UNIVERSITY OF THESSALLY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

IMPLEMENTATION OF A CAR MAKE AND MODEL RECOGNITION (CMMR) ALGORITHM ON A RECONFIGURABLE PLATFORM

Diploma Thesis

Xiradakis Nikolaos

Supervisor: Bellas Nikolaos

June 2022

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

IMPLEMENTATION OF A CAR MAKE AND MODEL RECOGNITION (CMMR) ALGORITHM ON A RECONFIGURABLE PLATFORM

Diploma Thesis

Xiradakis Nikolaos

Supervisor: Bellas Nikolaos

June 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΥΛΟΠΟΙΗΣΗ ΕΝΟΣ ΑΛΓΟΡΙΘΜΟΥ ΑΝΑΓΝΩΡΙΣΗΣ ΤΗΣ ΜΑΡΚΑΣ ΚΑΙ ΤΟΥ ΜΟΝΤΕΛΟΥ ΑΥΤΟΚΙΝΗΤΟΥ ΣΕ ΕΠΑΝΑΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΗ ΠΛΑΤΦΟΡΜΑ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

Διπλωματική Εργασία

Ξηραδάκης Νικόλαος

Επιβλέπων: Μπέλλας Νικόλαος

Ιούνιος 2022

Approved by the Examination Committee:

Supervisor	**Bellas Nikolaos** Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Potamianos Gerasimos** Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Lalis Spyros** Professor, Department of Electrical and Computer Engineering, University of Thessaly

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The Declarant

Xiradakis Nikolaos

Acknowledgements

First of all, I would like to convey my gratitude to Professor Nikolaos Bellas for his guidance and patience, not only for the completion of this thesis, but also for his collaboration all these years through his courses.

Additionally, I want to warmly thank the PhD candidate Alexandros Patras for his practical support and counsel on the embedded part of this thesis.

Lastly, I will be forever grateful for the faith and support my family and loved ones gave me during the course of my undergraduate career. This work and my development as an engineer could never happen without them.

Diploma Thesis

IMPLEMENTATION OF A CAR MAKE AND MODEL RECOGNITION ALGORITHM ON A RECONFIGURABLE PLATFORM

Xiradakis Nikolaos

Abstract

Computer Vision algorithms are widely used in many every-day life applications in recent years, taking advantage of the advancements in processing power and data transferring speeds. Car Make and Model Recognition implementations are using such algorithms and have become an important element of modern, intelligent transport systems. This diploma thesis presents a complete approach of a real-time Car Make and Model identification system from video stream frames. It analyzes, the creation of a car frontal images database, image data pre-processing, region of interest and frame features extraction and classification model training. The proposed solution, is time efficient and accurate when running on a modern x86 processor and an attempt in implementing it on a reconfigurable Xilinx platform utilizing its low frequency processor and FPGAs is also examined.

Keywords: Computer Vision, Car Make and Model Recognition, Histogram of Oriented Gradients, Support Vector Machines, FPGAs

Διπλωματική Εργασία

ΥΛΟΠΟΙΗΣΗ ΕΝΟΣ ΑΛΓΟΡΙΘΜΟΥ ΠΡΟΒΛΕΨΗΣ ΜΑΡΚΑΣ ΚΑΙ ΜΟΝΤΕΛΟΥ ΑΥΤΟΚΙΝΗΤΟΥ ΣΕ ΕΠΑΝΑΠΡΟΓΡΑΜΜΑΤΙΖΟΜΕΝΗ ΠΛΑΤΦΟΡΜΑ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΚΥΚΛΩΜΑΤΩΝ

Ξηραδάκης Νικόλαος

Περίληψη

Οι αλγόριθμοι Υπολογιστικής Όρασης χρησιμοποιούνται ευρέως σε πολλές εφαρμογές της καθημερινής ζωής τα τελευταία χρόνια, εκμεταλλευόμενοι τις εξελίξεις στην επεξεργαστική ισχύ και τις ταχύτητες μεταφοράς δεδομένων. Οι εφαρμογές αναγνώρισης της μάρκας και μοντέλου αυτοκινήτων, χρησιμοποιούν τέτοιους αλγόριθμους και έχουν εξελιχθεί σε ένα σημαντικό στοιχείο των σύγχρονων, ευφυών συστημάτων μεταφορών. Αυτή η διπλωματική εργασία παρουσιάζει μια ολοκληρωμένη προσέγγιση ενός συστήματος αναγνώρισης μάρκας και μοντέλου αυτοκινήτου πραγματικού χρόνου, από εικόνες μιας ροής βίντεο. Συμπεριλαμβάνει μεταξύ άλλων, τη δημιουργία μιας βάσης δεδομένων εικόνων αυτοκινήτων, την προεπεξεργασία των δεδομένων, την εξαγωγής περιοχής ενδιαφέροντος και χαρακτηριστικών, αλλά και την εκπαίδευση μοντέλων ταξινόμησης. Η προτεινόμενη λύση, είναι χρονικά αποδοτική και ακριβής όταν εκτελείται σε έναν σύγχρονο επεξεργαστή x86 και εξετάζεται επίσης μια προσπάθεια υλοποίησής της σε μια επαναπρογραμματιζόμενη πλακέτα της Xilinx χρησιμοποιώντας τον χαμηλής συχνότητας επεξεργαστή και τις FPGA που διαθέτει.

Λέξεις-κλειδιά: Υπολογιστική Όραση, Αναγνώριση Μάρκας και Μοντέλου Αυτοκινήτου, Ιστόγραμμα Προσανατολισμένων Διαβαθμίσεων, Μηχανές Διανυσμάτων Υποστήριξης, FPGAs

Table of Contents

List of Figures

List of Tables

Abbreviations

CMMR	Car Make and Model Recognition
ITS	Intelligent Transport Systems
LPR	License Plate Recognition
HOG	Histogram of Oriented Gradients
SVM	Support Vector Machine
CPU	Central Processing Unit
FPGA	Field Programmable Gate Arrays
SoC	System-on-a-Chip
SURF	Speed Up Robust Features
SIFT	Scale Invariant and Feature Transform
i.e	id est
FPS	Frames Per Second
CNN	Convolutional Neural Network
API	Application Programming Interface
HLS	High Level Synthesis
NVME	Non Volatile Memory
RAM	Random Access Memory
Mb, Gb	Megabyte, Gigabyte
ms, ns	milliseconds, nanoseconds
PL	Programmable Logic

Chapter 1

Introduction

The rising number of cars usage over the last decades has made matters of safety and efficiency on the roads of utmost importance. Modern Intelligent Transport Systems (ITS) offer a plethora of solutions that aim to improve the management and operation of transport systems by automating their functions. Such applications introduce automated vehicular surveillance (AVS) frameworks, advanced driver-assistance systems (ADASs), and traffic activity monitoring. Car Make and Model recognition grows great interest between these applications in recent years, due to the advanced security it can offer in an ITS.

A CMMR system can be used when buying or selling cars, where the interested buyer can have, after taking a photo of the vehicle, a good information about its specific make and model. It can further be used for statistical studies of traffic in a specific area but also in monitored parking areas where the owner of the site needs a complete knowledge of the cars it hosts at any time. However, the basic need of a CMMR system is its use in cases of imperative car identification for law enforcement. In surveillance applications, for example, it can be used as a complement to Automatic License Plate Recognition (LPR) systems for suspected cars detection.

LRP techniques are widely used for the recognition of a vehicle's make and model. They try to achieve this by cross-checking the detected license plate data with those existing in the registered vehicles database. However, such systems become inoperative when there is poor image quality or fraudulent use of license plates. A license plate can be forged, contain indistinguishable characters or be worn out, and in all these cases we will have failed car identification. CMMR offers a robust and accurate identification solution when LPR algorithms fail and can also be used as detection validation when they can be usable.

Figure 1: Cases in which LPR algorithms fail [1]

The process of creating an efficient CMMR system can be highly challenging. Different lighting conditions effect the car's appearance and strong intra-class similarity usually exists among models of the same car make, especially when they are of the same generation. However, studies have shown that these problems can be tackled with the correct analysis and usage of computer vision algorithms.

1.1 Related Work

Analysis of the literature presents that the majority of the recent CMMR publications approach the issue with a feature-based approach. Feature based methods use local or global invariant features to classify car models, and hence their performance depends on the accurate representation of those features. Some use the full front or rear view of the car to examine their features in their workflow and others try to localize certain parts of a car's body, basing their recognition process on those locations. For the classification process, those features are fed in a classifier network in order for their class to be predicted. Support Vector Machines (SVM), Nearest Neighbor Classifier (NNC) and Artificial Neural Networks (ANN) appear to be the most used classifiers in CMMR solutions.

Psyllos et al. [2] used an LPR algorithm for vehicle frontal view segmentation first and then a Probabilistic Neural Network that predicts the car manufacturer based on the logo. The same PNN network is fed with SIFT keypoints extracted from the frontal car view to output the predicted model. Baran et al. [3] developed a system that extracts SURF features from frontal car images and then after partitioned in a set of clusters (vocabulary generation), are given in a trained multiclass SVM model to predict their labels. Chen et al. [4] proposed a symmetrical SURF descriptor for vehicle detection, that after a car's ROI is detected, its

HOG features are extracted. The make and model are finally predicted using a novel Hamming distance classification scheme. The same detection technique was used by Manzoor et al. [5] that used HOG and GIST feature descriptors for their ROI. For the classification process, they applied SVM and Random Forest classifiers to the extracted features. Pearce et al. [6] included in their work a comparison of many feature extraction methods including Canny Edge Detection, Squared Mapped Gradients Harris Corner strengths. To classify their data, they examined Nearest Neighbors and Naïve Bayes techniques. A Bag of Expressions approach was presented by Jamil et al. [7], that use a combination of HOG descriptor and different keypoint detectors to extract car features. Their classification process was completed with a SVM classifier. Lee et al. [1] utilized and optimized the SqueezeNet, a smaller CNN network for their whole recognition process.

1.2 Thesis Subject and Contributions

As described above, the problem of CMMR has been addressed by many researchers in recent years as its importance in ITS has grown. This thesis describes a CMMR solution that could be able to work on a low power embedded platform, which was not the final goal for any of the examined approaches. The workflow it introduces was developed using C++ and Python programming languages, and tested on both Windows and Linux operating systems. We are going to present all the steps needed in order to create an efficient CMMR algorithm having zero starting resources.

At first, a database that could match our needs was created. Some projects that base their recognition on certain car parts such as the logo or the headlights could suffer from occlusion and poor image quality problems. For this reason, in our work the whole frontal view of a car was used. The robust HOG feature descriptor was applied for the extraction of cars image features. Those features are then fed into a SVM network for training the classifier model that will eventually predict a car's make and model. Our target was to create a system that would be fast enough to be used for real time applications, meaning to be able to recognize a moving car's model from video frames. This is achieved by finding the region where the car lies in each frame and then giving this area's features as an input to our pre-trained SVM model. A series of image processing algorithms and contour detection functions were utilized in order to find this region of interest. The computational performance and accuracy

were sufficient enough for our needs and they are compared with other similar methods from the literature.

The last part of this work includes an attempt to make the proposed framework work on the Zedboard evaluation kit provided by our university. It is an embedded platform that has a CPU together with reconfigurable hardware components. The need for less power while at the same time utilizing less resources and creating faster systems has become a norm in recent years. This is why such heterogenous platforms exist, to accelerate applications by using their reconfigurable hardware parts. Our method's performance while running through a real-time OS on this device's CPU is analyzed. Finally, hardware optimizations examined with the Vitis HLS tool are proposed.

1.3 Thesis Outline

Having described our thesis workflow, the remainder of this dissertation is organized as follows:

- Chapter 2. In this part, the database creation is analyzed along with the image manipulation taken into account for every inserted car image.
- Chapter 3. Here the feature extraction technique is firstly described. Then, the classifier training process and the classification metrics are reported.
- Chapter 4. This chapter gives a detailed description of the real time CMMR flow proposed and discusses the performance results along with further optimizations.
- Chapter 5. A description of the embedded platform and its performance while running the algorithm is given here, together with a detailed attempt for hardware optimization.
- Chapter 6. Finally, a conclusion is drawn about the proposed solution and future improvements are also presented.

Chapter 2

Database Creation

2.1 Database Selection

The first challenge in this problem was to create or find a database, from which we want to extract the features that will be later used in the pipeline of car identification. For this topic, several databases have been developed such as NTOU-MMR [4] which has been used by several related works [5,7] and others that are not publicly available from their creators [2,6]. All the existing databases, however, do not correspond to Greek reality, which is our goal for validation purposes, either because they consist of vehicles circulating only in the USA or China, or because they are based on other aspects of the car and not on the front, which is the requested for the proposed solution. Therefore, the creation of a new database became a priority. Due to the fact that it can be used in many cases of identification, as this is how the cameras are usually mounted on streets and parking lots, only images that clearly depict the front view of cars were selected for the new database. The selection of the images was made manually by websites of used and non-used cars such as autoscout24.com and car.gr and included cars that were high in sales in both the Greek and the European market in the last decade. The following table illustrates all the car models in the database and the number of images collected for each one.

Model	Date	Number of Images
AUDI A3	2012	24
BMW SERIES 1	2016	23
CITROEN C3	2014	27
FORD FIESTA	2017	28
MERCEDES A-CLASS	2019	26
OPEL CORSA	2015	21
RENAULT CLIO	2012	23
TOYOTA YARIS	2015	28
VW POLO	2014	22

Table 1: Models of the created database.

2.2 Selection Criteria

The point of attention at this stage, was the images chosen, only depict the front view of the vehicle and no other parts of it such as large part of the roof or side views such as the car doors. Thus, all the images in this database are taken from the imaginary straight line that connects the camera with the center of the vehicle's mask with a height deviation between them of not more than half a meter, so that only the car front is clearly visible. The edges of each image are a few centimeters away from each side of the car, i.e. right and left mirror, roof (top) and tires contact with the ground (bottom). Moreover, 25 images of non-cars were added to the database, so that if a picture of a non-car is examined, to be correctly predicted at the classification stage.

Since all the photos are not captured from the same distance, lenses of different focal lengths are needed for different distances from the object, in order to achieve the object always being the same centered and equidistant from the desired edges. In Figure 2 below, the W-width car is captured from points a and c. As the distance d1(c-b) is less than the distance d2(a-b), the angle θ is obtained greater than the angle φ to record the same desired view. In the images selected, care was taken to ensure that there is not a very large viewing angle due to ultra-wide angle lenses and thus no visual distortion caused to the photographed object (vehicle). Finally, when needed, images were cropped at their edges such as the appearance of the vehicle's front view is properly and equally centered.

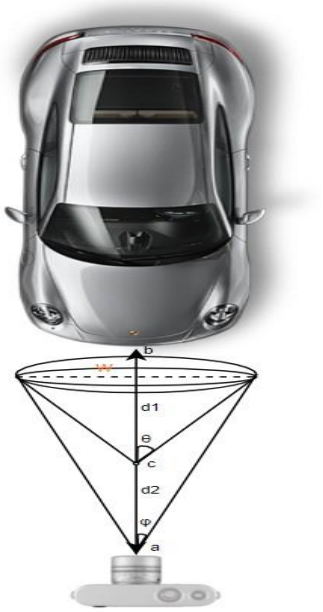

Figure 2: Distances and viewing angles when capturing an object.

Chapter 3

Classification Process

Having all the car photos collected and organized into specific folders, the next step is to develop a classification model for this multiclass problem, that takes image features as inputs and outputs their Make and Model. Image features should be extracted using a feature description algorithm and then fed into the selected classifier network in order to train the classification model.

3.1 Histogram of Oriented Gradients

The feature descriptor algorithm we use, should be descriptive enough for the differences between the models to be clearly visible in the next stage of identification, since it is a problem involving multiple car models (classes). For this, the Histogram of Oriented Gradients was chosen, as it has better accuracy than other descriptors like Canny Edge Detector and SIFT (Scale Invariant and Feature Transform) because it uses the magnitude along with the angle of the gradient to compute the features. HOG became known on a large scale in 2005, when Navneet Dalal and Bill Triggs presented their work on pedestrian detection using HOG descriptors at the Conference on Computer Vision and Pattern Recognition (CVPR) [8]. It is now widely used for object detection projects and other computer vision applications. Like most feature descriptors, HOG extracts useful information from an image and creates a feature vector of user specified length (depending on the algorithm options and need for accuracy). In our case, the information is collected by the distribution of directions (histogram) of gradients for each image. The steps for histogram calculation are described below.

3.1.1 Input Images Preprocessing

Every image should have the same feature vector length as an output of HOG and since they are not all of the same size, a resizing to specific dimensions must happen to all of them. It was noticed that the suitable size for those car photos is resizing them to 128x128 pixels, as

in this size not much useful information is lost and the feature extraction becomes more time efficient due to smaller feature vector length. To further limit algorithm's computation time, each resized image was converted from RGB to grayscale, reducing the color channels from three to just one and producing an image more fitting to edge detection functions.

3.1.2 Calculating Image Gradients

In order to create the histogram, we must initially calculate each pixel's vertical and horizontal gradient, which is achieved by filtering the image (convolutionally) with the kernels:

-1	0	1

-1
0
1

Table 2: Image Filtering Kernels.

The gradient of each pixel is computed as shown below:

$$Gx(x,y) = f(x + 1, y) - f(x - 1, y) \quad (3.1)$$

$$Gy(x,y) = f(x + 1, y) - f(x - 1, y) \quad (3.2)$$

where f(x,y) is the image pixel with coordinates x and y, Gx is the gradient of the horizontal direction and Gy is the gradient of the vertical direction.

Then, the magnitude and orientation of each pixel are computed as:

$$Magnitude = G(x,y) = \sqrt{Gx(x,y)^2 + Gy(x,y)^2} \quad (3.3)$$

$$Orientation = \theta = \tan^{-1}\frac{Gy(x,y)}{Gx(x,y)} \quad (3.4)$$

3.1.3 Creating Histograms of Gradients in 8x8 cells

Next step in the process, is the division of the image into 8x8 cells and the calculation of histogram of gradients for every such cell. In this way, we achieve not only a more compact representation but also a more robust to noise one.

For each cell, the values of magnitude from 64 pixels are placed depending on their direction (orientation) in 9 bins, one bin for every 20 degrees from 0 to 180, cumulatively. If the direction of a pixel gradient is between two bins, then the value of its magnitude is split accordingly, based on the distance from each respective bin, in 2 parts and then these are added to each bin. For example, if a pixel has a magnitude of 4 and an angle of 10 degrees, it is between the bins with degrees 0 and 20, so we will add 2 to 0 degree bin and 2 to 20 degree bin, as shown in Figure 3.

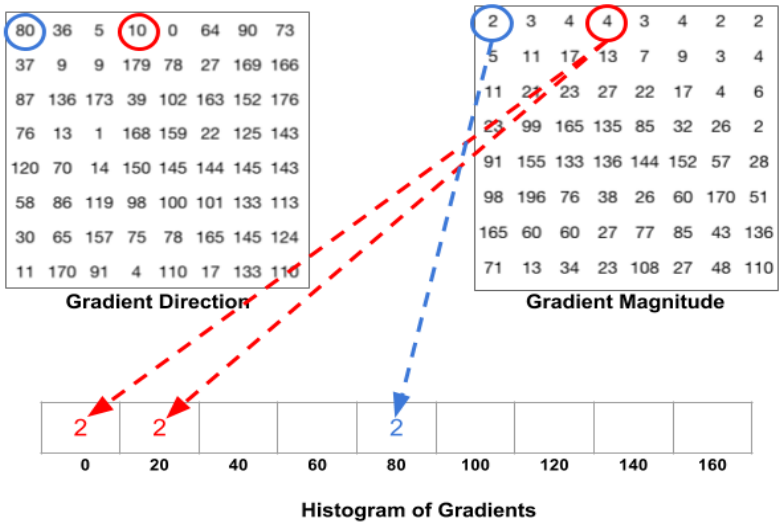

Figure 3: Attaching the magnitude values in bins. [9]

3.1.4 Block Normalization

To calculate the final feature vector, we divide the image into blocks of 16x16 pixels, i.e. into sets of 4 cells (2x2), so each block is described by 4 concatenated histograms combined, to form a vector of 4 cells x 9 bins = 36 elements. Since each area of the image does not have the same lighting intensity, the 36 values of each histogram have to be normalized with their L2 norm before being inserted into the final feature vector. For a given vector V = [v1, v2, v3…v36] in order to normalize it, will divide its values by the square root of the sum of squares of the values: $n = \sqrt{v1^2 + v2^2 + v3^2 + \cdots + v36^2}$ (3.5) and the normalized vector is computed as: $V_{normalized} = [\frac{v1}{n}, \frac{v2}{n}, \frac{v3}{n}, \dots, \frac{v36}{n}]$ (3.6)

Eventually, the image is scanned with 16x16 blocks with a stride of 8 pixels (half block) in an overlapping manner, into 15 horizontal and 15 vertical positions (128pixels/8pixels stride -1 block width = 15) as shown in Figure 4. Hence each block is described by a vector of 36 elements, the final feature vector length is 15 x 15 x 36 = 8100.

Figure 4: Blocks size, Stride size and scanning positions given the numbers described.

3.2 Classification Algorithm

After having extracted the "useful" data of every car depiction, those data need to be fed into a classifier network in order to train the classification model that will eventually predict the matching car make and model for every image it is given as an input. For this purpose, a Multiclass Support Vector Machine [10] was selected for our non-binary problem, as it is proven from related works [3,5,7] that it works respectably with HOG features and it is a memory efficient prediction method.

An SVM can be described as a representation of the input characteristics (samples) at points in space which aims for their maximum separation by a hyperplane in classes. Having two parallel hyperplanes to separate the two classes so that the distance between them is the maximum, the desired maximum-margin hyperplane is the one lying halfway between them. This hyperplane therefore depends on the marginal values – samples that are closest to it and these samples are called support vectors.

The above description follows the formula:

$$For\ input\ vectors\ x_i \in R^p\ ,\ i = 1, \dots, n\ with\ y_i \in \{-1,1\}^n\ our\ goal\ is\ to\ find$$
$$w \in R^p\ and\ b \in R\ such\ that\ the\ prediction\ given\ for\ every\ sample$$
$$by\ sign(w^T \varphi(x) + b)\ is\ correct\ for\ most\ samples.$$

More specifically we are solving the problem:

$$min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \quad \text{subject to} \quad y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \quad (3.7)$$

for $\zeta_i \geq 0\ , i = 0, \dots, n$, with w being the weight vector, b the distance from axis origin, C > 0 a regularization (penalty) parameter and ζ the distance from their correct margin boundary.

For this work, the LIBSVM library [11] was used, which provides support for many types of SVMs and is developed for use in several programming languages. This implementation was made with the C-SVM type with parameter C = 1.0 and linear kernel function, having as input the HOG characteristics of each image (X) and the number of its class (Y), corresponding to the name of the car model.

As this is a multiclass problem, LIBSVM implements a “one-against-one” approach to classification, i.e. for K number of classes, K*(K-1)/2 classifiers are created, each of which is trained with data from 2 classes. This way the problem splits into many binary problems. When solving - classifying for each binary problem, the class that will have in its total the most features - points gets a vote and so the final predicted class and output of the classifier is the class with the most votes.

From our database, 80% of the images were used for training (train set) the SVM predictor model and the remaining 20% for testing its accuracy (test set). The selection of images for training and testing was done in a random way each time and many SVM models were trained and saved in files to create a more complete statistical view of the whole algorithm. The testing results were similar between all the trained models and their overall performance is described in the next section.

3.3 Classification Results

In order to evaluate the classifier prediction results, three widely used metrics in classification problems were examined: precision, recall and F1-score. To explain the above metrics, the following table presents a possible prediction in relation to the expected (actual) value.

		PREDICTION	
		POSITIVE	NEGATIVE
REAL VALUE	POSITIVE	*TRUE POSTITIVE(TP)*	*FALSE NEGATIVE(FN)*
	NEGATIVE	*FALSE POSITIVE(FP)*	*TRUE NEGATIVE(TN)*

Table 3: Confusion matrix of binary prediction results.

Precision is derived from the fraction:

$$precision = \frac{TP}{TP + FP}$$

In other words, it presents the percentage: from the total positive predictions of the classification model, how many are really positive.

Recall or otherwise Sensitivity shows the percentage of how many from the set of really positive values the model predicted as positive and is calculated by the fraction:

$$recall = \frac{TP}{TP + FN}$$

The F1-score displays the harmonic mean of accuracy and recall.

$$F1 - score = 2 * \frac{precision * recall}{precision + recall} = \frac{2TP}{2TP + FP + FN}$$

As each test set is randomly selected from the entire database as mentioned at the end of subchapter 3.2, the samples given from each class are different in every trained and examined model. Taking into account this imbalance, the above metrics were noted for each trained SVM model based on the weighted average between the classes. The results obtained on average from 25 trained SVM models are presented in the table below.

Precision	Recall	F1-score
98,13%	97,56%	97,59%

Table 4: Average of performance metrics from trained classification models.

Although it may be a product of our relatively small database, the models trained on those data generate almost perfect results, with an average accuracy of 98%. The HOG features extraction and SVM model training were developed in Python and it took almost 5 seconds to complete the whole process in our system.

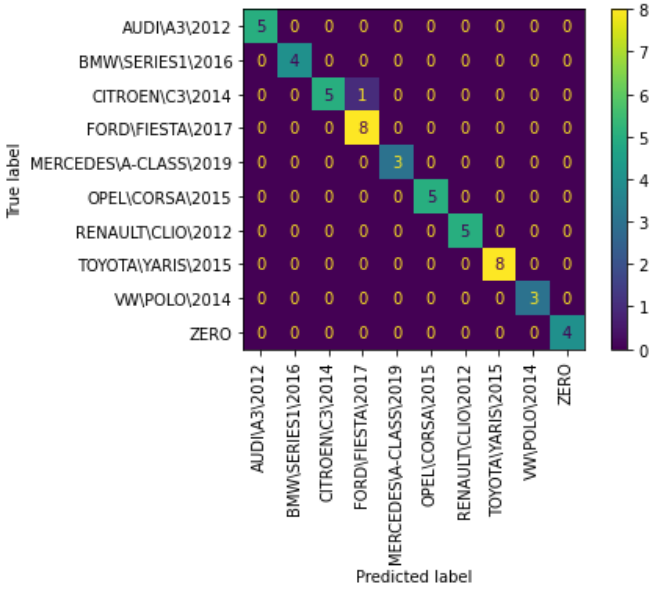

Figure 5: Confusion matrix of a trained SVM model.

Chapter 4

Car Make and Model Recognition from video frames

With the prediction process described, we can classify a static car's front view image to the specific Make and Model if it exits in the database. The goal however is developing a real-time recognition approach, where given a video stream, the algorithm will be able to recognize where and if a vehicle exists in each video frame and then predict its class using the classification process already discussed. This chapter analyzes the creation of such a real-time system presenting the frame preprocessing routine, the recognition technique and identification method and the performance measurements of this system. The open source computer vision library OpenCV [12] was used to form all the required steps.

4.1 Frame Preprocessing

Having a video file as input, the first thing is to grab every frame of it in order to process it. The approach selected, recognizes a moving object using a background segmentation method and all the functions below contribute to drawing a low noise foreground image in each frame.

4.1.1 Grayscale Conversion

After grabbing the video frame, we have an RGB image which has 3 color channels and a size of 1280x720 pixels in our case. We convert this image to Grayscale, which is a description of colors based on their lightness, with values ranging from 0 (black) to 255 (white). The new image may have less information but is a simplified depiction since it has only 1 color channel to process and that reduces the computational requirements. There are many ways to achieving this color conversion like the Average method, the Lightness method, the Gleam method etc. [13] OpenCV uses the Luminosity method which is a weighted one and tries to match the human brightness perception. The formula for converting every pixel value is:

$$Gray = 0.299 * R + 0.587 * G + 0.114 * B \qquad (4.1)$$

4.1.2 Gaussian Blurring

The grayscaled image is then convolved by a Gaussian kernel of size 3x3 in order to smooth sharp edges and reduce unwanted details. The formula calculating each pixel value using the Gaussian function is described as:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (4.2)$$

where x is the distance to the horizontal kernel center, y is the distance to the vertical kernel center and σ is the standard deviation of the Gaussian kernel. [14]

4.1.3 Background Subtraction

The next challenge is to have a clear view of foreground objects which may be moving cars, so a background subtraction algorithm is used in order to extract the moving foreground from the static background in the already blurred image. Several subtraction methods have been developed and their performance has been thoroughly examined in [15]. Most of these algorithms include: a) processing of N frames to provide the background image, b) definition of background representation, c) updating the model for every H number of frames, defined by the user and d) classifying pixels to sets of background and foreground. In this work a Gaussian Mixture-based segmentation method is used as described in [16].

In the Mixture of Gaussians technique, each background pixel is approached by modelling it as a mixture of k Gaussian distribution models, with k being between 3 and 5. In this method, it is assumed that each different distribution represents the different background and foreground colors. The model is weighted meaning that each distribution's weight is proportional to the amount of time each color remains on that pixel. Therefore, when a pixel's distribution weight is high, that pixel will be noted as background. This method has the ability to detect shadows if this option is activated. In our scenarios, the number of frames taken into account to define the background image, also known as history, was opted to be

60 and the threshold of distance between each pixel and the model to decide whether it belongs to the background or not, was selected to be 48.

4.1.4 Morphological Transformations

At this point, we have an image where segmented as foreground objects are totally white, while the areas forming the background are black. This subtraction may not be without noise however and some black spots may exist inside the foreground white areas. To eliminate those points, a technique known as Closing was followed, where Dilation and subsequently Erosion filtering is operated on the image.

4.1.4.1 Dilation

This function primarily expands the white area of foreground objects by dilating the binary image with a structuring element. This element is a kernel, which shape and size determines the pixel neighborhood over which the following check is involved:

$$\texttt{dst}(\mathrm{x},\mathrm{y}) = \max_{(\mathrm{x}',\mathrm{y}'):\,\texttt{element}(\mathrm{x}',\mathrm{y}')\neq 0} \texttt{src}(\mathrm{x}+\mathrm{x}',\mathrm{y}+\mathrm{y}') \qquad (4.3)$$

where dst is the destination – output image, src is the input image and x,y the coordinates of each pixel.

In this way, if at least one pixel under the kernel is “1” the pixel element is determined as “1”. The kernel shape in our case is a rectangle with a 3x3 size, meaning we have a 3x3 matrix of ones.

4.1.4.1 Erosion

The Erosion process is the opposite the Dilation one. It erodes the foregrounds’ objects boundaries by eroding the image with the structuring element mentioned above. The formula that determines each pixel’s color here is:

$$\texttt{dst}(x,y) = \min_{(x',y'):\,\texttt{element}(x',y')\neq 0} \texttt{src}(x+x',y+y') \qquad (4.4)$$

In this case, if only all pixels under the kernel are “1”, the pixel element is considered as “1”. Kernel shape and size here are the same with those used for Dilation. The final preprocessed image is presented in the upper part of Figure 7.

4.2 Object Recognition

Now that we have a clearer separation of foreground areas from the black background, we need to outline those areas and then threshold them depending on their size, in order to choose which of them are large enough to be considered as Regions of Interest for our prediction system. A useful method for object recognition problems is to find contours in the image.

4.2.1 Contours Detection

Contour is a curve, joining all continuous points that have the same color in our problem, forming this way the boundary of an object. The opencv findContours function [18], stores the (x,y) coordinates of object boundary points but has two Contour Approximation methods to decide which of all coordinates to keep. The first one is CHAIN_APPROX_NONE and stores all the boundary points for each contour. The other is CHAIN_APPROX_SIMPLE, that compresses horizontal, vertical, and diagonal segments and keeps only their end points. For example, in a straight line we do not need all of its points to describe it but the start and the end of it. The later method was selected as it has very lower memory requirements compared to the former one.

Another interesting option when searching for contours is the Contour Retrieval technique. An object may contain another shape-object inside its area and we can name the outer object as parent of the inner ones (children) declaring this way levels of contours hierarchy. The specific OpenCV function provides four Contour Retrieval modes: a) RETR_EXTERNAL which retrieves only the outer contours (the parent ones), b) RETR_LIST that does not establish any parent child relationship between the extracted contours, c) RETR_CCOMP where every contour in the image is retrieved and sorted in a 2 level hierarchy way: level 1 for all the outer contours (object boundaries) and level 2 for all the inner contours and d) RETR_TREE in which not only all the contours are retrieved but also organized with a multi-

level hierarchy, meaning that even if a contour is contained inside four other contour areas it will have its own hierarchy, with its level defined by the parent-child relationship described. It becomes obvious that RETR_TREE requires the maximum computation time between them, because of the level of detail it provides and RETR_EXTERNAL takes the least time to execute. With the preprocessing already discussed, it is almost guaranteed that no small "holes" will exist inside a bigger contour, resulting in a simpler hierarchy between the contours and for that, the RETR_TREE technique was selected.

Figure 6: Contours detected using CHAIN_APPROX_SIMPLE method. [17]

4.2.2 Contours Thresholding

With all the contours stored, we need to declare which of them can represent a Region of Interest, meaning an area inside the whole frame where a moving car may exist. Considering that the object we want to classify must be detailed enough for our classifier to predict, all contours with area less than 128x128 pixels are discarded. For those contours with area larger than that, we draw a straight rectangle around them.

Cars inside a video frame in our scenario appear in an almost rectangular shape, so our Region of Interest should be selected likewise. Therefore, we threshold all the aforementioned rectangles based on their height and width and keep only those that obey to the relation: $0{,}7 * width \leq height \leq 1{,}3 * width$. The remaining ones constitute our Regions of Interest of this frame. The area where this rectangle lies is cropped from the

original RGB frame (not the final processed image) and copied to a new image. This image is fed to our classification process described in chapter 3 in order for its class to be predicted. The SVM prediction (output) can then be stored and used in many ways, including drawn wherever in the frame we want or printed on the screen. An example of this can be seen in the lower part of Figure 7. Every prediction is written in a list and every (user specified) number of frames we apply majority voting in it in order to decide by the number of occurrences which Make and Model the depicted car is.

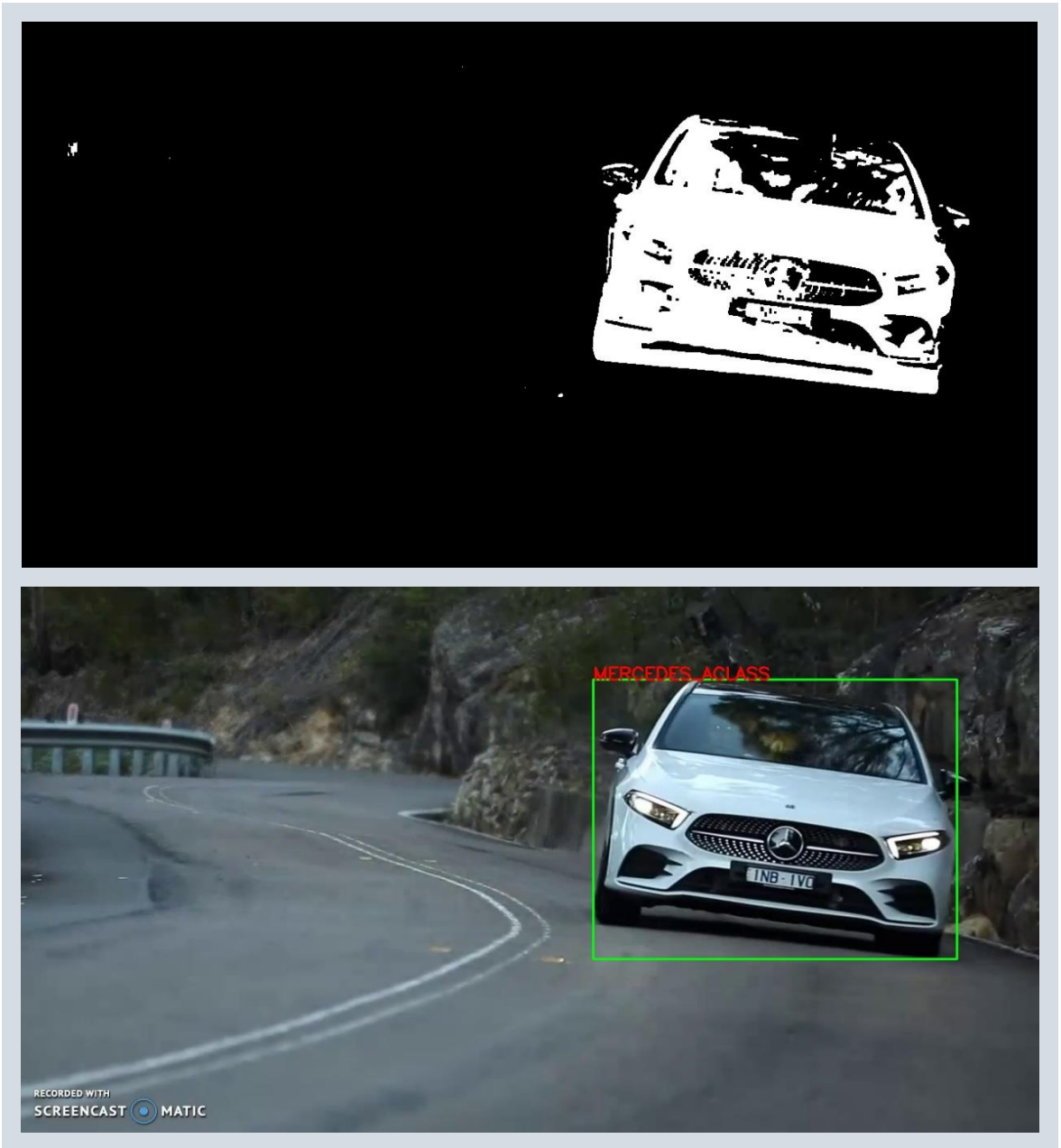

Figure 7: Final preprocessed image (Up) and predicted car model in this ROI (Down).

A full representation of the steps followed for training the classification model and recognizing the Make and Model of cars from video frames is presented in the following diagram.

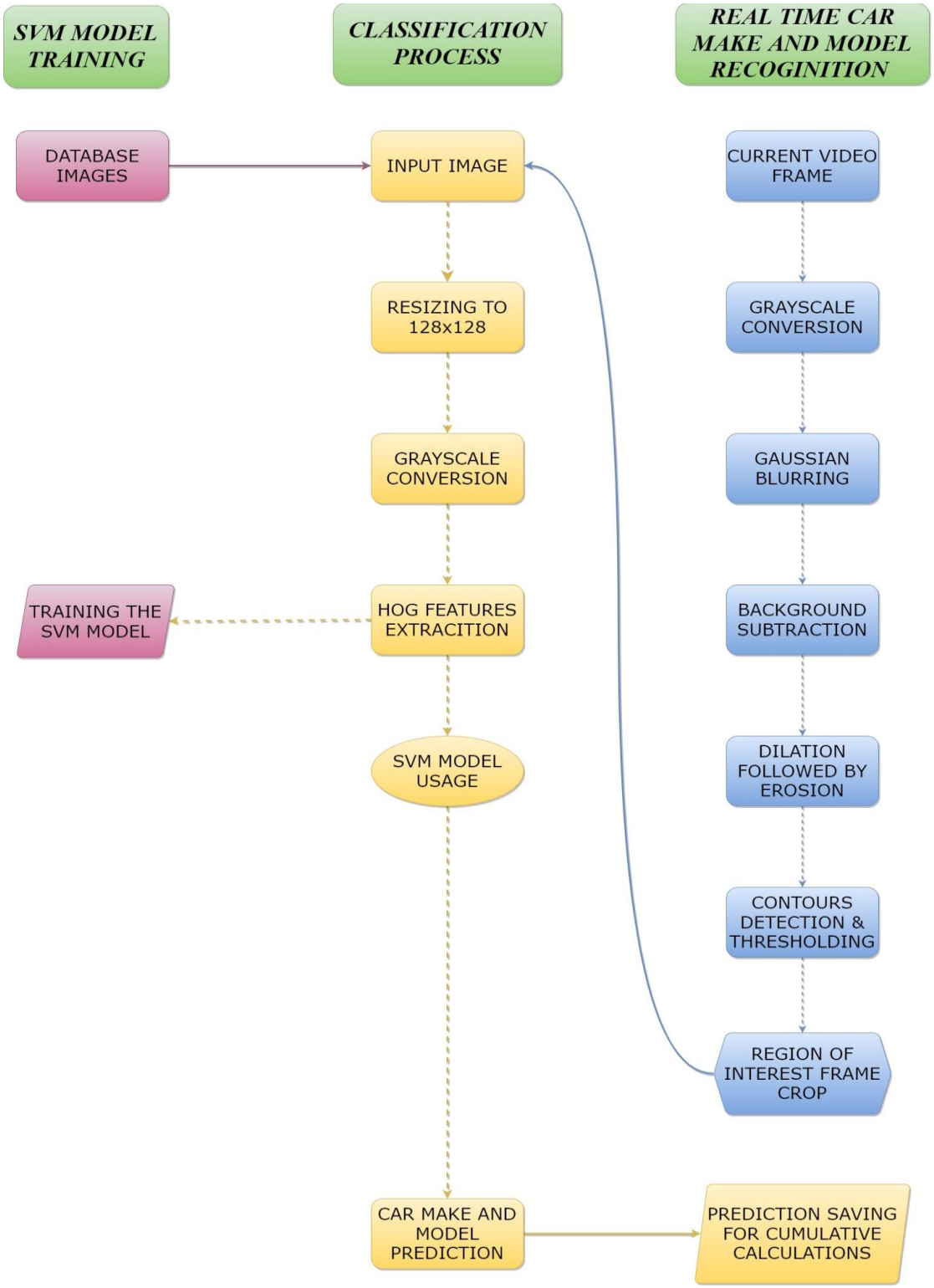

Figure 8: SVM Training and real-time CMMR process flows.

4.3 Recognition results and software optimizations

The above CMMR process was tested on a modern laptop with an 8-core AMD Ryzen 4800H CPU, 16 GBs of DDR4-3200 RAM and NVME storage. While having MP4 videos with a framerate of 25 to 30 FPS as input, the described solution can process more than 30 frames per second in this system, achieving to satisfy our real-time requirements. In particular, each frame's maximum time taken to process is less than 5ms. This makes this system able to process more than 250 FPS when image display functions are not used. The computation time is distributed as: 60-65% for the image preprocessing stage, 11-15% for Contours Detection and Thresholding and 16-22% for the classification process. The timing distribution for 3 videos of different total time and different capturing locations is presented in the next chart.

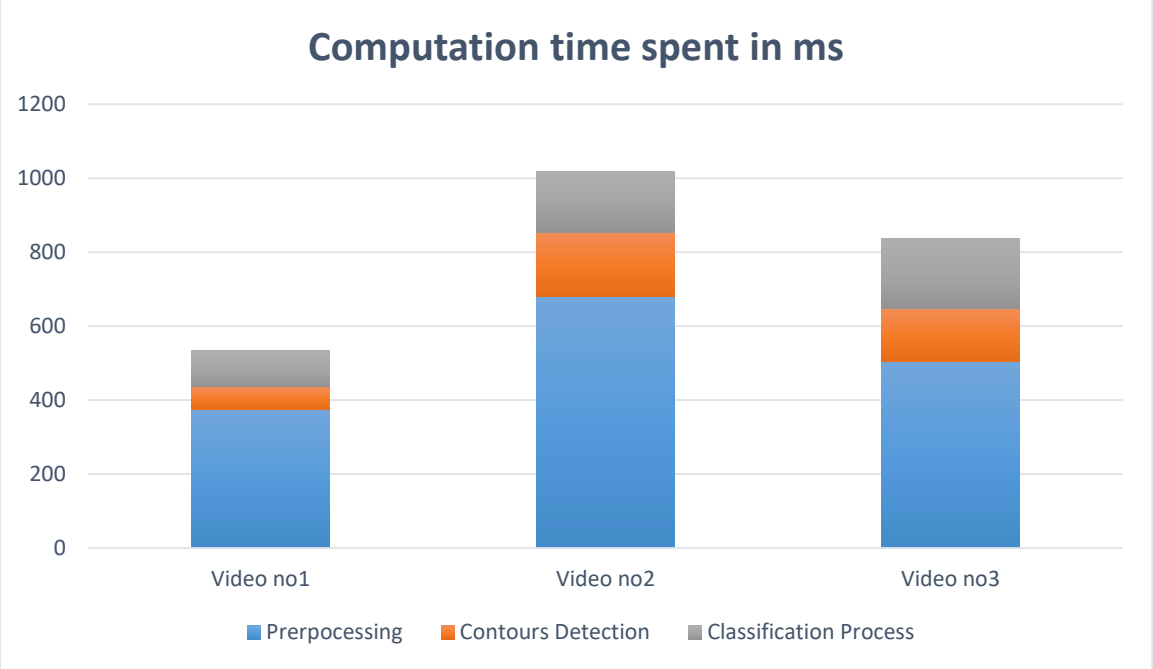

Figure 9: Timing totals of algorithm segments.

Since it is not possible to optimize the way SVM network predicts nor the contours detection functions we used, effort was made to reduce the frame's preprocessing computational complexity. The Background Subtraction method mentioned, took almost half of the preprocessing time and it became evident that this part could be improved. This foreground segmentation can happen manually, by subtracting our current frame's blurred image from

an N previous frame's one and then threshold this difference in order to have a binary image with black background and white foreground objects. The N previous frame image represents the background out of which we want to segment the moving object area existing in our current frame. N can be changed depending on video demands (how fast objects - cars move) and in tested scenarios it was sufficient enough to assign it to 45. To make it clearer, every 45 frames we copy this current frame and use it as a background image for the next ones. The accuracy falls down a bit but it is very acceptable compared to the timing gains this change offered. Additionally, the optical changes happening from frame-to-frame are usually minimal, so we can apply our whole recognition process every 2 or 3 frames if we want to drastically reduce the computation time.

Comparing our implementation with other CMMR approaches from the literature shows that a more simplistic approach like ours appears to be more suitable for a modern real-time recognition system. The accuracy when trying to identify a car model from a moving foreground object may be less than most of the examined methods but performance/accuracy ratio is far better than any other approach. The more recent computer specifications used in our work have surely affected this performance but the usage of of HOG and SVM instead of a combination of feature descriptors and Neural Networks, proved to be a lot more time efficient. A detailed comparison with other works is presented in the next table.

Work	Feature Descriptor	Classifier	Samples/Classes	Accuracy	Time in ms (FPS if real-time)
Psyllos et al. [2] (2011)	SIFT	PNN	110 / 11	85% make / 54% model	363 / no real-time
Pearce et al. [6] (2011)	Many feature descriptors	Nearest Neighbors / Naïve Bayes	262 / 74	96% on best combination	100 / no real-time
Chen et al. [4] (2015)	Symmetric SURF	Sparse representation and hamming distance	4502 / 39 NTOU-MMR dataset	91%	43.83 FPS
Baran et al. [3] (2015)	SURF / SIFT	SVM	3859 / 17	91	33.9 + 4.24
Manzoor et al. [5] (2019)	HOG	Random Forest / SVM	2725+3110 / 39 NTOU-MMR dataset	94.53% / 97.89%	35.7 FPS / 13.9 FPS
Lee et al. [1] (2019)	Same as Classifier	Residual Squeeze Net	291,602 / 766	96.33%	109.5
Jamil et al. [7] (2020)	HOG + Bag of elements	SVM	NTOU-MMR dataset	98.2% best options	24.7 FPS best options
Our approach	HOG	SVM	225 / 10	98% static / 85% real-time	~5ms / more than 250 FPS

Table 5: Comparison of our work with other implementations from the literature.

Chapter 5

Embedded System Analysis

Our system runs great on a high-performance laptop but what happens if we were to test it in a more compact and portable device with low power consumption requirements, appears as a challenge. Mobile platforms usually rely on less powerful batteries, making their power draw limited to low levels. As a result, although CMMR algorithms have worked with high accuracy levels on GPUs and traditional CPUs, those processing units do not pose as an option in our case, due to their high-power consumption. This problem can be tackled with the use of FPGAs. They have proved to be power efficient, highly adjustable as they are re-configurable and built for simultaneous computing. Many Computer Vision algorithms have been implemented in FPGAs over the last decade including LPR, Pedestrian Detection and Objects Recognition even with the use of CNN architectures. Before we analyze the performance of our system in an embedded device, we present some basic knowledge about the circuits and tools used.

5.1 FPGA

A field programmable gate array is an integrated circuit designed to be re-configurable by a developer for every application he wants to implement. FPGAs are formed using an array of configurable logic blocks that are wired together via reconfigurable interconnects and their configuration is specified by a hardware description language (HDL). Their hierarchy can be configured to create combinational logic gates or even simpler ones like XNOR and AND. Modern hard logic blocks usually consist of look up tables (LUTs), flip flops (FFs), high speed I/O buses as well as embedded memory objects (RAM blocks). The die shrinkage in recent years has allowed major FPGA manufacturers like Xilinx and Intel providing FPGAs with more than 40 billion transistors. Due to this advancement, they are nowadays used for AI applications and Data Centers acceleration.

Figure 10: Intel Stratix 10 FPGA. [19]

FPGAs can be regularly confused with Application-Specific Integrated Circuits (ASIC). Their most obvious difference is that FPGAs are multipurpose chips that can be reprogrammed for multiple applications while ASICs are manufactured for a specific application. ASICs are more efficient and consume less power compared to FPGAs but are not nearly as flexible as them. Although ASIC's more complex design results in higher performance, FPGA's reprogrammability is great for creating prototypes and for future changes in the application.

The performance of FPGAs can in some cases be compared with that of GPUs. Though GPUs can execute thousands of (usually floating point) operations in parallel, data processed should be uniformed, with the same operation run multiple times. Some algorithms may not be possible to be easily parallelized and that can give an edge to the FPGAs reconfigurable concept. Recently, attempts have been made to translate and port ML frameworks written in higher level languages for use in FPGAs. Many ML applications that require a certain environment different from the static office or server room can benefit from such implementations. Self-driving cars, robotics and smart city meters use neural networks nowadays to compute their tasks and compact, low-powered FPGAs can be used for their services.

Domain	Algorithm	Winner	Factor
Physics Simulation	HotSpot	GPU	0.59
Fluid Dynamics	Flux	GPU	0.35
Data Mining	KNN	GPU	0.32
Image Processing	SRAD	FPGA	4.52
Medical Imaging	GICOV	FPGA	7.76
Bioinformatics	HW	FPGA	19.79

Figure 11: FPGA and GPU comparison in selected algorithms by Cong et al. [20]

5.2 System-on-a-Chip (SoC)

A system-on-a-chip is an integrated circuit with all the necessary electronic elements and parts for a computer system, such as a smartphone or a tablet, combined into a single chip. A SoC always includes a CPU, peripheral controllers, memory interfaces and the most modern of them may include also GPUs, Digital Signal Processors and dedicated Neural Network hardware. The reduced space, power efficiency and lower manufacturing cost have made SoCs an un-changeable element of modern mobile and embedded systems. SoCs are designed to minimize latency and interconnection delays with boosted communication throughput between the parts. Their performance along with their durability have made them suitable for hardware accelerated applications. The embedded system used for this thesis has a SoC that combines its Processing System (CPU) with Programmable Logic (FPGA) and is presented next.

5.3 Zedboard Development Kit

The board we worked on uses the Xilinx Zynq®-7000 All Programmable SoC along with all the necessary ports and interfaces for embedded applications development. More specifically it mainly contains a Dual-core ARM Cortex®-A9 CPU, 28nm programmable logic, 512 Mb of DDR3 Memory, SD card reader, Ethernet USB and HDMI ports, 128x32 OLED screen, 8 User Switches and 7 user push buttons and 9 LEDs. Its detailed view is shown below.

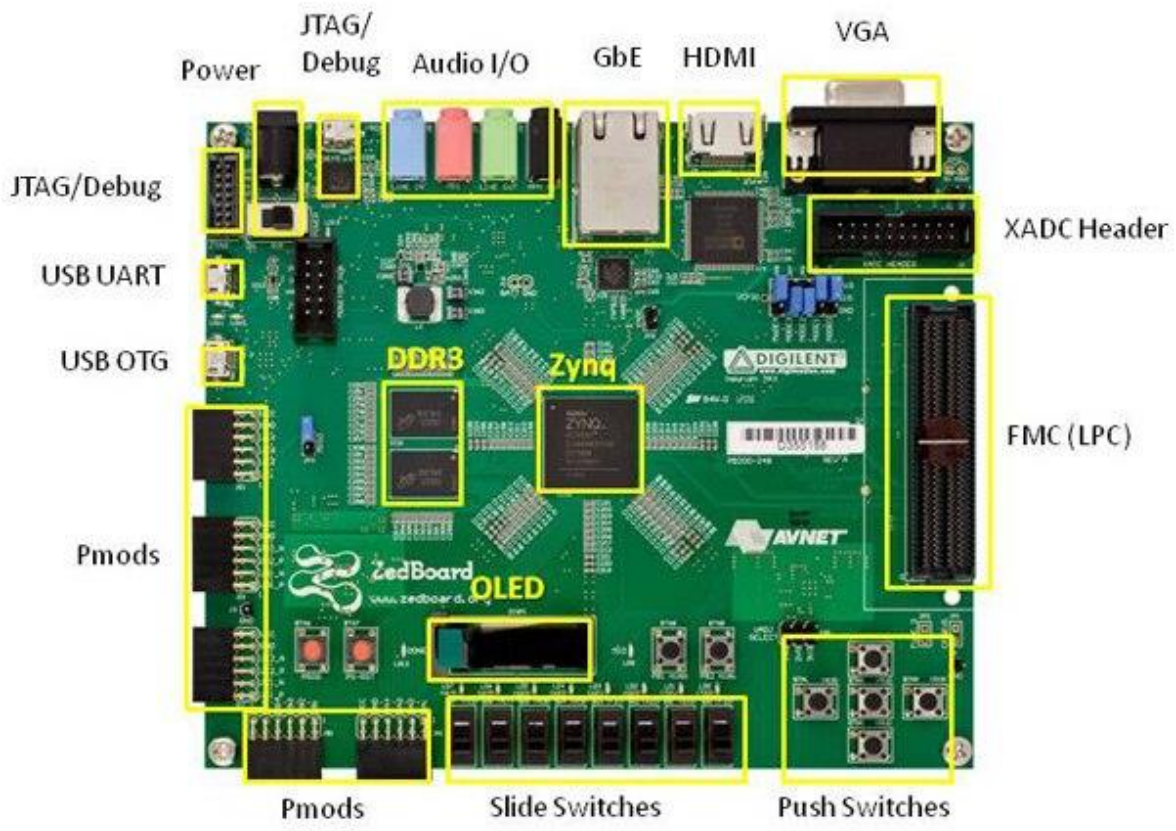

Figure 12: Overlay view of the Zedboard Development Board. [21]

5.3 Embedded Software Implementation

Trying to implement our solution on the embedded device, the first step is to build the system that will host our executables and binaries. PetaLinux [22] is an embedded Linux distribution for Xilinx Zynq chips which is supported by its own tools for building and deploying custom

embedded Linux solutions. By configuring these tools we were able to create a Linux filesystem with OpenCV libraries installed. Using commands from the Vitis Unified Software Platform [23] it was achieved to compile, build and package the host and kernel executables along with the device binary and other required files. Our system runs on the ARM processors through a SD card. For this to happen, we flashed the SD card with our data and two partitions were created: the root filesystem partition that contains the Petalinux filesystem and the BOOT partition where the bootloader and the application executables (host executable and kernel bitstream) are placed.

The first attempt, which is used as a baseline for our experiments, is to run our algorithm only on the Processing System. The results were far worse from what the laptop achieved, but this was to be expected for this low frequency processor. The board could process 2-3 FPS depending on the video demands every time. Images were read from a folder in a sequence and not from a video file, but that may had low impact on the overall performance. Timing distribution of the functions where similar to that of the x86 processor, described in chapter 4. This result is not close to a real-time solution and as a final step, optimization using the hardware resources (FPGA) must be done.

5.4 Hardware Optimization Attempt

Knowing that more than half of our algorithm time is spent during the frame pre-processing, we are going to analyze techniques that can decrease this number, when those functions are run through a hardware kernel. In recent years, Xilinx has developed the Vitis Vision Library [24]. It is a set of 90+ kernels based on the OpenCV library written in C++, optimized for Xilinx SoCs. Fortunately enough, all the functions that form our pre-processing step are included in this library. We used the Vitis HLS tool to put together and edit these functions in order to create and preview our kernel code. Optimization directives (pragmas) to modify and control the implementation of the internal logic and I/O ports are carefully placed inside the code.

5.4.1 Pipelining (pragma HLS pipeline)

By using this directive, we allow concurrent execution of operations. This way the initiation interval (II) of a function or a loop is reduced. For example, if there is no data dependency, functions can execute concurrently in an out of order execution way, with the first function of a second loop iteration executing before the first iteration finishes its last function, as shown in the next figure.

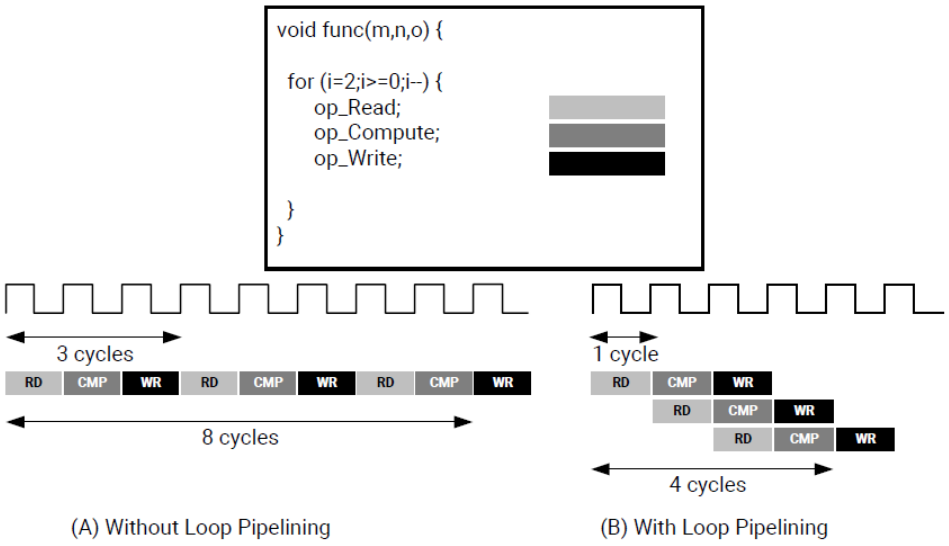

Figure 13: Loop Pipelining. [25]

5.4.2 Dataflow Optimization (pragma HLS dataflow)

This technique also known as task level parallelism, creates an architecture of concurrent processes for a series of sequential tasks. The compiler is allowed to schedule multiple functions inside the kernel to run concurrently achieving this way higher throughput and lower latency. The following figure shows an example of what can be achieved with this optimization. Function func_A is scheduled to execute as soon as data are available and not when func_C is completed.

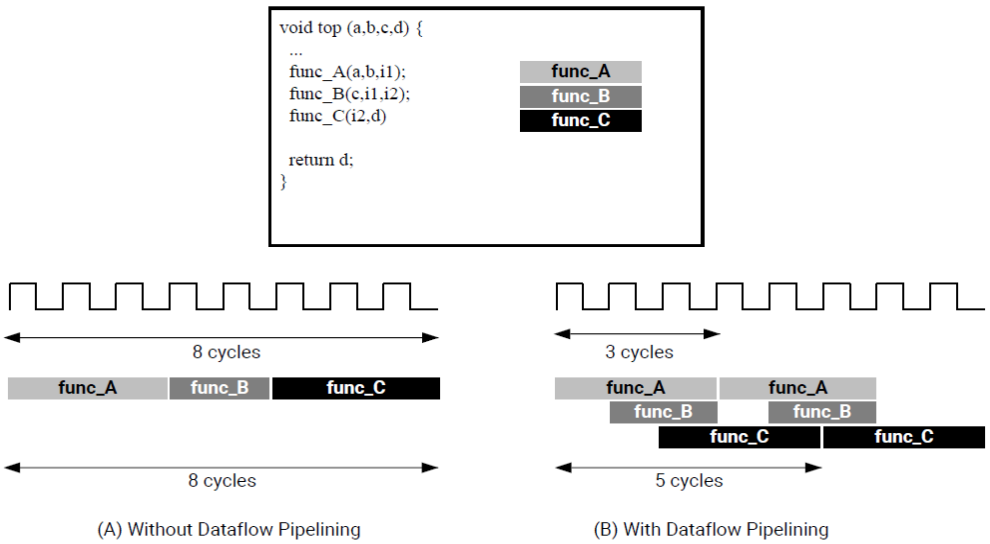

Figure 14: Dataflow Optimization. [25]

5.4.3 Synthesis Review

With the optimizations described applied to our functions, the HLS synthesis results are presented below. The top kernel function managed to reduce LUT utilization by more than 25% compared to the original attempt and latency is down by 20%. In addition, the initiation interval reduced to 12 cycles for the whole process. Unfortunately, this implementation uses a streaming interface where data are fed into our function as matrix pointers of arbitrary precision type (ap_uint) and the accelerated functions that convert them to accelerated OpenCV matrices (xf::Mat) must be used while being time consuming.

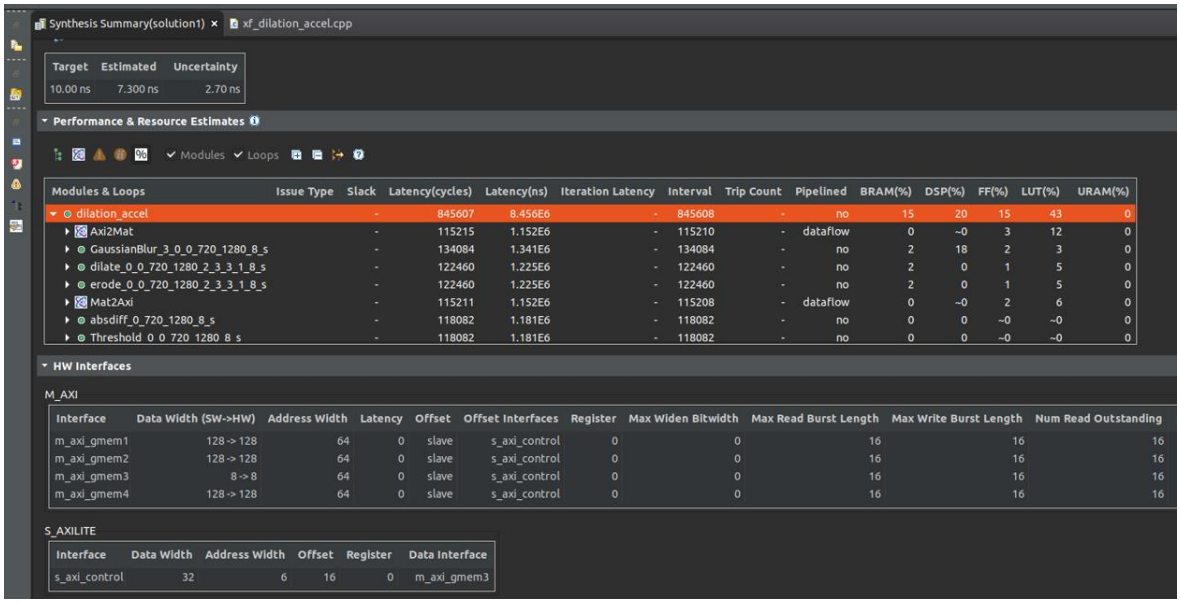

Figure 15: Synthesis Report of Vitis HLS GUI.

Analyzing the above metics, given a clock of 100MHz the Zedboard PL could run a loop cycle of pre-processing functions in less than 10ms compared to the ARM CPU that takes more than 100ms to complete the same process, a 10x speedup. OpenCL API can then be used by both the host application and the accelerated kernel in order for their interactions to happen efficiently.

Chapter 6

Conclusion and Future Work

The work presented can efficiently predict the make a model of a car appearing in a video frame in real-time, when running on a modern x86 CPU. The process of selecting a database, a feature extractor algorithm and a classification network can be used in many computer vision applications. The choice of SVM as classifier proved to be accurate enough even for our real-time approach where the position and size of the presented car changes every second. More complex networks like CNNs were not examined, due to their usually larger memory requirements that appeared as not suitable to be efficiently used in a less powerful embedded system. This network may be able to distinguish at a high-rate car models from different makes but it is unable to note the difference between intra-class models, due to our small dataset. A great upgrade could be to enrich this dataset with more models and more photos of every model and then re-train the SVM classifier. Our embedded system approach looks promising not only from the HLS report but also from the usage of recently developed accelerated OpenCV functions. Vitis Design Flow is demanding and requires more work to be done for this system to be ready to run in real hardware, which is the next step for this implementation. Finally, the described algorithm could be accelerated in GPUs with the help of the OpenCL API.

Bibliography

[1] Lee, H. J., Ullah, I., Wan, W., Gao, Y., & Fang, Z. (2019). Real-Time Vehicle Make and Model Recognition with the Residual SqueezeNet Architecture. Sensors (Basel, Switzerland), 19(5), 982. https://doi.org/10.3390/s19050982

[2] Psyllos, A., Anagnostopoulos, C. N., & Kayafas, E. (2011). Vehicle model recognition from frontal view image measurements. Computer Standards & Interfaces, 33(2), 142–151. https://doi.org/10.1016/j.csi.2010.06.005

[3] Baran, R., Glowacz, A., & Matiolanski, A. (2013). The efficient real- and non-real-time make and model recognition of Cars. Multimedia Tools and Applications, 74(12), 4269–4288. https://doi.org/10.1007/s11042-013-1545-2

[4] Chen, L.-C., Hsieh, J.-W., Yan, Y., & Chen, D.-Y. (2015). Vehicle make and model recognition using sparse representation and symmetrical SURFs. Pattern Recognition, 48(6), 1979–1998. https://doi.org/10.1016/j.patcog.2014.12.018

[5] Manzoor, M. A., Morgan, Y., & Bais, A. (2019). Real-Time Vehicle Make and Model Recognition System. Machine Learning and Knowledge Extraction, 1(2), 611–629. https://doi.org/10.3390/make1020036

[6] Pearce, G., & Pears, N. (2011). Automatic make and model recognition from frontal images of cars. 2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 373–378. doi:10.1109/AVSS.2011.6027353

[7] Jamil, A. A., Hussain, F., Yousaf, M. H., Butt, A. M., & Velastin, S. A. (2020). Vehicle Make and Model Recognition using Bag of Expressions. Sensors (Basel, Switzerland), 20(4), 1033. https://doi.org/10.3390/s20041033

[8] Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 1, 886–893. doi:10.1109/CVPR.2005.177

[9] Histogram of oriented gradients explained using OpenCV. LearnOpenCV. Available online: https://learnopencv.com/histogram-of-oriented-gradients/

[10] Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. Machine Learning, 20(3), 273–297. https://doi.org/10.1007/bf00994018

[11] Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2, 27:1-27:27.

[12] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

[13] Kanan, C., & Cottrell, G. W. (2012). Color-to-grayscale: Does the method matter in image recognition? PLoS ONE, 7(1). https://doi.org/10.1371/journal.pone.0029740

[14] Gaussian Blur. Wikipedia. Available online: https://en.wikipedia.org/wiki/Gaussian_blur

[15] Vacavant, A., Chateau, T., Wilhelm, A., Lequièvre, L. (2013). A Benchmark Dataset for Outdoor Foreground/Background Extraction. In: Park, JI., Kim, J. (eds) Computer Vision - ACCV 2012 Workshops. ACCV 2012. Lecture Notes in Computer Science, vol 7728. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37410-4_25

[16] Zivkovic, Z., & van der Heijden, F. (2006). Efficient adaptive density estimation per image pixel for the task of background subtraction. Pattern Recognition Letters, 27(7), 773–780. https://doi.org/10.1016/j.patrec.2005.11.005

[17] Contour detection using opencv (python/C++). LearnOpenCV. Available online: from https://learnopencv.com/contour-detection-using-opencv-python-c/

[18] Suzuki, S., & be, K. A. (1985). Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing, 30(1), 32–46. https://doi.org/10.1016/0734-189x(85)90016-7

[19] Intel's Stratix 10 FPGA: Supporting the smart and connected revolution. Intel Newsroom. Available online: https://newsroom.intel.com/editorials/intels-stratix-10-fpga-supporting-smart-connected-revolution/

[20] Cong, J., Fang, Z., Lo, M., Wang, H., Xu, J., & Zhang, S. (2018). Understanding Performance Differences of FPGAs and GPUs. 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 93–96. doi:10.1109/FCCM.2018.00023

[21] Zedboard. Avnet. Available online: https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/

[22] PetaLinux Tools Documentation Reference Guide (UG1144). Available online: https://docs.xilinx.com/r/2020.2-English/ug1144-petalinux-tools-reference-guide/Revision-History

[23] Vitis Unified Software Platform Documentation: Embedded Software Development (UG1400). Available online: https://docs.xilinx.com/r/en-US/ug1400-vitis-embedded/Installing-Embedded-Platforms

[24] Vitis Vision Library User Guide. Available online: https://xilinx.github.io/Vitis_Libraries/vision/2020.2/index.html

[25] Vitis High-Level Synthesis User Guide (UG1399). Available online: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls