VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

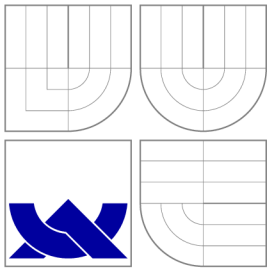
OBJECT DETECTION ALGORITHMS ON ANDROID PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

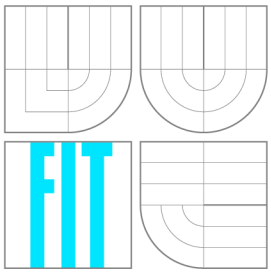
AUTOR PRÁCE
AUTHOR

VOJTĚCH DLÁPAL

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ALGORITMY DETEKCE OBJEKTŮ NA PLATFORMĚ ANDROID

OBJECT DETECTION ALGORITHMS ON ANDROID PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VOJTĚCH DLÁPAL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR MUSIL

BRNO 2014

Abstrakt

Cílem této práce je prozkoumat možnosti detekce objektů na platformě Android, navrhnout a implementovat demonstrativní aplikaci, otestovat ji a zhodnotit dosažené výsledky. Je představena platforma Android, knihovna pro počítačové vidění OpenCV a teorie pro detekci objektů. Byla navržena a implementována aplikace porovnávající detekci obličejů z OpenCV, Android API a navrženého detektoru používající klasifikátor. Aplikace byla důsledně otestována a výsledky vyhodnoceny.

Abstract

Aim of this thesis is to analyze possibilities of object detection on Android platform, design demonstrative application, test it and evaluate results. Android platform, computer vision library OpenCV and object detection theory are being introduced. Application for comparison of face detection from OpenCV, Android API and custom detector using classifier was designed and implemented. Application was tested and the results were evaluated.

Klíčová slova

Android, OpenCV, NDK, detekce objektů, WaldBoost, LBP

Keywords

Android, OpenCV, NDK, object detection, WaldBoost, LBP

Citace

Vojtěch Dlápal: Object Detection Algorithms on Android Platform, bakalářská práce, Brno, FIT VUT v Brně, 2014

Object Detection Algorithms on Android Platform

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Musila

.....
Vojtěch Dlápal
July 30, 2014

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Petru Musilovi za poskytnuté přínosné konzultace a za trpělivost při mém pobytu v zahraničí.

Contents

Chapter 1

Introduction

We live in a modern age when everyday life is closely connected to technology. From recent times people can put a computer into their pockets and take it to wherever they like to. This computer is represented by mobile devices such as modern mobile phones and tablets.

These devices are endowed with multiple sensors, such as camera, internet connection and operating system which offers an opportunity to install new applications. These applications could be more and more complex and fulfilling undreamed-of tasks. This is becouse of rapid technical development in the segment.

One of the fields with great potential for mobile devices is computer vision. This thesis is focused on subdiscipline of computer vision which is prerequisite for some another more advance subdisciplines. It is object detection, more precisely object detection on the most successful mobile platform which is Android.

In the first chapter is introduced Android platform from different points of view. As a platform what became the most successful, as a modern operating systems for not only mobile devices and last but not the least, as a platform suitable to develop new application for. The reader will get insight into specifics of application development for Android using Software development kit and Native development kit.

Chapter 3 is dedicated to object detection. In the first section is introduced library OpenCV which provides tools for computer vision and thus also for object detection. Further is discussed core of object detection and multiple approaches to it are listed. Last part of the chapter goes deeper into object detection using classifiers. Multiple terms such as weak classifier and strong classifier are introduced. Lastly are listed and explained some of the most popular features.

Chapter 4 analyses exploitation of object detection in applications available on the market and available tools for development of this kind of application.

Chapter 5 explains design of demonstrating application and use of selected classifier and chapter 6 goes into detail of implementation and used algorithm.

Testing of developed application is discussed in chapter 7. At first are introduced selected test cases and finally results are evaluated and future work is proposed.

Chapter 2

Android platform

This chapter aims to introduce the reader into the world of Android platform. At first is briefly mentioned modern history of mobile devices and then is introduced term „smartphone“. Later is introduced history of Android platform and importance of it as a market leader is supported by statistical data.

Further is Android introduced from developer's perspective. Android is looked upon as a system and main building blocks are mentioned. Further is explained what Android applications consist of and how to develop them. Last part of the chapter is dedicated to advance programing for Android using Native development kit.

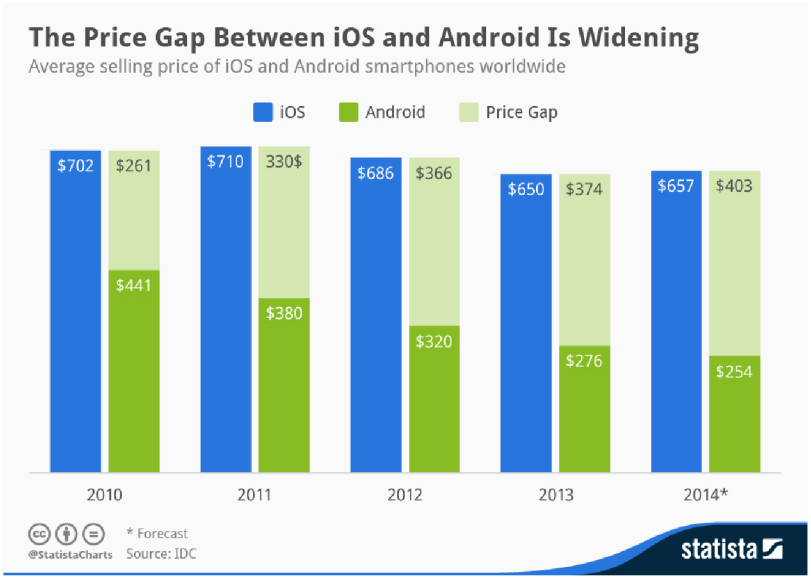


Figure 2.1: The difference in average price of iOS and Android devices. Taken from [27]

2.1 The smartphone era

Nowadays the most of people are having mobile phones. First phones were meant only for communicating through calls. Those phones were big, heavy and expensive. These days are different and people are familiar with word „smartphone“ [14]. Smartphones are providing to its users much more than just calling. The possibilities are nearly unlimited, as their

features could be extended with applications. These could combine device's hardware, such as different sensors, with internet connection into extraordinary user experience.

Beginning of latest generation of smartphones could be related to revelation of first mobile phone by Apple Inc. [16, 18] in 2007. Name of the product was iPhone, and it was huge success. It introduced the concept of controlling user interface using touchscreen to the masses. There was no hardware keyboard, just virtual one. Success of iPhone also meant the fact, that people became more aware of possibilities of phones with operating system. iPhone came with operating system called iOS [17]which comes with centralized place for getting applications called App Store.

2.2 Android progress

Operating system Android was released in 2008 [15] . It exploited some of ideas from iOS and catched up with it's popularity. Android is opensource and therefore it is possible for any manufacturer to use it. This resolved into that on the market appeared devices with modern operating system which were much cheaper then iPhone, see 2.1, and began to spread. In 2011 Android begun to have the largest installation base over mobile operating systems [15] and in 2013 sales of devices with Android were bigger than all the other platforms together, see figure 2.2. At the moment the market share of Android is about 78.4% [11]what makes it leader of mobile operating systems. Every day are activated about 1.5 millions of devices with Android [2]. Furthermore, with growing popularity of tablets are android devices slightly becoming substitutes for traditional PCs.

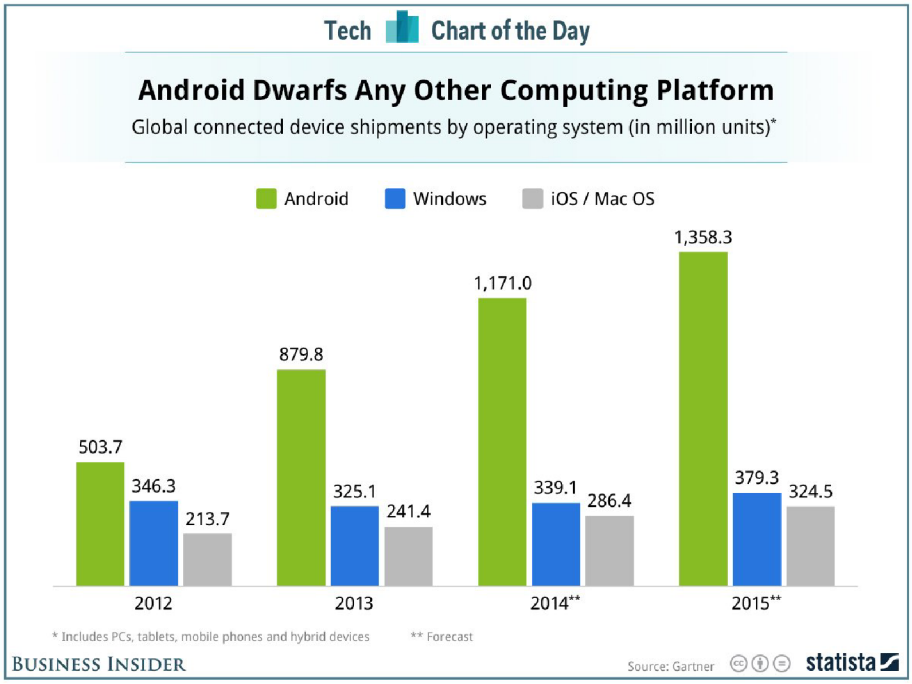


Figure 2.2: Worldwide shipments of devices by operating system. Taken from [34]. Data from [28]

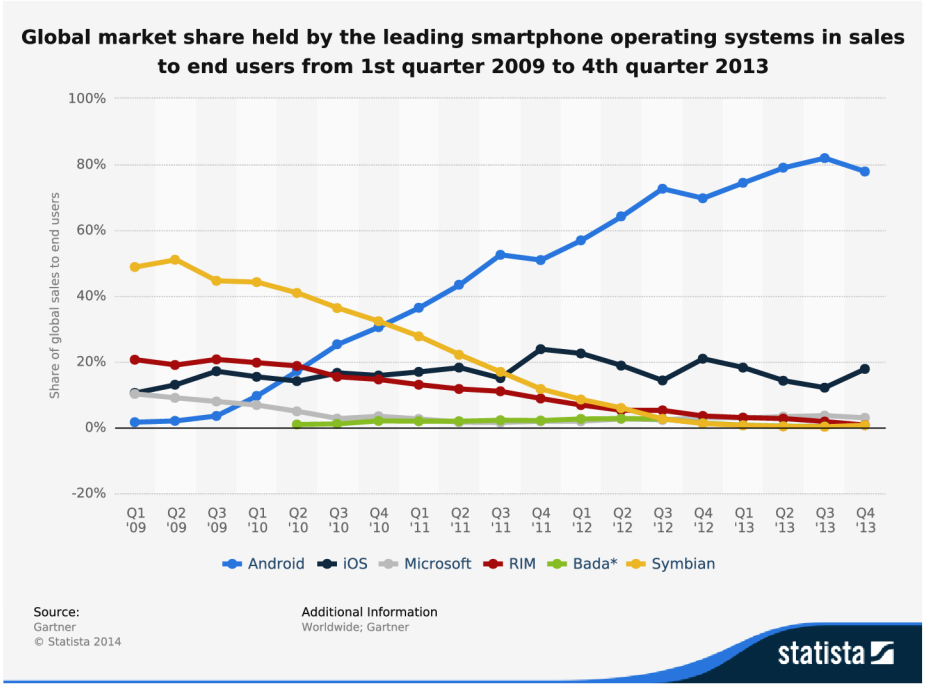


Figure 2.3: Global market share of smartphone operating systems. Taken from [11]

.

2.3 Programming for Android

Android is comprehensive open source platform designed for mobile devices [22]. Android appears in watches, phones, tablets, TVs, and cars [1]. It is developed and maintained by Open handset alliance[1]. The latest version while writing this thesis is KitKat 4.4. Even though it is not the most common version what is 4.1 Jelly Bean as is stated on the graph 2.4.

Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	13.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	11.4%
4.1.x	Jelly Bean	16	27.8%
4.2.x		17	19.7%
4.3		18	9.0%
4.4	KitKat	19	17.9%

Data collected during a 7-day period ending on July 7, 2014.
Any versions with less than 0.1% distribution are not shown.

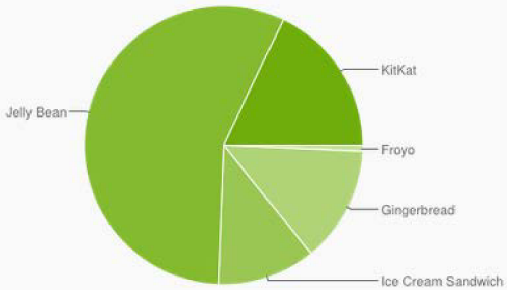


Figure 2.4: Distribution of different versions of Android. Taken from [13]

[1] www.openhandsetalliance.com

2.3.1 Android as a platform

Android is build around Linux kernel. It does not contain some parts of standard linux distributions like is X Window System. The presence of linux is hidden from the user but even from developers. Aplications compiled as a native code for linux based operating system can not be run on Android. Presence of Linux could be e.g. from filesystem structure. All the stack of software, what Android consist of, is illustrated on figure 2.5. Android contains bunch of libraries, what can developers rely on, as is SQLite database or WebKit rendering engine.

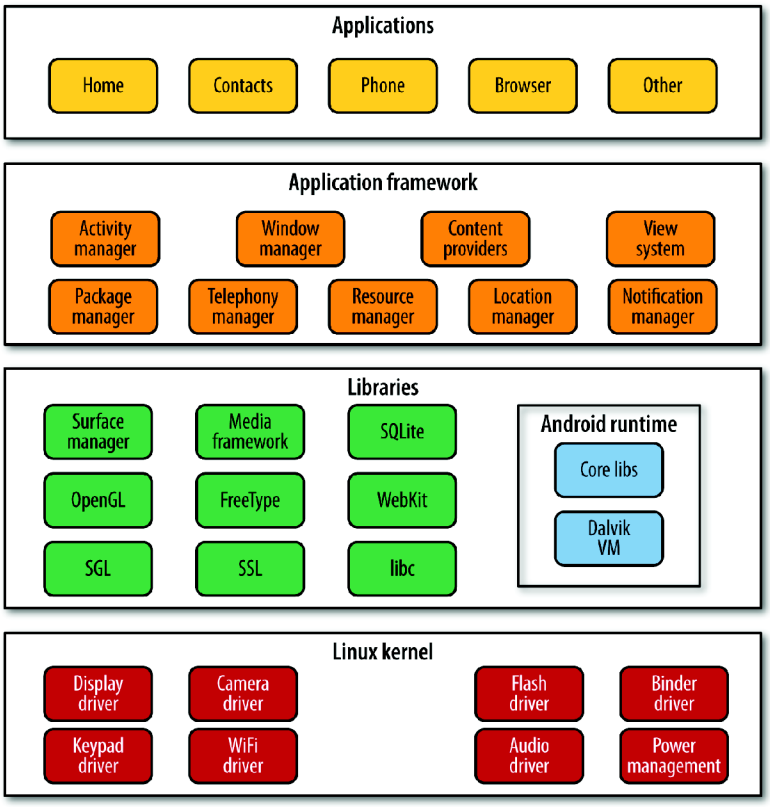


Figure 2.5: Illustration of all the software stack what Android platform consist of. Taken from [22]

Developer who wants to develop Android applications has to use Android software development kit (SDK). It provides API libraries and developer tools necessary to build, test, and debug applications for Android [5].

The standard integrated development environment for programming for Android is Eclipse. This offers plugin called Android Developer Tools which helps by providing graphical interface for creating layouts, debugging, project creation and much more [3].

Programming language for writing Android applications is Java. It is possible to use most of standard libraries included in Java SE (Standard edition) excluding AWT and Swing which are graphic user interface libraries which would have no use in Android since it has its own approach for graphic user interface. Although Java as a language is open source and free to use, Java virtual machine is not and Android does not use it. It uses

virtual machine called Dalvik. The compilation process consists of compiling Java code with Java compiler into Java bytecode and then with Dex compiler into Dalvik bytecode to be run on Dalvik virtual machine. There is a plan to replace Davik with virtual machine called ART [6] in future versions of Android.

2.3.2 Android application

Android application consists of couple of building blocks. These are activities, intents, services, content providers and broadcast receivers. An activity is a class which usually binds to one single screen of the application and is controlling its content and handling user interface. Every application have to have at least one activity which is main activity. Activities go through different states throughout applications life cycle as could be seen on diagram 2.6.

To create a new activity has to be created class which inherits from class Activity and implement its methods. Most importantly method `onCreate()` which is entry point for an activity, and thus for whole application in terms of main activity.

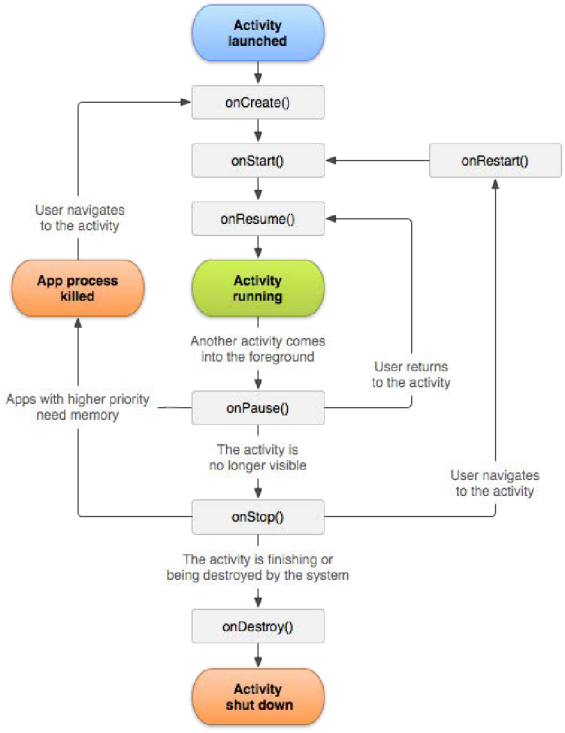


Figure 2.6: Lifecycle of Activity. Taken from [19]

Intents are tool for communicating between activities and therefore also for switching active activity. Services serve for tasks with no user inputs, the rest of building blocks is for work with data. User interface is hold in layouts, which are xml files with description and placement of control elements. Figure number A.1 shows how android project tree looks like in Eclipse integrated development environment.

Resources, such as images or strings, are stored in res folder. Those can not be accessed directly, but during the compilation is to every single resource being assigned id. Those id's are available in auto-generated R file and thus could be accessed only through this. This

approach is suitable e.g. for multilingual applications so string in different languages could be stored in `strings.xml` file and from the code accessed all the same way.

Big issue for Android developers are different densities of displays of all possible Android devices. Because of this, there are multiple folders for images with different densities in the res folder. So, when is created layout, some of drawable components have to be brought with application in different resolutions.

Every application has to contain manifest file which holds important information about the application such as declarations for all activities, permissions (e.g. permission to work with camera, to access data storage or use data connections), minimal SDK version for application to be able to build (newer version of target SDK will provide more functions, but will cause unavailability for older devices) and also stuff like application's icon and title.

Ready application comes in Android Application Package (APK). This is a zip file which consists of Android Manifest file, Dalvik executable, resources (parts of application which is not code eq. images), native libraries and signatures. Applications are mainly distributed to the user by so called „markets". The biggest market is Google Play Store run by Google.

2.4 Native development kit (NDK)

Sometimes, there is a need for using native code in Android application. This could be due to memory management or performance constraints in Java or a need of reusing of some existing code or library [12].

„Native development kit is a toolset which allows implementing parts of application using native code languages such as C and C++. It includes a set of cross-toolchains (compilers, linkers...) that can generate native ARM binaries, set of system headers for stable native APIs and build system." (Taken from official reference [4])

Official reference recommends using NDK only in specific cases such as game engines, signalling processing or physics simulation.

Parts of application written in managed code (Java part) and native code (C/C++ part) could interact through JNI what states for Java Native Interface [8].

By programming through the JNI is possible to use native methods to create, inspect, and update Java objects (including arrays and strings), call Java methods, catch and throw exceptions, load classes and obtain class information and perform runtime checking [7].

Chapter 3

Object detection and OpenCV

First part of this chapter will introduce library for computer vision OpenCV. It will give to the reader information about its history, use cases and future. Further will be discussed object detection. Detection will be described as a field of study and multiple approaches to it will be listed.

Main focus gets object detection using classifier. In part of the chapter dedicated to it will be explained what is classifier and which algorithms are used to train it. Namely AdaBoost and WaldBoost. In the next section will be commented on concept from Viola and Jones and the difference between weak and strong classifier will be explained. Last section is dedicated to different popular features. These will be explained and illustrated.

3.1 OpenCV

Computer vision is transformation of data from 2D/3D stills or videos into either a decision or a new representation [20]. When computer gets an image, it is just a grid of numbers. We need computer vision to represent it somehow or to get some information from it.

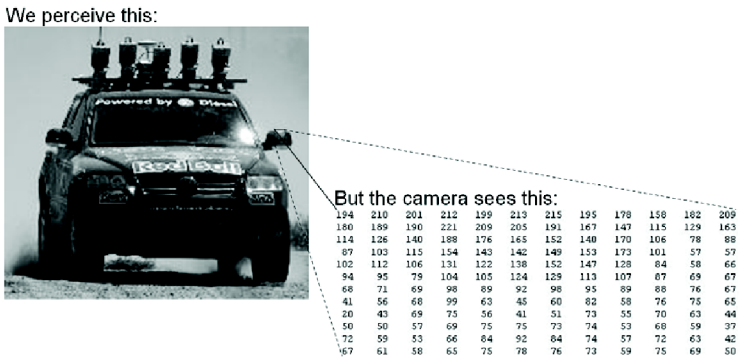


Figure 3.1: Images are represented as a grid of numbers. Taken from [21]

Today's use of computer vision is very wide. It could solve tasks from surveillance till quality checking's in mass manufacturing production. Author of [21] also highlights image-snitching for aerial and „street-map“ applications such as Google Street View.

OpenCV is an open source library for computer vision written in C and C++ having interface for various languages and running under multiple platforms including Android. It is highly optimized for computational efficiency. The library functions cover many areas but main usage is to work with images from camera and machine learning. Brief list of what OpenCV can do:

„Basic image processing (filtering, morphology, geometrical transformations, histograms, color space transformations), advanced image processing (like inpainting, watershed & meanshift segmentation etc.), contour processing and computational geometry, various feature detectors and descriptors (ranging from simple Harris detector to Hough transform, SURF, MSER etc.), object tracking, optical flow, object detection using cascades of boosted haar classifiers, camera calibration, machine learning tools (data clustering and statistical classifiers).“ Taken from [9].

Alpha release of the library was introduced in January 1999. At the beginning it was supported and developed by Intel. Now is OpenCV supported by OpenCV.org foundation and primary maintained by Itseez[1]. In 2013 library reached 6,000,000 downloads [9]. Since OpenCV is under BSD licence, everybody can use it even for commercial purposes without any obligations. Even though there are still people from commercial and also academic sector contributing to development.

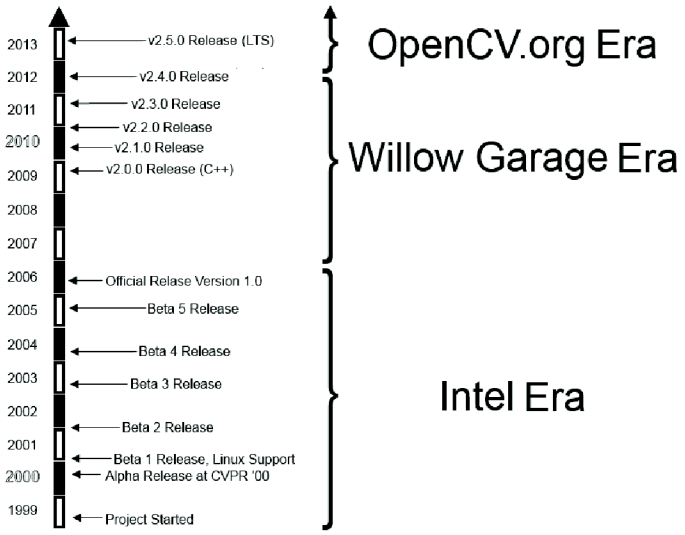


Figure 3.2: History of OpenCV. Taken from [21]

The biggest step forward was introducing version 2.0 which came with C++ interface. This resolved into much more comfortable way of using it for programmers. Good example is new structure for holding images `cv::Mat`. When using Mat, there is no need for manual allocation of memory, Mat is handling it by itself.

Nowadays all modern phones comes with camera. It is usual to use them for scanning QR codes, what is in fact a computer vision task. Optimization of OpenCV for mobile devices is one of the subjects of current development. Another way of optimization is exploiting graphic cards and parallelism for speeding up computations [26].

[1] www.itseez.com[9]

3.2 Detection

With everyday's innovation in computer science and computer hardware are opening new ways how to use technology. One of this ways is about how do people interact with computers and which tasks could be automated.

Detection means to decide whether there is a specific object present in the image or not. Detection of objects is fundamental task for some more advance tasks as is object tracking or object recognition, but it could be also goal by itself as for example in surveillance or quality control. Over the time there were found some approaches how to grasp detection task.

There are several ways how to detect an object. While choosing the right one there is a need to consider multiple factors such as rotations in the space, presence or absence of structural components (structural components may vary e.g. different shapes of lights on car, glasses on the face), occlusions (Objects could be occlude by another objects), image orientation and imaging conditions (like lighting).

Different methods have also different false positive and false negative detections. False positive detection is when the system decides that area is containing desired objects, but in fact it is not. False negative is when object is present on the picture, but the system does not mark it as a successful detection.

Existing methods could be sorted out into categories mentioned in subsections.

3.2.1 Knowledge based methods

This approach is based on specific knowledge about objects, which could be objectively stated by humans. According to this knowledge, there are some rules which apply for all desired objects. Therefore this approach is Top-Down oriented. In case of detecting faces, there is knowledge that there are eyes, nose, mouth... on human face, and these rules may refer to positioning of eyes according to each other and position of nose according to eyes etc. But the problem is to transform human knowledge into computer readable rules, which are appropriately general and thus do not resolve into too many false detections. See [33].

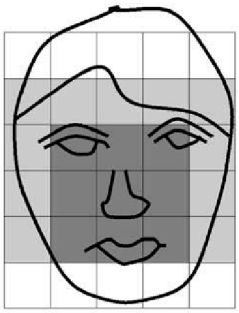


Figure 3.3: Illustration of face as seen by knowledge based methods. Rules are based of knowledge about facial regions. Taken from [33]

3.2.2 Feature based methods

This approach focus on finding invariant features of objects. Invariant means that these features remain the same, no matter the conditions such as lighting. This approach is Bottom-Up oriented. The feature could be e.g. shape of object segmented with edge detector, some specific texture of the object or other feature as could be skin colour for human face. See [33].

3.2.3 Template matching based methods

Template matching is based on having specific pattern for an object or parts of it, which is compared to an image and correlation is computed. This pattern is prepared in advance. The way how to do that could be e.g. comparing lines extracted from the image by following gradient changes with the template or also by comparing features extracted with using edge detector. Unfortunately this approach has issues with variations of shape, scale and pose. More in [33].

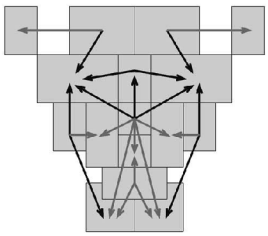


Figure 3.4: Typical template for face localization. Taken from [33]

3.2.4 Appearance–Based methods

In appearance-based methods, there are some typical characteristics learned from the image with desired object using statistical analysis and machine learning. This means that we need set of images for training. Nowadays the most of methods for objects detection fits into this category. See [33].

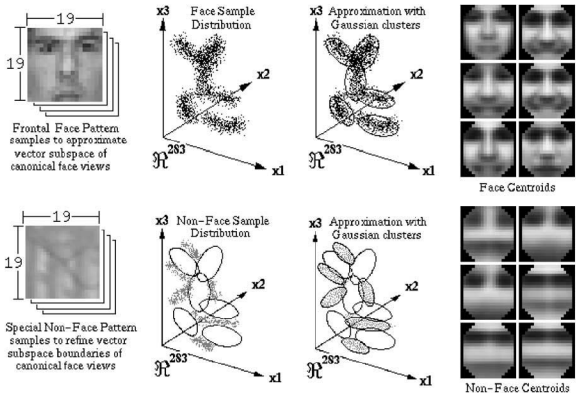


Figure 3.5: One of appearance-based methods. In this method are estimated density functions fo clusters of faces and non-faces. Taken from [33]

3.3 Object detection using classifiers

Classifier is a function which decides whether input belongs to the object or background class. Classifiers are made using machine learning algorithm. To the algorithm is given set of images with desired object and those are processed to make the best possible decision making function. This process is called learning or training.

There are many possible ways how to train a classifier. For instance artificial neural networks, support vector machines, decision trees and so on. In section 3.3.1 will be further discussed boosting.

3.3.1 Adaptive boosting

Sometimes it is possible to find classifier which has ability to divide data into two desired classes. For object detection, these classes could be called object (found) and background (not found). If error rate of this classifier is lower than 50%, then it means that it could be useful. Those classifiers are called weak classifiers. But this high error rate is not acceptable for any serious usage. So this is why came up the idea to combine these weak classifier together and therefore make one with acceptable error rate. This kind of classifier is called strong classifier and the process of making it is called boosting.

Boosting has its roots connected with machine learning algorithm PAC what stands for probably approximately correct. The learning algorithm in PAC was only random guessing, therefore it was weak. Then the question, whether this approach can be boosted to make it stronger, popped up and this is how whole idea of boosting came up.

The AdaBoost algorithm was introduced in 1995 by Freund and Schapire. The idea is to pick in each iteration one week classifier and add it to the strong classifier, so the error will lower. Classifier is trained on annotated set of images. Those are having weights, initialized to the same value, but over the time, images with higher error rate are getting bigger weight. AdaBoost is abbreviation of Adaptive Boosting. This is because the algorithm can adapt to the error rates of the individual weak hypotheses. More in [29].

3.3.2 Waldboost

The Waldboost algorithm introduced in [30] is based on AdaBoost (more specifically on real AdaBoost which is providing real numbers) enhanced by using Wald's optimal sequential probability ratio test. This is used for putting weak classifiers picked by AdaBoost into right sequence and to set thresholds. Both negative (under this threshold is investigated area not in class „object“ for sure) and positive (above this threshold is investigated area in class „object“ for sure). Thresholds are set in each iteration and these are computed using values False negative ratio and False positives, which are set beforehand. Because of these thresholds is final classifier able to eliminate great amount of frames in early stage and therefore is fast.

3.4 Weak classifiers

There are many possible features which could describe an image. Those differ mainly in discriminative power and computational power demands. The real breakthrough for object, respectively face, detection was work of Viola and Jones [31] from 2001. This meant beginning of rapid object detection era, where is possible to detect objects real-time.

In there were used Haar features in combination with "integral image" for speeding up computing process, AdaBoost algorithm for feature selection and "cascade approach" for combining classifiers and thus skipping unpromising regions what brought another speed up.

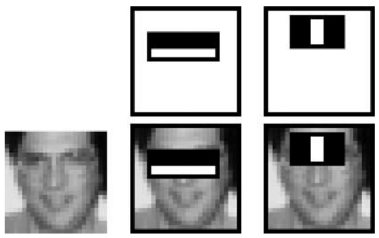


Figure 3.6: First and second Haar features selected by AdaBoost in Viola and Jones detector. The first feature exploits difference of intensity between region of eyes and region of upper cheek. Taken from [32]

3.4.1 Haar features

Haar features got its name from Haar wavelet. It is computed as difference of two adjacent rectangular areas, which are illustrated as black and white area. More theoretically this is convolution of image with Haar wavelet [24]. Those are strong in detecting lines and edges.

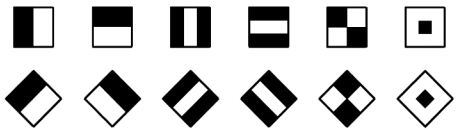


Figure 3.7: Standard and extended set of Haar features. Taken from [23]

Haar features are computationally efficient when computed on integral image. Integral image, is image where each pixel is sum of all pixel in rectangle to the left and top of it. This means that it is possible to get sum of whole area just by accessing one pixel.

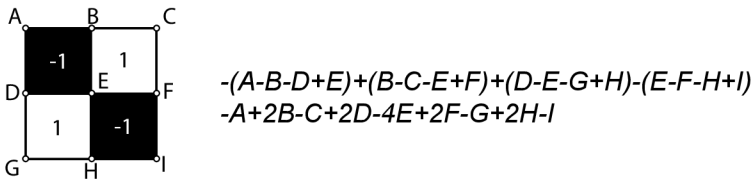


Figure 3.8: Calculation of Haar feature response. Values in corners are known from integral image and response is computed according to stated formulas. Note that there are only simple operations. Taken from [24]

Therefore Haar features are computed in constant time, but there is time needed to pre-process an image to get integral image and also for normalising it, as Haar features are

not invariant to lighting conditions [23, 24].

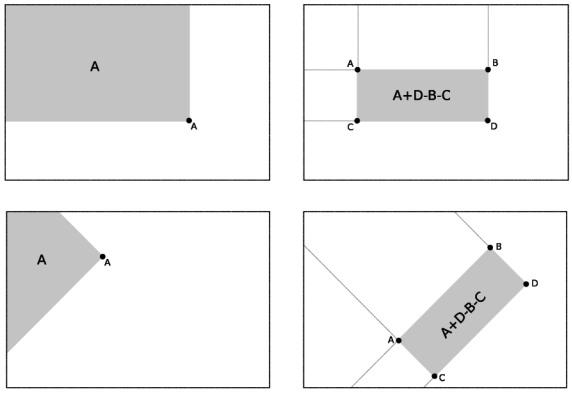


Figure 3.9: Computation of standard and extended integral image. Point A holds sum of whole area A. Taken from [23]

3.4.2 Local binary patterns

Local binary patterns were introduced in [25]. These are very simple, yet efficient. LBPs are computed from small matrices of pixels on grayscale images. The main idea is to threshold surrounding pixels with one in the middle. If the intensity of pixel is higher than intensity of one in the middle, then value for the position is 1, otherwise the value is 0. Each position has some weight. The result is computed by putting weighted values together, what makes a final result of LBP.

In [35] was proposed to use multi block LBPs. Those appeared to be more discriminative than original LBP or Haar features.

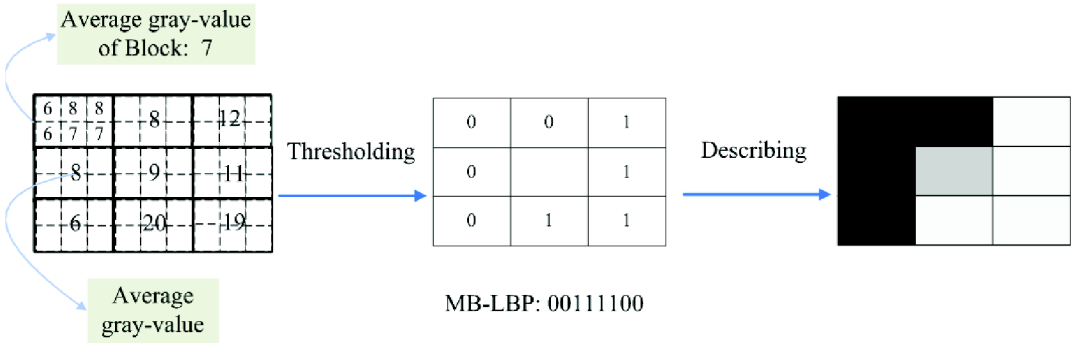


Figure 3.10: Evaluation of Multi block local binary patterns. One block is of size 3x2. Taken from [35]

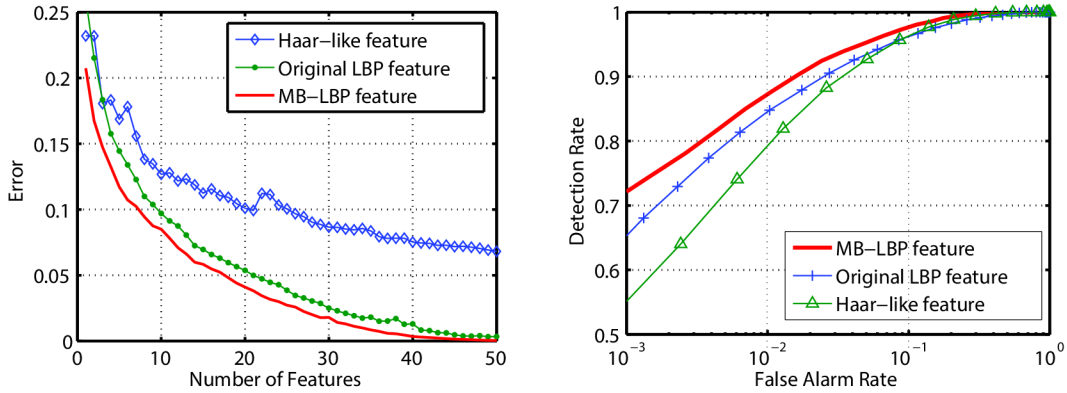


Figure 3.11: Comparison of Multi block LBPs with LBPs and Haar features on face samples. Taken from [35]

3.4.3 Local Rank Functions

Local rank functions are features developed at Faculty of Information Technology, Brno University of Technology as an alternative to existing features with high focus on possibility to implement evaluation in hardware. More in [23]. Main idea is to compute with order of intensities of pixels instead of intensity values. This order within the grid (usually 3x3 blocks) is called rank. This resolves into that LRF are invariat to lighting conditions or some adjustments of picture and also into that it is easy to optimize those for high computational efficiency.

From this concept were developed more specific features called Local Rank Differences (LRD) and Local Rank Patterns (LRP). LRD are computed as a difference between two ranks. LRP are computed as sum of two ranks, while the first rank is multiplied by 10. See [24].

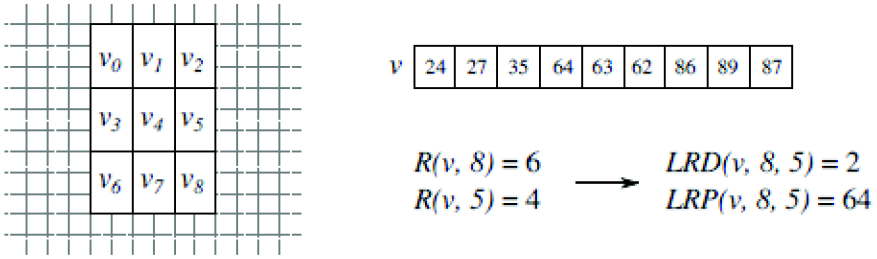


Figure 3.12: Evaluation of two LRF based functions, LRD and LRD. V is gotten from the images, then are computed ranks R for v from which are computed features using formulas 3.1. Taken from [24]

$$LR(v,a) = R(v,a) \tag{3.1}$$

$$LRD(v,a,b) = R(v,a) - R(v,b) \tag{3.2}$$

$$LRP(v,a,b) = 10R(v,a) + R(v,b) \tag{3.3}$$

3.4.4 Histograms of Oriented Gradients

Histogram of Oriented Gradients (HOG) is a feature which is based on computing with the most appearing directions and magnitudes of gradients in a specific part of an image. There are multiple possible approaches for making decision on what information from the histogram would be representing the feature. One of those could be seen on the picture 3.13.

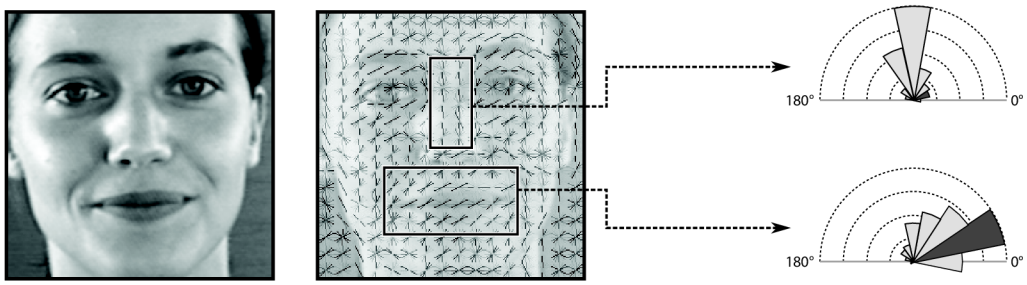


Figure 3.13: Calculation of HOG features. In this case, response of the feature are the second histograms which are highlighted. Taken from [24]

3.4.5 Other features

There are much more features what could by used for object detection. There is also big field for inventing new features as different types of object sensitive to different changes in images. The listed ones are state-of-the-art useful for this project as main priority for mobile devices is still fast evaluation as performance of those device cannot be as big as performance of machines determined for processing computationally complex tasks but also there is demand for mobile devices to make an object detection in real time as the purpose of object detection on these devices leads into this sphere.

Chapter 4

Problem analysis

In this chapter is analyzed, how is object detection used in common applications available on the market. The results of analysis are commented on and analysis go deeper. In next section are introduced approaches which could be used at the moment for developing Android application with object detection. Last section explains how are those approaches grasped in demontrational application.

4.1 Available applications with object detection on Android

The possibility of using information technologies for object detection is well spread around the globe for some time. Object detection in surveillance or industry has been mentioned. The most of people are familiar with face detection on web when using the biggest social network Facebook. In there, each uploaded photograph is processed and users are offered to tag their friends using rectangles which are marking areas where faces has been detected. But what about object detection on mobile devices and more precisely on devices running operation system Android which is so largely growing these days?

The right place to go when making an research about current state of market in terms of applications for Android is definitely Google Play store. In here could be found the most of existing applications for end users. The topic of this thesis is Object detection algorithms and so the first thing to do is to search for related keywords in Google Play store to relieve what possible solutions are ready to go.

First keyword to search for is naturally „Object detection“. Google Play store is very dynamic place and new applications are being added every day, but at the moment of search there were no applications for general object detection. Another useful keyword „face detection“, „smile detection“ and „car detection“. For those searches were found some applications, usually demo applications based on OpenCV or Android face detection. More notably application for detecting smiles for taking photos „SmileCam“ or applications for swapping faces on photos. Those applications comes with quite bad rating, about 3 points. SmileCam was not even possible to test, as it crashed on testing device right after launching. Most popular applications for „Face swap“ do not work in real time and detections have to be manually adjusted.

4.2 Approaches to build application with detection on Android platform

Aim of this thesis is to get familiar with object detection for most popular mobile platform, Android. Conclusions of this thesis might lead into answer to the question, why are not applications with object detection more popular.

When making application with object detection, it is possible to grasp it in three ways:

- Using only standard Android libraries
- Using external libraries
- Do it yourself approach

When it comes to Object detection, Android API is not very rich. It provides whole framework for working with camera and drawing images on the screen but object detection is restricted only to face detection. In API 1 is ready to use class `FaceDetector` which is able to detect faces on bitmaps. Detection is returning back objects of type `Face` representing detected faces. Those object contain information about position of detected face as middle point and distance between eyes. In API level 14 is added `Camera.Face` and its callback `Camera.FaceDetectionListener` which are detecting faces with more information (such as bounding rectangle or mouth position) right on camera frames, but for its use is need on-device hardware support.

The state-of-the-art library for computer vision OpenCV has it's API for Android and naturally, this library is providing tools for object detection, see [10]. From version 2.4.2, devices on which should be ran application with OpenCV have to have installed OpenCV manager which is availible in Google Play store. Becouse of this, application does not have to include whole library by itself, and therefore are smaller of size and also have their OpenCV part updated separately. Even though that full API is for Java, all the functions are native and highly optimized. OpenCV provides classifier in class `CascadeClassifier`. Older object detections using cascade classifier in OpenCV were performed only with Haar features, but now `CascadeClassifier` support also LBP features. User can train his own classifier using tools for OpenCV and provide is as an XML file. Therefore any kind of object which is feasible to detect with classifier based on Haar or LBP features, could by detected on Android with use of OpenCV library.

Last approach is to build own object detecting engine. This assumes deep knowledge about theory of object detection. Another problem is that Java API is too slow for such a computationally complex task and therefore at least computational part of the application has to be written in native code. This is an option in Android, as it provides NDK for compiling native C/C++ code into applications. NDK comes separately from standard development pack and programmer has to have knowledge about C/C++, JNI, Dalvik virtual machine fundamentals and NDK setting itself. This makes programming this way relatively demanding and even official documentation recommends to avoid NKD if possible [4].

4.3 Selected approach

This thesis is comparing all the approaches. Aim of the thesis was not to train own classifier and therefore there had to be chosen some type of objects, what already has some trained

classifiers available. The most common object to detect is human face. There are prepared databases of faces which were proven while training well known classifiers as is the one from Viola and Jones [31]. Faces has also detector in standard Android API thus application is tested and also developed with main focus on face detection task. Even though, is easy to change used classifier for another trained for another type of objects.

Chapter 5

Design

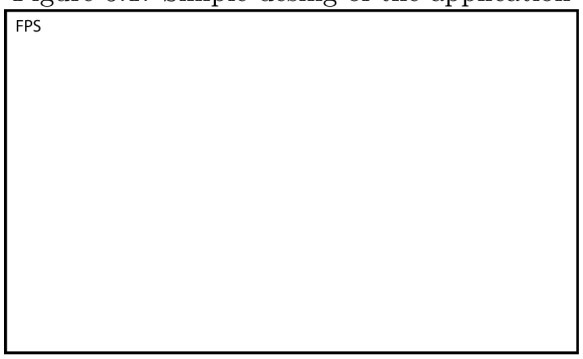
In this chapter is decribed what is the purpose of designed application and how is it reflected on user interface. Further is discussed selected classifier and its format.

5.1 Purpose of the application

Application is supposed to demonstrate object detection on Android platform. The best way how to do it is to show detection realtime as it is. Android devices are most notably smarphones and tablets with camera and processing frames from camera is the reason why is object detection for Android interesting as these devices are mobile and could be taken anywhere out. It should be possible to pull out the device and use an object detection application instantly, no matter the specific purpose of application. This means that demonstrating application should be able to process images from camera and do it as fast as possible to give to the user realtime feel.

Demonstration should be raw, without any „juicing“ like are visual effects, becouse this would bring attention of the user out of the main purpose of the application. On the other hand it has to visualize results. Detected objects should be highlighted. The best way is to draw rectangle right around area which was subject to detection. As this application is mainly for demonstrating performance, some information about this matter should be also provided to the user. Ideal thing for this is a counter of frames per second (FPS) in the corner of user interface.

Figure 5.1: Simple desing of the application



5.2 Selection of suitable classifier

Application is designed to compare multiple ways of detection object on Android platform. There was a decision to make, which algorithm to choose for implementation of detector in NDK. Type of object to test on was decided to be human face thus this was taken into consideration. The main thing to consider was that detector is going to run on mobile device which has some specifics. Very important condition was that chosen algorithm can not have any high demands on computational power or memory requirements. Characteristics as invariance to lighting conditions are very welcome as normalizing each frame before detection on mobile device might be a problem.

Features selected to use are Local binary patterns. These are proven to perform well in face detection tasks [35], are computing with integers which make evaluation fast, are invariant to lighting conditions, are currently state-of-the-art feature for object detection (cascade classifier in OpenCV uses LBP) and last but not the least, these are easy to implement in native code.

LBPs are evaluated on greyscale images. Therefore image from camera which is colorful has to be transformed to one channel greyscale before LBP evaluation.

Another thing is that when using classifier with these kind of features, objects to be detected are of constant size. But there is demand to detect all objects on the image. The way out form this is called image pyramid. Image pyramid is set of different sizes of images made from original one. Using this is classifier not adjusting to image, but image is adjusting to classifier. Images are downsampled with given ratio, interpolation strategy and the smallest size. For instance, if object to be detected is of size 26x26 pixels then if object is over whole space of an image then it will be detected on the smallest image from pyramid set.

As a classifier was chosen one trained using Waldboost algorithm trained on ČVUT face data set with parameters alpha=0.2, beta=0.1. There was no need to train new classifier for faces. The classifier is preserved as XML file and its structure is illustrated in figure 5.2.

```
<WaldBoostClassifier classifierName="NONAME" minStdDev="0" imageSizeX="24" imageSizeY="24">

 <stage posT="1e+50" negT="-1.0579">
  <HistogramWeakHypothesis predictionValues="-1.283203125 -0.228515625 -0.732421875 -0.521484375 ... ">
   <LBPFeature positionX="13" positionY="6" blockWidth="2" blockHeight="2" />
  </HistogramWeakHypothesis>
 </stage>

 <stage posT="1e+50" negT="-1e+50">
  <HistogramWeakHypothesis predictionValues="-0.79448686026923 -0.28992716270843 -0.26171028559741 ...">
   <LBPFeature positionX="16" positionY="0" blockWidth="2" blockHeight="1"/>
  </HistogramWeakHypothesis>
 </stage>
       ...
</WaldBoostClassifier>
```

Figure 5.2: Example of XML with classifier

The most important values are:

- `ImageSizeX` and `ImageSizeY` in `WaldBoostClassifier` element. This is information about size of object to be detected, provided by its width and heigh in pixels.
- `PosT` and `NegT` in `stage` element. These are positive and negative thresholds for one waldboost stage. If prediction value of the stage is higher than positive threshold, then no more stages are evaluated and detection is positive. If prediction value is lower than negative threshold, then no more stages are evaluated and detection have negative response.
- `PredictionValues` in `HistogramWeakHypotesis` is array with 256 values. Result of LBP evaluation is 8bit number. This number points into array of prediction values for current stage. Prediction values are cumulated with values from previous stages and compared with thresholds.
- `positionX` and `PositionY` in `LBPFeature` element are providing information about position of current LBP feature within image. This information could be represented by position of top left corner of grid of LPB block, shifted left and top border of an image by stated number of pixels.
- `blockWidth` and `blockHeight` from `LBPFeature` elements are providing information about how many pixels high and how many pixels wide is one block within the LPB grid.

Chapter 6

Implementation

In this chapter is described how is designed application implemented and all parameters are stated. Each implemented detection has its own section. The main focus is on custom detection, for which is included in-depth description of used algorithm.

6.1 Development environment

For appropriate development was used Eclipse IDE with ADT, NKD and OpenCV. All those components have to be set. Usage of Tegra Development Pack[1] made things smoother as it contains all necessary tools for development of appplication with computer vision.

6.2 Application and UI

The idea of implementing design of application is taken from demo samples which comes with OpenCV library for Android. Application consist of only one activity. In here is used camera from `org.opencv.android.CameraBridgeViewBase`. This procure all the work with camera. Frames from camera are processed in callback function `OnCameraFrame`. Images are provided as OpenCV `Mat` type.

6.2.1 Unprocessed camera frame

Camera frames given to callback are prepared in both interpretation as four channel RGBA matrix an also one channel grayscale matrix. For view mode without processing, which is used for comparison of speed to modes with processing. In this case, the given RGBA camera frame is simply returned back.

6.2.2 Android detection

Android face detection from API 1 assumes input to be a Bitmap type of image. Therefore OpenCV Mat has to be converted using OpenCV method `Utils.matToBitmap`. Then new instance of `FaceDetector` is created and detection performed. The only possible parameter to set is maximum number of detected faces. It is not possible to set minimal size of object and this detection is ignoring small faces. Detected faces are stored into array of `FaceDetector.Face` objects. From those are used atributes `MidPoint` and `EyeDistance`.

[1] https://developer.nvidia.com/tegra-android-development-pack

On position of midpoint are drawn circles with radius of eyedistance into OpenCV RGBA Mat which will be returned as an output.

6.2.3 OpenCV detection

This detection is all about `org.opencv.objdetect.CascadeClassifier`. At first, classifier is initiallized with XML file which contains classifier. In this case it is `lbpcascade_frontalface.xml` which is now OpenCV's default classifier for faces. It is based on Local binary patterns feature.

Detection itself is happening when calling `detectMultiScale` method. Parameters are:

- `scaleFactor` is stating density of image pyramid. In other words, this parameter says how much is reduced image size in each iteration (e.q. new width = width / scaleFactor). Default value is 1.1, this was changed to **1.3** according to setting of custom detector.
- `minNeighbors` is value which is stating how many positive detections has to be around detection to be marked as true positive. Higher values leads to more false negatives, lower values leads to more false positives. Default value is 3 which seems to be ideal. Custom detector is not exploiting the neighbour concept and therefore the value is set to **0** to be comparable.
- `flags` is unused for LBP classifier. It is remainder for classificator using Haar cascade.
- `minSize` and `maxSize` are parameters to set minimal and maximal size of object to be detected. This could speed up the classification, if it is known, that objects to be detected are supposed to have certain size. Those parameters are set to **0** as custom detector is searching for all sizes of objects.

Detection gives its result as a matrix of rectangles with areas of positives detection. Those rectangles are draw into input RGBA Mat and returned as a result.

6.2.4 Custom detector

The custom detector is written in native code using C++. This is possible because of use of NDK. The detect is performed in native function `MyDetector`. It had to be declared as a native function.

Usage of classifier

Classifier for custom detector is saved in asset folder of the project as an XML file. It needs to be loaded and parsed on each start of application. Class Stage is representing data from the classifier. Parsed data are stored as an array of Stages. Parsing of XML file is one of the things to make with standard Android libraries. There are even three possible parsers:

- DOM – Stores whole document tree to the memory. Is able to go backwards and add elements.
- SAX – Is based on events and callbacks assigned to them. Use less memory than DOM.
- XMLPullParser – Use less memory than DOM, but is more easy to use than SAX.

The chosen one is XMLPullParser. Parsing, as a long term task, has to be out of User Interface thread. This is one of the reasons why is it performed on start of the application as a background process using AsyncTask.

Usage of JNI

One of the problems to solve was how to pass arguments to the native function. Java objects with OpenCV Mat types has available method `getNativeObjAddr` which is returning address of Mat in the memory. This address is in native code casted to reference to Mat. So this is how frame with image from camera in both multichannel and greyscale are passed to native code. Another thing to pass are data of classifier. Those are passed as Java array of objects and thus are retrieved as `jobjectArray` type.

Data for classification are obtained in function `loadStages`. In there is used Java Native Interface (JNI) to get values from Java objects. It is iterated through the array of Java objects and values are extracted. When native function is called from Java, it comes with pointer to `JNIEnv` what is pointer to structure which is storing pointers to all JNI functions.

At first is needed to get object from array and cast it into `jobject`. The call for getting object is `GetObjectArrayElement`. Then is gotten class of the object using `GetObjectClass`. Then follow calls for getting values such as `getIntField` or `getFloatField` in combination with obtaining field id's of desired field using `GetFieldID` with given class, field name and type. Values are stored into arrays. Array with prediction Values is treated similarly but it is copied into new array so the old one could be released. All calls which are returning some sort of object are creating local reference which is stored in reference table. This could have the most 512 entries, so reference are cleaned using `DeleteLocalRef` in each iteration. Finally, all the arrays are wrapped and returned for further processing.

Image pyramid

Image pyramid is simulated by using function `resize` from OpenCV. This function takes parameters for new size and interpolation method. New size is set as 1/1.3 of previous size. It seemed like good compromise between number of images in pyramid and possibility miss positive detection. Interpolation method is set as `INTER_CUBIC`. This is a bicubic interpolation over 4x4 pixel neighborhood. It provides fair information loss and even that it is slower than some another methods, performance was not measurable within whole application.

Resizing is happening within the cycle on the highest level, so it wraps function for detection and this one is called again for every image in pyramid. This cycle breaks when size of next image would be smaller than scanning window (size of object to be detected).

Scanning window

Size of scanning window is determined by used classifier. Moving step for scanning window is set to 2. This means that scanning window is not activated on every position in image but it is skipping every other row and column. This does not resolve into false negative detections because window shifted by only one pixel represents barely the same information according to trained classifier. This also helps with speeding up the application as on positions with positive detections is high number of stages of classifier to be evaluated.

Scanning window is represented by rectangle initialized with position and size. This

rectangle is used for initializing new Mat what is passed to feature evaluation. This initialization is called in OpenCV terminology „creating Region Of Interest(ROI)“.

Feature evaluation

Function for feature evaluation is core of whole application. In here are the highest demands for optimization, becouse this function is called for every ROI which means for example for resolution 800x480 87,849x calls. The feature what is worked with is Local Binary Patterns.

The computation is happening inside the cycle what is iterating through all the stages of given classifier. Every stage has specific parameters for position of current feature within the ROI and block size. At first is computed value of center block. Position of topleft corner of central block of current feature within ROI is computed as position of topleft corner of feature summed with width of block on x axis and high on y axis. Then intensities of all pixels in the block are summed up.

Used LBP features are in 3x3 blocks. Algorithm is to give each block specific weight. The order of iterating through blocks in right order according to weights is achieved by using constant array where are stored positions of blocks within the 3x3 grid. In the most inner cycle is at first gotten sum of pixels for current block according to array with order. This sum of current block is compared to sum of central block. There is no need to compare average intensities within the block as average intensities could be compared as good as sums. If the sum of current block is bigger, then the final result is actualized using binary OR operation with variable which holds number which has in binary representation 1 on position of weight for current block. This variable is initialized as 1 and in each iteration binary shifted to the left, so e.g. in first iteration it changes from $00000001b \rightarrow 00000010b$.

After all 8 iterations, the number representing final result for the stage is used as key for stage's array of prediction values. From there is taken prediction value which is cumulated with prediction values from previous stages. This is compared to stage's thresholds. If the cumulated value is bigger than positive threshold, then ROI is decided to be positive detection, if the value is smaller, then the ROI is rejected as negative detection. Otherwise the iteration continues with next stage till all the stages are evaluated. Then the ROI is decided to be positive detection.

Positive detections highlighting

Finally, every positive detection is stored into vector of rectangles as a rectangle initialized by ROI adjusted to real size of detection according to current ratio of image downsampling.

Chapter 7

Testing

In this chapter is described how were designed test cases, which devices were used for testing and testing results will be listed and explained. Finally results are evaluated and future work following this thesis is proposed.

7.1 Test cases

Testing was focusing on comparison of speed of used detectors. Application was turned on and then were computed average frame rates. Frame rates were provided by FPS counter from OpenCV and recorded using Logcat tool from Android debug bridge.

For testing were chosen two scenarios. The first one was to aim camera of device with application on printed picture with people 7.1. The second scenario was to lay the device with application on the table so the camera was covered so the surface for detection consisted of black background with noise.

Figure 7.1: Image used for testing

For both scenarios were tested OpenCV detection, Android API detection and detection using custom detector. This was performed with camera setting for maximal resolution of 800x480 what represented high resolution and with maximal resolution 240x160, what

represented low resolution. Testing device Sony Xperia M do not offer resolution 800x480 so it was adjusted to 768x432.

It must be said that Android API detection suffered with plenty of false negative detections on given scenarios, namely on those with lower resolution. Custom detection appeared to have more false positives than OpenCV, but not obviously, see figures below.

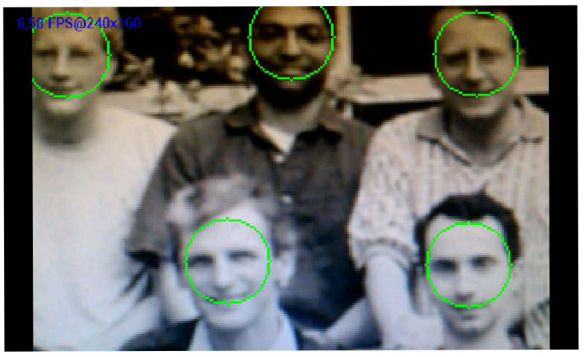


Figure 7.2: Android detection is set to detect only specific sizes of faces, therefore its hard to compare results.

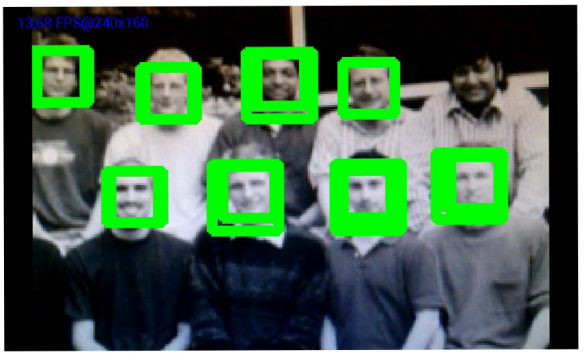


Figure 7.3: OpenCV detector has the best performance

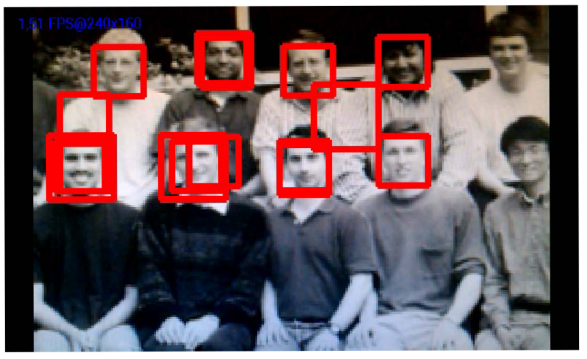


Figure 7.4: Custom detection suffers from false positives

7.2 Testing devices

Testing devices covers whole specter of phones from low-end to high-end. The first phone did not fulfil requirements of application and application was not able to run.

LG Optimus One	
Released	2010
OS	2.2 Froyo
CPU	600 MHz ARM 11
GPU	Adreno 320

Huawei Ascend G300 is device what was used for development. It is low-end device, new modern version of Android is running on it due to custom modification.

Huawei Ascend G300	
Released	2012
OS	4.3 Jelly Bean
CPU	1 GHz Cortex-A5
GPU	Adreno 200

Sony Xperia M is representative of devices with dual-core processor. It could be classified as a middle-class device.

Sony Xperia M	
Released	2013
OS	4.1 Jelly Bean
CPU	Dual-core 1 GHz Krait
GPU	Adreno 200

Devices in Nexus series are special phones designed by Google as a reference devices for Android developers. Testing device Nexus 4 could be classified as a high-end.

Nexus 4	
Released	2012
OS	4.4.2 KitKat
CPU	Quad-core 1.5 GHz Krait
GPU	Adreno 305

7.3 Testing results

The fastest detection was performed by device Nexus 4 using OpenCV detection. See 7.5 and 7.3. This is expected result as Nexus 4 is device with the highest computational power and OpenCV is highly optimized. Even though, this is not fast enough to be considered as real-time.

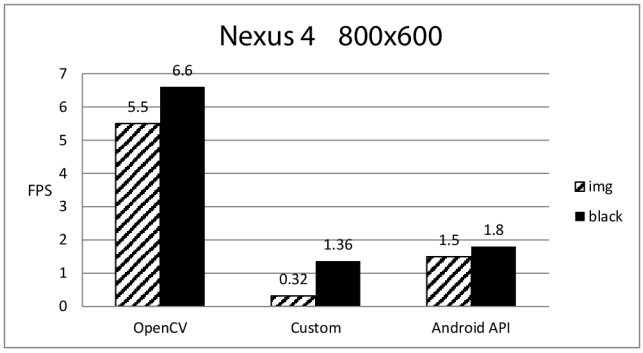


Figure 7.5: Comparison of achieved frames per seconds while performing detection on device Nexus 4 using camera resolution 800x480 pixels

Both custom and Android API1 detection are too slow for any real use on resolution as high as 800x480. Values for Sony Xperia M on high resolution are reflecting values for Nexus 4, but the slow down is not as big as expected. See 7.6 Android API detection for black background got even higher value than on Nexus. This could be given by conditions for measuring, as this trend is not confirmed by measurements on low resolution, see 7.3.

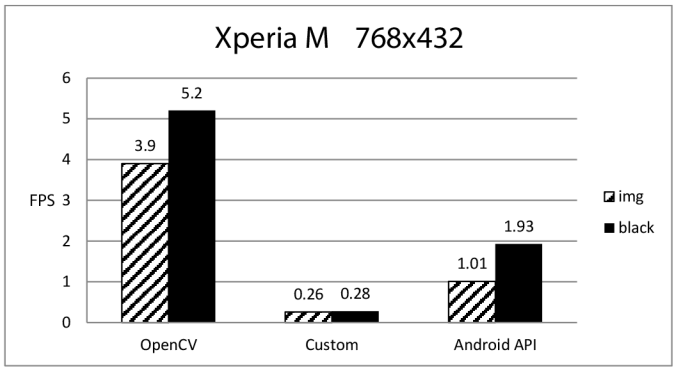


Figure 7.6: Comparison of achieved frames per seconds while performing detection on device Sony Xperia M using camera resolution 768x432 pixels

Testing on device, what was used for development, Huawei Ascend G300, revealed much smaller differences than tests on more modern phones. Results for OpenCV and Android API detection got so close together, that it could be considered as similar performance. This trend was confirmed by measurements on low resolution, see 7.3. Difference between custom and OpenCV was also shortened. In this case OpenCV performed only about 2.5x faster than custom detector. The gap on low resolution was bigger. OpenCV performed about 3.5x faster in there.

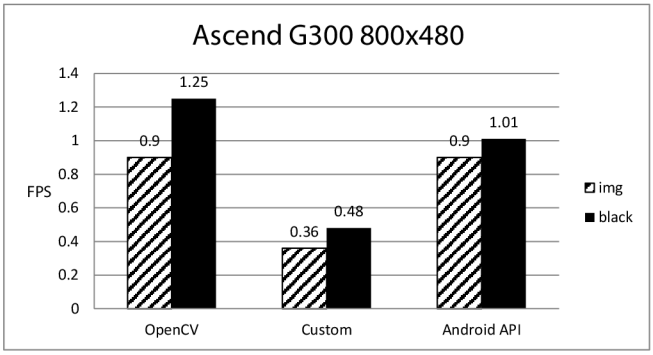


Figure 7.7: Comparison of achieved frames per seconds while performing detection on device Huawei Ascend G300 using camera resolution 800x480 pixels

Value of detection on image for OpenCV detection for low resolution on Nexus got over the milestone of 15 FPS. This could be considered as real-time detection. Custom detector for this case is about 5 times slower but giving performance what could be usable for some applications.

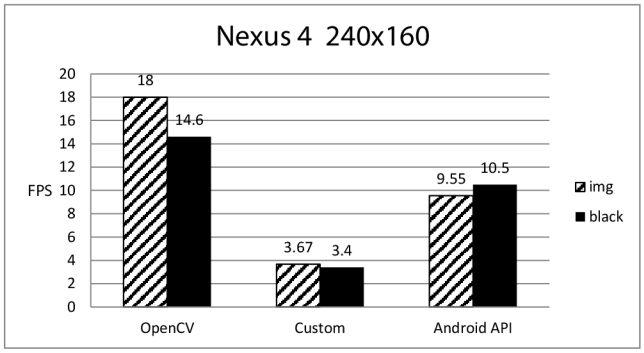


Figure 7.8: Comparison of achieved frames per seconds while performing detection on device Nexus 4 using camera resolution 240x160 pixels

Values for Xperia for low resolution are similar as for Nexus, but a bit slower. Value for black background for Android API detection is an exception wchich could be caused by various reason. This measuring was not getting steady in frame rates, so final value is an average of multiple different values with big differences. Android API detection for low resolution was not giving good detection results, because of its setting, what was not possible to change, so those results are inconclusive.

Lowening camera resolution is a must for object detection on Huawei Ascend G300. With as low resolution as 240x160, detection begins to be feasible for use in applications.

Noticeable is difference of performance of custom detector on different phones throwout the experiment. While for 800x480 is OpenCV on Nexus about 6x faster than on Ascend,

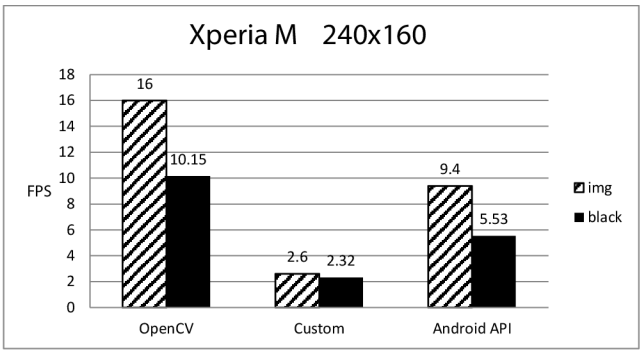


Figure 7.9: Comparison of achieved frames per seconds while performing detection on device Sony Xperia M using camera resolution 240x160 pixels

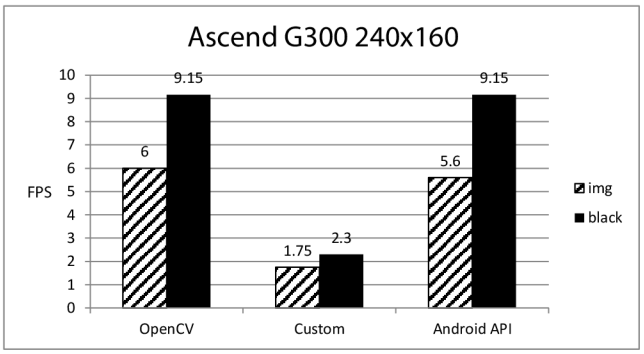


Figure 7.10: Comparison of achieved frames per seconds while performing detection on device Huawei Ascend G300 using resolution 240x160 pixels

custom detector have similar performance on all phones. For detection on image even Ascend appears to be the fastest. Detetion on black background looks more convincing. In there is Nexus about 2.8x faster than Ascend, but still Ascend appears to be about 1.7x faster than Xperia.

7.4 Result evaluation

This shows how much important is optimization. OpenCV detector is able to use all the hardware possibilities of modern devices much better than custom detector. These could be multicore CPUs, NEON instructions 7.12 or even computations on GPU 7.11. While CPU could process only one pixel in time, NEON coprocessor could process 16 of them [26].

The way how to improve custom detector leads into the same way. Another speedup would be achieved by hardcoding values of detector into native code. Now are those passed

from java and extracted using JNI which appeared to be quite expensive operation.

Performance of detectors on modern devices is sufficient for performing detection and more generally computer vision tasks in future applications. The key is to set parameters of detector to fit into task. For instance to skip needless computations as could detecting in frames of size where is not expected positive detections, as for some tasks are sizes of objects to detect known in advance.

The future is in new devices and optimization to its hardware capabilities. Possibilities for applications with object detection and computer vision on mobile devices are wide and exploiting new features of new devices as using GPU are very promising. See [26].

Figure 7.11: Here is stated how much are some of tasks performed by OpenCV speeded up by exploiting GPU for computations. Note the speed up for Viola-Jones detector. Taken from [26]

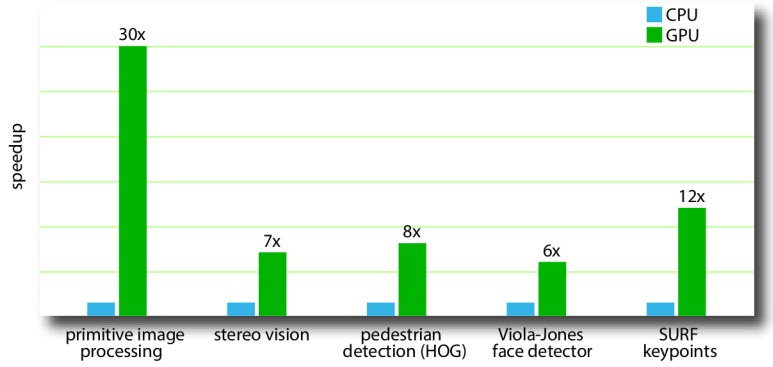
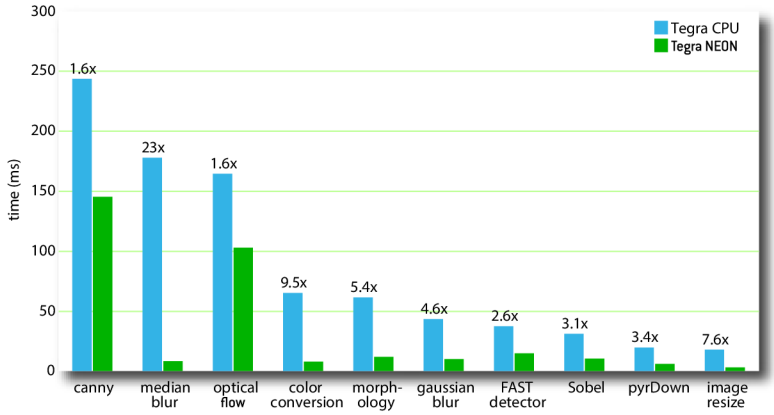


Figure 7.12: Here is stated how much could be some of basic tasks performed by OpenCV speeded up by Exploiting NEON instructions. Taken from [26]



Chapter 8

Conclusion

Goals of this thesis were to get familiar with mobile platform Android and application development for it using Software development kit and also Native development kit, to get familiar with OpenCV library for computer vision, to propose solutions demonstrating the detection of objects on Android platform using native calling, implement and test the designed program and evaluate the results. All the tasks from assignment were accomplished.

Getting familiar with Android and OpenCV was satisfied by studying textbooks and official documentation, practicing acquired knowledge by exercising when making multiple demo applications and integrating useful parts into final demonstrating application.

As a best demonstration of object detection algorithms on Android platform was chosen application for detection of human faces using detection from OpenCV library, Android API and self designed detector implemented in native code.

Demonstrative application was developed using ready made classifier for face detection which used LBP features and was trained using Waldboost algorithm. This approach was selected as current state of the art for the task.

Final application was tested on multiple Android devices from different parts of current Android device market spectrum. Tests revealed that custom implementation of detector is way behind detector from OpenCV in performance. It was proposed to redesign application to avoid JNI calls and put more attention into optimization and exploiting hardware capabilities of modern devices such are NEON instructions, muticore CPUs and computations on GPU.

Another way how to make better detector for Android than offers OpenCV could be to differentiate from it by using classifier which is using different kind of features such as LRF. That could bring improvement in both speed and precision for some types of objects.

Current state of object detection algorithms is sufficient for some purposes on some more modern devices. The future of object detection and more generally computer vision is very promising. With both hardware and software improvements, which are in sector of mobile devices moving rapidly forward, should be possible to use those technologies to improve our everyday life.

The future work following up this thesis should focus on improving overall detection performance using new hardware possibilities of devices such as exploiting mobile GPU for computations and also on designing application which uses object detection for solving some real problems.

Bibliography

[1] Android. www.android.com. Accessed: 2014-07-13.

[2] Android activations to reach 1 billion - Business Insider. https://intelligence.businessinsider.com/android-activations-to-reach-1-billion-2013-4. Accessed: 2014-07-20.

[3] Android Developer Tools — Android Developers. http://developer.android.com/tools/help/adt.html. Accessed: 2014-07-15.

[4] Android NDK — Android Developers. https://developer.android.com/tools/sdk/ndk/index.html. Accessed: 2014-07-10.

[5] Android sdk — android developers. https://developer.android.com/sdk/index.html. Accessed: 2014-07-20.

[6] Introducing ART — Android Developers. http://source.android.com/devices/tech/dalvik/art.html. Accessed: 2014-07-15.

[7] Introduction. http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/intro.html. Accessed: 2014-06-09.

[8] JNI Tips — Android Developers. http://developer.android.com/training/articles/perf-jni.html. Accessed: 2014-06-14.

[9] OpenCV. http://itseez.com/OpenCV/. Accessed: 2014-07-21.

[10] OpenCV4Android usage models — OpenCV. http://opencv.org/platforms/android/opencv4android-usage-models.html. Accessed: 2014-07-08.

[11] • Smartphone operating systems: global market share 2009-2013 — Statistic. http://www.statista.com/statistics/263453/global-market-share-held-by-smartphone-operating-systems/. Accessed: 2014-07-20.

[12] Native Development Kit (NDK) - An Android Tutorial. http://www.ntu.edu.sg/home/ehchua/programming/android/android_ndk.html, 2012-07-01.

[13] Dashboards — Android Developers. https://developer.android.com/about/dashboards/index.html, 2014-07-07. Accessed: 2014-07-20.

[14] Smartphone - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Smartphone, 2014-07-16. Accessed: 2014-07-16.

[15] Android (operating system) - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Android_(operating_system), 2014-07-21. Accessed: 2014-07-21.

[16] Apple Inc. - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Apple_Inc., 2014-07-21. Accessed: 2014-07-21.

[17] iOS - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Ios, 2014-07-22. Accessed: 2014-07-22.

[18] iPhone - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/IPhone, 2014-07-22. Accessed: 2014-07-22.

[19] Activity — Android Developers. http://developer.android.com/reference/android/app/Activity.html, 2014-07-23. Accessed: 2014-07-24.

[20] Gary Bradski. *Learning OpenCV : computer vision with the OpenCV library*. O'Reilly, Sebastopol, CA, 2008.

[21] Gary Bradski. *Learning OpenCV : Computer Vision in C++ with the OpenCV Library*. O'Reilly & Associates, Sebastopol, CA, 2012.

[22] Marko Gargenta. *Learning Android*. O'Reilly Media, Sebastopol, CA, 2014.

[23] Adam Herout, Pavel Zemcík, Michal Hradiš, Roman Juránek, Jiří Havel, Radovan Jošth, and Martin Žádník. Low-level image features for real-time object detection. *IN-TECH Education and Publishing*, page 25, 2009.

[24] Roman Juránek. *Acceleration of Object Detection Using Classifiers*. PhD thesis, 2012.

[25] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):971–987, 2002.

[26] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Real-time computer vision with opencv. *Communications of the ACM*, 55(6):61–69, 2012.

[27] Felix Richter. The price gap between ios and android is widening. http://www.statista.com/chart/1903/average-selling-price-of-android-and-ios-smartphones/, 2014-06-01. Accessed: 2014-07-20.

[28] Janessa Rivera and Laurence Goasduff. Gartner says worldwide traditional pc, tablet, ultramobile and mobile phone shipments are on pace to grow 6.9 percent in 2014. http://www.gartner.com/newsroom/id/2692318, 2014-03-27. Accessed: 2014-07-21.

[29] Robert E Schapire. A brief introduction to boosting. In *Ijcai*, volume 99, pages 1401–1406, 1999.

[30] Jan Sochman and Jiri Matas. Waldboost-learning for time constrained sequential detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 150–156. IEEE, 2005.

[31] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[32] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[33] Ming-Hsuan Yang, David Kriegman, and Narendra Ahuja. Detecting faces in images: A survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(1):34–58, 2002.

[34] Jay Yarow. This chart shows google's incredible domination of the world's computing platforms. http://www.businessinsider.com/androids-share-of-the-computing-market-2014-3, 2014-03-28. Accessed: 2014-07-21.

[35] Lun Zhang, Rufeng Chu, Shiming Xiang, Shengcai Liao, and Stan Z Li. Face detection based on multi-block lbp representation. In *Advances in biometrics*, pages 11–18. Springer, 2007.

Appendix A

Project structure

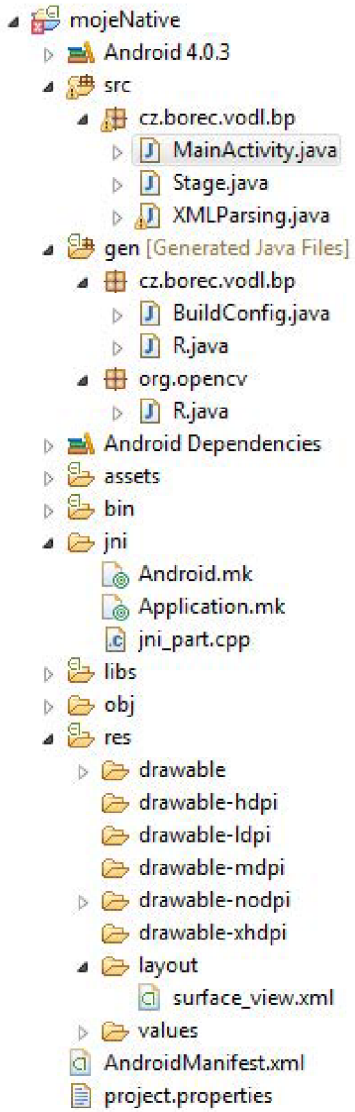


Figure A.1: Demonstration of application structure from developer's perspective. Image is taken from Eclipse development environment

Appendix B

CD content

Attached CD contains:

- project source codes
- Thesis in PDF
- Application project with source codes
- Application as a APK file
- Readme file