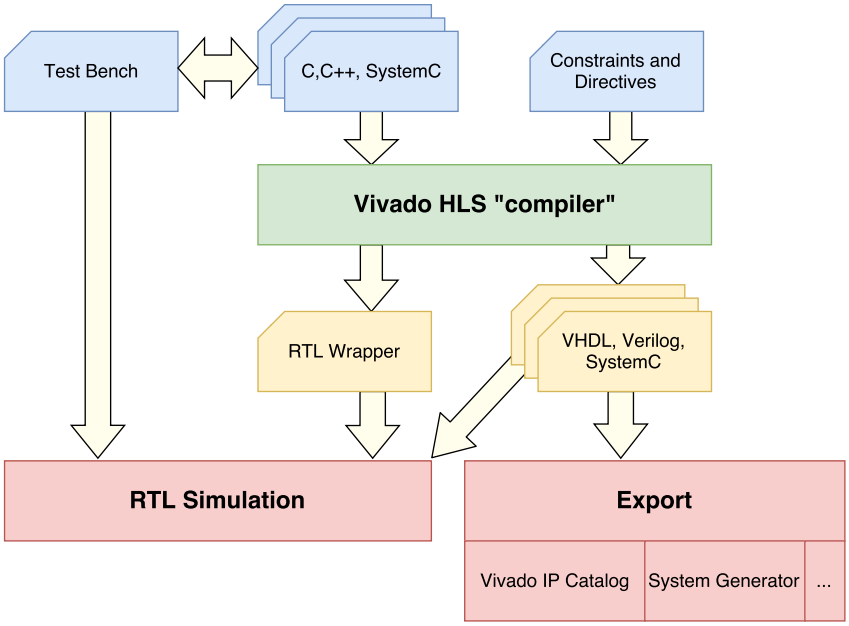


Figure 5.2: HLS flow

The design can be implemented as an IP core to be used in System Integration Environments such as System Generator for DSP, Vivado RTL Integrator, Vivado IP Integrator (Figure 5.3).

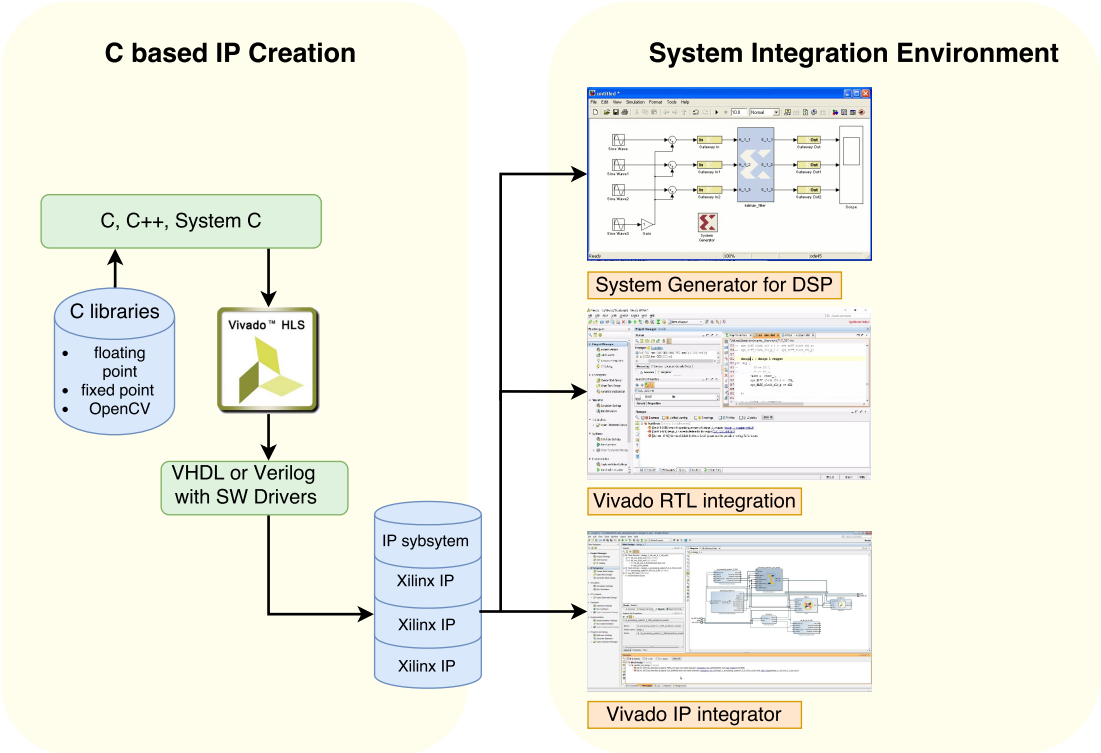


Figure 5.3: IP integration

Function arguments are synthetized into data ports. Clock and Reset ports are added to all RTL blocks. Port `ap_start` is used for controls when the RTL block starts execution, `ap_idle` indicates if RTL block is in idle state, `ap_done` signals when RTL block has completed operation or function return is valid, `ap_ready` is used if function is pipelined and it indicates that new sample can be introduced before done is asserted. Input/output protocols are added at the port level. I/O protocol can be selected from a list. However, not every combination of C argument and port protocol is supported (`ap_none`, `ap_stable`, `ap_ack`, `ap_vld`, `ap_ovld`, `ap_hs`, `ap_memory`, `ap_fifo`, `ap_bus`, `ap_ctrl_none`, `ap_ctrl_hs`). All ports are derivative of `ap_hs`: for input arguments (arguments which are only read) valid is input port indicating when the data is available to read, acknowledge is an out-

put port indicating that the data was read; for output arguments (arguments which are only written to) valid is an output port indicating when the data is ready, acknowledge is an input port indicating that the data was read; inout arguments are the combination of these, have both input and output ports. RAM ports are created by `ap_memory` protocol. There are ports for data, address and control. Function argument is an array FIFO ports are created by `ap_fifo` protocol. There are read/write ports and full/empty ports. Function argument can be an array, pointer of reference. Bus interfaces are used in design to connect the block in Vivado IP integrator environment. There are AXI4-stream, AXI4-master and AXI4-slave protocols. Ports are grouped into adapters and appropriate resources are specified in order to create AXI interfaces.

5.3 Designing IP blocks in HLS

As it has already been mentioned, by using certain algorithms, number of support vectors can be reduced significantly, with little effect on the final result. Consequently, less support vectors will result in more efficient processing. All the data obtained in Matlab, which is used to define SVM stages had been written to binary files. Binary files are being stored on SD card. Xilinx has provided **xilffs** library in SDK that can be included to the project's BSP (Board Support Package). This library implements FAT file system and consists of a file system and a glue layer. Glue layer supports SD/eMMC interface. API (*Application Programming Interface*) functions are provided in file system files. Input images that are being processed, as well as output results are being stored as binary files on SD card. Image height, and image width are represented with 4 bytes each, at the beginning of a binary file,

followed by bytes representing pixel values of the input image.
For linear SVM first stage there are 3 binary files:

1. **mulin**.**bin**, mean μ values for input sub-window scaling/normalization are fixed point, represented with 4 bytes each, in total 1600 bytes.

2. **sgbsclin**.**bin** standard deviation, kernel scale and beta coefficients all together are fixed point, represented with 4 bytes each, in total 1600 bytes.

3. **bslin**.**bin** bias value is stored as 4 bytes

For polynomial SVM second stage there are 5 binary files:

1. **mupoly**.**bin**, mean values for input sub-window scaling/normalization are fixed point, represented with 4 bytes each, in total 1600 bytes.

2. **aypoly**.**bin** $\alpha_i y_i$ values are fixed point, each corresponding to a different support vector (533 of them), represented with 4 bytes each, in total 2132 bytes.

3. **scpoly**.**bin** is reciprocal squared kernel scale parameter, in fixed point, represented with 4 bytes

4. **spspoly**.**bin** are 533 support vectors with 400 elements each, in fixed point, represented with 4 bytes, in total 852800 bytes

5. **bspoly**.**bin** bias value is stored as 4 bytes

For polynomial SVM third stage with LBP feature extraction there are 5 binary files:

1. **mulbp**.**bin**, mean values for input sub-window scaling/normalization are fixed point, represented with 4 bytes each, in total 4248 bytes.

2. **aylbpy.bin**, $\alpha_i y_i$ values are fixed point, each corresponding to a different support vector (875 of them), represented with 4 bytes each, in total 3500 bytes.

3. **sclbp.bin** is reciprocal squared kernel scale parameter, in fixed point, represented with 4 bytes

4. **spslbp.bin** are 875 support vectors with 1062 elements each, in fixed point, represented with 4 bytes, in total 3717000 bytes

5. **bslbp.bin** bias value is stored as 4 bytes

There are two types of memory that can be used inside FPGA when working in HLS: Block RAM and Distributed RAM. A block RAM is a dedicated (cannot be used to implement other functions like digital logic) two port memory containing several kilobits of RAM. Distributed RAM memory is implemented using LUTs inside FPGA. The configuration logic blocks (CLB) in most of the Xilinx FPGA's contain small single port or double port RAM. This RAM is normally distributed throughout the FPGA than as a single block (It is spread out over many LUT's) and so it is called "distributed RAM". A look up table on a Xilinx FPGA can be configured as a 16x1bit RAM, ROM, LUT or 16-bit shift register.

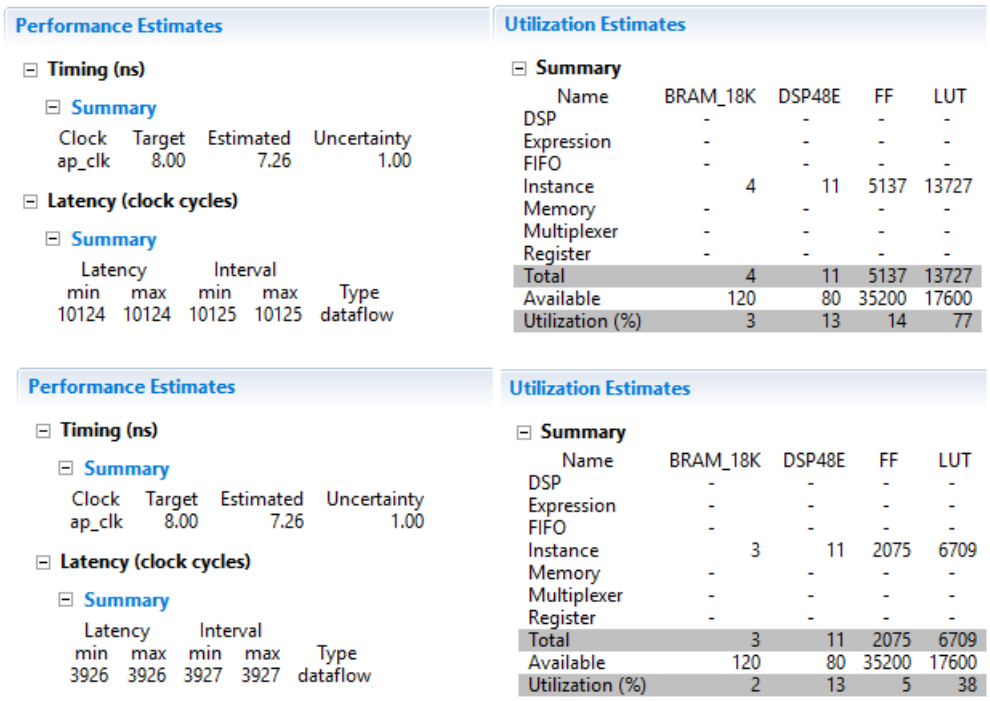
Given that there is a limited amount of resources (240 KB BRAM and $4,400$ logic slices, each with four 6-input LUTs and 8 flip-flops), the number of support vectors is too high to fit in RAM type of memory. Thus, support vectors coefficients for second and a third stage will be stored in DDR3 external memory, which is definitely slower but with higher storage capacity (512 MB).

5.3.1 Histogram equalization

Algorithm that was used in Matlab for histogram equalization has a certain level of complexity on which HLS block latency depends on. With histogram equalization function that was used in Matlab `J=histeq(I,N)`, intensity image I is transformed returning in J an intensity image with N discrete levels. A roughly equal number of pixels is mapped to each of the N levels in J, so that the histogram of J is approximately flat. (The histogram of J is flatter when N is much smaller than the number of discrete levels in I). The default value for N is 64 and it was used for training and testing purposes. 16 levels were used for the hardware implementation since it requires significantly less resources and latency is smaller (Figure 5.4). The deviation exists although it is acceptable.

```
void histeq_proc (uint32_t window_in [400],
                  uint32_t window_out1 [400],
                  uint32_t window_out2 [400],
                  uint32_t window_out3 [400])
{
#pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS
#pragma HLS INTERFACE bram port=window_out1
#pragma HLS INTERFACE bram port=window_out2
#pragma HLS INTERFACE bram port=window_out3
#pragma HLS INTERFACE bram port=window_in
```

`window_in` was used as input array with 400 elements of original sub-window pixel values, stored in BRAM. After exporting RTL to IP, this port was implemented as one-port BRAM interface. `window_out1`, `window_out2`, `window_out3`, were used to store 400 histogram equalized pixel values. Af-



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00	7.26	1.00

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
10124	10124	10125	10125	dataflow

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	4	11	5137	13727
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	4	11	5137	13727
Available	120	80	35200	17600
Utilization (%)	3	13	14	77

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00	7.26	1.00

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
3926	3926	3927	3927	dataflow

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	3	11	2075	6709
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	3	11	2075	6709
Available	120	80	35200	17600
Utilization (%)	2	13	5	38

Figure 5.4: Comparison of utilization estimates and performance estimates after synthesis for histogram equalization block and different level of complexity

ter the processing is done, the same values were stored to all of the three memories, so they could be used, first for calculating the linear first stage score, second for calculating the polynomial second stage score and the third one for extracting LBP histogram on CPU. After IP generation, all the ports were implemented as one-port BRAM. Final timing and resource usage are shown on Figure 5.5. An AXI4-Lite slave interface is used to allow a design to be controlled by the CPU. The Vivado HLS has synthetized AXI4-Lite slave interface with `s_axilite` mode enabled. IP block schematic used in Vivado IP integrator has on the left side one slave interface `s_axilite` set of ports, and on the right side four BRAM interface sets of ports (Figure 5.6).

Resource Usage

	VHDL
SLICE	793
LUT	2297
FF	1594
DSP	11
BRAM	3
SRL	14

Final Timing

	VHDL
CP required	8.000
CP achieved post-synthesis	7.924
CP achieved post-implemetation	7.505

Timing met

Figure 5.5: Utilization estimates and performance estimates after implementation of histogram equalization block

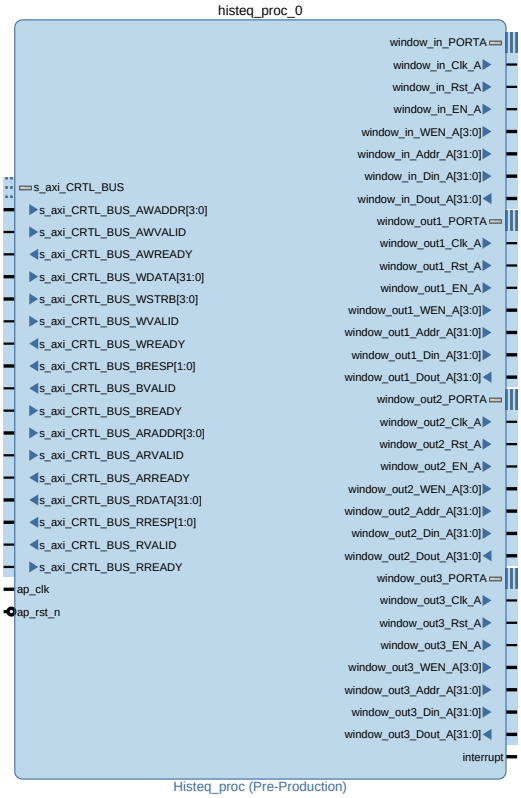


Figure 5.6: IP block schematic

5.3.2 Linear SVM first stage score calculation

```
void lin_stage_proc (uint32_t in_img [400],
                     uint32_t img_coeff [801],
                     float *score)
{
#pragma HLS RESOURCE variable=in_img core=RAM_1P_BRAM
#pragma HLS RESOURCE variable=img_coeff core=RAM_2P_BRAM
#pragma HLS INTERFACE s_axilite port=score bundle=CRTL_BUS
#pragma HLS INTERFACE bram port=in_img name=lin_w
#pragma HLS INTERFACE bram port=img_coeff name=st_coeffs
#pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS
```

`in_img` was used as input array with 400 elements of histogram equalized sub-window pixel values, stored in BRAM. After exporting RTL to IP, this port was implemented as one-port BRAM interface.

`img_coeff` was used in such a way, that first 400 array elements were occupied with mean coefficients μ_i , 401^{st} element represented a bias value, and next 400 values were β_i coefficients representing $\frac{\beta_i}{\sigma_i scale}$. After IP generation, this port was implemented as two-port BRAM. An AXI4-Lite slave interface is used to allow a design to be controlled by the CPU. The Vivado HLS has synthetized AXI4-Lite slave interface with `s_axilite` mode enabled. Multiple ports are grouped into the same AXI4-Lite slave interface. The code above shows an implementation of multiple arguments, `score`, including the function return, as AXI4-Lite slave interfaces in bundle = CRTL_BUS. Because each interface uses the same name for the bundle option, the HLS tool grouped each of the ports into the same AXI4-Lite interface. Performance and utilization estimates are given on Figure 5.7. Final timing and resource usage are shown on Figure (5.8). IP block schematic used in Vivado IP inte-

grator has on the left side slave interface `s_axilite` set of ports, and on the right side three BRAM interface sets of ports (Figure 5.9).

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00	6.85	1.00

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
227	227	113	113	dataflow

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	33
FIFO	-	-	-	-
Instance	0	16	2117	3330
Memory	-	-	-	-
Multiplexer	-	-	-	15
Register	-	-	15	-
Total	0	16	2132	3378
Available	120	80	35200	17600
Utilization (%)	0	20	6	19

Figure 5.7: Utilization estimates and performance estimates after synthesis of linear stage block

Resource Usage

	VHDL
SLICE	599
LUT	1490
FF	1751
DSP	16
BRAM	0
SRL	89

Final Timing

	VHDL
CP required	8.000
CP achieved post-synthesis	7.693
CP achieved post-implemetation	7.672

Timing met

Figure 5.8: Utilization estimates and performance estimates after implementation of linear stage block

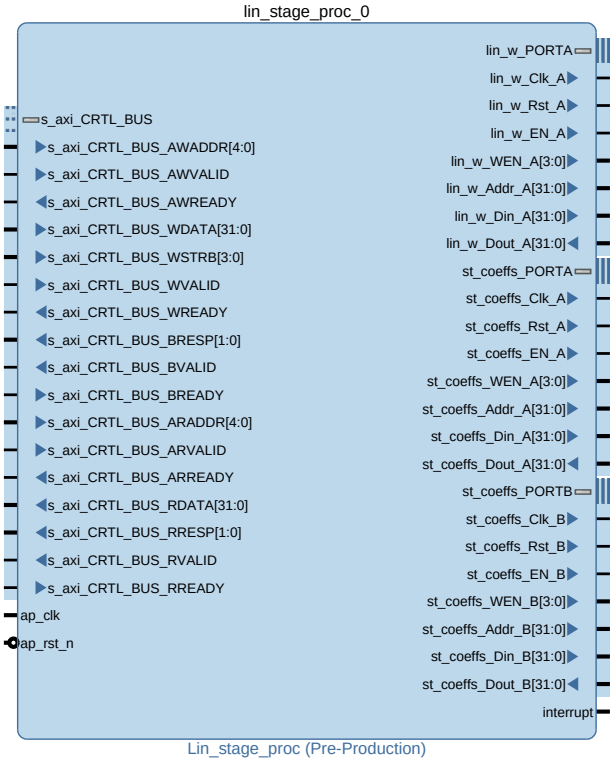


Figure 5.9: IP block schematic

5.3.3 Polynomial SVM second stage score calculation

```
void poly_stage_proc (int32_t in_img [400],
                      int32_t img_coeff [1500],
                      volatile uint32_t *poly_c,
                      uint32_t offset1,
                      uint32_t offset2,
                      float *score)
{
#pragma HLS RESOURCE variable=in_img core=RAM_1P_BRAM
#pragma HLS RESOURCE variable=img_coeff core=RAM_2P_BRAM
```

```
#pragma HLS INTERFACE m_axi depth=200 port=poly_c bundle=MASTER_BUS
#pragma HLS INTERFACE bram port=in_img name=in_image
#pragma HLS INTERFACE bram port=img_coeff name=img_coeffs
#pragma HLS INTERFACE s_axilite port=offset1 bundle=CRTL_BUS
#pragma HLS INTERFACE s_axilite port=offset2 bundle=CRTL_BUS
#pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS
#pragma HLS INTERFACE s_axilite port=score bundle=CRTL_BUS
```

`in_img` was used as input array with 400 elements of histogram equalized sub-window pixel values, stored in BRAM. After exporting RTL to IP, this port was implemented as one-port BRAM interface.
`img_coeff` was used in such a way, that first 400 array elements were occupied with mean coefficients, 401^{st} element represented a reciprocal squared kernel scale value, and next 533 values were $\alpha_i y_i$. After IP generation, this port was implemented as two-port BRAM. An AXI4-Lite slave interface is used to allow a design to be controlled by CPU. The Vivado HLS will synthetize AXI4-Lite slave interface when `s_axilite` mode is enabled. Multiple ports are grouped into the same AXI4-Lite slave interface. The code above shows an implementation of multiple arguments `offset1`, `offset2`, `score`, including the function return, as AXI4-Lite slave interfaces in bundle=CRTL_BUS. Because each interface uses the same name for the bundle option, the HLS tool groups each of the ports into the same AXI4-Lite interface. `offset1` has been used to position at support vector currently used, while `offset2` represents an index of $\alpha_i y_i$ coefficient corresponding to the current SV. `score` is used to return a calculation value. One additional interface is used in the design. It is AXI4 Master Interface and has been implemented on the pointer argument. Data is transferred from memory to local buffer in a burst-mode which is a result of using `memcpy` function.

Although burst-mode transfers data use a single base address to read 400 values, offset1 is used as an offset to this address. Higher data throughput is achieved using burst rather than individual data transfer. Final timing and resource usage are shown on Figure 5.11. IP block schematic used in Vivado IP integrator has on the left side slave interface `s_axilite` set of ports, and on the right side three BRAM interface sets of ports and one master interface `m_axi` set of ports (Figure 5.12).

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00	7.00	1.00

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
854	854	855	855	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1244
FIFO	-	-	-	-
Instance	2	16	1002	1366
Memory	1	-	0	0
Multiplexer	-	-	-	174
Register	-	-	1277	1
Total	3	16	2279	2785
Available	120	80	35200	17600
Utilization (%)	2	20	6	15

Figure 5.10: Utilization estimates and performance estimates after synthesis of polynomial stage block

Resource Usage

	VHDL
SLICE	525
LUT	940
FF	1846
DSP	16
BRAM	2
SRL	21

Final Timing

	VHDL
CP required	8.000
CP achieved post-synthesis	5.460
CP achieved post-implemetation	6.191

Timing met

Figure 5.11: Utilization estimates and performance estimates after implementation of polynomial stage block

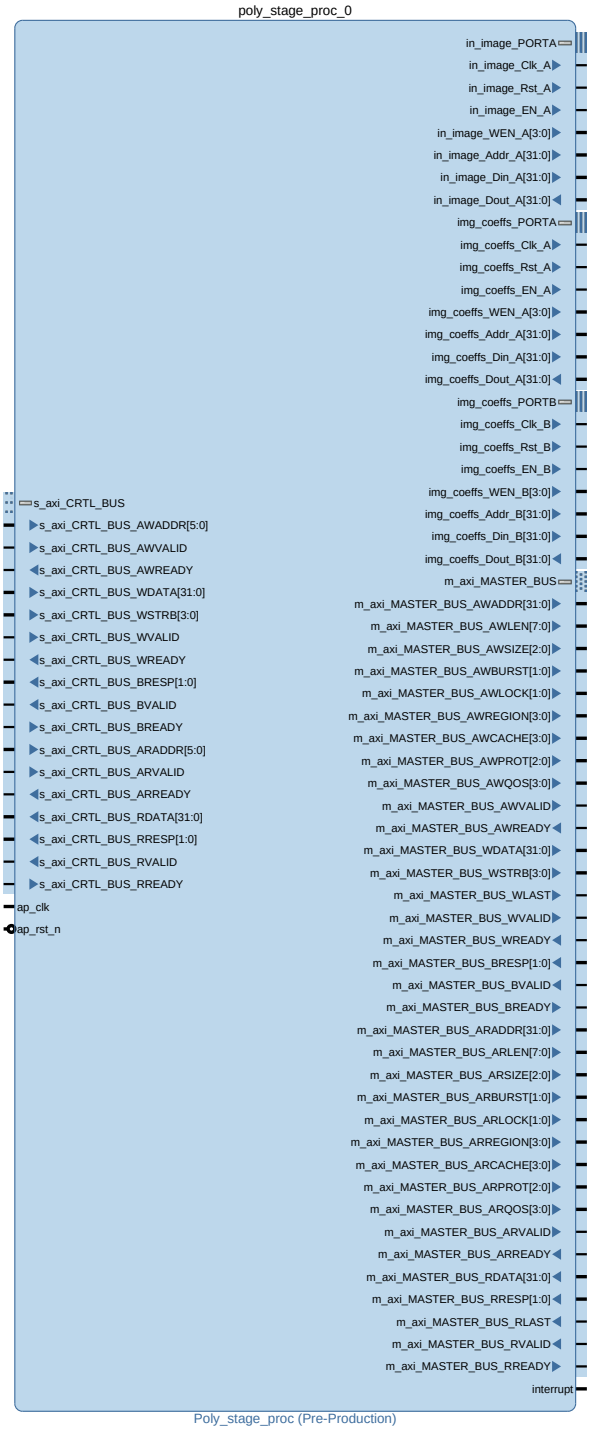


Figure 5.12: IP block schematic

5.3.4 Polynomial SVM third stage score calculation

```
void lbp_stage_proc (int32_t in_hist [1062],
                     int32_t hist_coeff [4000],
                     volatile uint32_t *lbp_c,
                     uint32_t offset1,
                     uint32_t offset2,
                     float *score)
{
#pragma HLS RESOURCE variable=in_hist core=RAM_1P_BRAM
#pragma HLS RESOURCE variable=hist_coeff core=RAM_2P_BRAM
#pragma HLS INTERFACE s_axilite port=offset1 bundle=CRTL_BUS
#pragma HLS INTERFACE s_axilite port=offset2 bundle=CRTL_BUS
#pragma HLS INTERFACE m_axi depth=200 port=lbp_c bundle=MASTER_BUS
#pragma HLS INTERFACE s_axilite port=score bundle=CRTL_BUS
#pragma HLS INTERFACE bram port=in_hist name=lbp_w
#pragma HLS INTERFACE bram port=hist_coeff name=st_coeffs
#pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS
```

`in_hist` was used as input LBP extracted histogram array with 1062 elements stored in BRAM. After exporting RTL to IP, this port was implemented as one-port BRAM interface.

`hist_coeff` was used in such a way, that first 1062 array elements were occupied with mean coefficients, 1063^{rd} element represented a reciprocal squared kernel scale value, and next 875 values were $\alpha_i y_i$. After IP generation, this port was implemented as two-port BRAM. An AXI4-Lite slave interface is used to allow a design to be controlled by CPU. The Vivado HLS will synthetize AXI4-Lite slave interface when `s_axilite` mode is enabled. Multiple ports are grouped into the same AXI4-Lite slave interface.

The code above shows an implementation of multiple arguments `offset1`, `offset2`, `score`, including the function return, as AXI4-Lite slave interfaces in bundle=CRTL_BUS. Because each interface uses the same name for the bundle option, the HLS tool groups each of the ports into the same AXI4-Lite interface. `offset1` has been used to position at support vector currently used, while `offset2` represents an index of $\alpha_i y_i$ coefficient corresponding to the current SV. `score` is used to return a calculation value. One additional interface is used in the design. It is AXI4 Master Interface and has been implemented on the pointer argument. Data is transferred from memory to local buffer in a burst-mode which is a result of using memcpy function. Although burst-mode transfers data use a single base address to read 1062 values, offset1 is used as an offset to this address. Higher data throughput is achieved using burst rather than individual data transfer. Final timing and resource usage are shown on Figure 5.14. IP block schematic used in Vivado IP integrator has on the left side slave interface `s_axilite` set of ports, and on the right side three BRAM interface sets of ports and one master interface `m_axi` set of ports (Figure 5.15).

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	8.00	7.00	1.00

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
2178	2178	2179	2179	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	1147
FIFO	-	-	-	-
Instance	2	16	1002	1366
Memory	4	-	0	0
Multiplexer	-	-	-	180
Register	-	-	1256	1
Total	6	16	2258	2694
Available	120	80	35200	17600
Utilization (%)	5	20	6	15

Figure 5.13: Utilization estimates and performance estimates after synthesis of polynomial lbp stage block

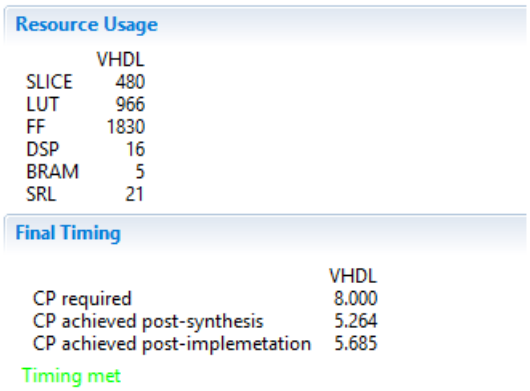


Figure 5.14: Utilization estimates and performance estimates after implementation of lbp polynomial stage block

5.4 IP components integration

Different IP components were instantiated and connected into a functional system in Vivado IP Integrator:

- Processor System Reset module `rst_ps7`
  - Reset modul, controlled by the Zynq Processing System, is used for resetting the peripheral and interconnection blocks
- AXI-Interconnect `ps7_axi` module
  - The processor implements an AXI Master interface, the peripherals implement an AXI Slave interface, and the interconnect implements both along with address decoding, arbitration, etc. to handle the movement of transactions from the processor to the peripherals. Slave input of the interconnection block is connected to the Zynq Processing System master interface at `M_AXI_GP0`. Block's master outputs are connected to the peripherals pins implementing slave interfaces. This way ARM core can directly ini-

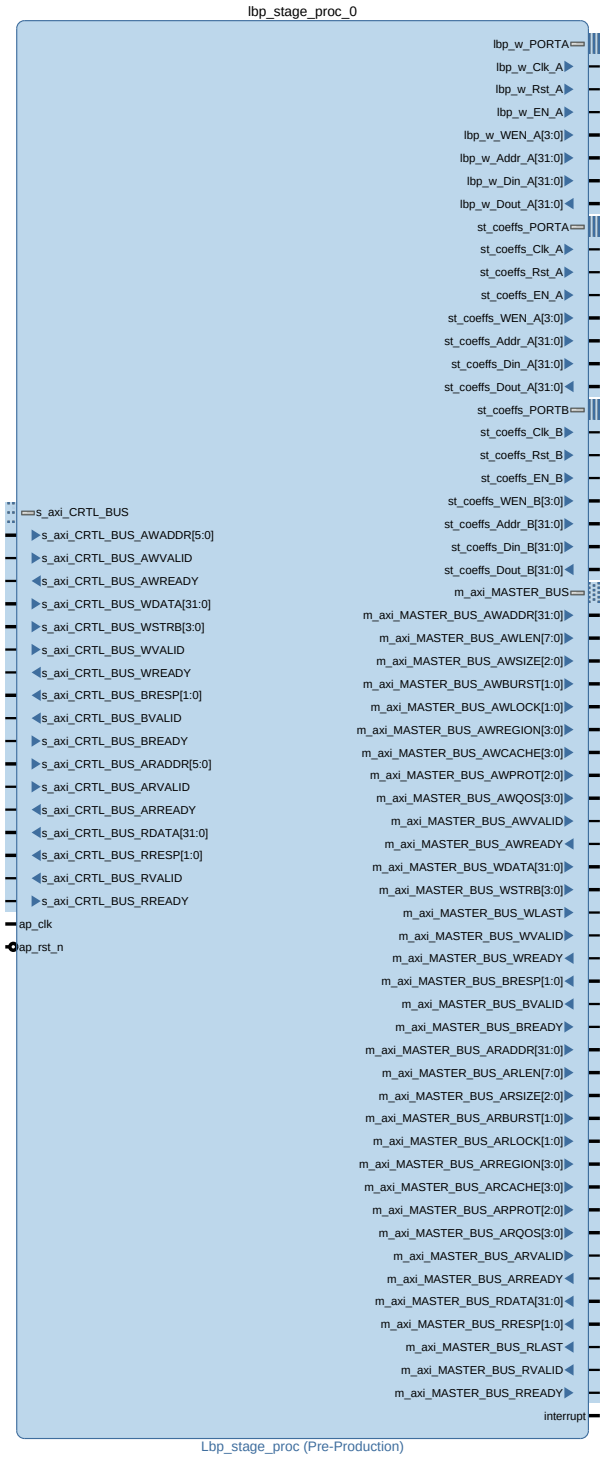


Figure 5.15: IP block schematic

tiate read and write transactions to the address space of the host through aforementioned `GP0` port.

- Zynq-7000 Processing System IP `zynq_processing`
  - This IP is the software interface around Zynq-7000 Processing System. It acts as a logic connection between PS and PL while integrating custom and embedded IPs with the processing system. It has DDR controller implemented for accessing external DDR memory.
- 11 Block Memory Generators `blk_mem_gen`
  - BMG is an advanced memory constructor implementing optimized memories using embedded BRAM resources in Xilinx FPGAs. These modules can be attached to different BRAM Interface Controllers. They can be configured as a single or two port memory, whose ports can even be attached to independent BRAM Interface Controllers.
- 9 AXI BRAM Controllers `axi_bram_ctrl`
  - BRAM controller is used to communicate with local BRAM. It has an AXI4 memory mapped slave interface. Moreover, it is compatible with AXI-Interconnect and has separate read and write channel interfaces to make an effective use of BRAM dual port memory.
- 2 AXI-Interconnect `axi_mem` modules
  - Interconnection blocks as well, with different direction. Master in the PL can initiate transactions to the PS. Their slave inputs are

connected to the custom processing IPs ports through AXI4 master interface, while their master outputs are attached to the Zynq Processing System through slave interfaces at ports `S_AXI_HP0` and `S_AXI_HP1`.

- 4 HLS custom processing modules
  - Histogram equalization, Linear SVM stage, Polynomial SVM stage, Polynomial SVM stage with LBP feature extraction.
- AXI-Timer/Counter `axi_timer`
  - 32-bit timer module that can be cascaded to work in 64-bit mode. It has AXI4-Lite Interface and it is used for measuring execution time.

Address editor was used to assign address space to individual components with memory mapped controlling interfaces. After synthesis and implementation of the design, power consumption and resource utilization reports were obtained (Figure 5.17). After exporting synthesized hardware into SDK environment, BSP was generated and included into a new software application project. FatFS library **xilffs** was enabled for using SD card interface. There are several functions which were used for accessing the files on the card:

- `f_mount()`
  - Mounting system
- `f_stat()`
  - Finding a file on the system and obtaining information about it
- `f_open()`

  - Opening a file to read (FA_OPEN_EXISTING | FA_READ) or write (FA_CREATE_ALWAYS | FA_WRITE)

- `f_write()`
  - Writing data to opened file
- `f_read()`
  - Reading data from opened file
- `f_close()`
  - Closing opened file

For some arrays memory was allocated in external memory using `malloc` function, while for others in BRAM memory, whose addresses are located in BSP header file `xparameters.h`. Function definitions used for controlling HLS custom created peripherals are located in:

- `xhisteq_proc.h`
- `xlin_stage_proc.h`
- `xpoly_stage_proc.h`
- `xlbp_stage_proc.h`

Using `_LookupConfig()` and `_CfgInitialize` four processing slave peripherals were initialized. `stopTimer()`, `startTimer()`, `getlapsedTimeInSeconds()` are functions used for controlling the timer, defined in `timer.h` header file. Image that was a read from a file is processed in blocks of 20x20 pixels. When this block is stored in BRAM the modules can be started.

```
XHisteq_proc_Start(&do_histeq);
while(!XHisteq_proc_IsDone(&do_histeq));

XLin_stage_proc_Start(&do_lin_stage);
while(!XLin_stage_proc_IsDone(&do_lin_stage));
s_lin = u32_to_float(XLin_stage_proc_Get_score(&do_poly_stage));

if (-s_lin > C_lin){
 s_poly = 0;
 for(int k=0; k<SVpoly; k++){
  XPoly_stage_proc_Set_offset1(&do_poly_stage,400*k);
  XPoly_stage_proc_Set_offset2(&do_poly_stage,k);
  XPoly_stage_proc_Start(&do_poly_stage);
  while(!XPoly_stage_proc_IsDone(&do_poly_stage));
  s_poly += u32_to_float(XPoly_stage_proc_Get_score(&do_poly_stage));
 }
 s_poly += b_poly;
 if (-s_poly > C_poly)
 {
  lbp_hist_cal(img_eq,imgs_lbp);
  s_lbp = 0;
  for(int k=0; k<SVlbp; k++){
   XLbp_stage_proc_Set_offset1(&do_lbp_stage,1062*k);
   XLbp_stage_proc_Set_offset2(&do_lbp_stage,k);
   XLbp_stage_proc_Start(&do_lbp_stage);
   while(!XLbp_stage_proc_IsDone(&do_lbp_stage));
   s_lbp += u32_to_float(XLbp_stage_proc_Get_score(&do_lbp_stage));
```

```
  }
  s_lbp += b_lbp;
 }
}
```

lbp_hist_cal is a function implemented in software, for extracting LBP block histogram. The output 2D array that is being stored on the SD card after calculating all the scores can have four different values:

- 0 - If the current sub-window did not pass 1st stage- FAIL
- 1 - If the current sub-window did pass 1st stage, but did not pass 2nd stage - FAIL
- 2 - If the current sub-window did pass 2nd stage, but did not pass 3rd stage - FAIL
- 3 - If the current sub-window did pass 3rd stage - PASS

While the PL is working at the frequency of 125 MHz, the PS is working with 650 MHz. DDR is operating at the frequency of 525 MHz. Execution time of the face detection system implemented in hardware, for images from Figures 4.15-4.21 is given in the Table 5.2. As a result of histogram equalization algorithm modification (number of discrete levels) there were slight changes in calculated score for every stage and therefore in TP and FP rates.

Table 5.2: Processing results for the system implemented in hardware

Image	Dim.	Sub-windows	Time[s]	TP	FP
n1	400x400	19896	18.718	14/16	3
n2	400x484	24331	21.412	12/12	2
n3	400x710	36293	31.342	10/11	0
n4	400x800	41055	33.101	9/11	0
n5	400x670	34291	29.873	14/19	0
n6	400x502	25179	24.773	5/5	0
n7	400x494	24781	22.029	6/6	1
n8	400x600	30416	25.833	6/7	0
n9	400x712	36412	33.984	6/6	2
n10	400x858	44134	35.843	5/5	3
n11	400x601	30555	23.731	4/5	2

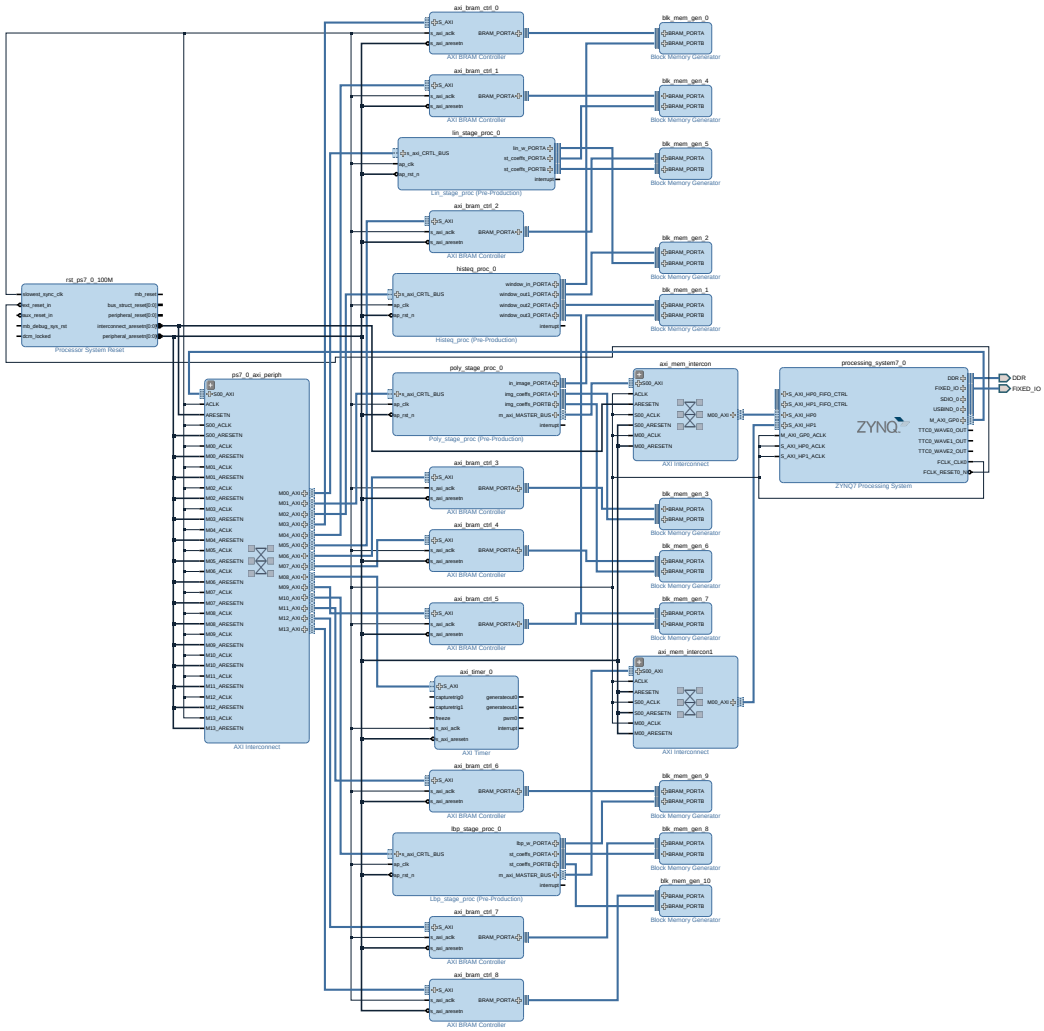
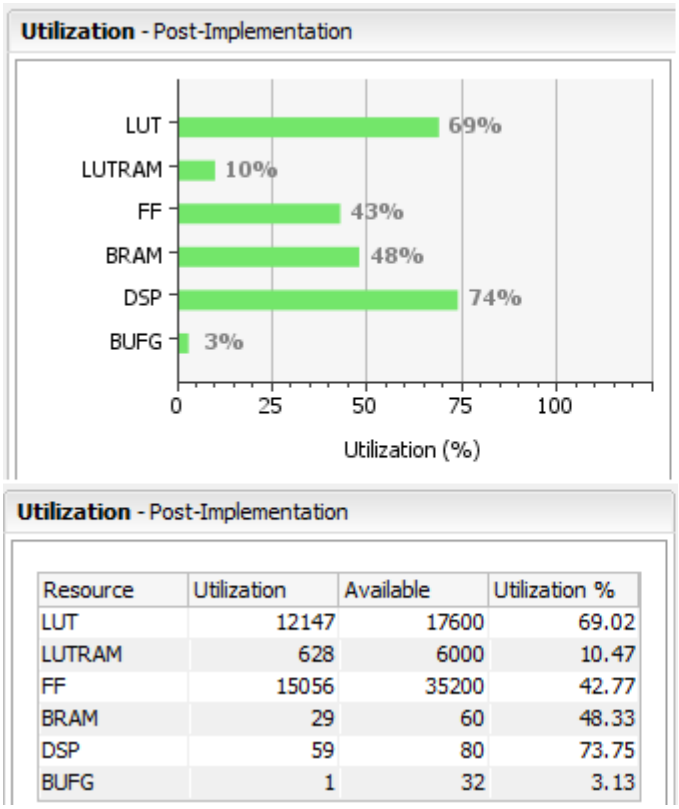


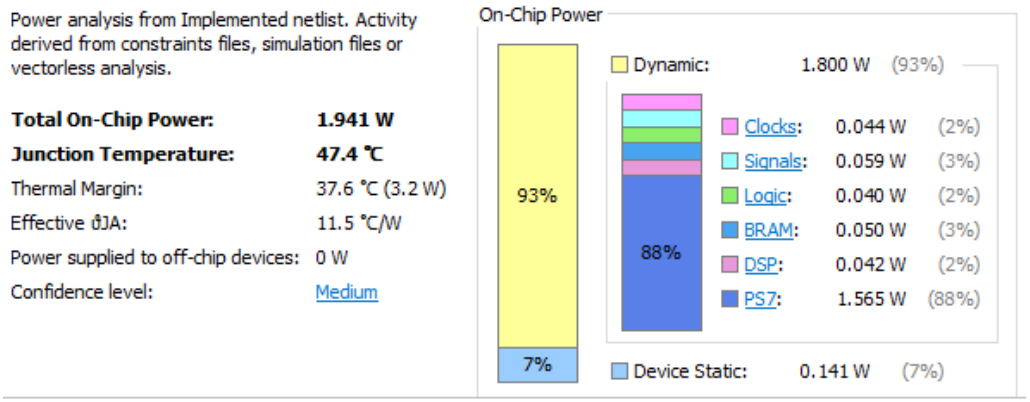
Figure 5.16: Design schematic



Utilization - Post-Implementation

Resource	Utilization	Available	Utilization %
LUT	12147	17600	69.02
LUTRAM	628	6000	10.47
FF	15056	35200	42.77
BRAM	29	60	48.33
DSP	59	80	73.75
BUFG	1	32	3.13

(a) Resource utilization report



(b) Power consumption report

Figure 5.17: Post-implementation reports

Chapter 6

Conclusion

In this master thesis, the review of some of the most efficient and most used face detection algorithms has been given. General concepts of Machine Learning, together with mathematical theory behind Support Vector Machine binary classification algorithm has been presented.

After having studied different solutions for face detection systems using SVM algorithm, the feasibility of the chosen architecture has been proven by obtaining Matlab reference model. Three SVMs of different complexity were trained. They were cascaded in order to reduce classification time, while maintaining classification accuracy. Classification score boundaries were chosen during training, as a compromise between true and false positive rates. In addition to testing the system on test face patterns from the chosen database, the algorithm was verified on the set of images chosen manually.

Coefficients defining an SVM, were converted from floating-point to fixed-point arithmetic. All these coefficients were stored on an SD card, making the system absolutely reusable and flexible in the terms of coefficients values and number of support vectors. Four processing blocks have been implemented using HLS. Using Vivado IP integrator, exported IP blocks were integrated

together with one ARM processor core, some interconnection blocks, memory and control modules.

The final testing has proven that the designed system is fully operational, wokring with images of different dimensions. Moreover, it is a flexible design, that can be further researched. LBP histogram extraction and image resizing are two remaining block that can be implemented in hardware. Applying the process of reducing the number of support vectors [5] will significantly impact the classification time, that is, will further decrease detection time.

Machine Learning has many applications in financial services, government, health care, oil and gas, transportation, etc. and as we have seen can provide exceptional results. Considering this alongside the data availability, inexpensive storage and rise in computation power, the extent of using this method of data analysis will undoubtedly continue to grow.

Furthermore, in this thesis, the possibilities of relatively new type of design methodology have been examined. HLS provides faster path to IP creation by abstraction of algorithmic description, data type specification and interfaces. However, it is still not as efficient as it should be, and additionally, there are many options and possibilities to consider while designing. When hardware designers transitioned from assembly to high-level languages for software development, their productivity increased. They were able to add more functionality with less effort, simplifying debugging, maintenance and improving reusability of the design. The same changes happened with the transition of hardware design from gate-level to RTL, and the same changes are happening once again, while transitioning from VHDL and Verilog to C++. While extending the capabilities of the technology, productivity and reducing time-to-market are one of the key drivers that will continue to support methodology transitions.

Bibliography

[1] *Detecting Faces in Images: A Survey* M.H.Yang, D.Kriegman, N.Ahuja IEEE 24(1): 34-58, 2002.

[2] *Rapid object detection using a boosted cascade of simple features.* P.Viola, and M.Jones. CVPR 2001.

[3] *A short introduction to boosting* Y.Freund, and R.Schapire. Journal of Japanese Society for Artificial Intelligence 14(5):771-780, 1999.

[4] B.E.Boser, I.M.Guyon and V.N.Vapnik. *A training algorithm for optimal margin classifiers.* In COLT '92 Proceeding of the fifth annual workshop on Computational learning theory 144-152, New York, NY, USA, 1992. ACM Press

[5] C.J.C. Burges. *Simplified support vector decision rules.* In International Conference on Machine Learning 71-77, 1996.

[6] B.Heisele, T.Poggio, and M.Pontil. *Face detection in still gray images.* 2000.

[7] B.Heisele, T.Serre, and M.Pontil. *A component-based framework for face-detection and identification.* International Journal of Computer Vision, 74(2):167-181 August, 2007.

[8] J.C.Platt. *Using analytic qp and sparseness to speed training of support vector machines.* International Journal of Computer Vision, 74(2):167-181 August, 2007. Advances in Neural Information Processing Systems, volume 11, Cambridge, MA, 1999. MIT Press.

[9] S.Romdhani, P.Torr, and B.Scholkopf. *Efficient face detection by a cascaded support vector machine expansion.* Royal Society of London Proceedings Series A, 460:3283-3297. November, 2004.

[10] V.N.Vapnik. *The nature of the Statistical Learning Theory.* 2nd Edition, Springer, 1999.

[11] L.Bottou *Comparison of classifier methods: a case study in handwritten digit recognition.* 1994.

[12] I.Kukenys, and B.McCane. *Classifier cascades for support vector machines.* 23rd International Conference Image and Vision Computing New Zealand 2008.

[13] B.Heisele, T.Serre, S.Prentice, and T.Poggio. *Hierarchical classification and feature reduction for fast face detection with support vector machines.* Pattern Recognit., 36(9):2007-2017, 2003.

[14] Y.Ma, and X.Ding. *Face detection based on cost-sensitive support vector machines.* Proc. 1st Int. Workshop Pattern Recognit. Support Vector Mach. 260-267, March 2002.

[15] M.Alvira, and R.Rifkin. *An Empirical Comparison of SNoW and SVMs for Face Detection.* Center for Biological and Computational Learning, MIT, 2001.

[16] K.K.Sung. *Learning and Example Selection for Object and Pattern Recognition.* Artificial Intelligence Laboratory and Center for Biological and Computational Learning, MIT, 1996.

[17] M.Papadonikolakis, and C.S.Bouganis *Novel Cascade FPGA Accelerator for Support Vector Machines Classification.* IEEE, May, 2012.

[18] T.Ojala, M.Pietikäinen, and D.Harwood. *A comparative study of texture measures with classification based on feature distribution.* Pattern Recongit. 29(1):51-59 1996.

[19] T.Mäenpää, and M.Pietikäinen. *Texture analysis with local binary patterns.* In eds. C.Chen, P.Wang Handbook of Pattern Recognition and Computer Vision 3rd Edition,pp.197 216. World Scientific, Singapore, 2005.

[20] T.Ojala, M.Pietikäinen, and T.Mäenpää. *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns.* IEEE, Transaction on Pattern Analysis and Machine Intelligence. (24):971-987, 2002.

[21] T.Ahonen, A.Hadid, and M.Pietikäinen. *Face description with local binary patterns: Application to face recognition.* IEEE, Transaction on Pattern Analysis and Machine Intelligence. (28):2037-2041, 2006.

[22] E.Osuna, R.Freund, and F.Girosi. *Training Support Vector Machines: An application to Face Detection.*

[23] *Face database 1* `http://www.ai.mit.edu/projects/cbcl` Center for Biological and Computational Learning, MIT