VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

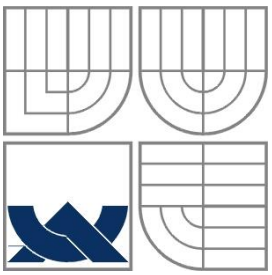
NÁVRH A OPTIMALIZACE OBRAZOVÝCH KLASIFIKÁTORŮ

DISERTAČNÍ PRÁCE
PHD THESIS

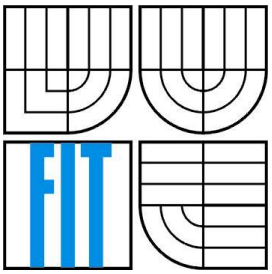
AUTOR PRÁCE
AUTHOR

Ing. FILIP KADLČEK

BRNO 2016



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH A OPTIMALIZACE OBRAZOVÝCH KLASIFIKÁTORŮ

DESIGN AND OPTIMISATION OF IMAGE CLASSIFIERS

DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE
AUTHOR

Ing. FILIP KADLČEK

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Dr. Ing. OTTO FUČÍK

BRNO 2016

Abstrakt

Detekce objektů je velmi důležitou operací v systémech zpracovávajících obrazová data, jako jsou například různé dohledové nebo bezpečnostní systémy. Tato úloha je výpočetně velmi náročná a konzumuje velké množství výpočetních zdrojů. Detekce se provádí například pomocí obrazových klasifikátorů. S vývojem nových obrazových snímačů, a s tím spojeným stoupajícím datovým tokem z nich, rostou požadavky na výpočetní prostředky, a klasifikaci je tak třeba provádět velmi rychle a efektivně. Disertační práce se proto zaměřuje na obrazové klasifikátory, optimalizaci jejich běhu, tvorby a využití FPGA technologie pro jejich implementaci. V práci jsou využity klasifikátory založené na obecné klasifikační metodě AdaBoost, pomocí které je možné detekovat různé typy objektů. AdaBoost patří do skupiny boostingových metod, které jsou založeny na kombinaci výsledků mnoha slabých klasifikátorů pro získání konečného výsledku. V této práci se pro vytváření slabých klasifikátorů využívá jednoduchých binárních obrazových operátorů – Local Binary Pattern, které mají dobrou diskriminativní sílu a jsou vhodné pro FPGA technologii. Jedním z cílů této práce je prokázat, že použití aplikačně specifického přístupu vede k lepším výsledkům z hlediska rychlosti a efektivity výpočtu, než při použití obecných klasifikátorů. Splnění tohoto cíle je rozděleno na několik částí, z nichž první je přizpůsobení tvaru obrazových operátorů pro aktuálně řešenou úlohu. Pro řešení této problematiky byl v práci zvolen genetický algoritmus. Pomocí něj jsou navrženy nové aplikačně specifické tvary příznaků a je dosaženo zpřesnění klasifikace až o 4 %. Předností nově navržených příznaků je, že jejich klasifikační výkonnost je optimalizována vzhledem k aktuální aplikační úloze. Hlavní přínos této práce však spočívá v představení multiparalelního detektoru objektů pracujícího v reálném čase – RT-MPOD. Ten je založen na vícenásobné aplikaci vyhodnocovací jednotky AdaBoost - *AC*, která využívá neseřazeného vyhodnocení slabých klasifikátorů pro dosažení plně paralelního běhu. *AC* jednotka vyhodnotí jedno detekční okno v každém hodinovém cyklu. RT-MPOD dosahuje velmi vysoké rychlosti zpracování dat - jeden pixel za jeden hodinový cyklus a může zpracovávat obrazový tok až s *300* megapixely za sekundu (to je například obraz o rozlišení *3000 x 2000* pixelů a *50* snímků za sekundu), což je téměř o řád více než doposud představené architektury. Vysoké rychlosti zpracování je dosaženo díky několika technikám použitých v práci. První technikou je využití několikanásobného paralelizmu při zpracování obrazu. Druhou technikou je využití aplikačně specifické implementace klasifikátoru pro každou z použitých AC jednotek. Každý klasifikátor je tak maximálně přizpůsoben aktuální úloze. Vyvinutá *AC* jednotka a také celá RT-MPOD jsou specifické svým proudovým charakterem zpracování dat, což znamená, že se již ke zpracovaným datům nevrací a není třeba dodatečných vyrovnávacích pamětí. Jelikož je každá z částí klasifikátoru silně aplikačně specifická, byla pro pohodlnou práci navržena metoda automatizovaného sestavení klasifikátoru na základě definovaných požadavků. Návrhář poté pracuje s nástrojem na vysoké úrovni abstrakce a je mu dána možnost určit parametry řešení, jako jsou například přesnost výsledného klasifikátoru, parametry cílového FPGA, data pro trénování, datová propustnost a podobně. RT-MPOD je také navržen s ohledem na jednoduché rozdělení do více výpočetních architektur tak, aby jej bylo možné použít například jako předzpracovávající jednotku pro provádění předvýběru zájmových objektů.

Klíčová slova

AdaBoost, zpracování v reálném čase, klasifikace, RT-MPOD, Local Binary Pattern, genetický algoritmus.

Citace

Kadlček Filip: Návrh a optimalizace obrazových klasifikátorů, disertační práce, Brno, FIT VUT v Brně, 2016.

Abstract

Object detection is a very important operation in image processing systems such as various surveillance and security systems. This operation is very computationally intensive and it consumes a large amount of resources. The detection can be performed by image classifiers. The development of new image sensors, which have a big resolution and data rate, brings higher requirements on computation resources and also the classification has to be very fast, precise and effective. For these reasons the thesis is focused on image classifier runtime optimization, creation and utilization of the FPGA technology for their implementation. The thesis utilizes classifiers based on AdaBoost, which is a universal classification method, by which whereby various objects can be detected. The AdaBoost belongs to a group of boosting methods, which are based on statistical combination of many weak classifiers to obtain the final result of classification. In this work simple binary image operators – Local Binary Pattern are utilized to create the weak classifiers. These operators have good discriminative capabilities and they are suitable for FPGA implementation. One goal of this thesis is to prove that using application specific classifiers can lead to better results of computation time and effectivity, than results achieved by general classifiers. The accomplishment of this goal is divided into a few parts of which the first is adjustment of the image operator shape for the current task. To solve this issue, genetic algorithm was chosen. The new application specific shapes of operators were designed by this approach and the total classification accuracy was improved by *4 %*. The main advantage of the newly designed features is their optimization of classification accuracy of the current application task. The main contribution of this thesis is in introducing a Real-Time Multi-Parallel Object Detector – RT-MPOD. It is based on multiple applications of the AdaBoost – *AC* evaluation unit, which utilizes an unordered evaluation of weak classifiers to achieve completely parallel processing. The *AC* unit is designed to evaluate one detection window in each clock cycle. The RT-MPOD reaches very high processing speed – one image pixel per clock cycle and it can process an image stream of up to *300* Megapixels per second (it is for example a video with a resolution of 3000 x 2000 pixels and *50* frames per second), which is nearly a magnitude higher than other introduced architectures. The high processing speed is reached by a few methods, which were used in this work. The first one is utilization of multiple parallelisms during the data processing. The second one is utilization of application specific implementation of each used AC unit. Each classifier is maximally adapted to the current task. The introduced *AC* unit and also the whole RT-MPOD architecture are specific by a stream character of data processing. It means that there is no need to return to already processed pixels and there is no need of additional buffers. Because each part of the classifier is strongly application specific, the method of automatized classifier design was introduced to simplify the classifier creation. After that the designer works with a tool on a high level of abstraction and he determines the parameters such as accuracy of the classifier, FPGA requirements, training data, throughput and so on. The RT-MPOD is also designed to be easily divided into several computational units in order to be used as a pre-processing unit to pre-select objects of interest.

Keywords

AdaBoost, Real-Time, classification, RT-MPOD, Local Binary Pattern, genetic algorithm.

Prohlášení

Prohlašuji, že jsem tuto disertační práci vypracoval samostatně pod vedením svého školitele doc. Dr. Ing. Otto Fučíka a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Filip Kadlček
30. 8. 2016

Poděkování

Chtěl bych poděkovat především školiteli doc. Dr. Ing. Otto Fučíkovi za odborné vedení, připomínky a rady při tvorbě mé disertační práce. Dále bych chtěl poděkovat prof. Dr. Ing. Pavlu Zemčíkovi za jeho náměty a cenné rady a Ing. Jirkovi Husákovy za jeho spolupráci.

Obsah

Seznam zkratek

AC	*AdaBoost Core* – AdaBoost jádro
ARM	*Acorn RISC Machine* – Acorn RISC procesor
ASUM	AdaBoost sčítací jednotka
ASC	Aplikačně specifický klasifikátor
ASIC	*Application Specific Integrated Circuit* – Aplikačně specifický integrovaný obvod
BlockRAM	Bloková paměť RAM
CCU	*Collection and Countation Unit* - Shromažďovací a výpočetní jednotka
CNN	*Convolutional neural network* – Konvoluční neuronové sítě
CU	*Control Unit* – Řídící jednotka
CPU	*Central Processing Unit* – Centrální výpočetní jednotka
CUDA	*Compute Unified Device Architecture*
DLP	*Data Level Paralelism* – Paralelismus na úrovni dat
DMA	*Direct Memory Access* – Jednotka přímého přístupu do paměti
DSP	*Digital Signal Processor* – Digitální signálový procesor
DW	*Detection Window* – Detekční okno
EA	Evoluční algoritmus
EU	*Evaluation Unit* – Vyhodnocovací jednotka
FIFO	*First In First Out* – První dovnitř, první ven
FPGA	*Field-Programmable Gate Arrays* - Programovatelná hradlová pole
FPS	*Frames per second* – Počet snímků za sekundu
Full HD	*Full High Definition* – Plné vysoké rozlišení
GA	Genetický algoritmus
GPU	*Graphical Processing Unit* – Grafická výpočetní jednotka
GP-GPU	*General Purpoase GPU* – GPU pro obecné použití
HD	*High Definition* – Vysoké rozlišení
HOG	Histogram Orientovaných gradientů
IBU	*Image Buffer* – Obrazová paměť
IPG	*Image Pyramid Generator* – Generátor obrazové pyramidy
ISB	*Image Strip Buffer* – Pamět pro uložení pásku obrazu
OpenCL	*Open Computing Language* – Otevřený výpočetní jazyk
OpenMP	*Open Multi-Processing*
LBP	*Local Binary Pattern* – Lokální binární vzor
LR	*Local Rank* – Lokální pořadí
LRP	*Local Rank Patter* – Pořadí lokálních vzorů
LRD	*Local Rank Differences* – Rozdíl lokálních pořadí

LUT	*LookUp Table* – Vyhledávací tabulka
MB-LBP	*Multi-scale Blok Local Binary Pattern*
Mpps	*Mega pixels per second* – megapixelů za sekundu
Mpx	Mega pixel
MMX	*MultiMedia eXtentsion* – Rozšíření pro multimédia
PAC	*Probably Approximately Correct*
PE	*Processing Element* – Výpočetní element
RAM	*Random Access Memory* – Paměť s nahodilým přístupem
RISC	*Reduced Instruction Set Computing* – Výpočet s redukovanou instrukční sadou
RZ	Registrační značka
SIMD	*Single Instruction Multiple Data* – Jedna instrukce více dat
SF	*Scale Faktor* – Poměr změny velikosti
SFU	*Special Functional Unit* – Jednotka speciálních operací
SP	Streamový (proudový) procesor
SPZ	Státní poznávací značka
SM	Symetrický Multiprocesor
SMC	*SM Controler* - Řízení SM
SSE	*Streaming SIMD Extension* – Streamové rozšíření SIMD
SoC	*System On Chip* – Systém na čipu
TLP	*Thread Level Paralelism* – Paralelismu na úrovni vláken
TPC	*Thread Processing cluster* - Vláknový cluster
VHDL	*Very-High Speed Integrated Circuits Hardware Description Language* – Jazyk pro popis hardware
WCU	*Weak Classifier Unit* – Jednotka slabého klasifikátoru

1 Úvod

Neustále rostoucí požadavky na kontinuální dohled za pomocí kamer a také na následnou analýzu dat vedou k tomu, že zpracování obrazu je dlouhodobě velmi žádanou úlohou a mnoho vývojových pracovníků jí věnuje velké úsilí. Je tak nutné jej neustále zdokonalovat jednak z hlediska rychlosti, ale i z hlediska kvality produkovaných výsledků.

Příkladem aplikací zpracovávajících obraz jsou například dohledové nebo bezpečnostní systémy. Základní operací, která se provádí u většiny uvedených systémů, je detekce (rozpoznání) objektů. Tato operace je velmi důležitá a je jí v mnoha pracích věnována velká pozornost. Detekce je jednou z prvotních operací při zpracování obrazu a na jejím výsledku přímo závisí výsledná přesnost a spolehlivost celého systému. Pracuje-li detekce objektů příliš pomalu nebo s příliš velkou chybou, tak se dohledový systém dozví o detekované události pozdě nebo dokonce nikdy v případě chyby.

V aplikacích a systémech zpracovávající obrazová data je kladen velmi vysoký důraz na zpracování v reálném čase, to tak vytváří vysoké požadavky na klasifikační jednotky. Takzvané následné (odložené) zpracování obrazu ve výpočetním centru není vhodné pro aplikace, které musejí pracovat v reálném čase a jejichž výsledky zpracování velmi rychle zastarávají. Data u takových aplikací se musejí zpracovat ihned, jakmile jsou dostupná; odklad zpracování je nepřípustný. V aplikacích, kde se neustále generuje velké množství obrazových dat, není zase vhodné je ukládat a zpracovávat až po čase, jelikož po zaplnění vyrovnávacích pamětí dojde k zahlcení a následně ke ztrátě cenných dat. Zpracování obrazu v reálném čase je tak velmi aktuální téma pro další vývoj.

Detekce objektů se provádí pomocí obrazových klasifikátorů. Velmi dobře známými a také velmi často používanými metodami jsou metody založené na *boostingu* [68], jako je například AdaBoost [9] nebo Waldboost [71]. AdaBoost dosahuje velmi dobrých klasifikačních výsledků a jeho vyhodnocení je možné paralelizovat, což je velmi důležité při implementaci na výpočetních architekturách poskytujících velkou míru paralelizmu. Uvedené univerzální klasifikační metody však vyžadují pro svůj běh velmi vysoký výpočetní výkon. A i přesto, že se v posledních několika desetiletích dočkala počítačová technika velkého rozvoje a výpočetní výkon je neustále vysokým tempem navyšován, není stále dostatečný k tomu, abychom se nemuseli zabývat optimalizací výpočetních metod.

Od uvedení metody AdaBoost v roce 1995 již bylo představeno mnoho různých implementací, ale také vylepšení této metody. Doposud představené architektury byly implementovány na různorodých výpočetních platformách zahrnující programovatelná hradlová pole (FPGA - *Field-Programmable Gate Arrays*), univerzální procesory (CPU - *Central Processing Unit*), grafické procesory (GPU - *Graphical Processing Unit*) a procesory se specializovanými vektorovými instrukcemi (SIMD – *Single Instruction Multiple Data*). Rychlost zpracování dat uvedených architektur je však limitována jen na desítky megapixelů za sekundu (Mpps). Doposud představené implementace jsou tak schopny zpracovávat vstupní obraz s relativně malým rozlišením a vysokým počtem snímků za sekundu (to je *640 x 480* pixelů při *160* snímcích za sekundu [96]) nebo vstupní obraz s vysokým rozlišením obrazu, ale nízkým počtem snímků za sekundu (to je *1920 x 1200* pixelů při *4* snímcích za sekundu). Taková rychlost je nepostačující vzhledem k dnes vyráběným a vyvíjeným obrazovým snímačům. Ty dokážou poskytovat data ve velmi vysokém rozlišení a současně s velkým počtem snímků (běžné rozlišení dnešních obrazových snímačů je až v desítkách megapixelů a poskytovaný počet snímků je v desítkách snímků za sekundu). Dnešní obrazové snímače tak vyžadují zpracování velkého množství dat, které je téměř o řád výše, než umožňují zpracovat doposud představené architektury.

Pro účely zpracování obrazu v reálném čase se jeví jako velmi perspektivní použití FPGA technologie pro její velmi vysokou schopnost paralelizace. Takovou možnost nabízí CPU jen omezeně pomocí vektorových instrukcí či běhu algoritmu na více jádrech či procesorech. U GPU jednotek je sice dostupné velké množství paralelně pracujících jednotek, ale ty jsou univerzální a jejich propojení je pevně definováno. Nelze je tak plně přizpůsobit pro jeden konkrétní druh operace. FPGA tak poskytuje možnost, jak vytvořit vlastní výpočetní procesor či ko-procesor, který může využívat velké míry paralelizmu a může být velmi přizpůsoben pro řešení aktuálního problému.

Doposud představené implementace pro detekci objektů v obraze na FPGA se, ať už více či méně, drží sekvenčního způsobu vyhodnocení metody. Pro dodržení sekvenčního schématu vyhodnocení mají architektury řadu omezení, jež je však možné odstranit. Navíc se implementace snaží ve většině případů vytvořit obecný klasifikátor, jenž je možné použit pro různé klasifikační úlohy, jako je například detekce obličejů nebo detekce vozidel a podobně. Ačkoliv se známé implementace prezentují jako univerzální jednotky, tak pouhá změna rozlišení vstupního obrazu u většiny jednotek vede minimálně ke změně parametrů generického návrhu nebo v horších případech dokonce i ke změně architektury samotné a tím plynoucí velké lidské práce. V obou případech je nutno provést opakovanou syntézu architektury pro FPGA a rekonfiguraci výsledného produktu.

Implementace univerzálních algoritmů poskytuje výhody především při zpracování na CPU, kde se zpracovává velké množství různých typů úloh. Pokud je však zpracování umístěno do FPGA, tak ve velkém množství případů průmyslových aplikací slouží jednotka jen pro zpracování jednoho typu úlohy (například detekce státních poznávacích značek). Pro uvedené aplikace je tak neopodstatněné mít v FPGA možnost změny klasifikační úlohy v krátkém čase (milisekundy). Zcela postačuje možnost změny klasifikátoru jen při servisním zásahu, který může trvat i několik minut a perioda změny může být i několik měsíců. Taková změna je poté provedena například rekonfigurací firmware v FPGA obvodu.

Univerzální algoritmy mají zpravidla menší přesnost a potřebují větší množství prostředků pro svůj běh v porovnání s aplikačně specifickými algoritmy. Na základě tohoto poznatku je možné předpokládat, že vytvoří-li se aplikačně specifická implementace pro FPGA, může dosahovat lepších výsledků než univerzální implementace. Práce prezentuje přístup, pomocí kterého se vytvářejí implementace klasifikátorů optimalizované jen na jednu úlohu, kterou se rozumí zpracování obrazu s pevně daným rozlišením, detekce pevně daného typu předmětu a podobně. Takový klasifikátor poté může vykazovat lepší vlastnosti než klasifikátor založený na univerzálním přístupu.

Návrh každého takového klasifikátoru samostatně by byl velmi složitý. Proto je v práci uvedena metoda jak takový klasifikátor navrhovat automatizovaně. Pro koncového uživatele, návrháře, se tak metoda jeví jako univerzální přístup. Změna klasifikátoru je provedena jen pouhou změnou vstupních parametrů a návrhář nemusí vytvářet žádný nový popis klasifikátoru pro FPGA technologii. Návrhář v uvedeném případě pracuje s nástrojem, který je vytvořen ve vyšším programovacím jazyce.

Celková spotřeba energií na zemi neustále roste a s rozvojem výpočetních jednotek je stále více a více věnována pozornost efektivitě výpočtu z hlediska jejich energetických nároků. O důležitosti tohoto trendu svědčí i řada organizací, které byly vytvořeny za účelem sledování a porovnávání efektivity výpočtů. Jednou z nejvýznamnějších organizací je Green 500 [12], která monitoruje, kolik operací jsou schopné systémy zpracovat na jeden spotřebovaný watt elektrické energie. Organizace je sice zaměřena především na supervýkonné výpočetní stroje, ale z dat jí poskytovaných lze sledovat trend a přenést požadavky i na obecnou implementaci algoritmů. Detekce objektů zpravidla konzumuje velké množství výpočetního výkonu, který následně vede i k velkému množství spotřebované energie. Provádění výpočtů na standardních procesorech CPU je velmi energeticky náročné (příkon procesoru je desítky nebo dokonce až stovky wattů). Na druhé straně provede-li se

vhodné rozdělení nebo přemístění algoritmu na jiné, efektivnější, výpočetní jednotky, jako je například FPGA, lze dosáhnout výrazné úspory energií. Většina doposud publikovaných prací se zabývala jen klasifikační přesností nebo rychlostí výpočtu, ale s ohledem na energii je nutné také začít řešit u algoritmů a jejich implementací i efektivitu z hlediska energetických nároků (příkon). V práci je uvedeno, jak lze vhodným rozdělením algoritmu dosáhnout výrazné úspory energií.

Implementace algoritmů, které jsou navrženy jen s ohledem na maximální výkonnost, jsou sice potřebné, ale z mnoha hledisek mohou být příliš omezující. Mnoho aplikací nepotřebuje mít implementován dokonalý algoritmus v jednom ohledu, ale potřebují dosáhnout rozumný kompromis mezi výkonností, spotřebou a přesností algoritmu. Možnost měnit vlastnosti operativně a rychle mezi uvedenými kritérii je velmi žádoucí. Aplikačně specifický přístup k implementaci prezentovaný v této práci ukazuje techniku, jak tyto potřeby velmi efektivně splnit.

V poslední době vzhledem k neustále se zlepšujícím obrazovým snímačům, které stále zlepšují své parametry, jako je poskytované rozlišení obrazu a poskytované množství snímků za jednu sekundu, vyvstává problém přenosu dat z obrazového snímače do systému pro zpracování dat. Například v dohledových systémech nebo systémech monitorujících dopravní situaci je velmi rozšířeno použití levné komunikační infrastruktury ethernet pracující na přenosové rychlosti *100* Mbs^{-1} nebo *1* Gbs^{-1} (viz [1] a [2]). Takto omezená infrastruktura není vhodná pro přenos nezpracovaných dat z obrazových snímačů s vysokým datovým tokem (vysoké rozlišení a vysoký počet snímků) v reálném čase, jelikož jeden obrazový senzor s rozlišením obrazu *1920 x 1200* pixelů (FullHD – *Full High Definition*) a s *50* snímky za sekundu produkuje datový tok *921* Mbs^{-1} pro obraz ve stupních šedi. Pouhá jedna kamera tak plně vytíží *1* Gbs^{-1} linku. Samozřejmě je možné pro přenos dat použít *10* Gbs^{-1} linku, ta je však poměrně drahá pro použití v reálných aplikacích a problém přenosu dat reálně neřeší, ale jen posunuje limity a dovolí přenést větší datový tok. Velmi často se nepřenáší nezpracovaná obrazová data, ale používají se kompresní nástroje pro zmenšení toku dat. Dnes jsou velmi oblíbené kodeky[3] H.264 [85] a H.265 [34]. Uvedené kodeky snižují množství přenášených dat, ale data stále musí být zpracována v reálném čase, to znamená komprimována na straně odesílatele a dekomprimována a zpracovávána na straně přijímače. Do celého procesu přenosu dat jsou tak navíc zařazeny dvě operace, které ve většině případů vedou k ztrátě informace a jsou tak pro řadu aplikací neakceptovatelné. U většiny aplikací je převážná část přenášených dat po úvodním zpracování zahozena, jelikož nenesou potřebnou informaci. Proto se nabízí jiné řešení, jak provést redukci přenášených dat, kterým je přenos pouze zájmových dat, to je přenosu pouze snímků, které obsahují například hledané objekty, nebo snímků, u kterých je vysoká pravděpodobnost, že je obsahují. Přenos dat po komunikační lince tak může být výrazně redukován a nemusí docházet ke ztrátě informace.

Od uvedení algoritmů AdaBoost, či WaldBoost byla představena řada metod, které poskytují lepší výsledky z různých hledisek. Jednou z metod, která produkuje lepší výsledky z hlediska klasifikační přesnosti, je například konvoluční neuronová síť CNN (convolutional neural network) [50]. Tato metoda je však výpočetně velmi náročná a velmi obtížně s ní lze dosáhnout vysoké datové propustnosti. Pokud se však provede zkombinování například metody AdaBoost a CNN dostaneme takzvanou hybridní architekturu, která může mít výhody obou řešení, což je vysoká datová propustnost (AdaBoost v FPGA) a dobrá klasifikační přesnost (CNN). Nová architektura může vypadat tak, že malá ale velmi rychlá předzpracovávající jednotka založená na AdaBoost metodě (například implementována v FPGA) bude provádět filtraci zájmových oblastí a pouze před vybrané oblasti

[1] Mbs^{-1} – Mega bitů za sekundu

[2] Gbs^{-1} – Giga bitů za sekundu

[3] Kodek – nástroj pro kompresi a dekompresi obrazových dat

budou zpracovány pomocí CNN (například na CPU). Tím tak bude dosaženo výhod z obou řešení. Metoda AdaBoost tak díky vhodnosti pro implementaci v FPGA a tím získané vysoké datové propustnosti nalezne uplatnění i v moderních systémech.

Práce [4] představuje výsledky výzkumu za posledních deset let z hlediska pokroku klasifikace. Autoři uvádějí, že největší pokrok v přesnosti klasifikace byl udělán pomocí vývoje nových obrazových operátorů (příznaků). Užití metody pro návrh aplikačně specifického klasifikátoru přímo vybízí i k použití aplikačně specifických obrazových operátorů, které budou vždy navrženy pro danou aplikační úlohu. V práci je uveden přístup, jak za pomocí genetických algoritmů takové operátory automatizovaně navrhovat a uzpůsobovat je pro běh v FPGA.

1.1 Cíle práce

Primárním cílem této disertační práce je výzkum v oblasti tvorby metodiky a architektur pro automatizovaný návrh aplikačně specifických obrazových klasifikátorů. Pro práci byly stanoveny následující cíle a podcíle:

Prvním dílčím cílem je výzkum v oblasti obrazových operátorů. V rámci tohoto cíle bude ověřeno, že pomocí aplikačně specifických příznaků navržených pro konkrétní úlohu je možné dosáhnout zpřesnění výsledků klasifikace. Dále bude ověřena možnost vytváření příznaků optimalizovaných pro implementaci v FPGA.

Druhým dílčím cílem práce je ověřit, že aplikačně specifické klasifikátory mohou dosahovat lepších parametrů než obecně sestavené klasifikátory, a to především z hlediska rychlosti zpracování dat. V rámci tohoto cíle bude navržena nová architektura pro obrazové klasifikátory, jejíž konkrétní podoba bude závislá na typu aktuální řešené úlohy a budou ověřeny její parametry. Architektura bude navržena tak, aby umožňovala vytváření klasifikačních jednotek podle aktuálních potřeb návrháře, a to především z hlediska možnosti volby mezi klasifikační přesností, zdrojů potřebných pro implementaci, rychlosti zpracování a množství spotřebované energie. Architektura bude též navržena tak, aby ji bylo možné rozdělit na více výpočetních jednotek.

Třetím dílčím cílem je odstranění sekvenčního charakteru zpracování z obrazového klasifikátoru a navržení plně proudové architektury, která bude mít konstantní rychlost zpracování dat, tedy propustnost architektury bude necitlivá na obsah dat a bude mít zaručenou propustnost.

Čtvrtým dílčím cílem je ověření možnosti vytvoření nástroje, který bude automatizovaně vytvářet aplikačně specifické klasifikátory na základě popisu na vysoké úrovni abstrakce a přenést tak generičnost algoritmu z jazyků pro popis hardware do jazyků vyšších úrovní.

V práci bude postupně ukázáno splnění jednotlivých cílů a celková funkčnost nově navržené architektury bude demonstrována na reálné aplikaci.

1.2 Struktura práce

Práce je členěna do sedmi kapitol. Druhá kapitola se zabývá detekcí objektů z obecného hlediska a jsou v ní představeny jednotlivé části, ze kterých se skládá klasifikátor. Nejprve je popsána extrakce informace z obrazu a její transformace do vektoru příznaků. Následně jsou uvedeny jednotlivé obrazové operátory, které se používají pro extrakci informace z obrazu. Nejdůležitějšími operátory z hlediska práce jsou *Local Binary Patterns* (LBP), ale jsou uvedeny i jiné operátory, které jsou relevantní k práci, jako jsou například široce používané Haarovy vlnky. Po představení operátorů je

popsáno vytvoření slabého klasifikátoru z operátorů a následně i vytvoření celého silného klasifikátoru na základě množiny slabých klasifikátorů. Pro vytvoření silného klasifikátoru jsou představeny metody založené na takzvaném *boostingu* a zvláštní pozornost je věnována především metodě AdaBoost, na které je práce založena. V závěru kapitoly jsou uvedeny důvody, proč byla zvolena právě metoda AdaBoost.

Ve třetí kapitole je uveden přehled známých implementací *boostingových* metod na různých výpočetních platformách. První část kapitoly se zabývá vyhodnocením algoritmu na standardních CPU, případně na CPU s využitím vektorových instrukcí. Následně jsou uvedeny výsledky dosažené pro implementace využívající CPU ve spojení s grafickým procesorem (GPU). Dosažené výsledky těchto prací jsou uvedeny především pro porovnání výkonnosti s FPGA implementacemi. V podkapitole 3.3 jsou uvedeny vybrané implementace na FPGA, které jsou zajímavé buď přístupem k řešení problému anebo dosahovanou rychlostí. Ke každé architektuře je uveden její stručný popis a dosahované výsledky. Na závěr kapitoly je uvedeno porovnání výkonnosti doposud známých implementací, které ukazuje dosavadní limity z hlediska rychlosti zpracování.

Čtvrtá kapitola popisuje způsob dosažení prvního dílčího cíle – návrh nových aplikačně specifických tvarů obrazových operátorů. V úvodu kapitoly je popsána metodika návrhu nových tvarů operátorů a je vybrána metoda pro řešení problému – genetický algoritmus. Následně jsou rozebrány nejdůležitější části metody, jako je například tvorba fitness funkce. Po představení metody jsou prezentovány dosažené výsledky.

V páté kapitole je popsáno jádro AdaBoost klasifikátoru. Jelikož však pracuje na zcela odlišném principu než ostatní doposud představené metody, je na počátku kapitoly uveden způsob práce nově navrhnuté architektury klasifikátoru. Následně je uveden popis samotné klasifikační jednotky, jejíž výsledná podoba je vždy unikátní pro každý klasifikátor. Koncepce klasifikátoru je založena na paralelním zpracování všech obsažených slabých klasifikátorů v jednom kroku. Tato kapitola ukazuje první část řešení třetího dílčího cíle práce. Po představení jednotlivých částí, ze kterých se jednotka skládá, jsou diskutovány její vlastnosti. Následně se kapitola zabývá automatickou syntézou klasifikátoru a ukazuje první část řešení čtvrtého dílčího cíle práce. V kapitole je uveden iterační proces návrhu klasifikátoru, který v několika málo krocích provede kompletní návrh klasifikátoru pro danou úlohu. Tento proces provede jednak trénování klasifikátoru, ale také jeho iterační sestavení a přizpůsobení vzhledem k zadanému FPGA. Poslední část kapitoly se zabývá dekompozicí klasifikátoru na více výpočetních jednotek za účelem snížení příkonu a zvýšení datové propustnosti klasifikátoru.

Šestá kapitola ukazuje splnění hlavního cíle práce, kterým je návrh metodiky a výpočetní architektury kompletního aplikačně specifického klasifikátoru v FPGA. Jako první je v kapitole ilustrován sekvenčně paralelní klasifikátor, který je použit jako referenční implementace a následně je popsán multiparalelní detektor objektů pracující v reálném čase. Je představena architektura, která využívá několik stupňů paralelizace a poskytuje velmi rychlé zpracování dat. Následně jsou uvedeny její nároky na implementaci vzhledem k FPGA zdrojům, možnosti její dekompozice, aplikace a experimentální ověření klasifikátoru. V závěru kapitoly je uvedena druhá část třetího a čtvrtého cíle práce a je provedeno shrnutí dosažených výsledků.

V závěrečné kapitole jsou zhodnoceny dosažené výsledky, splnění vytyčených cílů práce a jsou diskutovány další možnosti pokračování v práci.

2 Detekce objektů

Detekce objektů je nedílnou součástí každodenního života. Lidský mozek ji provádí automaticky, aniž bychom si to uvědomovali. A je to právě detekce objektů, která člověku pomáhá orientovat se v běžném životě. Na základě detekce a rozpoznání objektů dokáže lidský mozek řídit své chování a přizpůsobovat jej měnícím se okolním podmínkám. Stejně tak je i detekce objektů velmi důležitá pro systémy strojového vidění, které se snaží pomocí algoritmů a výpočetních prostředků simulovat právě chování mozku. Detekci objektů nebo také klasifikaci objektů lze popsat následovně. Běžný člověk dnešní doby dokáže velmi snadno identifikovat například osobní vozidlo, i když každé osobní vozidlo vypadá mírně odlišně. To je dáno především tím, že i když vozidla vypadají odlišně, tak stále mají stejné základní rysy. Pokud se na vozidlo podíváme zepředu, tak zjistíme, že má dvě světla v krajích karoserie, že má přední masku s otvory pro větrání, dále pak významným prvkem je kapota, čelní sklo nebo státní poznávací značka (SPZ). Na základě těchto charakteristických rysů dokáže člověk rozpoznat objekt a říci, že se jedná o vozidlo. Lidský mozek však sám o sobě tuto činnost provádět neumí. Schopnost rozpoznání určitého typu předmětu mu byla dána na základě učení. A i přesto mozek neprodukuje diskrétní výstup, ale své výsledky produkuje ve vágní logice, a teprve další rozhodovací krok určí, zda se opravdu jedná o objekt daného typu nebo ne.

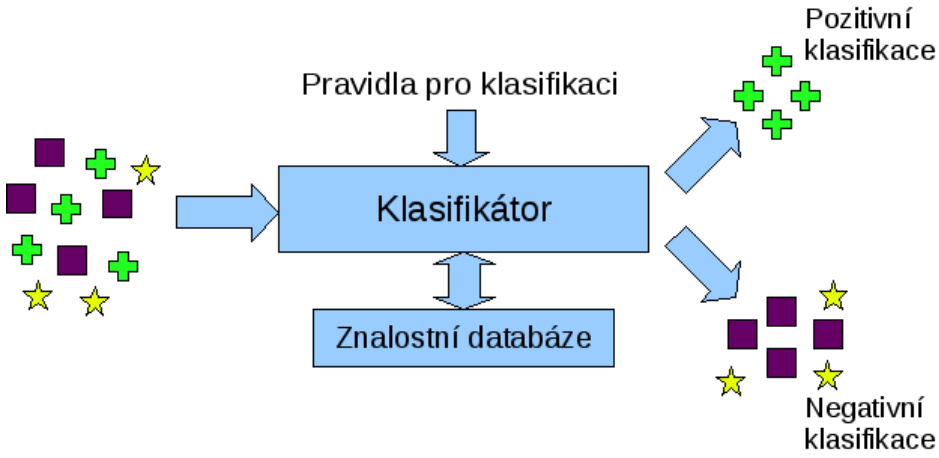
Výpočetní technika a algoritmy pro detekci objektů v mnoha ohledech pracují velmi podobně jako lidský mozek. Například metoda pro detekci čelního pohledu na auto musí být nejdříve naučena, nebo sestavena – to lze srovnat s procesem učení lidského mozku. Následně sestavený algoritmus pak zpravidla pracuje tak, že hledá ve vstupním obraze význačné body, které jsou společné pro daný typ objektů. Srovnáme-li činnost mozku s počítačovým algoritmem, tak zjistíme, že i on se naučí rozpoznávat například světla automobilu. Algoritmus produkuje pravděpodobnostní výstup, který stanovuje, s jakou mírou nejistoty se vyskytuje v obraze hledaný objekt. S použitím rozhodovacího prahu lze poté tento výstup snadno převést na diskrétní hodnotu – ANO či NE.

Mozek je velmi složitý výpočetní stroj, který poskytuje velmi vysoký výpočetní výkon a umožňuje provádět velké množství operací paralelně. Proto mozek v jeden okamžik může provádět jednak detekci objetu jednoho typu, ale také detekci objektů několika typů současně. Kdežto obrazové detektory velmi často klasifikují jen do jedné třídy, a pokud se má provést klasifikace do více tříd, je nutné buď klasifikátor upravit, nebo provádět současně klasifikaci do více tříd.

2.1 Detektor objektů

Detektorem objektů v rámci této práce chápeme stroj nebo strojový kód (případně program), který provádí rozdělování objektů, jež jsou mu dány na vstup do dvou výstupních tříd. V této práci se neuvažují metody, které dokážou klasifikovat objekty do více tříd. První výstupní třída je tvořena hledanými objekty a do druhé třídy spadají všechny zbývající objekty. Tato práce se dále zabývá pouze detektory objektů, které pracují na základě dvoudimenzionálního vstupu dat. Uvedený vstup představuje rastrový obraz, který je poskytován běžným obrazovým snímačem. Výstupem obrazových detektorů objektů je obvykle množina obdélníků, které obalují detekované předměty.

Obrázek 1 zobrazuje základní schéma funkce klasifikátoru. Uvedený klasifikátor využívá ke své práci znalostní databázi a takzvaná pravidla pro klasifikaci. Na vstup klasifikátoru jsou přivedena obrazová data, která obsahují 3 typy obrázků (čtverec, křížek a hvězdu). Klasifikátor je sestaven tak, aby dokázal rozpoznat křížky. Na výstupu klasifikátoru jsou tedy křížky dávány do takzvané množiny pozitivních obrázků a všechny ostatní obrázky obsahující jiné předměty jsou umísťovány do množiny negativních obrázků. Pravidla pro klasifikaci jsou obvykle dána použitou metodou – to znamená, že byla určena expertem. Znalostní databáze může být určena buď expertem nebo získána procesem učení (viz následující podkapitola).



Obrázek 1: Ilustrace práce klasifikátoru.

Algoritmy pro detekci objektů mohou být dle práce autorů Yang a kolektiv [94] rozděleny na čtyři základní skupiny:

- **Znalostně založené algoritmy** (*knowledge-based*) – jsou založeny na pravidlech, které jsou určeny expertem – člověkem.
- **Příznakově invariantní** (*feature invariant*) – algoritmy se snaží naleznout příznaky, které jsou invariantní vůči vnějším vlivům, jako je například osvětlení nebo vůči rotaci.
- **Vyhledávání vzorů** (*template matching*) – algoritmy jsou založeny na výpočtu shody (korelace) mezi testovaným objektem a vzorovým objektem.
- **Vzhledově založené** (*appereance based*) – jsou založeny na technice strojového učení, kdy se během učení snaží extrahovat množinu diskriminativních (popisných) příznaků na základě předem známé množiny vstupních dat

Tato práce se zabývá poslední skupinou uvedených metod - vzhledově založené. Tyto metody jsou zajímavé především z toho hlediska, že jedna metoda může být obecně použita pro detekci různých druhů objektů. Pro každý typ objektu však musí být natrénována nová znalostní databáze, dle které se bude metoda řídit.

Na detektor objektů lze zjednodušeně pohlížet jako na kombinaci dvou základních stavebních bloků. Prvním je extrakce příznaků, která provádí překódování obrazové informace do podoby, se kterou se klasifikátoru lépe pracuje. A druhým blokem je klasifikátor samotný, který na základě vektoru příznaků rozhoduje o výsledku klasifikace.

2.2 Vektor příznaků

Reálný obraz, tak jak jej vnímá lidské oko, se zpravidla neukládá do paměti počítače. Oko má totiž velmi vysokou rozlišovací schopnost a uložení takového obrazu není snadné provést v paměti počítače, jelikož by k tomu bylo třeba velké množství paměťového prostoru. V paměti počítače se proto ukládá diskretizovaný obraz. Úrovní rozlišení ukládaného obrazu se stanoví, jaké množství dat, ale hlavně jaké množství informace (detailů) v obraze se bude ukládat do paměti počítače. Již tuto prvotní transformaci reálného obrazu lze považovat za extrakci vektoru příznaků. Celý rastrový obraz je poté považován za vektor příznaků. Jeden pixel je tedy jeden příznak.

Takový vektor příznaků je však stále ještě velmi obsáhlý a práce s ním vyžaduje velmi vysoký výpočetní výkon a také vyžaduje velkou paměť. Proto pro účely detekce objektů se s rastrovým obrazem provádí takzvaná operace extrakce příznaků. Pro tuto operaci byly vytvořeny obrazové operátory, které jsou uvedeny v podkapitole 2.4. Aplikací těchto operátorů na vstupní obraz se provede extrakce nejdůležitějších prvků v obraze do vektoru příznaků. V něm je tak koncentrovaná obrazová informace a jeho úkolem je co nejlépe popsat obsah dat v obraze pro danou aplikaci. Na základě těchto dat se následně provádí další zpracování obrazu. To, jaké příznaky se budou extrahovat, vždy záleží na daném typu aplikace. Pro detekci obličejů se zřejmě bude extrahovat jiná množina příznaků, než pro detekci čelního pohledu na vozidlo. Zvolení vhodné množiny operací pro extrakci příznaků je klíčové a velmi ovlivňuje funkci výsledného klasifikátoru.

V aplikacích počítačového vidění se rozlišují dvě základní skupiny příznaků – globální a lokální příznaky. Globální příznaky se pokouší popsat celý obraz a obecně nejsou příliš robustní a malá byť i nepodstatná změna v obraze je může ovlivnit natolik, že příznak nebude pracovat správně. Zatímco lokální příznaky popisují menší regiony v obraze a jsou obecně vhodnější pro rozpoznávání objektů v obraze. Malá změna v obraze pak ovlivní pouze malou skupinu příznaků, do jejichž oblastí změna přímo zasahuje.

Vhodné příznaky pro detektory objektů musí vykazovat žádané vlastnosti. Nejdůležitější vlastností je invariantnost (necitlivost) vůči jistým obrazovým změnám, jako je například posun objektu, velikost objetu, rotace objektu, osvětlení scény a geometrické vady [3], [54]. Operátory se nazývají invariantní, pokud jejich výsledek není ovlivněn uvedenými změnami. Ve skutečnosti není žádný příznak invariantní vůči všem uvedeným faktorům, jednotlivé příznaky jsou více či méně citlivé na některé změny. Necitlivost vzhledem ke změnám osvětlení je často prováděna pomocí normalizace obrazu. Invariance pro posun, rotaci a změnu velikosti objektu je často prováděna pomocí průměrování přes okolní pixely.

Práce autorů Benson a kolektiv [4] ukazuje, že největší posun z hlediska vylepšení detekčních schopností klasifikátorů byl udělán právě pomocí vylepšení extraktorů příznaků. Vytvoření nové množiny příznaků tak může být mnohem přínosnější než vytvoření nové klasifikační metody nebo vylepšení procesu učení klasifikátoru. Jedním z příkladů velmi významného vylepšení extrakce příznaků je použit v konvolučních neuronových sítích CNN (convolutional neural network) [50], [66], [98], kde je použito několika konvolučních vrstev pro zpracování výstupu ze základních příznaků. Takto extrahované příznaky jsou poté vstupem několikavrstvé neuronové sítě. CNN dosahuje velmi dobrých výsledků v úloze detekce objektů.

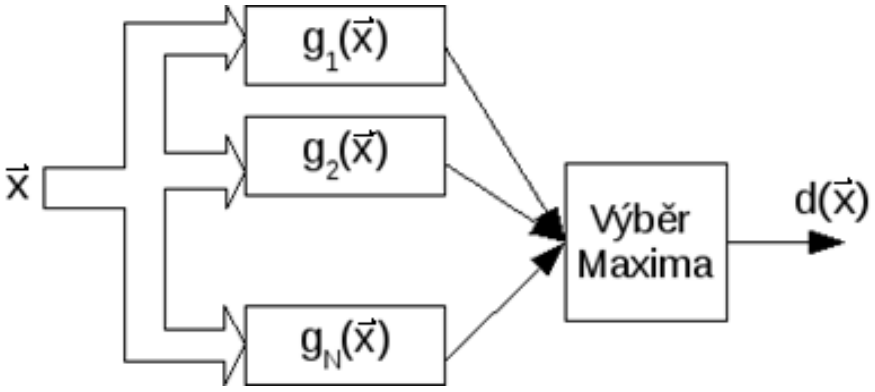
2.3 Znalostní databáze klasifikátoru

Detektor objektů (klasifikátor) obvykle pracuje na základě znalostní databáze. Taková databáze však musí být nejprve vytvořena. Základní možností je vytvoření znalostní databáze na základě znalostí experta nebo vytvoření pomocí automatizovaného procesu učení (trénování). V první variantě se znalosti experta přímo přenesou do klasifikátoru a ten na jejich základě provádí vyhodnocení. Takové pravidla jsou do klasifikátoru přenesena přímo (jsou v klasifikátoru takzvaně zakódována) a výsledný klasifikátor poté slouží pro klasifikaci jednoho typu předmětu – jednoúčelový klasifikátor. Jednoduchým příkladem takového klasifikátoru je detektor obličejů v obraze, který pracuje jen na základě barevného schématu obrazu a detekuje obličeje na základě barvy lidské kůže. Pravidla takového klasifikátoru nelze přenést na zcela jiný typ úlohy.

Vstup klasifikátoru je zpravidla tvořen vektorem příznaků, jež se extrahují ze vstupních dat. Vektor příznaků může být zapsán jako:

$$\vec{x} = (x_1, x_2, x_3, \dots x_N) \tag{2.1}$$

Kde x_i definuje hodnotu příznaku a N je počet příznaků. Typický klasifikátor je poté tvořen několika diskriminačními funkcemi $g(\vec{x})$, kde $\vec{x}$ je vektor příznaků. Výstupem diskriminační funkce je vyjádření míry, jak mnoho zkoumaný předmět patří do třídy, jež je dána diskriminační funkcí (v případě rozpoznání obličeje na základě barvy vyjadřuje například vzdálenost dvou barev). Je-li klasifikátor tvořen pomocí množiny diskriminačních funkcí (například klasifikace do více tříd nebo více kritérií pro jednu třídu), pak je zkoumaný objekt zařazen do nejbližší třídy dle výsledků diskriminačních funkcí. Příklad takového klasifikátoru je na obrázku 2, výstupní třída je označena jako $d(\vec{x})$. Do diskriminačních funkcí $g(\vec{x})$ je napevno přenesena veškerá logika a její změna je obtížně proveditelná.

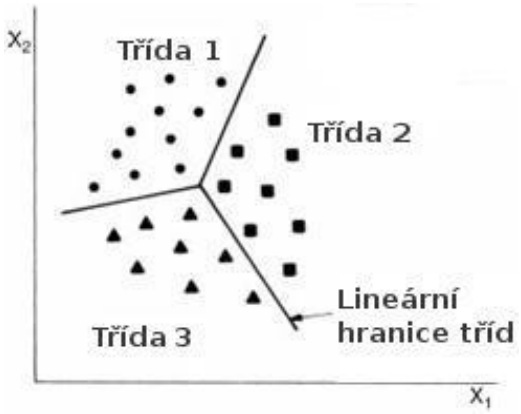


Obrázek 2: Diskriminační funkce klasifikátoru.

Druhým přístupem pro vytvoření znalostní databáze klasifikátoru je využití algoritmu, který ji vytvoří na základě poskytnutých dat – trénovací proces. Ten tak nahrazuje experta (člověka). Výhodou tohoto procesu je, že se dá využít opakovaně a sestavit tak vícero různých klasifikátorů pro různé druhy objektů. Znalosti experta sice nejsou využity pro tvorbu klasifikátoru, ale jsou využity pro sestavení procesu trénování a sestavení obecné diskriminační funkce. Taková funkce poté pracuje jednak se vstupními daty pro klasifikaci, ale také s daty, která řídí její vyhodnocení. Tato data jsou získána v procesu trénování. Metody pro klasifikaci založené na učení se tak skládají ze dvou základních částí. První část se zabývá natrénováním klasifikátoru a druhá část je zaměřena na samotné zpracování dat a rozpoznání objektů.

Učení klasifikátorů je zpravidla složité a časově velmi náročné. Dle typu poskytnutých vstupních dat se učení může rozdělovat na učení bez učitele a učení s učitelem. Oba druhy učení předpokládají na svém vstupu data. Ale při využití metody učení bez učitele se nepředpokládá žádná znalost obsahu dat. Kdežto při použití metody s učitelem se předpokládá znalost obsahu dat.

Pro učení bez učitele jsou vstupní data tvořena množinou $X = \{x_1, x_2, x_3, \dots x_N\}$, kde x_N je jeden vstupní vzorek dat. Dalším vstupem procesu trénování je například definice funkce blízkosti dat. Tato funkce $c(x_N,x_M)$ definuje vzdálenost dvou vzorků dat ze vstupní množiny. Na základě funkce blízkosti se pokusí proces trénování naleznout rozdělení vstupní množiny do několika výstupních tříd (viz obrázek 3). Tento proces však nemusí vždy naleznout vhodné rozdělení prostoru. Výsledek je závislý především na separovatelnosti vstupních dat dle použitých kritérií. Zástupcem této metody je například shluková analýza K-Means [56]. Učení bez učitele je náročné a probíhá v několika iteracích. Tento způsob učení není možno použít ve všech případech.



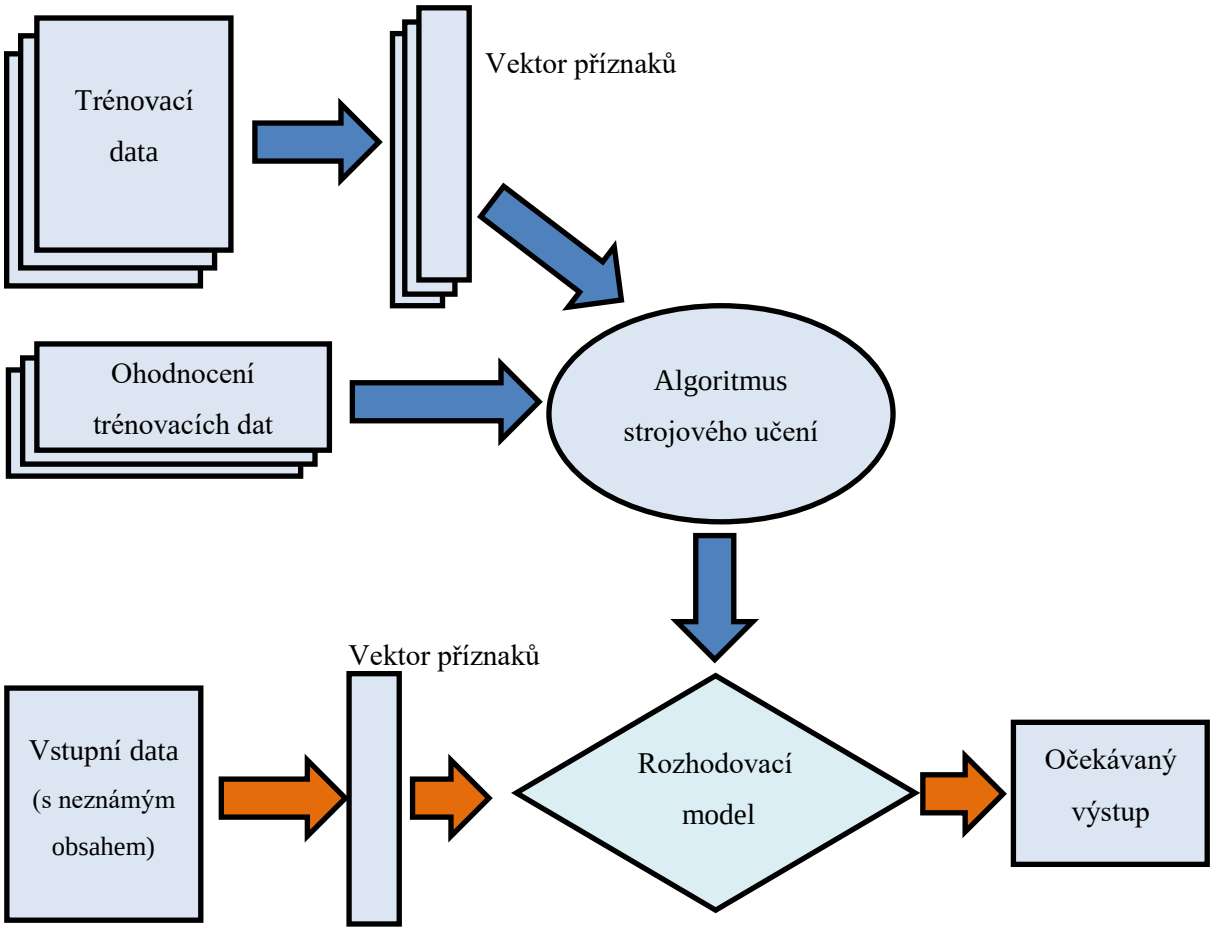
Obrázek 3: Rozdělení objektů to tříd na základě jejich blízkosti.

Druhou metodou učení je takzvané učení s učitelem. Tato metoda využívá vstupní data ve formátu množiny dvojic $X = \{(x_1, Y_1), (x_2, Y_2), ..(x_N, Y_N)\}$, kde x_N je vstupní vzorek dat a Y_N je zařazení dat do výstupní množiny. Proces trénování tak má o každém vstupním vzorku znalost, do které výstupní množiny patří. Během učení se tak proces trénování snaží obvykle iterativně naleznout diskriminační funkci či funkce a po každé iteraci může sám provést ohodnocení dosažených výsledků pro danou iteraci. Proces učení je zpravidla ukončen, nalezne-li se diskriminační funkce, která rozděluje vstupní objekty do výstupních tříd s dostatečnou přesností. Správné ukončení procesu učení je důležité také z důvodu možného přeučení klasifikátoru, které spočívá v tom, že se klasifikátor naučí klasifikovat pouze vzorky dat z trénovací množiny a všechny ostatní vzorky bude považovat za data nepatřící do dané množiny, i pokud budou obsahovat hledaný typ předmětu. Takový stav je nežádoucí a během procesu trénování je třeba mu zabránit. Klasifikátor musí zůstat dostatečně univerzální pro rozpoznávání daného typu objektu. Hlavní předností učících se metod je schopnost automaticky vybrat vhodnou množinu příznaků a na základě ní následně provádět klasifikaci.

Složení a reprezentativnost trénovací množiny je velmi důležitá pro správné natrénování klasifikátoru. To znamená, že množina musí obsahovat dostatečný počet objektů všech možných variací a reprezentací včetně objektů s vadou, které však mají být klasifikovány. Stejně tak musí obsahovat i velkou spoustu objektů, které nepatří do dané množiny. Požadovaná velikost trénovací množiny je v řádu tisíců až statisíců vzorků v závislosti na vybrané metodě.

Výstupem trénování je tedy diskriminační funkce, která dělí *N* rozměrný prostor vstupních dat pomocí *N-1* rozměrné hyperplochy na dvě části. Jedna část obsahuje hledané předměty a druhá část obsahuje všechny ostatní předměty.

Na obrázku 4 je uveden model trénovací a klasifikační části učícího klasifikátoru dle [62]. První část metody se skládá ze sestavení množiny trénovacích vzorků a jejich anotace (ohodnocení). Z množiny vzorků je následně prováděna extrakce příznaků do vektoru příznaků. Ten je poté společně s ohodnocením množiny trénovacích dat vstupem do algoritmu strojového učení, jež na základě poskytnutých dat vystaví rozhodovací model (diskriminační funkci). Horní část obrázku tak odpovídá trénování. Ve spodní části obrázku je uveden klasifikační proces, který využívá dříve sestavený rozhodovací model. Ze vstupních dat je nejprve extrahován vektor příznaků a na jeho základě je rozhodovacím modelem určena výsledná třída objektu.



Obrázek 4: Model trénovací a klasifikační části učícího se algoritmu [62].

2.4 Obrazové operátory

Obrazové operátory slouží pro extrakci příznaků z obrazu a následně k vytvoření celého vektoru příznaků. Na základě obrazových operátorů jsou vystavěny takzvané slabé klasifikátory, jejichž výsledky jsou již přímo použity pro získání konečného výsledku klasifikace. Slabým klasifikátorem může být jakákoliv funkce, která má přesnost klasifikace lepší než náhodná funkce. To znamená, že výsledek klasifikace musí být správný ve více než *50 %* případů. Úspěšnost rozpoznávání jednotlivých

slabých klasifikátorů tak může být poměrně malá, ale jelikož se téměř nikdy nepoužívají samostatně, tak tato skutečnost není na škodu a naopak jí může být využito. Pro získání výsledku klasifikace se tedy používá množina slabých klasifikátorů, jejichž výsledky se zpracovávají pomocí takzvaných silných klasifikátorů (viz podkapitola 2.5).

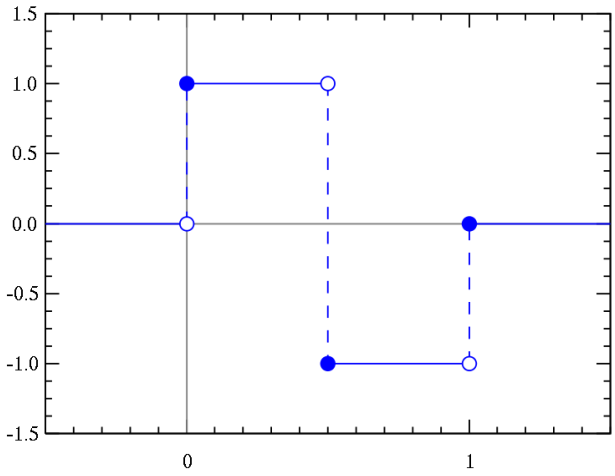
Slabé klasifikátory jsou zpravidla vystavěny nad obrazovým operátorem. Operátory se z hlediska návratové hodnoty dělí na spojité a diskrétní. Spojité operátory mohou navracet libovolnou hodnotu z oboru reálných čísel, kdežto diskrétní příznaky zpravidla vracejí jen omezenou množinu výstupních hodnot (celá čísla či reálná). Příkladem spojitých operátorů jsou například Haarovy vlnky [13] ve 2D nebo Gáborovy vlnky [30] ve 3D. Operátory jsou tvořeny různě velkými konvolucemi (viz dále) ve vybraných místech obrazu. Zástupcem diskrétních příznaků je například LBP [60], LRP [24] a LRD [64]. Tyto příznaky boudu podrobně uvedeny později. Každý slabý klasifikátor je v obraze definován svými rozměry a pozicí.

Zpracování obrazu na základě množiny příznaků je mnohem výhodnější než jen na základě pixelových dat, jelikož příznaky přinášejí řadu výhod, jako je například invariantnost vůči vybraným vlivům (osvětlení), přinášejí vyšší flexibilitu a možnost popsání obecného typu předmětu.

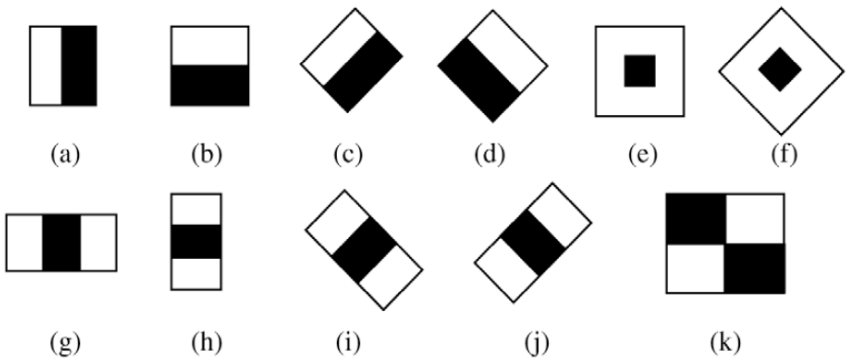
2.4.1 Haarovy vlnky

Jedním z nejpoužívanějších příznaků při tvorbě slabých klasifikátorů jsou právě příznaky založené na Haarových vlnkách. Ty byly prvně představeny autorem Haar v práci [13] v roce 1909 jako matematický operátor. První použití Haarových vlnek pro tvorbu klasifikátorů bylo v práci autorů Viola a Johnes [82], kteří je využili v aplikaci pro detekci obličejů. Haarovy vlnky se v průběhu času ukázaly jako velmi univerzální a byly použity v mnoha dalších pracích.

Haarovy vlnky byly autorem zprvu definovány pro jednorozměrný prostor, jak je ukázáno na obrázku 5. Následně byly Haarovy vlnky rozšířeny i do 2D prostoru, jak je ukázáno na obrázku 6 [51], [86].



Obrázek 5: 1D Haarova vlnka.



Obrázek 6:2D Haarovy vlky [51].

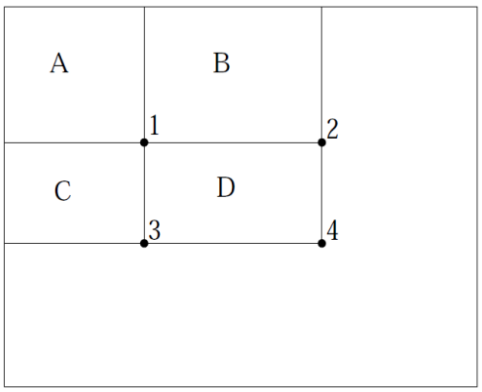
Vyhodnocení příznaků využívajících 2D Haarových vlnek je založeno na rozdílu sum pixelů pod vybranými oblastmi - například bílou a černou oblastí, jak je uvedeno na obrázku 6. Pro efektivní vyhodnocení Haarových příznaků byl vytvořen přístup, který je založen na integrálním obraze. Ten byl prvně představen v práci autorů Viola a Johnes [82]. Integrální obraz je založen na myšlence, že každý pixel integrálního obrazu je tvořen sumou všech pixelů nalevo a nahoru od aktuálního pixelu. Pixely integrálního obrazu jsou definovány vzorcem:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.2}$$

Kde *ii(x, y)* představuje pixel integrálního obrazu a *i(x',y')* představuje pixely původního obrazu. Nad integrálním obrazem je možné velmi efektivně počítat sumy libovolně velkých čtvercových oblastí s konstantní složitostí. Pro výpočet libovolného obdélníku je zapotřebí maximálně čtyř přístupů do paměti a maximálně tří aritmetických operací (dva rozdíly a jeden součet). Na obrázku 7 je uveden příklad výpočtu oblastí čtyř čtverců *A, B, C a D* v integrálním obraze. Dále jsou zobrazeny čtyři významné body *1, 2, 3* a *4*, jejichž hodnoty postačí k výpočtu všech oblastí.

- Oblast *A* – hodnota oblasti se rovná hodnotě bodu *1* v integrálním obraze. K získání hodnoty je tak třeba jen jednoho přístupu do paměti.
- Oblast *B* – hodnota oblasti se vypočte jako rozdíl bodů *2* a *1 (2-1)*.
- Oblast *C* – hodnota oblasti se vypočte jako rozdíl bodů *3* a *1 (3-1)*.
- Oblast *D* – hodnota oblasti se vypočte dle vzorce *4 – 3 – 2 + 1*.

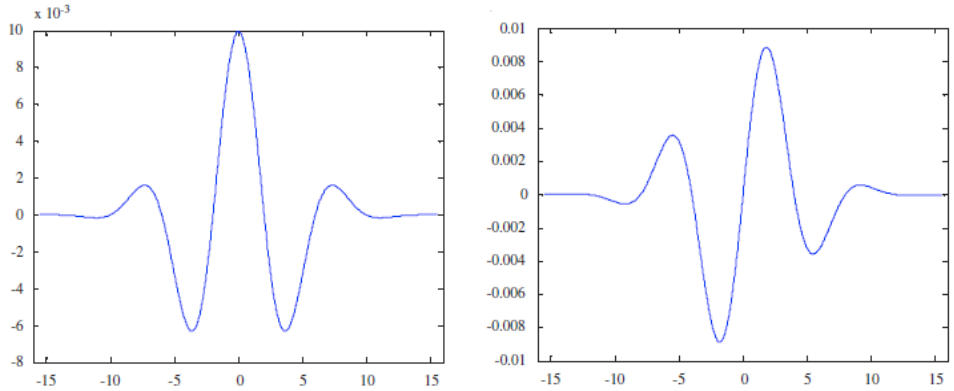
Tímto způsobem lze velmi efektivně počítat Haarovy příznaky.



Obrázek 7: Integrální obraz [82].

Nevýhoda užití slabých N-arních klasifikátorů založených na Haarových vlnkách spočívá především v nutnosti normalizovat intenzitu obrazu před zpracováním (normalizaci není nutné provádět u binárních klasifikátorů s prahem rovným nule). Normalizace se často provádí pomocí derivace obrazu. Na obrázku 6 jsou mimo jiné zobrazeny Haarovy vlnky (c, d, f, i, j), které jsou pootočené. Výpočet takových příznaků však nelze provést za pomoci integrálního obrazu, a tak se většinou nepoužívají.

Autor Lee ve své práci [49] popsal Gaborovy vlnky, jež jsou založeny na odlišné vlnkové bázi. Hlavní rozdíl oproti Haarovým vlnkám je v jejich spojitém průběhu, jak ukazuje obrázek 8. Jejich významnou vlastností je velká schopnost popisu obrazu a velmi dobrý kompromis mezi frekvenčním a prostorovým rozlišením obrazu. Gaborovy vlnky se svým popisným způsobem velmi blíží tomu, jak vnímá obraz člověk pomocí šedé kůry mozkové, a tak se zdá být jejich použití velmi přirozené. Avšak výpočet Gaborových vlnek je velmi výpočetně náročný, tudíž se používají jen v časově nekritických systémech.



Obrázek 8: Gaborovy vlnky v 1D [30].

2.4.2 Local Binary Pattern (LBP)

Local Binary Pattern (LBP) je diskrétní obrazový operátor, který byl představen v prací [60] a později také v [55]. LBP je strukturální operátor pro analýzu obrazu, který převádí obraz ze vstupní reprezentace na výstupní reprezentaci. Pixely výstupní reprezentace obrazu jsou získány tak, že se pro každý pixel vstupního obrazu vezme jeho nejbližší osmi-okolí a všechny hodnoty osmi-okolí se porovnají s hodnotou středového pixelu. Výstupem každé z 8 operací porovnání je binární hodnota *1* nebo *0*. *1,* je-li hodnota pixelu z osmi-okolí větší nebo rovna hodnotě středového (referenčního) pixelu, nebo *0,* je-li hodnota pixelu z osmi-okolí menší než hodnota středového pixelu. Takto se získá *8* hodnot – bitů, které se složí v jednu výslednou *8*bitovou hodnotu pro reprezentaci ve výstupním obraze (výstupní hodnota tak může nabývat až *256* různých stavů).

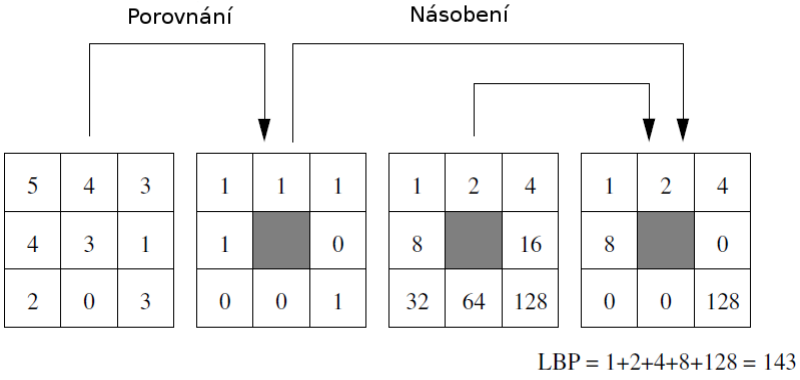
Vyhodnocení LBP operátoru je vyjádřeno následující rovnicí [17]:

$$LBP(x_c, y_c) = \sum_{n=0}^{7} 2^n \, s(x_{i_n} - y_c) \tag{2.3}$$

Kde y_c je středový pixel, x_c jsou pixely osmi okolí vstupního obrazu, x_{i_n} je příslušná hodnota z osmi okolí a *s(x)* je funkce ve tvaru:

$$s(x) = \begin{cases} 1, if\ x \geq 0 \\ 0, if\ x < 0 \end{cases} \tag{2.4}$$

Princip vyhodnocení dle rovnice 2.3 je ilustrován na obrázku 9. Nejprve je provedeno porovnání všech hodnot se středovým pixelem (hodnota 3). Výsledek této operace je uveden v druhé části obrázku zleva. Následně je provedeno vynásobení předdefinovanou maskou, která má hodnoty rovny druhé mocnině (třetí část zleva), a výsledek násobení (první část zprava) je pak sečten pro získání výsledné hodnoty. Pořadí hodnot masky pro násobení se může lišit a je jen na autorovi, jak ji sestaví. Výsledná funkcionalita tím nebude nijak ovlivněna. Avšak pro dosažení invariantnosti vůči rotaci se doporučuje zachovat sousednost.

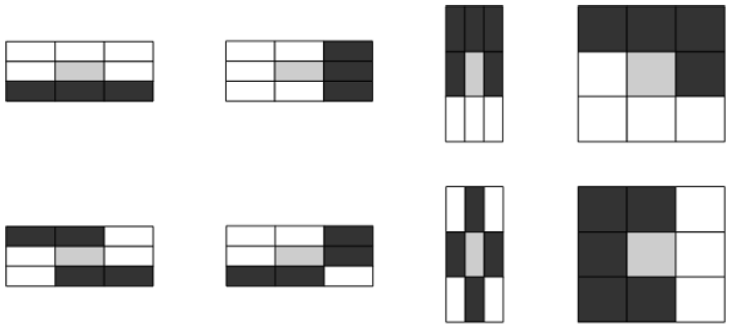


Obrázek 9: Schéma LBP klasifikace [60], [55].

Multi-scale Block Local Binary Pattern (MB-LBP)

LBP je velmi robustní operátor, který je invariantní vůči změně intenzity světla v obraze. Pro získání větší robustnosti však bylo vyvinuto jeho rozšíření na MB-LBP - *Multi-scale Blok Local Binary Pattern* [53]. Rozšíření spočívá v nahrazení jednoho pixelu v mřížce *3 x 3* pixelů původního LBP za regiony. Jeden region může být tvořen několika sousedícími pixely a výsledná hodnota se získá jako konvoluce hodnot pixelů (velmi často je konvoluce nahrazena pouhou sumou hodnot pixelů). Uvedené rozšíření činí algoritmus mnohem robustnějším. LBP se vzhledem ke své velikosti (3x3 pixelů) zaměřuje především na mikrostruktury v obraze. MB-LBP však díky proměnné velikosti regionů může navíc popisovat i makrostruktury. Pro výpočet jednotlivých hodnot regionů může být stejně jako v případě Haarových příznaků využit integrální obraz (viz podkapitola 2.4.1). Vyhodnocení MB-LBP je velmi podobné vyhodnocení LBP. Pouze jednotlivé pixely jsou nahrazeny hodnotou regionů.

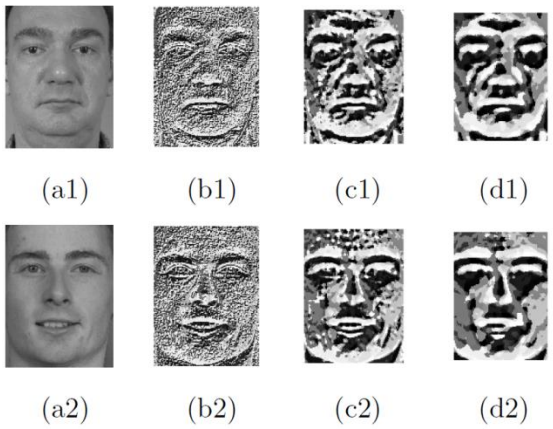
MB-LBP operátor lze použít s různými velikostmi regionů. Několik základních tvarů je vyobrazeno na obrázku 10. Základní mřížka má vždy rozměr *3 x 3*, celkové velikosti operátorů se však mohou lišit, ale velikosti jednotlivých regionů jednoho operátoru jsou stejné (nemusí být ale pravidlem, jak bude ukázáno v kapitole 4).



Obrázek 10: Typy mřížek pro MB-LBP [97].

Na obrázku 11 je uveden výsledek aplikace MB-LBP operátoru na vstupní obraz. Část obrázku *(a)* představuje vstupní obraz ve stupních šedi. Obrázky *(b)*, *(c)* a *(d)* jsou výsledky po aplikaci MB-LBP operátoru s různými velikostmi, a to v pořadí zleva *3 x 3*, *9 x 9*, *15 x 15* pixelů.

LBP, případně MB-LBP operátory se s velkou výhodou používají vzhledem k jejich výhodným vlastnostem pro implementace v FPGA. LBP se používá pro statickou strukturní analýzu obrazu, ale lze jej použít i pro dynamickou strukturní analýzu. Velmi často se LBP operátory používají pro detekci obličeje v obraze [97]. Další možnosti využití LBP může být nalezeno v pracích [58], [63], [75].



Obrázek 11: Výsledek obrázku po aplikaci MB-LBP operátoru [97]. (a) původní obraz (b) 3x3 MB-LBP, (c) 9x9 MB-LBP, (d) 15x15 MB-LBP.

2.4.3 Local Rank Differences (LRD)

Další ze skupiny disktrétních obrazových operátorů je *Local Rank Differences* – LRD. Hlavní výhodou LRD je jeho inherentnost k šedo-tónovým transformacím v obraze. Metoda tak dokáže pracovat s obrazem, který má vysoký jas, ale i s obrazem, který má nízkou hodnotu jasu. LRD dosahuje při klasifikaci obdobných výsledků jako Haarovy příznaky. Vyhodnocení LRD operátoru je velmi podobné vyhodnocení LBP operátoru. Na obrázku 12 je uvedena mřížka *3 x 3* hodnot pixelů nebo regionů. Nejdříve jsou koeficienty v_1 - v_9 získány pomocí konvolucí nad vstupním obrazem obdobně jako pro MB-LBP operátor. Následně je provedeno vyhodnocení LRD příznaku pomocí rovnice [36]:

$$LRD(\mathbf{v}, a, b) = r(v_a, \mathbf{v}) - r(v_b, \mathbf{v}) \tag{2.5}$$

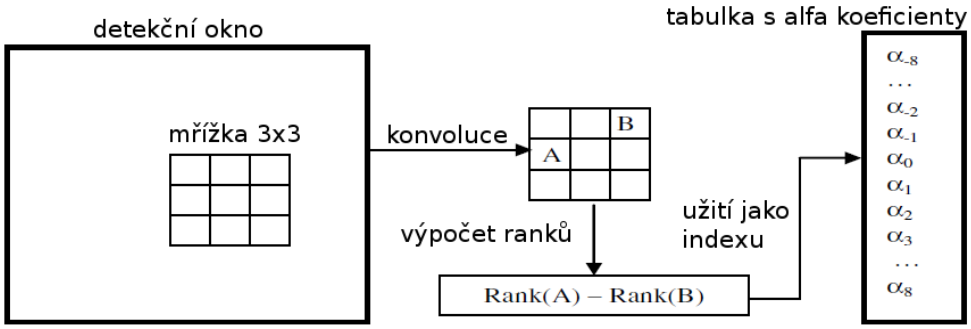
Kde **v** je vektor vstupních dat odpovídající mřížce *3 x 3* na obrázku 12, a a b jsou parametry operátoru v rozsahu *1* až *9* – indexy do vektoru **v** a funkce *r(v,***v***)* je definována následovně:

$$r(v, \mathbf{v}) = \sum_{i=1}^{9} \begin{cases} 1, když\ v > v_i \\ 0, jinak \end{cases} \tag{2.6}$$

v_1	v_2	v_3
v_4	v_5	v_6
v_7	v_8	v_9

Obrázek 12: Příklad operátoru pro LRD a LRP.

Na obrázku 13 je ilustrováno vyhodnocení kompletního slabého klasifikátoru založeného na LRD příznaku v rámci detekčního okna. Velikost detekčního okna je *31 x 31* pixelů a dimenze masky klasifikátoru je *3 x 3* pixelů. Získaná hodnota LRD operátoru je použita jako index do tabulky alfa koeficientů, která obsahuje data získaná trénovacím procesem. Indexovaná hodnota z tabulky alfa koeficientů je konečným výsledkem slabého klasifikátoru.



Obrázek 13: LRD klasifikátor [64].

Výstupní hodnota LRD operátoru může nabývat hodnoty od $\langle -8,8 \rangle$, což je celkem 17 možných výstupních hodnot (poznamenejme, že LBP poskytuje 256 různých výstupních hodnot). LRD je navržen vhodně tak, aby jej bylo možné efektivně implementovat v FPGA a ASIC technologii. Činnost LRD operátoru lze srovnat s činností Haarových vlnek. Haarovy vlnky produkují rozdíl intenzit obrazu dvou oblastí (obecně však mohou i více) a LRD produkuje rozdíl dvou ranků. Výhoda LRD oproti Haarovým vlnkám však spočívá v nezávislosti na celkové intenzitě obrazu. Příznak je invariantní stejně jako LBP vůči intenzitě světla v obraze, a není tak nutno provádět složitou normalizaci vstupního obrazu.

V detekčním okně o velikosti *24 x 24* pixelů (velikost okna používaného pro detekci obličeje) je možné vygenerovat celkem *304 704* různých LRD operátorů s mřížkou velikosti *3 x 3* buňky. Kdežto ve stejném obraze můžeme vygenerovat jenom *86 400* příznaků založených na Haarových vlnkách. To dává LRD výhodu při použití v silných klasifikátorech, jelikož je možné vytvořit větší množství různých slabých klasifikátorů.

2.4.4 Local Rank Patterns (LRP)

Local Rank Patterns - LRP vyvinuli autoři na základě znalostí získaných při práci s LRD operátorem. LRP má snahu vylepšit klasifikační schopností LRD operátoru. LRP je založen na obdobném principu jako LRD operátor ve smyslu výpočtu ranků. Následná manipulace s ranky je však u LRP rozdílná. Výpočet ranku je definován rovnicí [36]:

$$LRP(\mathbf{v}, a, b) = N \cdot r(v_a, \mathbf{v}) + r(v_b, \mathbf{v}) \tag{2.7}$$

Kde koeficient *N* je roven počtu možných hodnot funkce *r(v,***v***)* - tedy 10; význam ostatních proměnných je stejný jako v případě LRD příznaku. S výsledkem operátoru se pracuje stejně jako s výsledkem LRD. LRP není omezeno jen na součet dvou ranků, ale může být aplikováno i na více ranků. Pro součet čtyř ranků by rovnice vypadala následovně:

$$LRP(\mathbf{v}, a, b, c, d) = N^3 \cdot r(v_a, \mathbf{v}) + N^2 \cdot r(v_b, \mathbf{v}) + N \cdot r(v_c, \mathbf{v}) + r(v_d, \mathbf{v}) \tag{2.8}$$

LRP pro případ uvedený v rovnici 2.7 tedy může nabývat celkem 100 různých výstupních hodnot, a má tak mnohem větší diskriminativní sílu než LRD. LRP má vybrané vlastnosti shodné s LRD. Je jí například invariantnost vzhledem k intenzitě jasu obrazu. Diskriminativní síla LRP byla v práci [24] porovnávána s LDR a bylo ukázáno, že je na vyšší úrovni. Dále pak bylo ukázáno, že je lepší i než příznaky založené na Haarových vlnkách. Lepší výsledky byly dosahovány jak v úspěšnosti klasifikace, tak i v menším počtu příznaků (slabých klasifikátorů) potřebných k úspěšné klasifikaci. LRP stejně jako LRD je možné efektivně implementovat v FPGA technologii.

2.4.5 Local Rank (LR)

Operátory LRD a LRP jsou založeny na výpočtu ranků. Výpočet samotného ranku je však možné použít jako samostatný operátor a vystavit slabý klasifikátor jen s jeho pomocí. Tento příznak stejně jako LRD a LRP je invariantní vzhledem k intenzitě jasu obrazu. Počet výstupních hodnot, kterých však může nabývat LR operátor, je poměrně malý (10), a tak jsou i poměrně malé diskriminativní schopnosti daného operátoru. Výpočet příznaku se provádí pomocí rovnice (2.6).

2.4.6 Slabý klasifikátor

Slabé klasifikátory jsou založeny například na dříve uvedených obrazových operátorech. Slabý klasifikátor je vytvořen tak, že je operátor instancován na definované místo v obraze a je mu dána velikost. Jeden operátor tak může být instancován do všech pozic detekčního okna a s různými velikostmi, ale tak, aby byl v detekčním okně a nepřesahoval mimo něj. Na základě jednoho operátoru je tak vytvořena velká množina slabých klasifikátorů. Slabý klasifikátor dále vzniká během trénování tak, že jsou všem možným výstupním stavům přiřazeny hodnoty, které určují míru příslušnosti k dané klasifikační třídě. Tyto hodnoty jsou uloženy v takzvané tabulce alfa koeficientů, jak je například vidět na obrázku 13. Během klasifikace jsou tyto hodnoty vyčítány a je z nich tvořen celkový výsledek klasifikace, kterým je slabý binární statistický klasifikátor.

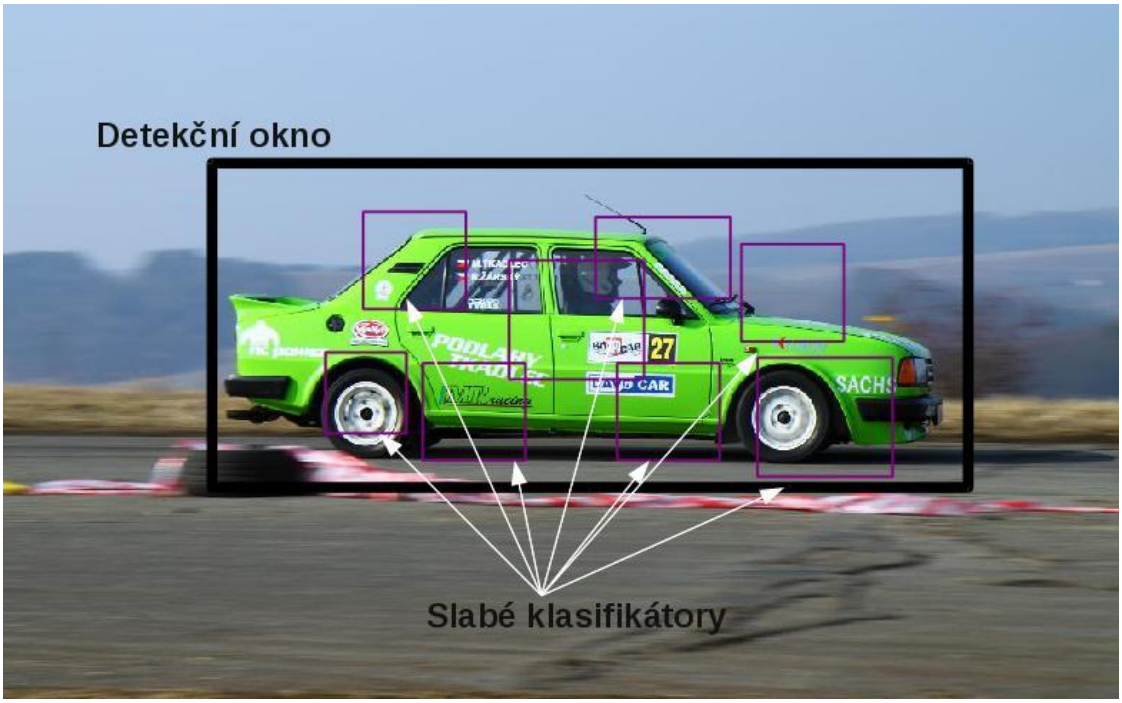
Jeden slabý klasifikátor pracuje nad lokální oblastí detekčního okna a produkuje binární rozhodnutí, zda celé detekční okno obsahuje hledaný předmět. Takové rozhodnutí však samo o sobě nemá požadovanou přesnost pro rozhodování o výsledku klasifikace. Pro trénování slabých klasifikátorů se používá takzvaný slabý učitel. Ten je pro účely uvedených operátorů založen na tvorbě statistiky na základě produkovaných výstupů příznaků a příslušnosti vstupních dat do výstupní třídy. Ke každé výstupní hodnotě operátoru je tak přiřazena hodnota určující míru příslušnosti do dané třídy v závislosti na vstupním objektu.

2.5 Klasifikátory

Tato kapitola je zaměřena na statistické binární obrazové klasifikátory. To je na klasifikátory, které pracují nad detekčním oknem a produkují binární rozhodnutí, zda detekční okno obsahuje hledaný předmět nebo zda jej neobsahuje. Slabé klasifikátory uvedené v předchozí podkapitole nemohou samostatně tvořit takový klasifikátor, ale s využitím celé množiny takových slabých klasifikátorů je již možné vytvořit takzvaný silný klasifikátor, který pracuje s dobrou přesností. Slabé klasifikátory použité v této práci jsou založeny na LBP, LRD, LRP a LR obrazových operátorech. Práce neuvažuje využití slabých klasifikátorů založených na Haarových nebo Gaborvých vlnkách či dokonce neuronových sítích a podobně. Klasifikátory uvedené v této podkapitole pracují tak, že mají na vstupu množinu slabých klasifikátorů a s jejich výstupy se provádí vybraná operace (dle zvolené metody) pro

získání konečného výsledku klasifikace. Natrénování klasifikátoru a běh klasifikátoru se zpravidla liší pro každou klasifikační metodu.

Princip funkce klasifikátoru (rozpoznání předmětu, nikoli trénování klasifikátoru) je založen na skenování vstupního obrazu pomocí takzvaného detekčního okna. V každém kroku klasifikace se detekční okno posouvá v obraze tak, aby se prozkoumal celý obraz (pohyb detekčního okna je o 1, případně i více pixelů doprava nebo o jeden řádek dolů s případným návratem na počátek řádku). Klasifikovaný předmět (či jeho hledaná část) musí být v detekčním okně. Obrázek 14 znázorňuje úlohu rozpoznávání bočního pohledu na automobil. Na obrázku 14 je znázorněno detekční okno, ve kterém probíhá aktuální krok klasifikace. V detekčním okně je umístěno několik slabých klasifikátorů, které společně tvoří výsledný klasifikátor. Každý slabý klasifikátor je přesně určen svým typem, tvarem příznaku a pozicí. Slabé klasifikátory poskytují dílčí výsledky klasifikace. Výstup jednotlivých slabých klasifikátorů má zpravidla různou váhu dle jejich významnosti získané v procesu trénování. Celkový výsledek se získá jako kombinace všech slabých klasifikátorů dle zvoleného principu klasifikátoru (například AdaBoost). Může jím být například vážená suma výsledků slabých klasifikátorů. Běžně se využívá velké množství slabých klasifikátorů (cca 100 – 5000), na obrázku je jich však pro jednoduchost použito jen několik. Detekované předměty mohou mít v obraze různou velikost, ale velikost detekčního okna se nemění (v případě této práce). Proto je nutné měnit velikost obrazu, a provádět tak nepřímo změnu velikosti detekovaných předmětů. Jeden obrazový vzorek je tak zkoumán několikrát, ale vždy s jinou velikostí.



Obrázek 14: Detekční okno se sedmi slabými klasifikátory. Výsledné vyhodnocení je založeného na metodě AdaBoost.

2.5.1 Boosting

Boosting je jedna z prvních metod, která pro své rozhodnutí nevyužívá jednoho pravidla, ale množiny pravidel. Tím se snaží nahradit jedno velmi komplikované pravidlo, které je také velmi obtížně sestavitelné, řadou jednoduchých pravidel, jejichž společnou kombinací se však získá velmi dobrý výsledek. V případě obrazových klasifikátorů chápeme pravidlem jeden slabý klasifikátor. Kořeny

boostingu jsou v projektu PAC (*Probably Approximately Correct*), jenž se zabýval strojovým učením [80]. Myšlenkou využití množiny jednoduchých pravidel se prvně zbývali autoři Kearns a Valiant ve svých dílech [43], [42]. Autoři zkoumali, zda kombinace množiny pravidel, které jsou jen o trochu lepší než náhodná funkce, může vytvořit libovolně přesný silný učící se algoritmus. První algoritmus využívající *boosting* pro zpracování obrazu představil Schapire ve svém díle [68] v roce 1989. Uvedený algoritmus měl polynomiální časovou složitost. V roce 1995 přestavil Freund ve své práci [9] mnohem efektivnější *boostingový* algoritmus, který i když je v určitém aspektu optimální, stále trpí jistými nevýhodami. *Boostingové* metody byly prvně implementovány v práci autorů Drucker, Schapire a Simard [6], která se zabývá čtením znaků v obraze. Rané *boostingové* metody se však dočkaly velkého rozvoje, řada jejich nedostatků byla odstraněna a byly navrženy nové metody, jako je AdaBoost či WaldBoost (viz dále).

2.5.2 AdaBoost

Metoda AdaBoost patří do skupiny *boostingových* metod a je jedním z jejich vylepšení, kterých se dočkala. Název metody AdaBoost je odvozen ze slovního spojení adaptivní *boosting* (přizpůsobivý *boosting*). Adaptivnost klasifikátoru se chápe ve smyslu přizpůsobení klasifikátoru na vstupní data. To se děje tak, že jednotlivým vzorkům z trénovací množiny jsou přiřazeny váhy, a ty se v každé iteraci trénování upravují dle toho, zda je vstupní vzorek klasifikován správně nebo ne. Tyto úpravy mají za následek to, že vzorky, které se obtížně klasifikují, mají s postupem iterací větší váhu a klasifikátor se jím více přizpůsobuje.

Metoda AdaBoost byla představena autory Freund a Schapire v roce 1995 [8]. Algoritmus odstraňuje řadu nevýhod původní *boostingové* metody.

Trénování klasifikátoru

Proces učení AdaBoost klasifikátoru probíhá v několika iteracích $t = 1,2,\ldots T$. Celkový počet iterací T je dán počtem slabých klasifikátorů, které má výsledný klasifikátor obsahovat. Počet iterací nemusí být dopředu znám, pokud se provádí trénování klasifikátoru na stanovenou přesnost. V takovém případě je poté celkový počet iterací určen dynamicky během procesu trénování klasifikátoru. V každé iteraci je prováděno ohodnocení množiny slabých klasifikátorů na množině vstupních dat S s váhami D_t. V každé iteraci dochází k aktualizaci vah D_t za účelem zlepšení výsledného klasifikátoru. V každé iteraci je vybrán nejlepší slabý klasifikátor (hypotéza) z množiny všech dostupných klasifikátorů H_P. V každé iteraci se hledá taková hypotéza $h_t : X \rightarrow \{-1,+1\}$, která minimalizuje chybu klasifikátoru s aktuálními vahami D_t trénovací množiny vzorků. Chyba hypotézy ε_t se vypočte následovně:

$$\varepsilon_t = P_{i \sim D_t}[h_t(x_t) \neq y_t] \tag{2.9}$$

Nejlepší slabá hypotéza h_t je vyjmuta z množiny dostupných hypotéz H_P a vložena do množiny hypotéz, které tvoří výsledný klasifikátor s koeficientem α_t, který je stanoven s ohledem na vypočtenou chybu ε_t a váhami D_t. Koeficient α_t definuje jak významná (přesná) je hypotéza v kroku t. Jeho výpočet se provede vzorcem:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \tag{2.10}$$

Výsledný klasifikátor H je poté tvořen lineární kombinací výsledků dříve určené množiny slabých klasifikátorů. Celkový výsledek klasifikace se získá vzorcem:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{2.11}$$

V každé iteraci trénování se po vybrání hypotézy provádí přepočet množiny vah D_t. Váhy D_{t+1} pro následující iteraci $t+1$ jsou vypočteny z původních vah D_t na základě nově vybrané hypotézy h_t. Hodnoty vah jednotlivých vzorků se tak navyšují pro nesprávně klasifikované vzorky a naopak hodnoty vah se snižují pro správně klasifikované vzorky z trénovací množiny. Aktualizace vah se provede pomocí vzorce:

$$D_{t+1}(x_i) = \frac{D_t(x_i) \cdot \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \tag{2.12}$$

kde

$$Z_t = \sum_{i=1}^{m} D_t(x_i) \cdot \exp(-\alpha_t y_i h_t(x_i)) \tag{2.13}$$

Aktualizace vah D_t je základní myšlenkou AdaBoostu a vyjadřuje výsledky klasifikací v předešlých iteracích procesu učení pro daný vzorek trénovací množiny. Váha $D_t(x_i)$ vstupního vzorku i, v iteraci t vyjadřuje úspěšnost klasifikace pomocí dříve vybraných hypotéz. Níže je uveden kompletní algoritmus AdaBoostu [8].

Mějme $S = \langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$, kde $x_i \in X$ vstupní data a $y_i \in Y = \{-1, +1\}$

Inicializace vah $D_1 = \frac{1}{m}$

For t=1,…,T:

1. Nalezni $h_t = \arg\min_{h_j \in H} \varepsilon_j$; $\varepsilon_j = \sum_{i=1}^{m} D_t(x_i) I[h_j(x_i) \neq y_i]$
2. If $\varepsilon_t \geq 1/2$ then stop
3. $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
4. Aktualizuj váhy

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

kde $Z_t = \sum_{i=1}^{m} D_t(x_i) \cdot \exp(-\alpha_t y_i h_t(x_i))$.

Výsledný klasifikátor má poté tvar:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Algoritmus AdaBoost [8].

Vstup algoritmu je tvořen trénovací množinou ve tvaru $S = \langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$, kde $x_i \in X$ jsou vstupní data a $y_i \in Y = \{-1, +1\}$ je ohodnocení vstupních dat (-1 nepatří mezi hledané objety, 1 patří mezi hledané objekty). V prvním kroku algoritmu se provádí inicializace vah trénovací množiny na iniciální hodnotu, která je nepřímo úměrná počtu vstupních dat. Následuje iterační proces pro hledání jednotlivých hypotéz. Ten se skládá ze čtyř základních kroků. V prvním kroku iterace se hledá taková hypotéza h_t, která má nejmenší váženou chybu na množině trénovacích dat s aktuálními váhami. (Funkce *I* vrací *1*, pokud je podmínka splněna, nebo *0*, pokud podmínka není splněna.) V druhém kroku se kontroluje základní podmínka pro slabé klasifikátory v AdaBoostu a tou je, že chyba slabého klasifikátoru nesmí být větší než *50 %* (lepší než náhodná funkce). Podmínka je důležitá vzhledem ke konvergenci algoritmu. Pokud by nebyla splněna, tak by s přibývajícím počtem hypotéz výsledný klasifikátor ztrácel svou přesnost.

Ve třetím kroku se provádí výpočet koeficientu α_t pro lineární kombinaci výsledků slabých klasifikátorů v rámci silného klasifikátoru. Výpočet α_t se provádí tak, aby se minimalizoval horní odhad chyby výsledného klasifikátoru.

$$\varepsilon_{tr}(H) \leq \prod_{t=1}^{T} Z_t = \frac{1}{2^T} \prod_{t=1}^{T} \sqrt{\varepsilon_t (1 - \varepsilon_t)} \tag{2.14}$$

V posledním čtvrtém kroku je provedena aktualizace vah, která provede zmenšení vah správně klasifikovaných trénovacích vzorků a zvětšení vah nesprávně klasifikovaných vzorků trénovací množiny. Tím je zajištěna lepší adaptace na trénovací množinu a v dalším kroku se bude hledat klasifikátor, který lépe klasifikuje doposud nesprávně klasifikované vzorky trénovací množiny.

Uvedený algoritmus je ukončen buď po nalezení *T* hypotéz, nebo není-li možné naleznout hypotézu h_t, která má chybovost ε_t menší než *0,5*. V každé iteraci algoritmu je nutné pro vybrání nejlepší hypotézy h_t provést natrénování všech slabých klasifikátorů (hypotéz) s aktualizovanými váhami trénovací množiny *S* pomocí slabého učitele. Pro každou hypotézu h_t se sestrojí strom s *256* listy, kde každý z nich odpovídá jedné výstupní diskrétní hodnotě LBP operátoru. Slabý klasifikátor může být poté definován následovně:

$$h_t(x) = \begin{cases} a_0, & x^k = 0 \\ \ldots & \\ a_j, & x^k = j \\ \ldots & \\ a_{255}, & x^k = 255 \end{cases} \tag{2.15}$$

Kde x^k značí *k*-tý prvek příznakového vektoru *x* (výsledek LBP operátoru na vstup *x*) a a_j, $j = 0, \ldots, 255$, jsou regresní parametry, které se získávají v procesu trénování. Slabý učitel je často nazýván rozhodovacím či regresním stromem. Nejlepší stromově založený slabý klasifikátor (parametry *k* a a_j jsou nalezeny tak, aby byla minimalizována chyba vzhledem k vzorci 2.13) lze získat naučením jednotlivých listů stromu pomocí následujícího vzorce:

$$a_j = \frac{\sum_i D_i y_i \delta(x_i^k = j)}{\sum_i D_i \delta(x_i^k = j)} \tag{2.16}$$

Kde funkce δ vrací hodnotu 1, pokud je výraz v podmínce pravdivý, jinak vrací 0. Takto vytvořený strom (funkce) je velmi podobná vyhledávací tabulce.

Předností AdaBoost metody je, že velmi rychle konverguje k nalezení klasifikátoru, který má malou chybu na trénovacích datech. Mnoho prací [8] se zaměřuje na detailní popis minimalizace chyby v procesu trénování. Tato práce se však primárně nezabývá problémem, jak vylepšit samotný algoritmus, a proto se touto problematikou dále nezabývá.

Běh klasifikátoru

Výstupem trénovacího procesu AdaBoost klasifikátoru je množina slabých klasifikátorů, kde každý z nich je definován hypotézou $h_t(x)$ a koeficientem α_t. Hypotéza $h_t(x)$ představuje natrénovaný slabý klasifikátor, například klasifikátor založený na LBP příznacích. Koeficient α_t vyjadřuje kvalitu slabého klasifikátoru. Čím vyšší je koeficient α_t, tím je daný slabý klasifikátor významnější a tím více se uplatňuje v celkovém výsledku klasifikátoru. Rovnice pro výpočet klasifikátoru nad daným vstupním obrazem x je vyjádřena následujícím vztahem:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{2.17}$$

Kde funkce *sign* je definována jako:

$$sign(x) = \begin{cases} 1\ když\ x > 0 \\ -1\ jinak \end{cases} \tag{2.18}$$

Výsledná hodnota klasifikátoru je dána lineární kombinací výsledků slabých klasifikátorů. V tomto případě je lineární kombinace nahrazena pouhou sumou výsledků slabých klasifikátorů. Ve vzorci se vyskytuje násobení koeficientu α_t s výsledkem funkce $h_t(x)$. Toto násobení však lze eliminovat přesunutím α_t do funkce $h_t(x)$. Tato modifikace je možná, jelikož funkce $h_t(x)$ je tvořena obrazovým operátorem, jehož výstup se používá jako index do tabulky alfa koeficientů (natrénovaných dat). A ta je také součástí natrénovaného slabého klasifikátoru. Pokud se na hodnoty získané v trénování aplikuje α_t, lze získat (předpočítat na konci trénovacího procesu) novou funkci $h_{\alpha t}(x)$. Dále je pak možné upravit vztah pro výpočet AdaBoost klasifikátoru na nový tvar:

$$H(x) = sign\left(\sum_{t=1}^{T} h_{\alpha t}(x)\right) \tag{2.19}$$

Takto upravený vzorec je poté velmi vhodný pro implementaci v FPGA, jelikož obsahuje jen základní matematické operace, jako je sčítání a porovnávání, které je možné snadno implementovat v FPGA.

2.5.3 WaldBoost

AdaBoost se dočkal mnoha rozšíření, jako je například vylepšení autory Viola a Johnes [81], kteří přestavili upravený AdaBoost klasifikátor, jež byl rozdělen do jednotlivých částí. A po vyhodnocení každé z části provádí test příslušnosti do klasifikační třídy. Není tak nutné vyhodnotit všechny příznaky jako v původním AdaBoost od autorů Freund a Schapire. Tato optimalizace vedla k významnému zvýšení rychlosti práce klasifikátoru (běh klasifikátoru).

Později bylo autory Šochman a Matas v práci [71] představeno další vylepšení AdaBoost metody jménem WaldBoost. Toto rozšíření lze považovat za nejvýznamnější z hlediska navýšení rychlosti

klasifikace. WaldBoost je založen na kombinaci AdaBoost metody s Wald's sekvenčním pravděpodobnostním testem. Použití této metody si klade za cíl vyřešit problém s vyhodnocením velkého počtu slabých klasifikátorů, jako je tomu u AdaBoost metody. U AdaBoostu [9] je počet slabých klasifikátorů, které se mají vyhodnotit, konstantní a je dopředu znám. U WaldBoost metody tomu však tak není, a počet klasifikátorů, které se vyhodnotí pro jednotlivá detekční okna, závisí především na obsahu dat. Algoritmus předpokládá, že většina zkoumaných detekčních oken nebude obsahovat hledaný objekt, a proto u nich provede jen „rychlý test" a poté jejich vyhodnocení ukončí. Pro většinu detekčních oken je tak provedeno vyhodnocení jen několika málo slabých klasifikátorů. Díky tomu je vyhodnocení metody WaldBoost o cca *1,5 – 2* řády rychlejší než AdaBoost při srovnatelné klasifikační přesnosti.

Hlavním problémem je v uvedeném případě problematika určení správného prahu pro ukončení vyhodnocení. Autoři ve své práci formulovali klasifikační problém jako sekvenční rozhodovací proces, který je charakterizován mírou chybovosti a průměrnou dobou vyhodnocení klasifikátoru. Na základě této formulace autoři ukázali, že optimální strategie ve smyslu nejkratšího průměrného času rozhodnutí v závislosti na požadavcích na míru chyb je dána Wald's sekvenčním pravděpodobnostním testem. Jejich práce tedy ukazuje, jak co nejlépe zvolit hranici mezi rychlostí vyhodnocení a mírou chyb.

Nevýhodnou WaldBoostu je, že nastavení prahů spoléhá na proces, který zahrnuje odhad hustoty pravděpodobnosti. Vyhodnocení klasifikátoru pomocí metody WaldBoost také velmi přísně zavádí sekvenční charakter vyhodnocení. U AdaBoostu je možné nejprve provést vyhodnocení všech příznaků nezávisle na sobě a až následně provést vyhodnocení jejich výsledků. U WaldBoost je nutné zachovat sekvenční charakter a po každém vyhodnocení slabého klasifikátoru (skupiny slabých klasifikátorů) provést test, zda se má pokračovat ve vyhodnocení. Takové provedení testu však zabraňuje provedení masivně paralelního vyhodnocení.

2.6 Shrnutí

Práce se bude dále zabývat statistickou metodou klasifikátoru AdaBoost podle autorů Freund a Schapire [9], případně pak také dle autorů Viola a Johnes [81]. Tato metoda byla vybrána vzhledem k jejím velmi dobrým klasifikačním vlastnostem, a také z toho důvodu, že umožňuje použít pro její zpracování masivně paralelní techniky. AdaBoost není z hlediska rychlosti zpracování na univerzálním CPU příliš vhodný, jelikož je jeho zpracování zdlouhavé a byl v mnoha případech nahrazen právě algoritmem WaldBoost. Avšak při zpracování na FPGA lze výhodu masivní paralelizace využít, a použití AdaBoostu se pak může jevit mnohem vhodnějším, jak bude ukázáno v kapitolách 5 a 6.

AdaBoost je univerzální metoda, kterou je možné použít pro řešení velkého počtu úloh. Díky tomu však bude vždy o něco horší než aplikačně specifické metody, které jsou optimalizovány jen pro řešení jednoho typu problému, a to ať už z hlediska rychlosti, tak i z hlediska přesnosti. Výhoda univerzálnosti však převládá, a možnost jednou metodou zpracovávat více úloh je hlavní předností.

3 Implementace boostingových metod

Od prvního představení adaptivní *boostingové* metody autory Freund a Schapire [8], [7] v roce 1995 bylo publikováno mnoho prací, které se snaží efektivně tuto metodu implementovat. První práce se zabývaly implementací metody na standardních CPU [8], případně se snažily výpočet akcelerovat využitím speciálních instrukcí CPU (jako jsou například vektorové instrukce SIMD, SSE - *Streaming SIMD Extension*, a další). Jelikož tyto implementace spotřebovávají velmi mnoho výkonu CPU, či případně bylo použití CPU výkonově nedostatečné, začaly se objevovat nové implementace, které již mířily na jednotky s jinými architekturami než je CPU. Jelikož je možné *boostingové* metody různými způsoby paralelizovat, tak se směr implementací obrátil především k výpočetním architekturám poskytující vysokou míru paralelizmu. Proto se další implementace objevily pro GPU a také pro FPGA. Implementace *boostingových* metod na těchto architekturách dosahují rozmanitých výsledků.

V následujících podkapitolách je uveden přehled o nalezených nejvýznamnějších implementacích pro nejpoužívanější architektury. Ke každé implementaci jsou dány její hlavní přednosti či zápory.

3.1 Standardní CPU

První vytvořené implementace *boostingových* metod byly provedeny právě na standardním CPU, jelikož tato architektura umožňuje velmi rychle a jednoduše vytvořit výsledné řešení. První implementace byla dána autory Freund a Schapire v práci [8]. Jelikož však AdaBoost není příliš optimalizován pro zpracování na CPU a plýtvá výpočetním časem, byla metoda boostingu postupně rozvíjena tak, aby její zpracování bylo efektivnější. Autoři jednoho z vylepšení Viola a Johnes představili v roce 2001 [81] upravenou AdaBoost metodu, která výrazně redukuje čas zpracování na CPU. Tato metoda byla posléze ještě dále vylepšována a jedno z dalších vylepšení nese jméno WaldBoost (viz 2.5.3).

3.1.1 AdaBoost dle Freund a Schapire

Implementace původního AdaBoostu dle Freunda and Schapire [8] je na CPU z implementačního hlediska sice jednoduchá, ale velmi výpočetně neefektivní. Tato metoda neobsahuje žádná urychlení a pro vyhodnocení jednoho detekčního okna se musí vyhodnotit všechny slabé klasifikátory, které tvoří silný AdaBoost klasifikátor. Tato vlastnost vede k faktu, že vyhodnocení každého detekčního okna vyžaduje stejné množství operací, a tedy i stejný čas nezávisle na obsahu vstupních dat. Rychlost takového algoritmu je tak v závislosti na počtu slabých klasifikátorů až řádově nižší než u dále uvedených vylepšených metod. Rychlost takto implementovaného klasifikátoru je cca 3,3 Mpps na CPU Intel Core i5-3570K – čtyři jádra, frekvence 4.1 GHz [35].

3.1.2 AdaBoost – vylepšení

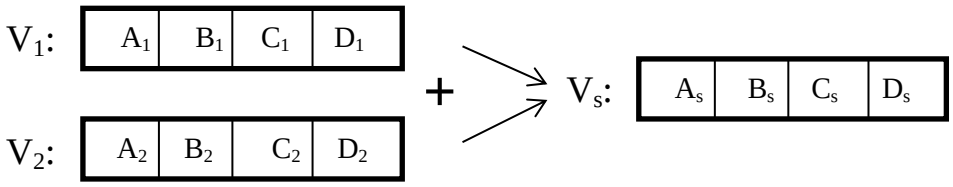
Původní AdaBoost metoda se dočkala řady vylepšení, z nichž nejvýznamnější z hlediska rychlosti vyhodnocení je úprava představená autory Viola a Johnes [82]. Tato úprava spočívá v rozdělení celého klasifikátoru do několika částí (takzvaných *stages*) a možnost ukončení vyhodnocení detekčního okna po každé z částí. AdaBoost byl dále vylepšován například v pracích [5], [52] a [72], [73]. Další významné vylepšení z hlediska rychlosti zpracování dat je WaldBoost [71] (viz 2.5.3),

který je založen na Wald sekvenčním rozhodování [84] v kombinaci s AdaBoostem při zachování klasifikační přesnosti.

3.1.3 Implementace s využitím SIMD instrukcí

Originální vyhodnocení AdaBoostu na PC je sekvenční, což také odpovídá sekvenčnímu charakteru CPU. Novější CPU však obsahují rozšíření instrukční sady o takzvanou SIMD instrukční sadu. Jedná se o instrukce, které zpracovávají najednou několik dat – jedna instrukce, vícero dat. Za pomoci těchto instrukcí je tak možné provést například operaci součtu nad dvěma vektory dat. A jako výsledek operace získat vektor součtů.

Obrázek 15 zobrazuje příklad součtové operace pro SIMD instrukci. Vstupem jsou dva vektory V_1 a V_2 a výstupem je vektor V_s. SIMD instrukce jsou v procesorech často zahrnuty v instrukční sadě SSE instrukcí (z anglického *Streaming SIMD Extension*). SIMD bylo poprvé použito v roce 1999 jako rozšíření instrukční sady x86 pod názvem MMX, takzvané rozšíření pro multimédia (z anglického *Multi-Media eXtentsion*). SIMD instrukce v dnešních procesorech mají zpravidla *128* a více bitové registry. Do *128*bitového registru je možné uložit až šestnáct osmibitových hodnot a provést nad všemi současně společnou operaci. SIMD instrukce tak ve své podstatě představují zavedení paralelního zpracování do sekvenčních CPU.



Obrázek 15: Ilustrace SIMD instrukce.

Vlastnost částečného paralelního zpracování u sekvenčních procesorů byla využita v pracích [17] a [18]. V práci [17] jsou ukázány techniky pro dosažení rychlejšího zpracování snímků. Autoři uvádějí, že zrychlení bylo dosaženo pomocí dvou hlavních technik. První je minimalizování počtu přístupů do paměti a druhou technikou je právě použití SIMD instrukcí. Minimalizace počtu přístupů do paměti je dosažena za pomoci použití předpočítané reprezentace obrazu, takzvaný prokládaný konvoluční obraz [18] a [19], díky kterému je možné rychle vyčítat aktuálně potřebná data z paměti pomocí malého počtu paměťových operací.

Autoři ve své práci využili slabé klasifikátory založené na LBP operátorech (v dalších obdobných pracích také na LRP či LRD operátorech). Výpočet množiny příznaků AdaBoostu byl autory rozdělen do dvou nezávislých částí. V první části se provedl výpočet všech konvolucí pro všechny LBP operátory, výsledky se uložily a následně použily jako vstup do další fáze. Provedením tohoto kroku autoři zajistili, že konvoluce všech příznaků se budou počítat pro všechny LBP operátory jen jedenkrát. Aby mohli autoři provést efektivní implementaci, omezili se pouze na obdélníkové tvary operátorů a velikost jedné konvoluce omezili na *2 x 2* pixely, což dává maximální velikost LBP operátoru *6 x 6* pixelů. Pro použité tvary příznaků vznikly celkem čtyři nové reprezentace obrazu – takzvané konvoluované obrazy. Autoři zajistili vhodné uložení konvoluovaných dat tak, aby byly následné paměťové operace čtení velmi rychlé.

Pro otestování implementace autoři použili PC s následujícími parametry CPU – Intel Core i7 (osm jader) a *4* GB RAM paměti. Na uvedeném procesoru dosáhli rychlosti zpracování 10 snímků za

sekundu pro obraz o velikosti *1280 x 720* pixelů, to je v přepočtu rychlost cca *9,3* Mpps. Rychlost byla dosažena jak s LBP, tak s LRP i s LRD operátory, cílová požadovaná chyba byla nastavena na $\alpha=0.1$ (cílový *error rate*). Autoři však neudávají detailní složení klasifikátoru. Dá se však spekulovat, že pokud by byla požadována větší přesnost klasifikátoru $\alpha=0.02$, zřejmě by klesla rychlost zpracování třeba až na polovinu, jelikož by se výrazně zvětšil počet použitých příznaků. Tato spekulace je založena na faktu, že autoři provedli měření i na méně přesném klasifikátoru s $\alpha=0.2$ a dosáhli o *30 %* větší rychlosti.

V práci je pro srovnání prezentována rychlost zpracování stejného klasifikátoru bez SIMD instrukcí na stejném PC. Tato rychlost je cca *1,4* snímku za sekundu, což je přibližně *1,3* Mpps. Použitím SIMD instrukcí tak bylo dosaženo přibližně šestinásobného urychlení.

3.1.4 Vícevláknové implementace na CPU

Práce [33] se také zabývá problémem rychlosti zpracování AdaBoost metody na CPU. Tato práce se však na problematiku dívá mírně odlišně a zaměřuje se především na efektivní zpracování AdaBoostu na více jádrových procesorech. Práce vznikla již v roce 2008, a tak jsou výsledky dosažené na dřívějších procesorech mírně zkreslené a nelze je přímo porovnávat s výše uvedenou implementací. Zajímavé jsou však poměry dosažených urychlení pro různé úrovně použitého paralelizmu. Nicméně i tato práce využívá SIMD instrukční sadu stejně, jako je tomu v práci uvedené v podkapitole 3.1.3. AdaBoost použitý v práci využívá slabé klasifikátory založené na HOG příznacích (Histogram Orientovaných Gradientů).

V této práci autoři definovali základní úrovně, jak může být zpracování AdaBoostu paralelizováno:

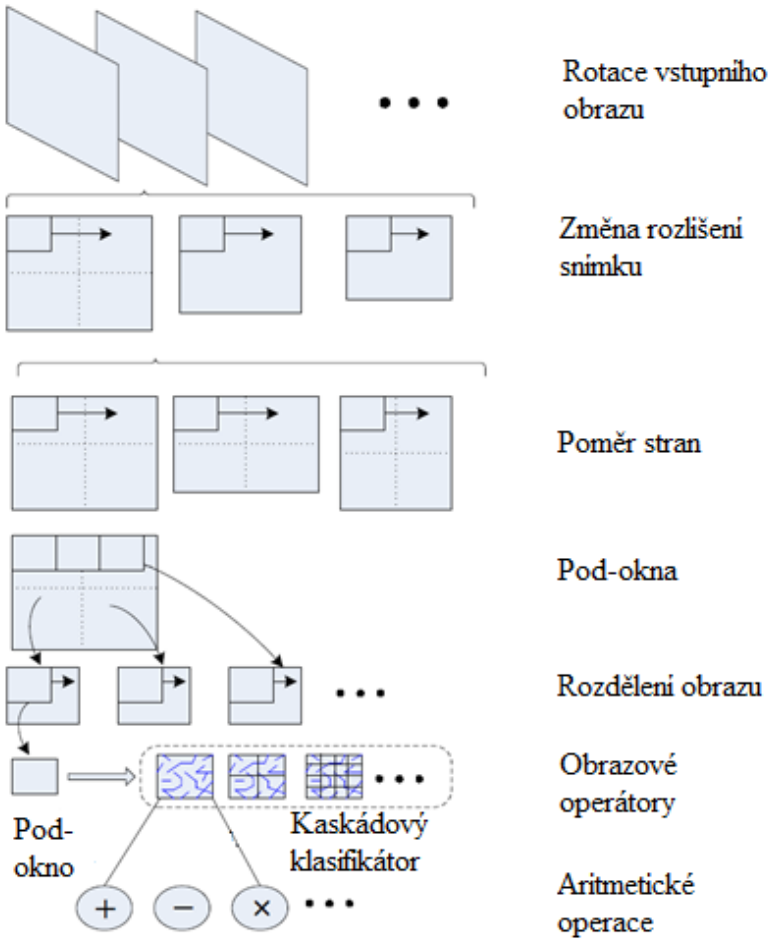
- Rotace vstupního obrazu – každý rotovaný snímek může být zpracováván samostatně.
- Změna rozlišení snímků – zpracování snímků v různém rozlišení, hledání objektů různé velikosti.
- Poměr stran vstupního obrazu – každý deformovaný obraz je zpracován samostatně.
- Rozdělení obrazu do podoken a zpracování každého podokna samostatně.
- Zpracování více detekčních oken.
- Paralelní zpracování obrazových příznaků.
- Paralelizace aritmetických operací.

Na obrázku 16 jsou jednotlivé úrovně paralelizace vizualizovány. Tyto úrovně pak autoři rozdělili do 3 základních skupin a definovali, co do nich patří, následovně:

- Hrubý vláknový paralelizmus:
  - rotace obrazu,
  - změna rozlišení obrazu,
  - změna poměru stran obrazu.
- Jemný vláknový paralelizmus:
  - rozdělení obrazu na části,
  - zpracování několika detekčních oken,
  - zpracování příznaků.

- Jemný datový paralelizmus (DLP):
    - jednotlivé datové operace.

Právě DLP (*Data Level Paralelism*) může být velmi snadno implementován pomocí SIMD instrukcí a může být použit v obou případech vláknového paralelizmu (TLP – *Thread Level Paralelism*).



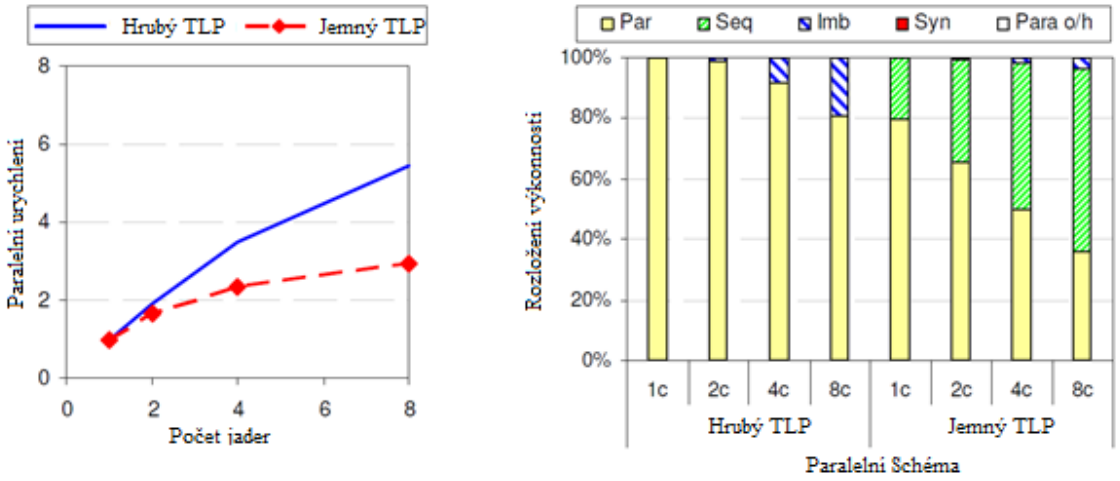
Obrázek 16: Sedm základních úrovní paralelizace dle [33].

Každý z uvedených paralelizmů má své výhody a nevýhody. Jako jedna z hlavních výhod hrubého vláknového paralelizmu se může jevit jeho snadná implementace, jelikož hranice dělení jsou velmi snadno definované a téměř veškerý kód klasifikátoru může být vykonáván paralelně. Jemný vláknový paralelizmus je obtížnější na implementaci a musí obsahovat větší sekvenční část kódu (neparalelizovatelnou). A také další nebezpečí jemného paralelizmu je jeho možná nevyváženost vzhledem k využití výpočetních prostředků. Avšak použitím jemného paralelizmu snížíme latenci na získání výsledků z jednoho snímku (jedné úrovně změny rozlišení).

Autoři provedli experimenty s jemným a hrubým vláknovým paralelizmem na počítači s dvěma CPU Intel Core 2 Quad s pracovní frekvencí *2,33* GHz. Pro paralelizaci použili OpenMP (*Open Multi-Processing*) programovací model [29]. Dosažená rychlost čistě sekvenčního zpracování bez paralelizmu byla *0,5* snímku za sekundu při rozlišení obrazu *704 x 576* pixelů, což je rychlost *0,2* Mpps. Použitím jemného vláknového paralelizmu bylo dosaženo urychlení *2,9*krát při použití *8* jader. Použitím hrubého vláknového paralelizmu dosáhli urychlení *5,5*krát taktéž při použití *8* jader.

Použitím DLP dosáhli dalšího přibližně *30%* urychlení, a tedy poté celkové urychlení bylo *3,77* pro jemný vláknový paralelizmus a *7,1*krát pro hrubý vláknový paralelizmus. I přesto se však rychlost zpracování pohybuje na hranici cca *1,5* Mpps.

V porovnání s prací uvedenou v podkapitole 3.1.3 jsou dosažené výsledky cca *6*krát horší, což je dáno jednak použitím staršího typu procesorů a také zřejmě menší optimalizací pro SIMD instrukce. Tato práce však především ukazuje, jakým způsobem je vhodné provádět paralelizaci AdaBoostu na více jádrových procesorech. A ukazuje, že nejlepší škálovatelnost je právě u hrubé paralelizace. Na obrázku 17 jsou uvedeny výsledky škálovatelnosti. Z grafu nalevo je možné vidět, že jemný paralelizmus se již při použití *4* a více jader začíná nevyplácet vzhledem k dosaženému urychlení (viz pravá část obrázku). Avšak hrubý paralelizmus má smysl použít i u *8* jader. V grafu napravo lze vidět, že u jemného vláknového paralelizmu je velká sekvenční část, která neumožňuje provést masivní paralelizaci. Graf nalevo udává škálovatelnost AdaBoostu a graf napravo udává podíl jednotlivých složek při vykonávání (paralelní část, sekvenční část, nevyváženost, synchronizace a OpenMP režie).

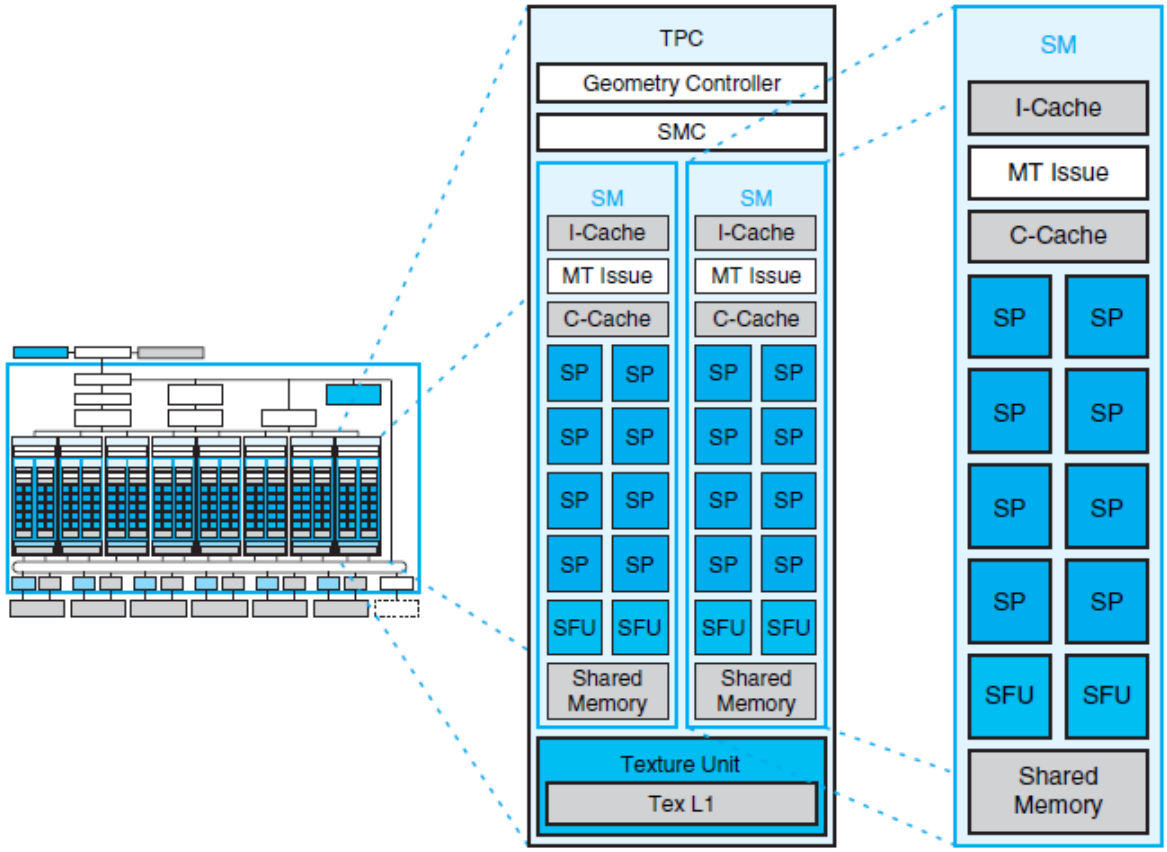


Obrázek 17: Vlevo: škálovatelnost AdaBoostu. Vpravo: Podíl jednotlivých složek při vykonávání [33].

3.2 GPU

Jelikož rychlost zpracování AdaBoostu na CPU u představených metod není zcela postačující, pro zpracování obrazu ve vysokém rozlišení (například FullHD – *1920 x 1080* pixelů při 25 snímcích za sekundu) v reálném čase byly vyvíjeny další metody pro akceleraci. Metody v sekcích 3.1.3 a 3.1.4 jsou založeny především na využití paralelizmu, který poskytuje standardní CPU, a to ať už SIMD instrukce nebo více jader ve výpočetní jednotce. Ovšem i tak CPU poskytuje jen řádově desítky samostatně pracujících (jádra procesoru) nebo lokálně řízených (SIMD) jednotek. Na druhou stranu jsou dostupné výpočetní prostředky, které poskytují až o několik řádů více paralelně pracujících jednotek (stovky až tisíce). Příkladem takové výpočetní jednotky je GPU. Ačkoli bylo GPU primárně určeno na grafické operace, tak postupem času byl jeho potenciál pro obecné výpočty tak silný, že byly vyvinuty jazyky a sady nástrojů pro řešení obecných problémů na GPU (GP-GPU – *General Purpoase GPU*). Jedním z prostředků pro vytváření obecných algoritmů na GPU je například CUDA (*Compute Unified Device Architecture*) [59] od nVidia nebo jazyk OpenCL (*Open Computing Language*) [74]. Hlavní předností GPU pro obecné výpočty je především to, že obsahuje velké množství paralelně pracujících jednotek. Tyto jednotky jsou sice velmi jednoduché a umí zpracovat jen základní operace, ale jejich počet je významný pro urychlení řady problémů. Architektura

jednotek v GPU je zpravidla hierarchická a je rozdělena do několika úrovní. Na obrázku 18 je uveden příklad uspořádání GPU architektury. Základní jednotkou je SM – symetrický multiprocesor. Ten je dále složen z cache pamětí pro instrukce a data, proudových procesorů – SP, nebo jinak řečeno procesorových jader, paměti a jednotek pro speciální funkce – SFU (*Special Functional Unit*). SP jednotky dokážou zpravidla vykonávat jen základní instrukce. Pro vykonání složitějších instrukcí, jako například výpočet trigonometrických funkcí, se používá právě SFU jednotek. Skupina SM je sdružena do TPC – vláknového clusteru (*Thread Processing Cluster*), který obsahuje řídící logiku (*Geometry Controller*- mapování výpočtů na SM, SMC – *SM Controler*), jednotky pro práci s texturami (*Texture units*) a další paměť.



Obrázek 18:Ilustrace struktury GPU [78].

Akcelerací AdaBoostu využitím GPU jednotek se zabývá řada prací [20], [61], [32]. Jedna z prvních prací, která využila GPU pro výpočet AdaBoostu dle Viola a Jouhnes [81], byla práce autorů Krpec a Němec [46]. Ta tak poprvé ukázala sílu GPU pro výpočet AdaBoostu. Autoři ukázali, že GPU je ve srovnání s jednovláknovými implementacemi na CPU výrazně rychlejší. Následně byly představovány další práce, které postupně zvyšovaly maximální dosaženou rychlost zpracování na GPU. Autor Herout ve své práci [20] prezentoval detektor obličejů založený na GPU s využitím LRP operátorů a dosáhl rychlosti zpracování *60* snímků za sekundu pro obraz o rozlišení *1280 x 720* pixelů. Autoři Sharma a kolektiv [69] prezentovali architekturu, která dokáže zpracovat *19* snímků za sekundu při rozlišení *1280 x 960* pixelů.

Poslední zde uvedená a prozatím nejzajímavější a nejrychlejší implementace byla vytvořena autory Chouchene a kolektiv [32] v roce 2015. Architektura v uvedené práci dosahuje rychlosti zpracování *124* snímků za sekundu při rozlišení obrazu *1024 x 1024* pixelů. Chouchene použil v práci

slabé klasifikátory založené na Haarových vlnkách. Autoři dosáhli rychlého zpracování na GPU pomocí dvou různých optimalizačních technik. První technika se zabývá využitím sdílené paměti, zatímco druhá technika se zabývá použitím vhodné velikosti výpočetních bloků na CUDA (vhodná úroveň paralelizovatelnosti kódu). Autoři ve své práci postupovali tak, že nejprve vytvořili sekvenční implementaci na standardní CPU a následně provedli analýzu, které části kódu spotřebovávají nejvíce výpočetních prostředků. Tímto způsobem určili části algoritmu pro urychlení a ty se pak následně snažili akcelerovat použitím GPU. Například vyhodnocení algoritmu AdaBoost dle Viola a Johnes zabíral *66,95 %* času (ohodnocení kaskády klasifikátorů). Zbývající čas je potřebný pro načtení snímku, uložení snímku, předzpracování snímku a podobně. Autoři tak pro akceleraci na GPU vybrali právě tuto část. První z optimalizací, kterou autoři prováděli, se týkala úrovně paralelizace ohodnocení kaskády klasifikátorů. Autoři došli pomocí experimentů k hodnotě *32 x 32* či *64 x 64* vláken v závislosti na použité velikosti vstupního obrazu. Počet vláken *32 x 32* se dle autorů však jeví být tím nejvhodnějším. Optimalizace paměťových operací byla provedena za pomoci vytvoření vyrovnávací paměti mezi globální pamětí CPU a jednotlivými registry výpočetních jader. Tím bylo zajištěno zmenšení počtu přístupů do globální paměti CPU a zvýšení rychlosti výpočtu. Autoři v jedné z části své práce také uvádějí, že provedli implementaci i WaldBoostu na GPU, kde dosáhli rychlosti zpracování *196* snímků za sekundu při rozlišení vstupního obrazu *512 x 512* pixelů, což je v přepočtu *51,4* Mpps.

V tabulce 1 je uvedeno porovnání dosažených výsledků několika vybraných prací. Z výsledků je možné určit, že současný hardware grafických karet je dostačující pro zpracování obrazu v HD rozlišení (*1280 x 720* pixelů) při *50* snímcích za sekundu a stejně tak by měl postačovat i pro zpracování FullHD obrazu (*1920 x 1080* pixelů) při *25* snímcích za sekundu. Použití GPU však není vždy možné, jelikož výpočetní jednotka s GPU má velké rozměry a také vyžaduje nemalé nároky na napájení (spotřeba CPU + spotřeba GPU). Není tak možné vytvořit velmi malé a energeticky nenáročné zařízení. Stejně tak pro napsání kódu pro GPU jednotku je třeba vyvinout určité úsilí navíc, než je tomu při klasické implementaci na CPU.

Tabulka 1: Dosažené rychlosti známých implementaci na GPU.

Autor	**Rychlost zpracování**		**Hardware**
	Rozlišení; počet snímků	**Mpps**	**CPU/GPU**
Herout [20]	1280 x 720 px; 60 FPS	55,5	Intel Core I7-920 NVIDIA GTX280
Chouchene [32]	1024 x 1024 px; 124 FPS	117,8	Intel Core i5 2.6 GHz NVIDIA GeForce 310 M
Pertsau [61]	1280 x 960 px; 25 FPS	30,2	Intel Core i7-920 2,66 GHz NVIDIA GeForce GTX 460
Sharma [69]	1280 x 960 px; 19 FPS	23,3	Intel Xeon 3.33 GHz NVIDIA GeForce GTX 285

3.3 FPGA

V podkapitole 3.1 byly prezentovány vybrané implementace obrazových klasifikátorů na CPU a na jejich základě lze stanovit, že výkonnost dnes používaných CPU společně s doposud používanými metodami výpočtu není dostatečná pro zpracování obrazu s vysokým rozlišením (FullHD) v reálném čase. V kapitole 3.2 bylo ukázáno, že použití GPU může přinést dostatečný výkon pro zpracování

FullHD obrazu na GPU se spojení s CPU. Toto řešení má však své nevýhody a pro řadu aplikací je nepoužitelné. Jedním z příkladů takového použití může být u zařízení, která mají požadavek na nízkou spotřebu elektrické energie (*<10*W) a také u zařízení, která musí splňovat omezení z hlediska velikosti. GPU je tak téměř nemožné použít v malých vestavěných systémech, jelikož GPU potřebuje pro svoji práci vždy i CPU. Stejně tak i pokud bychom potřebovali systém s extrémně velkou propustností dat, tak by se u GPU narazilo na technologické limity. Uvedené problémy a omezení však mohou být eliminovány, použije-li se technologie FPGA.

Vývoj architektur klasifikátorů běžících na FPGA probíhá již téměř 10 let a za tu dobu bylo představeno mnoho rozmanitých variant implementací. Jako první klasifikátory na FPGA se objevily implementace založené na neuronových sítích. Tyto první implementace byly většinou cíleny na jeden specifický hardware a byly určeny pro zpracování jednoho specifického problému. Příkladem takových implementací jsou například [76], [57], [21]. Autoři v těchto pracích jen zřídkakdy zvažovali AdaBoost za alternativu k jimi uvedeným řešením. Ačkoliv je AdaBoost univerzální klasifikátor, tak se první implementace na FPGA velmi často objevovaly jen ve spojení s detekcí obličejů. Tento jev je dán tím, že detekce obličejů patří mezi snadnější typy úloh, ale zato v praxi velmi žádaných a využívaných. Ale také tím, že si jej vybírá mnoho autorů, a tak se snadněji provádí porovnání výsledků. Příklady takových implementací jsou [77], [76], [22], [31], [83], [67], [48] a [14]. Většina uvedených řešení je založena na základních principech, jež byly představeny v práci Theocharidese [79]. Těmito základními principy je využití Haarových příznaků a s nimi spojený výpočet integrálního obrazu. Tyto implementace pak přistupují k výsledkům integrálního obrazu jako k systolickému poli (homogenní síť úzce svázaných paměťových buněk) raději, než aby použili centralizovanou nebo dokonce distribuovanou paměť.

Hiromoto ve své práci [22] použil koncept rozdělení skupiny slabých klasifikátorů na dvě části. První část slabých klasifikátorů pak byla vykonávána paralelně a druhá část sekvenčně. Tato optimalizace vede k tomu, že slabé klasifikátory, jež jsou často vyhodnocovány, jsou implementovány paralelně a jejich vyhodnocení je rychlé, a klasifikátory, jež se používají jen zřídkakdy, jsou implementovány sekvenčně, tedy pomalým způsobem. Tento přístup tak nutně vede k tomu, že při každé změně klasifikátoru je třeba syntetizovat nový design klasifikátoru. Poznamenejme, že tato vlastnost není omezující, pokud se jí vhodně využije, jak bude ukázáno v podkapitole 5.2.6. Implementace AdaBoost klasifikátorů využívající Haarovy příznaky bývaly velmi často implementovány tak, že neprováděly změnu rozlišení obrazu, aby našly předměty různých rozlišení, ale provádějí změnu velikosti Haarových vlnek. Změna velikosti Haarových vlnek při použití integrálního obrazu je velmi jednoduchá, efektivní a téměř nevyžaduje žádnou další přidanou práci. To je dáno výpočtem Haarových příznaků, jak je definováno v podkapitole 2.4.1. Propustnost řešení navrženého Hiromotem je *30* snímků za sekundu při rozlišení *640 x 480* pixelů.

Autor Cho [31] přišel s implementací, která provádí změnu velikosti Haarových příznaků, ale i změnu rozlišení samotného obrazu. Příznaky jsou tak v jeho práci zvětšovány a obraz je zmenšován. Práce je mimo jiné zajímavá tím, že přišel s paralelním zpracováním všech úrovní změn rozlišení. Tím se mu podařilo zvýšit propustnost systému na *7* snímků za sekundu při rozlišení obrazu *640 x 480* pixelů. Jeho implementace však vyžadovala mnoho paměti pro implementaci obrazové pyramidy, která byla použita v rámci změny rozlišení vstupního obrazu.

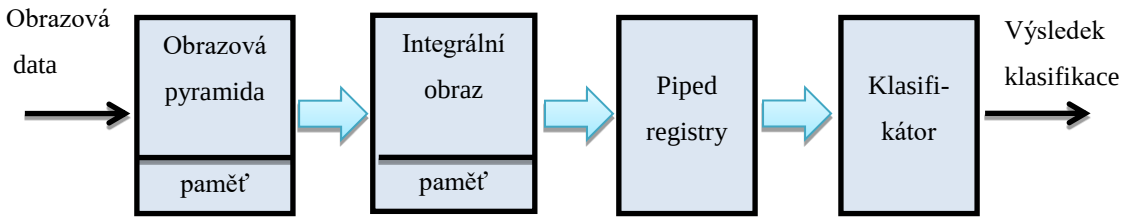
Zjednodušená verze Choova přístupu byla implementována v práci autorů Wei a Bing [83]. Tato implementace byla význačná počtem použitých slabých klasifikátorů, jejichž počet byl výrazně redukován právě díky změně rozlišení obrazu. Redukce počtu příznaků byla z 2135 v [31] versus 225 v [83]. V této práci byla opět použita obrazová pyramida pro generování a následně zpracování

snímků v různém rozlišení. Propustnost této implementace je *15* snímků za sekundu při rozlišení *640 x 480* pixelů.

Huang [27] se ve své práci pokusil redukovat nároky na použité FPGA zdroje pro uložení integrálního obrazu pomocí multiplexované sítě a vhodných podmínek při trénování. Shi ve své práci [67] představil vylepšený koncept pro práci s integrálním obrazem, který spočíval v zavedení vertikální a horizontální řetězené linky. Vertikální řetězená linka počítala standardní integrální obraz, ale horizontální řetězená linka počítala obdélníkové příznaky. Tím bylo dosaženo dalšího urychlení a celková propustnost klasifikátoru navrženého řešení je *102* snímků za sekundu při rozlišení *640 x 480* pixelů.

3.3.1 Implementace Lai [48]

Řešení uvedené autory Lai a kolektiv [48] je zajímavé, jelikož se jako první známé řešení snaží využít potenciálu FGPA a rozvíjí tak masivní paralelizmus. Autoři využívají ve své práci AdaBoost ve spojení se slabými klasifikátory založenými na Haarových vlnkách. Uvedené řešení dokáže zpracovat jedno detekční okno za jeden hodinový cyklus, což je velmi podobné řešení, které bude představeno v kapitole 5. Obě architektury se však liší v principu vyhodnocení. Architektura navržená autorem Lai je založena na speciální paměti (pole registrů) pro uložení výsledků integrálního obrazu, tak aby se ke každému výsledku mohlo přistupovat v každém hodinovém cyklu, a případně i z více míst. Paměť potřebná pro tuto komponentu však není zanedbatelná.



Obrázek 19: Blokové schéma architektury navržené v [48].

Na obrázku 19 je zobrazeno blokové schéma architektury navržené autory Lai a kolektiv. Architektura je založena na 4 základních komponentách. První komponentou je generátor obrazové pyramidy, který generuje všechna požadovaná rozlišení vstupního obrazu. Následující komponentou je FIFO vyrovnávací paměť integrálního obrazu. Vyrovnávací paměť pojme celkem *22* řádků obrazu (dáno velikosti detekčního okna). Paměť tedy slouží pro ukládání pruhu obrazu o výšce *22* řádků. Každá paměťová buňka má velikost *32* bitů, tak aby v ní mohl být uložen integrální obraz. Vyrovnávací paměť primárně slouží k výpočtu a také k ukládání výsledků integrálního obrazu. Je konstruována tak, že má jeden čtecí port a *21* výstupních portů, které jsou připojeny na navázanou jednotku. Autoři uvádějí, že použitím jejich přístupu se ušetří až *99,7 %* paměti oproti standardnímu přístupu. Další připojenou jednotkou jsou takzvané *Piped registry*, jež mají zajistit velmi rychlý přístup k datům integrálního obrazu, nad nimiž se právě vyhodnocuje. Piped registry jsou tvořeny maticí *21 x 21* registrů. Tyto registry následně umožňují zpracovat všechny příznaky v detekčním oknu najednou, a vyhodnotit tak detekční okno za jeden takt. Každý registr uchovává *32* bitů dat. Poslední připojenou jednotkou je již samotný klasifikátor s Haarovými příznaky. Autoři uvádějí, že používají pouze příznaky, které mají *6*, *8* nebo *9* vstupních hodnot. To znamená, že příznaky mají *2*, *3* nebo *4* čtvercové oblasti. Příznaky použité autory jsou dostatečně obecné pro danou úlohu.

Klasifikátor, který však autoři použili v práci, má jen omezenou množinu *52* příznaků a nedosahuje stejných klasifikačních výsledků jako jiné dříve uvedené architektury. U testů provedených na MIT-CMU data setu byla úspěšnost pouze *86 %*. Uvedená architektura byla implementována na FPGA Virtex-II Pro XC2VP30, což je dnes již poměrně zastaralé FPGA. Autoři uvádějí, že maximální dosažitelná rychlost jejich implementace pro uvedené FPGA je *143* snímků za sekundu při rozlišení obrazu *640 x 480* pixelů. Maximální dosažitelná frekvence udávaná autory je *126* MHz. Tabulka 2 ukazuje nároky na FPGA zdroje pro uvedené řešení. Z tabulky lze vidět, že nároky na zdroje jsou poměrně velké vzhledem k použití pouhých 52 slabých klasifikátorů.

Tabulka 2: Nároky na FPGA zdroje, Lai, Virtex-II Pro XC2VP30 [48].

	Počet zdrojů		**Využití**
	Dostupné	**Použité**	
Slices	13696	11505	84 %
Slice Flip Flops	27392	7872	28 %
4 input LUT	27392	20901	76 %
BRAM	136	44	32 %
GCLKs	16	1	6 %

Spotřebované zdroje v FPGA jsou vyjádřeny za pomoci výčtu FPGA komponent, které spotřebují pro svou implementaci. Význam jednotlivých komponent je následující:

- *Slice* – Základní konfigurovatelná jednotka na FPGA, která se skládá z *Flip Flop*, *LUT* a multiplexorů.
- *Slice Flip Flop* – klopný obvod, má dva stabilní stavy a může uložit jeden bit.
- *Slice Register* – skupina *Flip Flop* obvodů pro uložení několika bitů.
- *4 input LUT* – čtyřvstupové vyhledávací tabulky, ukládají předefinované hodnoty a implementují logické funkce.
- *BRAM* – Bloková paměť RAM – *BlockRAM*.
- *GCLKs* – generátor hodin.
- *DSP* – jednotka pro zpracování digitálních signálů (operace sčítání, násobení a podobně).

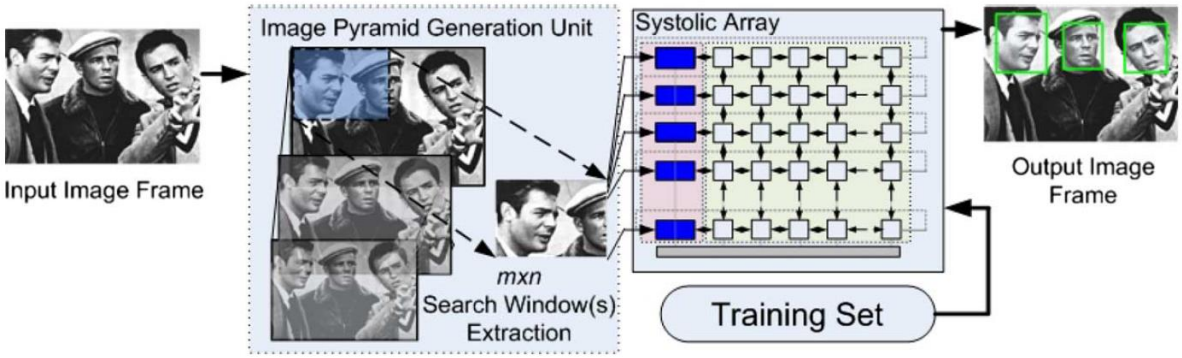
3.3.2 Implementace Kyrkou [47]

Kyrkou a Theocharides představili ve své práci [47] klasifikátor, o kterém tvrdí, že je to plně generická hardwarová implementace AdaBoostu. Autoři ve své práci uvádí, že předchozí implementace byly vždy svázány s řadou omezení, jako je například omezená nebo fixní velikost vstupního obrazu, detekce jen jednoho typu objektů či dokonce detekce například jen omezeného počtu objektů stejného typu v obraze. To znamená, že dřívější implementace dokázaly detekovat například jen 5 objektů v jednom snímku. Autoři vytvořili tedy plně uživatelsky generickou architekturu, která se má vypořádat se všemi dříve uvedenými omezeními. Architektura může přijímat obraz v libovolném rozlišení, může detekovat libovolný počet objektů stejného typu, může být použita pro detekci různých typů objektů a autoři uvádějí jako hlavní přednost to, že se architektura nemusí měnit, dojde-li ke změně klasifikátoru. To znamená, že se pouze natrénuje nový klasifikátor, který se dá na vstup klasifikátoru v FPGA. Architektura má tedy představovat první zcela generický AdaBoost.

Autoři v práci detekovali pět hlavních oblastí, se kterými se musejí efektivně vypořádat. Jsou jimi:

- změna rozlišení obrazu,
- výpočet integrálního obrazu,
- výpočet příznaků,
- výpočet stages a
- identifikace regionů s objektem zájmu.

Architektura představená autory se skládá ze dvou základních bloků. Prvním je generátor obrazové pyramidy (IPG) a druhou částí je takzvané systolické pole. IPG generuje regiony obrazu a systolické pole je zpracovává. Systolické pole ohodnocuje kandidátní regiony obrazu, které mohou potencionálně obsahovat detekované předměty. Autoři použili hybridní systém změny rozlišení vstupního obrazu, který jednak provádí zmenšování vstupního obrazu, ale také provádí změnu velikosti příznaků (zvětšování) dle původního schématu představeného autory Viola a Johnes [82]. IPG generuje takzvané vyhledávací okno, které je zpravidla větší než detekovaný předmět (například *80 x 60* pixelů), a v tomto okně provádí detekci a především změnu velikosti Haarových příznaků. Vyhledávací okno je zpracováváno paralelně příznak za příznakem, *stage* za *stage* pomocí systolického pole. To provádí výpočet integrálního obrazu, výpočet obdélníků pro každý příznak, ohodnocení příznaků a také výpočet výsledků celé *stage*. Po kompletním zpracování vyhledávacího okna je pomocí IPG vygenerován obsah nového vyhledávacího okna, dokud není obraz kompletně zpracován. Výstupem systolického pole jsou pozice a velikosti detekovaných předmětů. Blokové schéma architektury je dáno na obrázku 20.

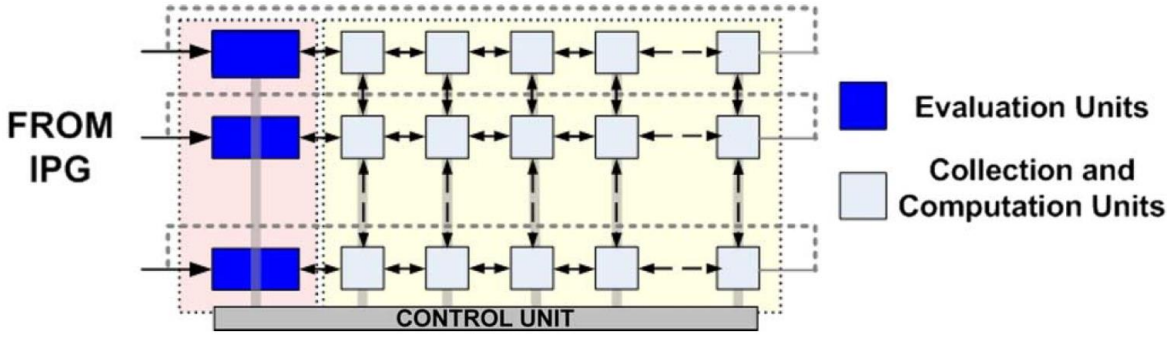


Obrázek 20: Blokové schéma AdaBoost klasifikátoru navrženého autorem Kyrkou [47].

IPG generuje vyhledávací okno o velikosti *m x n* pixelů, jehož rozměr je dán velikostí systolického pole. Velikost systolického pole je zpravidla omezena dostupnými prostředky na FPGA. IPG a systolické pole pracují v režimu zřetězeného zpracování (*pipelining*). IPG se skládá ze tří částí. První část je přijímací, která přijímá pixely obrazu z obrazové vyrovnávací paměti; další část je dělící, která generuje jednotlivá vyhledávací okna, a části produkující obraz v různých stupních rozlišení. IPG předává do systolického pole vždy celý první sloupec vyhledávacího okna.

Systolické pole provádí převážnou část výpočtů a je složeno ze dvou typů výpočetních elementů (PE). Prvním je shromažďovací a výpočetní jednotka (CCU), druhým typem jsou ohodnocující jednotky (EU). Struktura systolického pole je na obrázku 21. EU jednotky jsou umístěny nalevo jako první buňka v každém řádku. Zbytek pole je pak tvořen CCU jednotkami. Každá EU jednotka je propojena se svou sousední CCU jednotkou na řádku a také s poslední CCU jednotkou na řádku ve

smyslu kruhového propojení. Systolické pole také obsahuje řídící jednotku (CU), která globálně řídí všechny prvky pole. CU slouží jako koordinátor ohodnocení vyhledávacího okna a je v podstatě odpovědná za správné vykonání dle principu AdaBoost. CCU jednotky pracují synchronně a mezi sousedními jednotkami je vytvořeno obousměrné propojení. Minimální velikost systolického pole je dána originální velikostí použitých příznaků. Se zvětšením systolického pole se zvýší i rychlost zpracování, jelikož je pak možné v jednom kroku zpracovat více detekčních oken. Horní hranice velikosti je dána především použitým FPGA.



Obrázek 21: Systolické pole [47].

Systolické pole pracuje následujícím způsobem. Jako první je vypočten integrální obraz. Následně jsou vypočteny všechny obdélníkové oblasti pro všechny příznaky, a to paralelně a pro všechny úrovně velikostí příznaků. Dále je provedeno ohodnocení všech *stages* paralelně v rámci jednoho vyhledávacího okna pro získání výsledků *stages*. Regiony, které nebyly detekovány žádnou *stages,* jsou označeny za nevalidní a vyloučeny z dalšího zpracování. Detailní popis CCU a EU je dán v [47]. Veškeré operace v systolickém poli jsou založeny na lokálních výpočtech, které však potřebují získat data od svých sousedů. Pro každou operaci je tak třeba provést řadu paralelních přenosů dat, a tedy každá z operací, kterou systolické pole provádí, spotřebuje určitý čas, který musí být delší než 1 hodinový cyklus. Výpočet integrálního obrazu například zabere pro obraz o velikosti *m x n* pixelů *2·[m+(m-1)+(n-1)]* hodinových cyklů. Pro výpočet obdélníkových oblastí je třeba *dx + dy* operací pro získání dat. Kde *dx* a *dy* jsou dány offsetem souřadnic počítaného obdélníku. Následně je třeba udělat výměnu výsledků detekčních oken, která opět zabere *2n* hodinových cyklů, kde *n* je dáno počtem obdélníků v Haarově příznaku. Výsledky jednotlivých obdélníků se shromažďují v EU jednotkách, které provádějí jejich ohodnocení na základě dat získaných při trénování. Po výpočtu všech příznaků je proveden výpočet *stages* a na základě těchto výsledků jsou pak jednotlivé regiony obrazu označovány, zda obsahují hledaný předmět nebo ne.

Autoři spatřují hlavní přínosy práce v tom, že je architektura paralelní, ve velké rychlosti zpracování systolického pole, v konstantní rychlosti zpracování nezávisle na počtu detekovaných předmětů v obraze a možnosti přizpůsobení architektury na požadavky aktuálního řešení, což je dostupné místo na FPGA a případně rychlost zpracování.

Pro ověření konceptu své práce autoři vytvořili testovací implementaci na Xilinx XUP Virtex II Pro platformě – XC2VP30 [88]. Pro uložení výsledků trénování autoři použili externí paměťové moduly. Nároky na implementaci v FPGA jsou uvedeny v tabulce 3. Autoři také provedli implementaci v ASIC (*Application Specific Integrated Circuit*) technologii. Tyto výsledky jsou dostupné v jejich práci.

Maximální pracovní frekvence uvedeného řešení na FPGA XC2VP30 je 100 MHz. Při této frekvenci udávají autoři propustnost architektury *64* snímků za sekundu při rozlišení obrazu *640 x 480*

pixelů. Přesnost implementovaného klasifikátoru je *93 %*. Autoři stejně jako Lai [48] použili Haarovy příznaky, které se skládají ze *2*, *3* nebo *4* obdélníkových oblastí.

Autoři ve svém článku tvrdí, že architektura je plně generická a není třeba ji měnit s novým klasifikátorem. Toto však v praxi neznamená, že může být jeden implementovaný klasifikátor na FPGA a jen mu měnit vstupy (natrénovaná data klasifikátorů), ale ve skutečnosti to znamená, že pro každý nový klasifikátor je třeba přegenerovat popis hardwarových jednotek, tedy předně upravit velikost systolického pole, které je závislé například na velikosti detekovaných předmětů. Nemění se tedy architektura jako celek, ale musí se změnit implementace.

Autoři ve svém díle přestavují architekturu jako paralelní, avšak z globálního hlediska se nejedná o plně paralelní architekturu, ale o architekturu sekvenční, která má však optimalizovány jednotlivé své části pomocí paralelního vykonávání. Nelze ji tedy považovat za plně paralelní implementaci, ale jen za pseudoparalelní implementaci. Potenciál FPGA pro paralelizaci tak nebyl ani v této velmi slibné práci plně využit.

Tabulka 3: Nároky na FPGA zdroje Kyrkou, Xilinx Virtex II Pro XC2VP30.

	Počet zdrojů		**Využití**
	Dostupné	**Použité**	
Slices	136	68	50 %
Slice Flip Flops	27392	23744	87 %
4 input LUT	27392	25818	94 %
BRAM	136	24	17 %
GCLKs	16	1	6 %

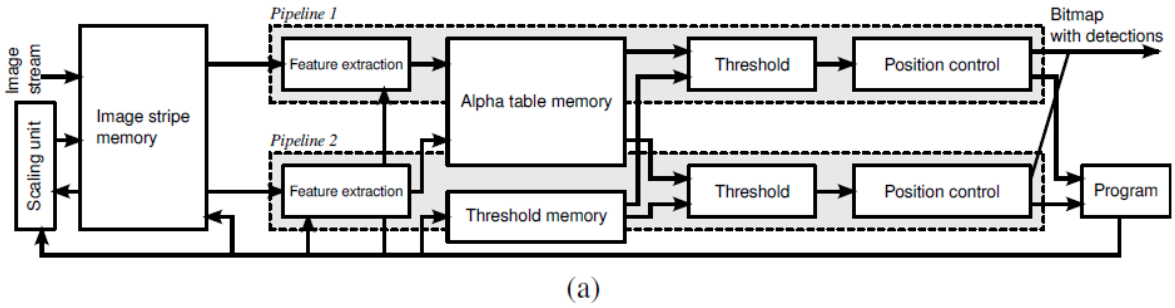
3.3.3 AdaBoost engine - Zemčík [96]

Zemčík a Žádník představili první implementaci AdaBoost engine již v roce 2007 v práci [95]. Engine je založen na využití principu jemného multi-threadingu (viz podkapitola 3.1.4) kombinovaného se zřetězeným zpracováním. Multi-threading je použit především pro překrytí zpoždění generovaného v zřetězené lince. V této práci byl vytvořen klasifikátor malých rozměrů, ale poměrně dobrých parametrů. Architektura byla implementována na FPGA Virtex-II 250 [88], kde dokázala zpracovat *22* snímků za sekundu při rozlišení *640 x 480* pixelů a frekvenci FPGA *100* MHz. Engine potřebuje *2980 Slices* a *14 BlockRAM* pamětí k tomu, aby mohl implementovat klasifikátor s pouhými 20 slabými klasifikátory. Tyto nároky jsou poměrně malé. Klasifikátor implementuje takové malé množství slabých klasifikátorů, jelikož jeho hlavním účelem mělo být použití jako předzpracovávající jednotka. Engine byl také omezen z hlediska šířky zpracovávaného obrazu na *352* pixelů. Pro zpracování širšího obrazu bylo nutné obraz zpracovávat po sloupcích s překryvem *128* pixelů. Zvolený klasifikátor v této práci byl AdaBoost ve spojení s LRD příznaky (i když autoři práce uvádějí, že architektura byla připravena i na použití WaldBoostu). Architektura se vyznačovala ukládáním malého proužku obrazu do paměti. Detekce objektů různých velikostí byla umožněna pomocí externího DSP, které provádělo výpočet obrazové pyramidy s požadovanými úrovněmi rozlišení. Tato implementace se také vyznačuje velmi složitou logikou pro adresování pixelů obrazu, které se mají vyhodnotit v rámci slabých klasifikátorů.

Později Zemčík ve spolupráci s dalšími autory svůj engine přepracoval do nové podoby, kterou prezentoval v práci [96]. V této práci je navržena vylepšená detekce předmětů s různou velikostí, a tedy i změna rozlišení vstupního obrazu a proudové zpracování v porovnání s předchozí

implementací [95]. V této práci autoři použili jak LRD, tak i LBP příznaky pro vytvoření slabých klasifikátorů. Jako silný klasifikátor byl nyní zvolen již WaldBoost, který již ze svého principu přinesl zvýšení rychlosti zpracování. Autoři provedli změnu trénovacího algoritmu tak, aby minimalizovali vliv kvantizační chyby způsobené použitím celočíselné reprezentace hodnot v tabulce alfa koeficientů. V architektuře zůstala zachována vyrovnávací paměť pro ukládání tenkého pásku vstupního obrazu přímo na čipu FPGA, který má sloužit k velmi rychlému přístupu k datům, nad kterými se právě pracuje.

Na obrázku 22 je vyobrazeno blokové schéma nového engine. Ten pracuje jako programovatelný automat, jež je řízen *64*bitovou instrukční sadou. Klasifikátor je vyhodnocován na každé pozici v obraze. Engine implementuje zřetězené zpracování za účelem paralelizování zpracování jednotlivých slabých klasifikátorů. O rychlý přístup k obrazovým datům se stará jednotka pro ukládání pásu obrazu. V této jednotce však není uložen původní *8*bitový obraz ve stupních šedi, ale pouze obraz s *6*bitovým rozlišením pro každý pixel obrazu. Autoři uvádějí, že touto redukcí nejsou ovlivněny detekční schopnosti uvedeného řešení a současně je tímto dosaženo určité úspory paměti na FPGA čipu. Z vyrovnávací paměti pro pás obrazu se vyčítá oblast *6 x 6* pixelů, na které se provádí vyhodnocení slabých příznaků. Paměť je uzpůsobena tak, aby bylo možné tuto oblast vyčítat každý hodinový cyklus.



Obrázek 22: Architektura AdaBoost engine [96].

Autoři v práci navrhli dva způsoby, jak provádět změnu rozlišení obrazu. Prvním způsobem je jemná změna rozlišení obrazu, kdy se vstupní obraz postupně zmenšuje vždy na *5/6* původní velikosti v prvních *8* krocích. A následně pak vždy na *1/2* ve zbývajících krocích. Tímto způsobem je ušetřena část výpočetního výkonu. Druhý způsob, který byl navržen autory, spočívá v hrubé změně rozlišení obrazu vždy na *1/2* velikosti, ale aby dodrželi celkový poměr zmenšování 5/6, tak provádějí změnu velikosti příznaků směrem nahoru. Změna velikosti příznaků je v této práci v podstatě prováděna pomocí dalšího klasifikátoru, kdy pro každou úroveň změny velikosti příznaků je vyhodnocen klasifikátor s jinými velikostmi příznaků. Celkem pro vyhodnocení tak musejí být použity *4* klasifikátory lišící se právě velikostí příznaků. Jak je možné již z tohoto popisu vidět, tak tento přístup povede zcela jistě k implementaci s menším datovým tokem, ale využitím tohoto přístupu by mělo dojít k úspoře paměťových zdrojů, které jsou potřebné pro implementaci (pro implementaci s LBP došlo k významnému poklesu o cca *40 %*, ale u implementace s LRD byl pokles téměř zanedbatelný). Tento způsob zpracování byl ale autory následně zavržen, jelikož poskytoval horší výsledky. LRD a LBP příznaky použité v práci jsou omezeny na maximální velikost *6 x 6* pixelů. Pro vyhodnocení příznaků jsou implementovány speciální jednotky.

Blokové schéma na obrázku 22 obsahuje jednotku pro extrakci příznaků, které jsou načítány z proužkové paměti, paměť alfa tabulek, která slouží k ohodnocení hypotéz, jednotku pro porovnávání částečných výsledků s prahem a paměť instrukcí. V architektuře jsou dvě linky zřetězeného zpracování

pro výpočet klasifikátoru. Některé komponenty, jako je například paměť instrukcí a paměť alfa koeficientů, jsou sdílené, ale samotné výpočetní jednotky jsou replikovány.

Autoři provedli implementaci na FPGA Xilinx Spartan 6 LX45T, které bylo na vývojové desce Xilinx SP605. Použitý detektor obsahoval pouhých *128* slabých klasifikátorů. Tabulka 4 udává počet zdrojů potřebných pro implementaci na vybraném FPGA v konfiguraci LRD 5/6 (nejvýkonnější implementace). Použitím LBP namísto LRD se cca dvojnásobně zvýší spotřeba pamětí BRAM a navíc klesne cca o *1/4* rychlost zpracování dat.

Tabulka 4: Spotřeba zdrojů AdaBoost engine v konfiguraci LRD 5/6.

	Počet zdrojů		**Využití**
	Dostupné	**Použité**	
Slice Flip Flops	54576	1732	3 %
4 input LUT	43661	7373	27 %
BRAM	116	31	31 %

Propustnost jejich implementace je *163* snímků za sekundu při rozlišení *640 x 480* pixelů. Pro implementaci uvedeného řešení je třeba poměrně málo zdrojů na FPGA a autoři uvádějí, že na jednom FPGA může být instanciováno více engine pro zvýšení rychlosti zpracování nebo pro detekci několika typů objektů najednou. Ale vzhledem k omezením, jež jsou u implementované architektury (128 slabých klasifikátorů, rozlišení pouze *640 x 480* pixelů), se dá očekávat, že sestavením klasifikátoru, který by měl alespoň *500* slabých klasifikátorů LBP nebo LRD a zpracovával by obraz o rozlišení minimálně FullHD, by se významně změnila spotřeba zdrojů na FPGA (především pamětí BRAM) a zřejmě by i poklesla snímková rychlost architektury v důsledku výrazného navýšení počtu detekčních oken, které by se musely zpracovat. Výsledky této práce je tak třeba posuzovat v omezeních, které si stanovili autoři, a nelze je přímo srovnávat například s řešením vytvořeným Kyrkou [47]. Architektura však může být stejně jako její předchůdce velmi výhodně použita v systémech, kde má pracovat jako malá předzpracovávající jednotka. Pro takové systémy je architektura autory zřejmě zamýšlena a v takových systémech výrazně vyniknou její přednosti.

3.3.4 Obličejový detektor dle He [16]

Autor He a kolektiv ve své práci [16] představili velmi rychlý obličejový detektor. Jejich řešení překonává v rychlosti zpracování násobně doposud známá řešení pro detekci objektů. Toto řešení se však nedá považovat za přímého konkurenta této práci či ostatním řešením představeným v předchozích podkapitolách. Je zde však uvedeno především pro porovnání výsledků univerzálního řešení a čistě specializovaného řešení pro jeden typ úlohy. He prezentuje svoji práci jako SoC (*Systém on Chip*) architekturu pro ultra rychlou detekci obličejů ve videu. Práce je založena na efektivním a robustním algoritmu, který používá kaskádu klasifikátorů založených na neuronových sítích, které jsou však dále založeny na natrénovaných Haarových příznacích pomocí AdaBoostu. Práce však pro dosažení velké rychlosti využívá ještě dvou dalších metod pro před-detekci zájmových zón. První metodou je detekce a sledování pohybu zájmových předmětů a druhou metodou je před-detekce pomocí barevného schématu lidské kůže.

Autoři ve svém přístupu používají integrální reprezentaci obrazu, aby mohli rychle počítat výsledky Haarových vlnek. Architektura se skládá ze tří základních částí předzpracování obrazu, detekční části a následného zpracování obrazu. V jednotce pro předzpracování je provedena změna rozlišení vstupního obrazu (autoři pracují s obrazem v rozlišení *640 x 480* pixelů) na velikost pouhých

80 x 60 pixelů. Na takto velkém snímku provedou detekci pohybu a detekci lidské kůže. Detekce pohybu použitá v práci je založena na rozdílu dvou následujících snímků, což vede k použití vyrovnávací paměti pro předcházející snímek. Detekce kůže je založena na barvě pixelů a je pro ni použito třívrstvé umělé neuronové sítě s vstupní vrstvou s třemi neurony pro červenou, zelenou a modrou složku obrazu, šest neuronů pro zpracování barevné informace obrazu ve skryté vrstvě a poslední vrstva pro generování výsledků. Ve druhé části je generována integrální reprezentace obrazu a je provedeno vyhodnocení kaskádového klasifikátoru. Vyhodnocení je však provedeno pouze v oblastech, jež jsou označeny v před-detekční fázi, a probíhá na obraze o velikosti *80 x 60* pixelů. V obraze se pak hledají objekty jen tří různých velikostí dle přednastavených masek. Klasifikace je prováděna pomocí kaskády devíti *stages* neuronových sítí, které využívají jako vstup slabé klasifikátory založené na Haarových vlnkách. V poslední části následného zpracování jsou detekované výsledky použity pro eliminaci artefaktů a šumu (to je například pro odstranění vícenásobné detekce jedné tváře).

Autoři implementovali architekturu na FPGA Xilinx Virtex-5 FPGA (XC5VFX130T) a použili vývojovou desku ML510. Tabulka 5 uvádí nároky na zdroje spotřebované architekturou v FPGA. Propustnost architektury je na velmi vysoké úrovni a dosahuje *625* FPS při rozlišení obrazu *640 x 480* pixelů při frekvenci *73* MHz. Tato hodnota je mnohem vyšší než u dříve uvedených architektur. Ale stejně tak vzroste i spotřeba zdrojů, a to především pamětí *BlockRAM*. Nutno podotknout, že použitá implementace pro rozpoznávání barvy kůže je velmi často citlivá na světelné podmínky a také na to, jaká barva pleti se detekuje.

Tabulka 5: Spotřebovaných zdrojů He [16] pro Xilinx Virtex-5.

	Počet zdrojů		**Využití**
	Dostupné	**Použité**	
Slice Flip Flops	81920	37828	46 %
4 input LUT	81920	67704	83 %
BRAM	298	276	93 %
DSP48Es	320	161	50 %

3.4 Porovnání dosažených výsledků známých architektur

V tabulce 6 jsou uvedeny výsledky dosažené jednotlivými architekturami. Výsledky jsou získány pro vstupní obraz o rozlišení *640 x 480* pixelů mimo práci autora Cho, jehož výsledky jsou uvedeny pro poloviční rozlišení obrazu. Rozlišení, na základě kterého je provedeno porovnání, je dnes již nepostačující, ale bohužel nejsou známy další možnosti uvedených architektur vzhledem k možnosti zpracování obrazu s velkým rozlišením (FullHD a více). Dá se však spekulovat, že řada architektur, které jsou uvedeny v tabulce, nejsou postaveny pro zpracování obrazu s tak velkým rozlišením a jejich přepracování na možnost zpracovávat takový obraz by zabralo nemalé úsilí. Předně by zřejmě u některých architektur výrazně narostly požadavky na FPGA zdroje. Stejně tak, pokud by měly uvedené architektury pracovat na své maximální rychlosti, tak by téměř každá architektura potřebovala vyrovnávací paměť vstupních snímků, jelikož nemají konstantní čas zpracování jednoho snímku s výjimkou Kyrkou implementace, která ji má konstantní. Jedním z největších omezení uvedených architektur je jejich sekvenční přístup ke zpracování obrazu. Téměř každá z uvedených architektur popisuje, jak paralelizovala výpočet, ale po delším prozkoumání čtenář zjistí, že se jedná jen

o paralelizaci určité části algoritmu. Nejdále v paralelizaci dospěl autor Kyrkou, ale i jeho řešení je z globálního pohledu sekvenční s paralelizací jednotlivých kroků. Potenciál FPGA tak zůstal nevyčerpán a stále je v tomto oboru co zkoumat a vylepšovat.

Tabulka 6: Dosažené výsledky jednotlivých implementací pro rozlišení obrazu 640x 480 pixelů.

	Zemcik [95]	**Hiromoto [22]**	**Cho [31]**	**Wei [83]**	**Shi [67]**	**Lai [48]**	**Kykou [47]**	**Zemcik [96]**	**He [16]**
FPGA	Virtex-II 250	XC5VLX330-2	XC5VLX110	XC2V2000	-	XC2VP30	XC2VP30	Spartan6 LX-45T	XC5VFX130T
Snímků za sekundu	22	30	26 (320x240)	15	102	143	64	164	625
Slices LUTs	2980	63443	66851	13853	-	20901	25818	7014	67704
Slices Flip Flop	-	55515	21902	4573	-	7782	23744	1673	37828
BRAM paměti	14	-	41	56	-	44	24	29	276
Násobičky	0	-	-	28	-	-	68	-	-
Frekvence (MHz)	100	160,9	-	91	200	126	100	152	73
Přesnost detekce	-	-	-	85 %	-	86 %	93 %	~91 %	-
Změna rozlišení	není	0,8	0,8	0,75	-	0,75	0,75	0,8	-
Počet slabých klasifikátorů	20	2913	2135	225	2913	52	2913	128	115
Stages	-	25	22	3	25	1	25	-	9

4 Návrh nových tvarů LBP operátorů

Práce [4] ukazuje, že vytvořením nových příznaků a také nových metod extrakce příznaků lze dosáhnout významného vylepšení přesnosti klasifikace a v řadě případů to může vést k lepším výsledkům než vytvoření zcela nové klasifikační metody. Tato kapitola je tak zaměřena na návrh nových aplikačně specifických tvarů příznaků pro LBP, LRP, LRD nebo LR operátory (všechny uvedené operátory mohou využívat stejnou množinu tvarů). V současných implementacích, které využívají LBP klasifikátory, se používají jen základní tvary masek (matic) pro výpočet příznaků. Tyto základní tvary byly navrženy lidmi a nevyužívají potenciál možného přizpůsobení vzhledem k vybrané třídě problémů. Lidmi navržené příznaky sice pracují v širokém spektru úloh velmi dobře, ale téměř pro žádnou úlohu nejsou nejvhodnější.

V této kapitole je představena metoda návrhu nových tvarů operátorů pro řešení jednoho konkrétního problému. Při takovémto přístupu je velká pravděpodobnost, že se podaří nalézt nový tvar operátoru, který bude ve vybrané úloze pracovat velmi dobře, ale pro jiné úlohy může být zcela nepoužitelný. Tato zdánlivá překážka je však nedůležitá, jelikož pro každou specifickou úlohu můžeme navrhnout nový specifický tvar operátoru.

Získáním nového tvaru příznaku, který bude vykazovat dobré výsledky při klasifikaci, se podaří zvýšit účinnost celého silného klasifikátoru. Další výhodou tohoto přístupu je, že pro vybudování silného klasifikátoru s požadovanou účinností budeme potřebovat menší počet slabých klasifikátorů. Tato vlastnost je velmi žádoucí, pokud budeme výsledný silný klasifikátor implementovat v FPGA. Sníží se tím požadavky klasifikátoru na zdroje v FPGA.

Čím větší je množina slabých klasifikátorů, které je možné použít v rámci jednoho silného klasifikátoru, tím lepší klasifikátor může být sestaven. Velikost množiny slabých klasifikátorů, která je vstupem do trénovacího procesu, je závislá na počtu všech tvarů operátorů, které máme k dispozici. S návrhem nových příznaků se tato množina rozroste.

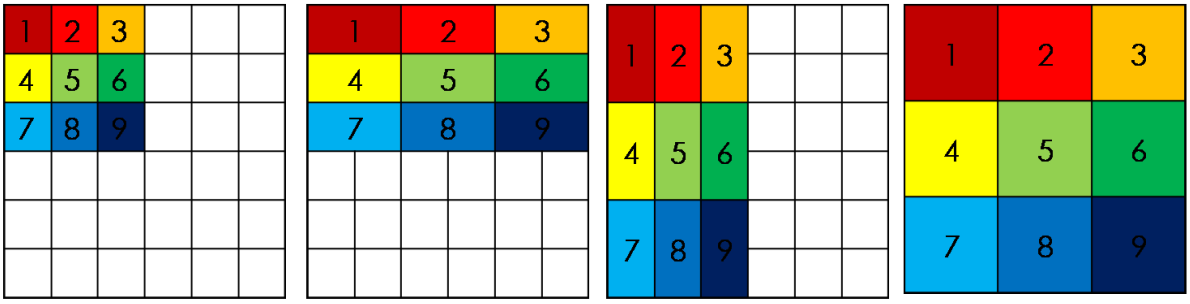
Při návrhu nových tvarů operátorů můžeme zavést řadu kritérií, dle kterých se bude navrhovat a posuzovat jejich kvalita. Například ze znalosti technologie, možností a zdrojů FPGA a cílové architektury můžeme sestavit kritérium, dle kterého se bude posuzovat vhodnost nově navrženého příznaku právě vzhledem k implementaci v FPGA. Takovým omezením můžeme generovat příznaky vhodné pro implementaci v FPGA.

4.1 Tvar operátoru

Tato podkapitola uvádí popis základních tvarů příznaků pro LBP operátory a následně ukazuje, jak by mohly vypadat nové tvary operátorů (příznaků) navržené automatizovaně. Velikost operátorů pro klasifikátor je omezena a nesmí přesáhnout velikost detekčního okna. Pro implementaci v FPGA jsou však operátory o velikosti detekčního okna příliš velké a jejich implementace by spotřebovala velké množství zdrojů FPGA. Experimentálně však bylo zjištěno, že pro většinu úloh postačují příznaky s omezenou velikostí [19]. Uveďme, že pro detekci obličeje se běžně používá velikost detekčního okna *24 x 24* pixelů [19] nebo *20 x 20* pixelů. Avšak pro slabé klasifikátory postačuje používat operátory o maximální velikosti *6 x 6* pixelů. Toto omezení je velmi vhodné nejen pro implementaci v FPGA, ale i pro hledání nových tvarů operátorů, jelikož se tímto omezením výrazně redukuje stavový prostor, který je nutné prohledat.

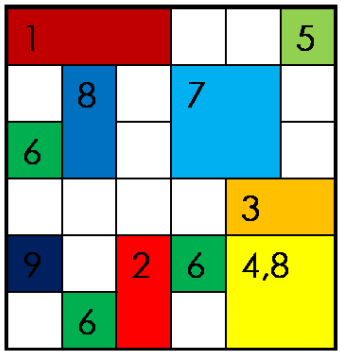
Obrázek 23 ukazuje ručně stanovené a v současnosti používané tvary operátorů [17]. Uvedené příznaky jsou charakteristické tím, že tvoří spojitou oblast. Všechny vyobrazené příznaky jsou vsazeny do obrazové matice o velikosti *6 x 6* pixelů. Jednotlivé oblasti (konvoluce) pro MB-LBP příznaky mají velikosti: *1 x 1* pixelů, *2 x 1* pixelů, *1 x 2* pixelů a *2 x 2* pixelů.

Každá oblast (jedna barva v obrázku) tvoří vždy jen jednu spojitou obdélníkovou oblast, stejně tak i všechny příznaky dohromady tvoří další spojitou oblast. Žádné dvě oblasti se nepřekrývají a žádný pixel není součástí více než jedné oblasti v jednom tvaru příznaku.



Obrázek 23:Standardní tvary příznaků [17].

Takovéto lidmi definované příznaky tedy splňují výše uvedená omezení. Metoda popsaná v této práci však neklade na návrh nových tvarů příznaků žádná omezení a novým příznakům dovoluje překrývání jednotlivých oblastí. Dovoluje, aby oblasti byly prázdné nebo aby byly oblasti pro konvoluce nespojité. Oblasti tak mohou být umístěny kdekoliv v předdefinované matici *6 x 6* pixelů. A tato matice je tak jediným omezením při návrhu nových tvarů příznaků. Obrázek 24 ukazuje možný tvar nově navržených příznaků a ukazuje možné tvary a překrytí jednotlivých oblastí (volnost evolučního návrhu). Pomocí uvedeného přístupu je dokonce možné provést nasimulování Haarových vlnek pomocí LBP. A to tak, že budou aktivní jen dvě oblasti (ostatní oblasti nebudou obsahovat žádný prvek) a aplikací LBP operátoru získáme poté jen dva možné výsledky stejně jako u Haarových vlnek.



Obrázek 24: Možný nový tar příznaků

Pomocí LBP operátorů se snažíme zachytit a popsat nejdůležitější rysy zkoumaného obrazu. Snažíme se tak de facto zachytit texturu objektu v obraze. Dle rovnice pro LBP 2.3 [60] víme, že se při výpočtu LBP příznaku provádí porovnání všech okrajových oblastí (hodnot jejich konvolucí) s hodnotou konvoluce prostřední oblasti (pro případ naší práce je konvoluce nahrazena prostou sumou pixelů v oblasti). Aby porovnání hodnot konvoluovaných oblastí dle rovnice 2.3 bylo smysluplné, musí mít porovnávané hodnoty stejný dynamický rozsah. Oblast o velikosti dvou pixelů má v našem případě dynamický rozsah *0 – 511*; u velkosti oblasti pět pixelů je dynamický rozsah *0 – 1279*. Tyto

dva rozsahy není vhodné bez další úpravy porovnávat. Proto je nutné provést úpravu dynamických rozsahů u obou hodnot do stejného „normalizovaného“ rozsahu. Základní dynamický rozsah, který se používá při výpočtu LBP, je $0 - 255$. Převod do tohoto základního rozsahu je velmi jednoduchý, jelikož postačuje provést pouze vydělení hodnoty dané oblasti počtem prvků oblasti. Po tomto kroku je již možné provést výpočet dle standardního algoritmu LBP operátoru.

4.2 Metoda pro řešení problému

Problematiku uvedenou v předchozí podkapitole je možné řešit pomocí velkého počtu optimalizačních technik pro prohledávání stavového prostoru možných řešení. Jelikož je velikost prohledávaného stavového prostoru enormně velká, není možné použít techniky, které provádějí jeho kompletní prohledání za účelem nalezení nejlepšího možného řešení dle zadaných kritérií (při hledání nejlepšího řešení se pokoušíme maximalizovat fitness funkci). Velikost stavového prostoru, který je nutné prohledávat, je $2^{W \times H \times K}$, kde W a H jsou rozměry maximální velikosti tvaru operátoru, a K je počet oblastí. V našem případě jsou $W = 6, H = 6, K = 9$. Velikost stavového prostoru pro uvedený případ je $2^{6 \times 6 \times 9} = 2^{324}$. S rostoucí maximální velikostí příznaku narůstá prostor pro řešení problému exponenciálně. Problém spadá do časové třídy EXP problémů (třída s exponenciální časovou náročností). Nalezení nejlepšího řešení pomocí náhodných neřízených prohledávacích technik není možné provést, jelikož prostor pro prohledávání je příliš veliký vzhledem k dnes dostupnému výpočetnímu výkonu. Pro řešení je tak nutné použít některou z technik, které využívají heuristiku. Struktura problému v prostoru je velmi komplikovaná a prostor řešení není spojitý. Řešení by bylo možné například technikami:

- horolezecká metoda [65],
- genetické algoritmy (GA) [65],
- metoda simulovaného žíhání [65] a tak dále.

Pro řešení našeho problému byla vybrána metoda založená na genetickém algoritmu. Tato metoda byla vybrána na základě její schopnosti řešit problémy v uvedené třídě složitosti. GA vykazují velmi dobré výsledky při řešení problémů, které jsou nespojité v prostoru a které mají také velký stavový prostor. GA je vhodný pro řešení uvedeného problému, jelikož není třeba naleznout globální maximum, ale postačuje nalézt lokální maximum, neboli dostatečně dobré řešení. Pokud bychom hledali globální maximum, bylo by vhodnější použít některou jinou optimalizační techniku, která zaručí jeho nalezení – například Horolezeckou metodu. Pomocí GA se však daří navrhovat velmi neobvyklá řešení, na která by běžný návrhář s pomocí standardních postupů a technik nepřišel. Takový výsledek však může být velmi zajímavý vzhledem k možnému nalezení vysoce optimalizovaných tvarů příznaků.

4.3 Genetický algoritmus pro návrh tvarů operátorů

Při řešení uvedené problematiky pomocí genetických algoritmů se musí vyřešit několik základních problémů. Tato podkapitola se zabývá řešením dílčích problémů, kterými jsou zakódování problému, schéma evolučního výpočtu a návrh fitness funkce.

Obrázek 24 ukazuje možný tvar nových operátorů. Jelikož algoritmus bude hledat jen tvar operátoru, není třeba do problému zakomponovat pozice příznaků v rámci detekčního okna. Evoluční algoritmus (EA) bude hledat jen nejvhodnější rozložení devíti oblastí ve vymezeném prostoru.

4.3.1 Zakódování problému

Obecná schémata evolučních algoritmů požadují reprezentaci vstupního problému v definované formě – chromozom. Ten je představován řetězcem znaků, ve kterém je provedeno jednoznačné zakódování kandidátního řešení problému. Chromozom musí být navržen tak, aby pomocí něj bylo možné provést zakódování jakékoliv možného kandidátního řešení. Operace prováděné nad chromozomy, jako jsou křížení a mutace, se většinou nezabývají vnitřní strukturou chromozomu a provedou jimi definovanou operaci na libovolném chromozomu. V některých případech se může stát, že aplikací uvedených operací získáme řešení, které není možné interpretovat (nemá validní strukturu). Takovému stavu se však vyvarujeme, pokud použijeme zakódování problému, ve kterém bude každý vygenerovaný chromozom představovat validní řešení. EA však mají i svá omezení. Jedním z omezení, které je vhodné uvést, je délka chromozomu. Pokud by byla příliš velká, tak by se řešení problému hledalo jen velmi obtížně. Proto je vhodné naleznout zakódování problému tak, aby výsledný chromozom byl co nejkratší.

V uvedeném problému je třeba zakódovat devět tvarů oblastí. Každá oblast může mít různou velikost a může být nespojitá. Minimální velikost oblasti je stavena na *0* prvků a maximální velikost oblasti je omezena na $6 \times 6 = 36$ prvků. Toto omezení je dáno maximální velikostí navrhovaných tvarů operátorů definovaných v podkapitole 4.1.

Problém zakódování příznaku tedy můžeme rozložit na devět stejných dílčích problémů – zakódování jedné oblasti. Jednu oblast můžeme zakódovat pomocí dvou základních způsobů:

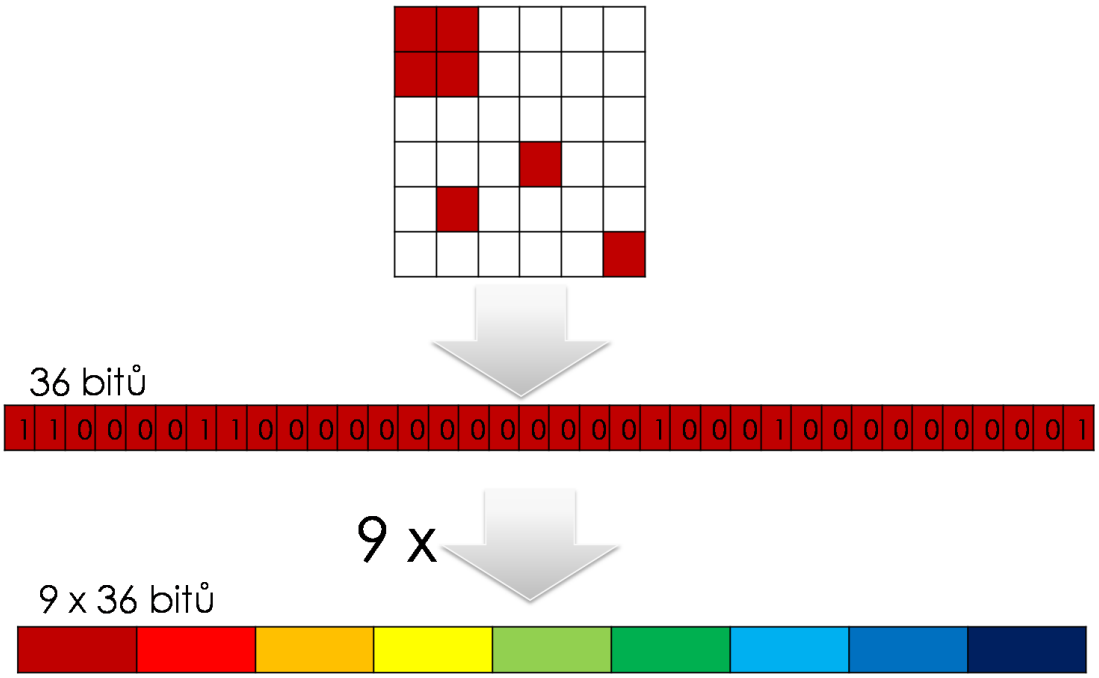
- variabilní délka řetězce,
- pevná délka řetězce.

Zakódování řetězce pomocí variabilní délky, by sice vedlo k relativně krátkému chromozomu, který by se snadno ohodnocoval, avšak toto řešení vede k řadě nevýhod. První nevýhodou je, že bychom museli mít v chromozomu separátory pro oddělení jednotlivých částí oblastí a také separátory pro oddělení konvolucí samostatných. Separátory jsou pak neužitečným znakem, který zvyšuje délku chromozomu. Operace křížení a mutací by musely být naprogramovány speciálně tak, aby produkovaly jen validní chromozomy. Zakódování problému pomocí variabilní délky řetězce by v důsledku nepřineslo příliš mnoho. Proto bylo v práci použito zakódování pomocí pevné délky řetězce. Zvolené kódování má velkou výhodu v tom, že se může libovolně provádět operace křížení a mutace a ve všech případech se získá platný jedinec. Zcela pak může být vyloučena kontrola validity vygenerovaného kandidátního řešení.

Jedna oblast se zakóduje jako posloupnost $6 \times 6 = 36$ bitů. Každý bit posloupnosti reprezentuje jeden pixel v oblasti 6×6 pixelů. Pixel na pozici dané souřadnicemi x a y je reprezentován bitem na pozici $x \cdot y$ v chromozomu. Pixel se souřadnicemi $x = 2$ a $y = 5$ je v bitovém řetězci reprezentován hodnotou na 10. místě. Hodnoty v bitovém řetězci udávají, zda se daný pixel uplatní v konvoluci dané oblasti nebo ne. Pokud je hodnota bitu rovna 1, tak se bit uplatní, jinak se neuplatní.

Obrázek 25 znázorňuje zakódování problému do chromozomu. V horní polovině obrázku je vyobrazeno zakódování jedné oblasti do bitového řetězce o délce *36* bitů. Tato operace je provedena celkem devětkrát, a to pro každou z oblastí. Ve spodní části obrázku je vyobrazeno získání výsledného chromozomu po zakódování celého problému. Výsledný chromozom má tedy délku $9 \times 6 \times 6 = 324$

bitů. Délka chromozomu není příliš velká, a tak se dá předpokládat úspěšné řešení problému pomocí EA.



Obrázek 25: Zakódování kandidátního řešení do chromozomu.

4.3.2 Schéma genetického algoritmu

Pro řešení problému evolučních výpočtů bylo použito jedno ze základních schémat genetických algoritmů, které poprvé představil Holland ve své práci [23]. GA se skládá z několika základních kroků, kterými jsou:

- inicializace populace,
- ohodnocení kandidátních řešení pomocí fitness funkce,
- selekce,
- rekombinace (mutace a křížení) a
- sestavení nové populace.

První problém při řešení vybraného problému pomocí GA je inicializace populace. Při řešení problému můžeme začínat buď již ze známého řešení anebo z náhodně vygenerovaného řešení. V případě, kdy začínáme z již známého řešení, mluvíme o takzvané řízené inicializaci. Celá populace se inicializuje pomocí již známých řešení a EA se je poté snaží vylepšovat. Druhou možností je provést náhodnou inicializaci populace. Pokud nejlepší globální optimum leží daleko od již známých řešení, na která se inicializuje, je vysoká pravděpodobnost, že algoritmus uvázne v některém z lokálních maxim a nedostane se ke globálně nejlepšímu řešení. Proto může být vhodnější používat pro řešení problémů náhodnou inicializaci, a pokusit se tak dosáhnout lepšího řešení.

Nejdůležitějším bodem pro řešení evolučního algoritmu je ohodnocení kandidátních řešení pomocí fitness funkce. Této problematice je proto věnována celá následující podkapitola 4.3.3.

Problém selekce jedinců pro rekombinaci je možné řešit pomocí několika základních způsobů. Pro tuto práci byla zvolena turnajová selekce [11]. Jedinci vybraní pomocí selekce postupují ve zpracování k rekombinaci – mutaci a křížení. Nejprve se provádí křížení jedinců, v algoritmu se uvažuje jen jednobodové křížení. Operace křížení se díky vhodné struktuře chromozomu provádí velmi jednoduše a postačuje tedy určit jen bod pro křížení a poté operaci křížení provést. Křížení spočívá ve výměně posloupnosti bitů za/před bodem křížení mezi dvěma chromozomy. Touto operací získáme ze dvou původních chromozomů dva nové chromozomy. Operace křížení se v algoritmu provádí s definovanou pravděpodobností. Díky této pravděpodobnosti se někteří jedinci nezmění a pokračují tak v původní formě do nové generace populace. Po operaci křížení následuje operace mutace, která mění hodnotu jednoho náhodně vybraného bitu v chromozomu. Operace mutace se může provádět i několikrát nad jedním chromozomem a opět se provádí s definovanou pravděpodobností, která opět zajistí, že někteří jedinci mohou projít beze změny do další generace.

Poslední operací v genetickém algoritmu je sestavení nové populace. Při této operaci je opět na výběr mezi několika technikami:

- obnova celé populace,
- částečná obnova populace.

V této práci byl zvolen princip částečné obnovy populace. Nejlépe hodnocení jedinci se tak vždy dostanou do nové populace (hranice byla nastavena na 5 % nejlepších jedinců), uplatňuje se zde takzvaný elitismus. Někteří další jedinci se dostanou do nové populace přes rekombinaci, v případě že nejsou pozměněni. Nová populace je však převážně složena z jedinců, kteří vznikli operacemi křížení a mutace.

4.3.3 Fitness funkce

Fitness funkce slouží u GA k ohodnocování kvality populace kandidátních řešení. Kvalitu každého jedince lze posuzovat dle několika rozličných hledisek. Na její kvalitě závisí výsledek celého genetického algoritmu a také rychlost konvergence algoritmu.

Pro návrh co nejvhodnějších tvarů příznaků můžeme použít několik různých fitness funkcí. V následujících odstavcích jsou popsány vybrané základní fitness funkce. Ty jsou sestaveny tak, aby je bylo možné kombinovat a vytvořit tak vícekriteriální fitness funkce.

Globální maximum

Aby bylo možné vysvětlit fitness funkci, je třeba nejprve popsat způsob výpočtu váhované chyby slabého klasifikátoru. Pro ohodnocení jednoho jedince populace (chromozomu) je třeba sestavit množinu slabých klasifikátorů, které jsou založeny na tvaru příznaku, který popisuje daný jedinec populace. Pokud budeme uvažovat, že výsledný silný klasifikátor pracuje nad oblastí obrazu o velikosti *24 x 24* pixelů, tak můžeme pomocí jednoho tvaru příznaku vytvořit celkem *361* různých klasifikátorů, které se budou lišit svou pozicí v detekčním okně. Pro evaluaci jednoho chromozomu musíme provést natrénování celé uvedené množiny slabých klasifikátorů a následně provést jejich ohodnocení na sadě testovacích dat. Ohodnocení takto velké množiny klasifikátorů je velmi pomalé a jsou tak kladeny velké nároky na výpočetní výkonnost. Výsledkem ohodnocení každého slabého klasifikátoru je váhovaná chyba, která udává přesnost klasifikátoru. Čím menší je váhovaná chyba, tím vyšší přesnosti dosahuje slabý klasifikátor. Nyní byl velmi stručně popsán způsob evaluace jednoho člena populace. Výsledkem evaluace je tedy množina hodnot (pro uvedený příklad *361*).

Fitness funkce globálního maxima je vytvořena tak, že najde slabý klasifikátor s nejmenší váhovanou chybu a tuto chybu bude nadále propagovat jako hodnotu fitness funkce. Název „globální maximum“ vychází z toho, že hledání maxima se provádí přes všechny možné pozice příznaku.

Tato funkce se tak snaží optimalizovat příznak pro jednu konkrétní pozici v detekčním okně. Takto natrénovaný příznak má velmi dobré vlastnosti. Avšak pokud bychom sestavili celý klasifikátor jenom z takovýchto příznaků, tak by výsledný silný klasifikátor nepracoval příliš dobře, jelikož by se příliš prosazoval jen jeden slabý klasifikátor a celý výsledný klasifikátor by měl úspěšnost jen o málo větší, než je právě tento jeden příznak. Pomocí této fitness funkce jsme schopni natrénovat příznaky, které mají účinnost okolo *70 – 75 %*. Pokud se však použije tento příznak jen v jednom slabém klasifikátoru (nebo několika málo), velmi dobře zahájí klasifikační proces a například ve spojení s WaldBoost [71] může produkovat velmi dobré výsledky (zamítne většinu detekčních oken již v prvním kroku).

Průměrná a střední hodnota

Fitness funkce průměrné a střední hodnoty vycházejí ze stejného základního principu jako předchozí fitness funkce. Avšak funkce neposkytují výslednou hodnotu jako maximum z množiny natrénovaných slabých klasifikátorů, ale vrací průměrnou nebo střední hodnotu z množiny natrénovaných slabých klasifikátorů. Do výsledné fitness funkce se tak promítne celá množina slabých klasifikátorů. Uvedené fitness funkce se snaží naleznout tvar příznaků, který bude fungovat co nejlépe na většině pozic detekčního okna. Na některých pozicích však může být úspěšnost slabého klasifikátoru na velmi dobré pozici a někde zase může klasifikátor poskytovat velmi špatné výsledky. Se špatnými výsledky slabých klasifikátorů na některých pozicích se však vyrovná trénovací algoritmus AdaBoost tak, že je nevybere do výsledného silného klasifikátoru.

Příznaky natrénované pomocí těchto fitness funkcí jsou uplatnitelné v kterékoliv části trénovacího procesu klasifikátoru. Průměrná úspěšnost takto natrénovaných příznaků je cca *50 - 55 %*. Příznak vykazuje velmi podobné hodnoty jako doposud ručně navržené příznaky. Takto natrénované příznaky jsou prospěšné, jelikož zvětší množinu slabých klasifikátorů pro AdaBoost, a je tak možné natrénovat větší a také dokonalejší klasifikátor.

Průměrná a střední hodnota z N nejlepších pozic

Další metoda pro hodnocení kvality tvarů nových příznaků je založena na kombinaci předešle uvedených metod. Nyní se však nehledá maximum ani průměrově nejlepší vzorek, ale je provedena kombinace obou myšlenek. Metoda je založena na hypotéze, že je možné naleznout příznak, který vykazuje velmi dobré vlastnosti při klasifikaci jen na několika pozicích v detekčním okně.

Metoda pro výpočet fitness funkce je opět založena na sestavení a ohodnocení množiny slabých klasifikátorů. Po tomto kroku je vybráno N nejlepších slabých klasifikátorů a z hodnot těchto klasifikátorů je vypočten průměr či střední hodnota. Slabé klasifikátory, které jsou založeny na tomto způsobu výpočtu, dosahují v průměru úspěšnosti cca *65 %*.

Z takto získaných slabých klasifikátorů opět není možné sestavit kompletní silný klasifikátor, jelikož tyto klasifikátory jsou natrénovány jen pro úvodní část klasifikátoru. Pokud je však použijeme v úvodní části, tak vykazují velmi dobré výsledky. Vhodnost klasifikátorů (tvary příznaků) jen pro úvodní část klasifikátoru je dána tím, že jejich trénování probíhá jen na sadě trénovacích dat s váhami, které odpovídají právě počátku trénovacího algoritmu AdaBoost. Pokud bychom chtěli mít natrénovány příznaky i pro další části AdaBoost klasifikátoru, museli bychom GA opakovaně spouštět v průběhu trénování silného klasifikátoru na datech s aktualizovanými vahami. Tento postup však vede

k enormní výpočetní náročnosti a následně pak k enormnímu času trénování (až několik měsíců, viz podkapitola 4.4.1).

Přizpůsobení příznaků pro FPGA

Poslední základní typ fitness funkce, který je v této práci použit, se zabývá přizpůsobením příznaků pro použití v FPGA. Pro každý navržený příznak můžeme stanovit typy operací, které využívá, a jejich četnosti. Dále pak pro každou operaci můžeme rovněž stanovit její cenu v FPGA. Rovnice pro výpočet cenové funkce má tvar:

$$FPGA_{cena} = \sum_{o \in O} f_{cena}(o) \cdot f_{četnost}(o) \tag{4.1}$$

Kde *O* je množina všech operací pro výpočet konvolucí příznaku *{+,/,>1, >2, >3, >4,>5}*, kde *>1* značí operaci logického posunu vpravo o 1 bit nebo také operaci dělení *x/2* (v FPGA vypuštění posledního bitu), *>2* značí operaci *x/4* (vypuštění dvou posledních bitů) a tak dále. Funkce f_{cena} je cenovou funkcí pro jednotlivé operace a $f_{četnost}$ je funkcí vyjadřující četnost použití této operace. Výstup funkce $FPGA_{cena}$ je již přímo výstupem této fitness funkce. Cenová funkce je nastavena tak, že vyžaduje vhodný počet pixelů v oblastech (mocniny dvou) a současně velmi diskriminuje operaci dělení, která není pro FPGA příliš vhodná. Poznamenejme, že v FPGA jsou i operace celočíselného dělení, které jsou velmi levné z hlediska spotřebovaných zdrojů. Takovou operací je dělení pomocí mocniny dvou. Fitness funkce je poté nastavena tak, aby se snažila přizpůsobit oblasti (jejich velikosti) tak, aby se pro výpočet jejich hodnoty používali jen vhodné operace z hlediska FPGA. Uvedená funkce by nemohla sloužit jako samostatná fitness funkce, jelikož nezkoumá úspěšnost příznaků. Funkce je však nastavena tak, aby ji bylo možné použít společně s již dříve představenými fitness funkcemi.

Dvoukriteriální fitness funkce

Předešlá podkapitola pojednávala o přizpůsobení tvarů příznaků pro FPGA. Avšak dle navržené fitness funkce není možné natrénovat úspěšný klasifikátor. Nyní je však ukázána možnost, jak vytvořit dvoukriteriální fitness funkci, která bude obsahovat kombinaci dvou dříve uvedených fitness funkcí. Složení dvou fitness funkcí do jedné se provede například pomocí váženého součtu. Vzorec pro výpočet dvoukriteriální fitness funkce, která bude složena z fitness funkce pro výpočet globálního maxima $f_{globmax}$ a fitness funkce pro FPGA přizpůsobení f_{FPGA}, bude vypadat následovně:

$$F_{multi}(c) = f_{FPGA}(c) \cdot W_{FPGA} + f_{globmax}(c) \cdot W_{globmax} \tag{4.2}$$

Kde W_{FPGA} a $W_{globmax}$ jsou váhy pro každou z fitness funkcí a *c* je slabý klasifikátor. Pomocí dvoukriteriální fitness funkce lze navrhovat velmi dobré slabé klasifikátory, které jsou současně velmi dobře implementovatelné v FPGA technologii. Váha W_{FPGA} je ve výsledné aplikaci nastavena tak, aby výsledné tvary příznaků byly navrženy vhodně pro implementaci v FPGA (experimentálně byla zjištěna hodnota váhy cca *40 %*), což se v důsledku projeví tak, že velikost konvolucí příznaku je u výsledných tvarů příznaků rovna mocnině dvou. Tvary nesplňující tuto podmínku jsou velmi silně penalizovány. Fitness funkce byla experimentálně ověřena a aplikována v algoritmu. Výsledky z klasifikačního hlediska je možné naleznout v podkapitole 4.4.2.

4.4 Dosažené výsledky

S navrženým GA byla provedena řada experimentů. Získané výsledky jsou rozděleny do dvou hlavních oblastí. První oblast výsledků se zabývá dosaženými výsledky z hlediska genetických algoritmů. A druhá oblast se zabývá úspěšností klasifikátoru natrénovaného pomocí metody AdaBoost s využitím nových tvarů příznaků získaných pomocí uvedeného genetického algoritmu.

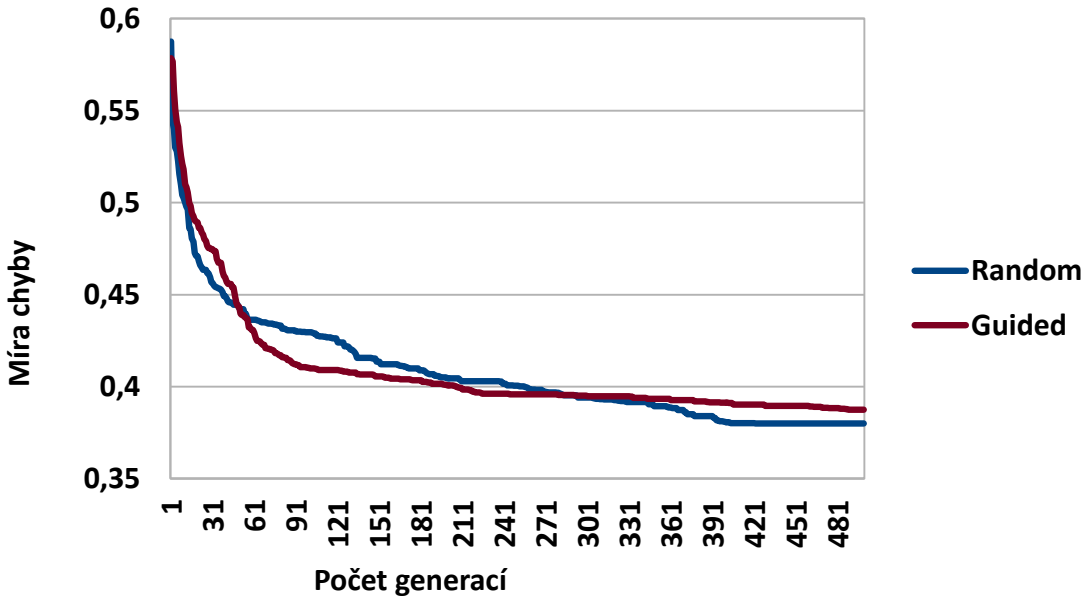
4.4.1 Výsledky genetického algoritmu

U genetického algoritmu bylo prozkoumáno velké množství variací parametrů pro vhodné nastavení algoritmu. V této části jsou prezentovány nejpodstatnější výsledky, kterými jsou rychlost konvergence algoritmu v závislosti na nastavených parametrech, a současně také dosažené výsledky vzhledem ke klasifikační přesnosti možných slabých klasifikátorů. Ve všech zde uvedených případech byla použita fitness funkce „průměr z N nejlepších pozic“. Všechny vykreslované grafy byly získány jako průměr z M běhů, kde M bylo nastaveno na 3. Takto malé číslo bylo zvoleno vzhledem k příliš dlouhé době běhu GA a velkému počtu experimentů pro vyhodnocení. Ohodnocení experimentů probíhalo na počítači s dvěma osmijádrovými procesory Intel Xeon E5640 na frekvenci *2.66* GHz. Ohodnocení běhu GA s pouhými *100* jednici v populaci, maximálním počtem *500* generací a pouhými *10 000* vzorky v trénovací množině slabých klasifikátorů zabere na všech *16* jádrech více než *3,5* hodiny. Poznamenejme však, že GA není optimalizován na rychlost výpočtu, jelikož to není předmětem této práce. Pro tuto práci jsou důležité pouze nově navržené tvary příznaků, a nikoli doba jejich návrhu, jelikož návrh se provádí jen jednou a není to časově kritická operace. Dlouhý čas výpočtu je dán především nutností natrénovat a ohodnotit v každé generaci všechny slabé klasifikátory. Pro ohodnocení slabých klasifikátorů byla z hlediska úspory času použita jen omezená trénovací množina.

Inicializace algoritmu

V této části práce je analyzován vliv inicializace na rychlost konvergence GA a úspěšnost výsledných slabých klasifikátorů. Jsou analyzovány výsledky získané použitím řízené inicializace a následně také použitím náhodné inicializace. Při generování chromozomu náhodnou inicializací je každý bit nastaven na logickou hodnotu *1* nebo *0* pouze v závislosti na pravděpodobnosti, která je vstupem do algoritmu. Vhodná míra pravděpodobnosti byla také předmětem zkoumání, kdy se ukázalo, že je vhodné používat jen velmi malou míru pravděpodobnosti – okolo *5 %*. (Menší pravděpodobnost znamená, že bude použito méně pixelů v oblasti obrazu pro konvoluce.) Z experimentálních měření se potvrdilo (obrázek 26), že pro účely inicializace je lepší používat náhodnou inicializaci. Řízená inicializace z počátku GA konverguje velmi rychle, ale poté GA uvázne v lokálním extrému, ze kterého se dostává jen s velkými obtížemi. Náhodná inicializace konverguje zpočátku pomaleji, ale následně se pomocí ní podaří dosáhnout lepších výsledků než s řízenou inicializací.

Vodorovná osa na obrázku 26 (a na dalších dále uvedených) představuje počet generací GA, které byly prozkoumávány (maximálně *500*), svislá osa představuje míru chyby nejlepšího nalezeného řešení pro danou generaci GA. Míra chyby *0,4* znamená, že nejlepší slabý klasifikátor dosahuje přesnosti *60 %*.

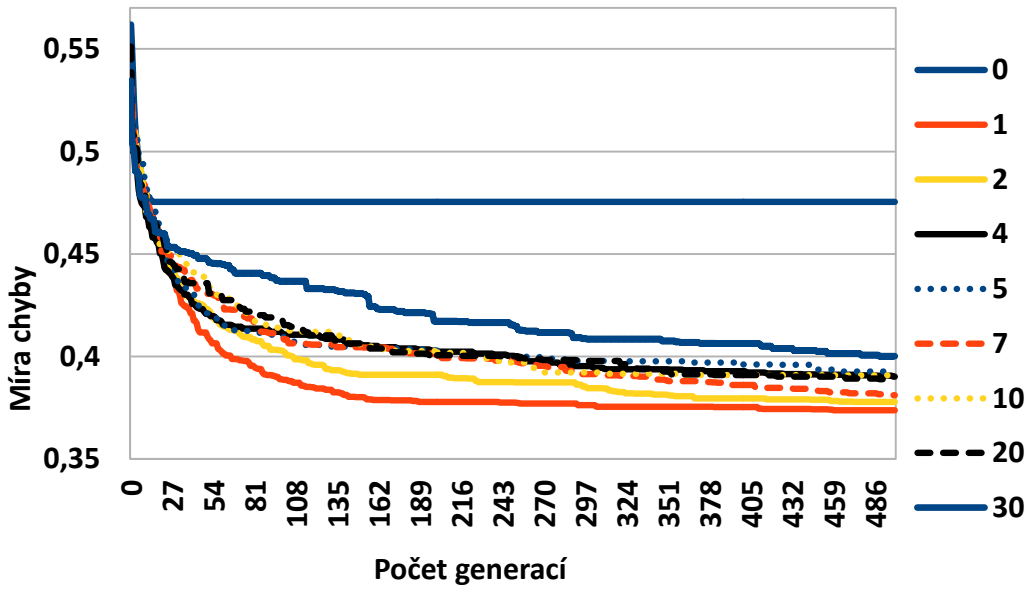


Obrázek 26: Inicializace GA (náhodná a řízená).

Počet mutací

V tomto experimentu je zkoumán vliv počtu mutací na rychlost konvergence algoritmu. Počtem mutací se myslí počet pokusů provedení mutace v jednom chromozomu. Počet mutací byl zkoumán v rozsahu *0-30* mutací. (Nastavení evolučního algoritmu: pravděpodobnost mutací = 40 %, pravděpodobnost křížení = *80 %*, velikost populace = *40*, počet generací = *500*.)

Obrázek 27 ukazuje výsledky experimentu. Na vodorovné ose grafu je vyznačen počet generací, po které algoritmus běžel, a na svislé ose je vynesena míra chyby, což je převrácená hodnota úspěšnosti klasifikátoru. Z grafu je možné vypozorovat, že nejlepších výsledků dosahuje algoritmus při použití jedné mutace. Druhý nejlepší výsledek je pro tři mutace. Pokud mutace vynecháme, tak algoritmus velmi rychle uvázne v lokálním extrému, ze kterého se jen za pomoci operací křížení není schopen vymanit. Jako nejvhodnější nastavení počtu mutací se tedy jeví jedna mutace.

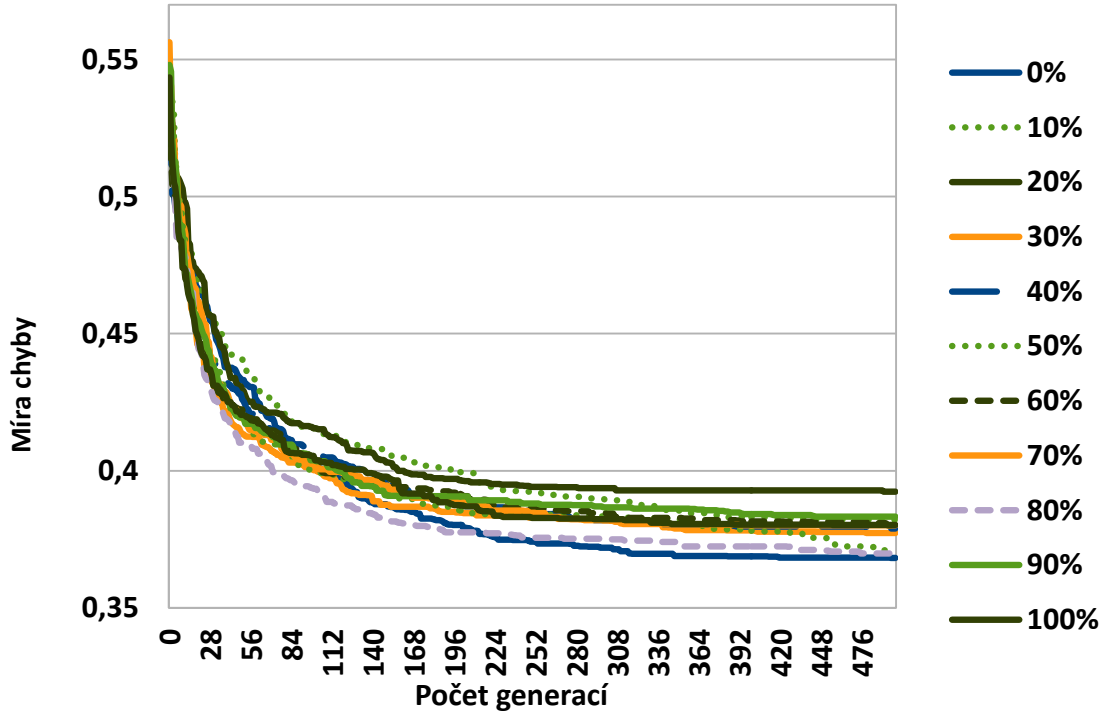


Obrázek 27: Závislost rychlosti konvergence GA na počtu mutací.

Pravděpodobnost křížení

Experiment se zabýval zkoumáním pravděpodobnosti, že dojde ke křížení dvou jedinců. Pravděpodobnost křížení byla zkoumána pro rozsah hodnot *0 – 100 %*. (Nastavení genetického algoritmu bylo: pravděpodobnost mutací = *40 %*, počet mutací = *1*, velikost populace = *40*, počet generací = *500*.)

Obrázek 28 ukazuje výsledky experimentu. Význam os je stejný jako u předchozího příkladu. Nejlepších výsledků bylo dosahováno při *0%* a *80%* pravděpodobnosti použití křížení. Jelikož by bez křížení mohl algoritmus velmi často uváznout v lokálních extrémech, byla jako nejvhodnější nastavení pravděpodobnosti křížení zvolena hodnota *80 %*.

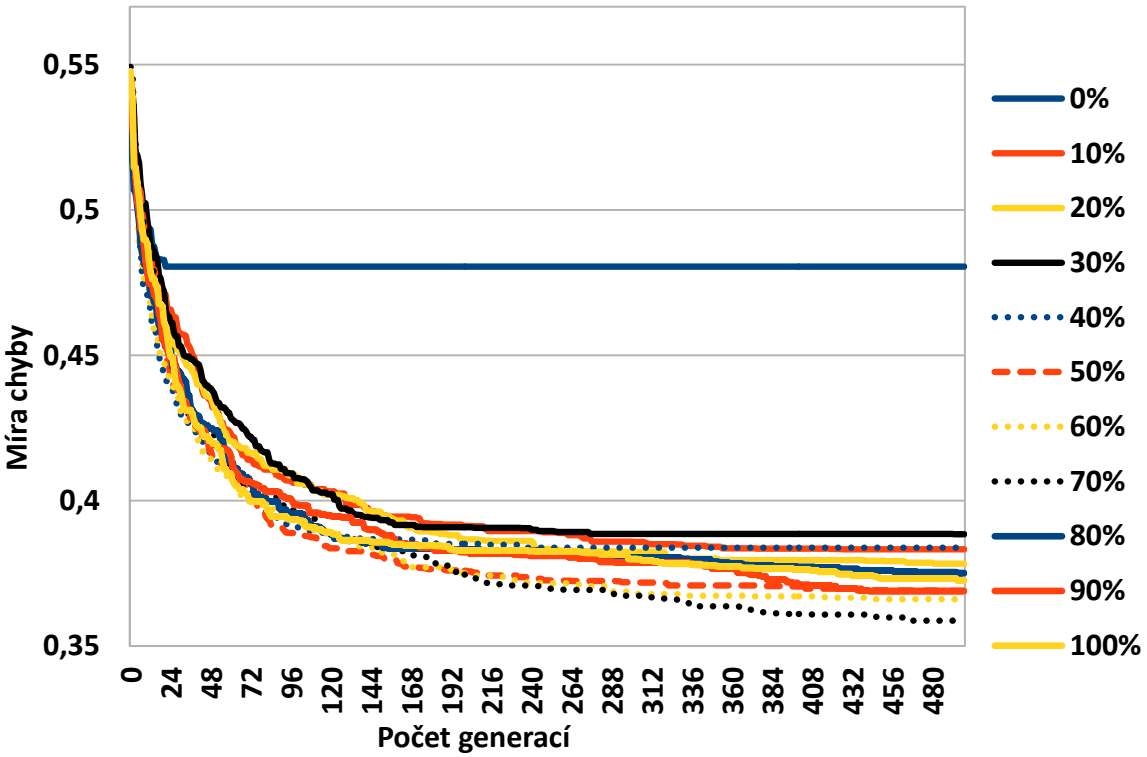


Obrázek 28: Závislost rychlosti konvergence GA na pravděpodobnosti křížení.

Pravděpodobnost mutací

Experiment se zabýval zkoumáním vlivu pravděpodobnosti provedení mutace. Pravděpodobnost mutací byla zkoumána v rozsahu *0 – 100 %*. (Nastavení genetického algoritmu bylo následující: pravděpodobnost křížení = *80 %*, počet mutací = *1*, velikost populace = *40*, počet generací = *500*.)

Význam os je opět stejný, jako v předešlých případech. Obrázek 29 ukazuje, že nejlepších výsledků bylo dosaženo při pravděpodobnosti mutací *70 %*. Tudíž mutace jsou nepostradatelnou částí uvedeného GA.



Obrázek 29:Závislost rychlosti konvergence GA na pravděpodobnosti mutací.

Nastavení genetického algoritmu

Z naměřených výsledků byly stanoveny nejvhodnější parametry pro genetický algoritmus:

- pravděpodobnost mutací = *70 %,*
- počet mutací na chromozom = *1*,
- pravděpodobnost křížení = *80 %* a
- jednobodové křížení.

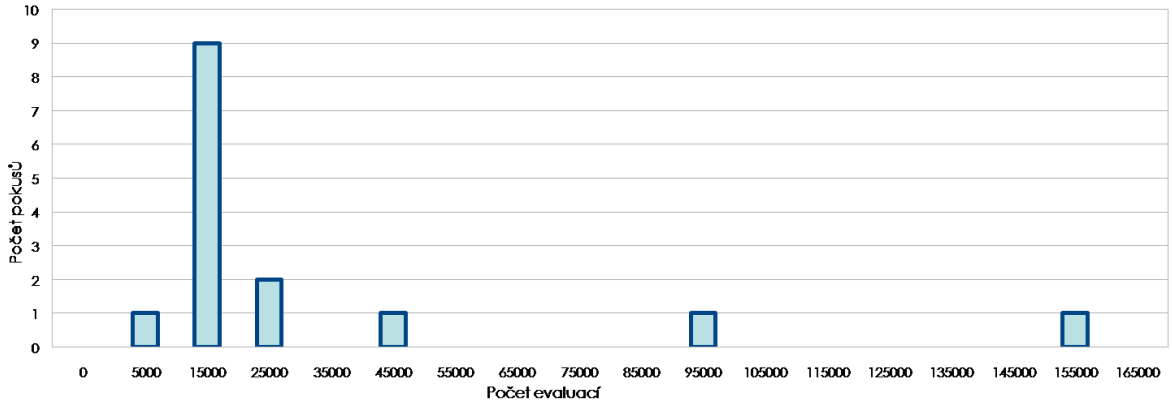
Ověření konvergence

Po určení co nejvhodnějších parametrů GA byla provedena řada testů za účelem ověření, zda algoritmus s určenými parametry konverguje a zda je schopen dosáhnout požadovaných výsledků. První experiment pro ověření konvergence byl proveden pro následující parametry: pravděpodobnost mutací = *70 %*, počet mutací na chromozom = *1*, pravděpodobnost křížení = *80 %*, jednobodové křížení, velikost populace = *100*.

Ve všech dále uvedených experimentech byly nastaveny dvě ukončující podmínky pro GA. První podmínka stanovovala míru chyby slabého klasifikátoru, po jejímž překročení došlo k zastavení algoritmu (nalezení řešení). Druhá podmínka omezovala maximální počet evaluací jedinců. Tento počet byl stanoven na *120 000*. Evaluací jedince se rozumí ohodnocení (výpočet fitness funkce) jednoho kandidátního řešení z populace. Pokud má tedy populace 100 jedinců, na ohodnocení jedné generace potřebujeme 100 evaluací.

Histogramy na obrázcích 30, 31 a 32 ukazují rychlost a stabilitu konvergence GA pro definované počáteční podmínky. Histogramy vznikly tak, že se jeden GA s definovanými počátečními podmínkami spustil opakovaně a do histogramu se vynesly potřebné počty evaluací jedinců pro jednotlivé běhy. Délka běhu GA byla závislá na tom, kdy bylo dosaženo stanovené míry chyby.

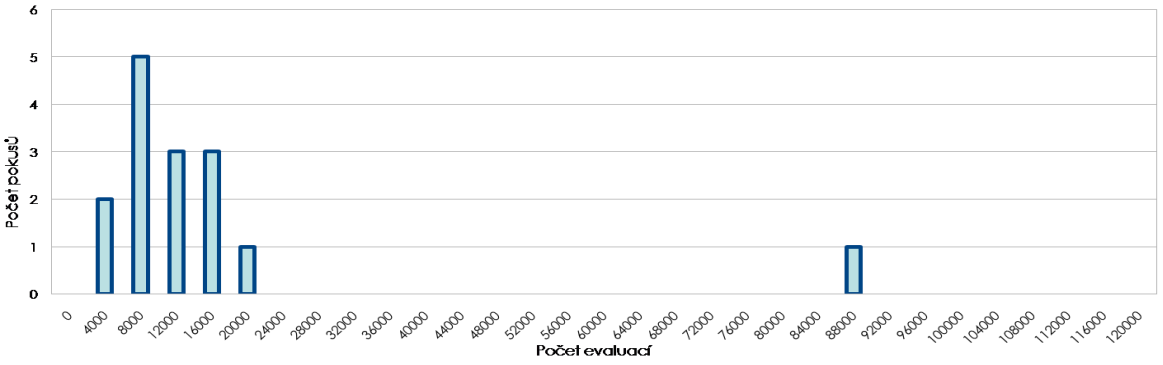
Výsledky prvního experimentu jsou uvedeny na obrázku 30. Například první sloupec histogramu vyjadřuje, že pro dosažení stanovené úspěšnosti bylo třeba cca *5 000* evaluací v jednom případě běhu GA. Druhý sloupec vyjadřuje, že pro dosažení stanovené hranice úspěšnosti bylo třeba cca *15 000* evaluací v devíti běžích GA. Z grafu lze vidět, že ve všech případech se GA ukončil dříve, než se podařilo dosáhnout maximálního stanoveného počtu evaluací. To znamená, že algoritmus ve všech 15 běžích nalezl řešení.



Obrázek 30: Histogram počtu pokusů v závislosti na počtu evaluací GA (parametry: pravděpodobnost mutací – *70 %*, počet mutací – *1*, pravděpodobnost křížení – *80 %*, jednobodové křížení, velikost populace – *100*).

Další provedený experiment je pro nastavení: pravděpodobnost mutací = *70 %*, počet mutací na chromozom = *1*, pravděpodobnost křížení = *80 %*, jednobodové křížení, velikost populace = *40*.

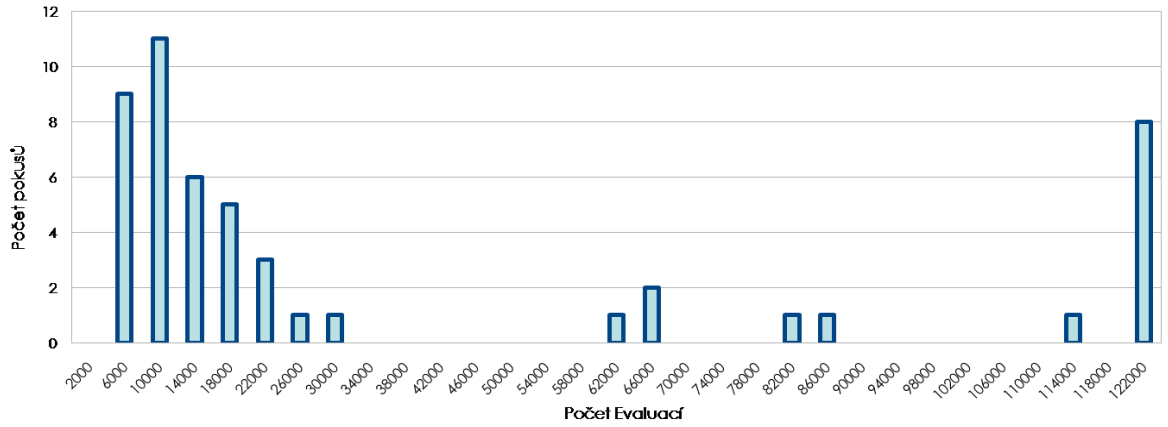
Obrázek 31 ukazuje fakt, že pokud použijeme menší množství jedinců v populaci, tak bude konvergence opět zaručena, a to ve velmi vysokém počtu případů, a navíc klesne i průměrný počet evaluací pro dosažení stanovené hranice míry chyby. Význam os histogramu je stejný jako v předešlém případě. Menší velikost populace je výhodná jak z hlediska rychlosti, tak i z hlediska konvergence.



Obrázek 31: Histogram počtu pokusů v závislosti na počtu evaluací GA (parametry: pravděpodobnost mutací = *70 %*, počet mutací na chromozom = *1*, pravděpodobnost křížení = 80 %, jednobodové křížení, velikost populace = *40*).

Poslední provedený experiment ukazuje situaci, že pokud se zvolí nevhodné počáteční parametry pro GA, tak ne vždy algoritmus konverguje, a může tedy uváznout v některém z lokálních extrémů. Nastavení posledního experimentu je: pravděpodobnost mutací = *40 %*, počet mutací na chromozom = *1*, Pravděpodobnost křížení = *80 %*, jednobodové křížení, velikost populace = *40*. Výsledky pro experiment jsou na obrázku 32. Ten ukazuje, že algoritmus v mnoha případech konvergoval poměrně

rychle, ale také ve velkém počtu případů (cca *20 %*) nedosáhl stanovené hranice míry chyby a genetický algoritmus byl přerušen pro dosažení maximálního počtu povolených evaluací jedinců.



Obrázek 32: Histogram počtu pokusů v závislosti na počtu evaluací GA (parametry: pravděpodobnost mutací = *40 %*, počet mutací na chromozom = *1*, Pravděpodobnost křížení = *80 %*, jednobodové křížení, velikost populace = *40*).

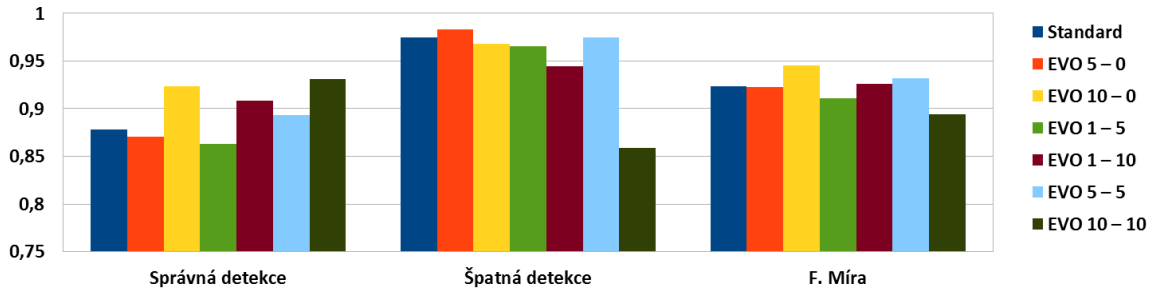
4.4.2 Výsledky klasifikace

Tato podkapitola uvádí dosažené výsledky silných AdaBoost klasifikátorů, které jsou sestaveny právě s využitím evolučně navržených tvarů příznaků. Pro ověření úspěšnosti klasifikátorů byl použit dataset (množina dat) UIUC [1], který obsahuje obrázky bočních pohledů na auta. Ten je rozdělen do dvou hlavních částí – trénovací dataset a testovací dataset. Trénovací dataset obsahuje celkem *1050* obrazových vzorků, z nichž je *550* pozitivních (vzorek auta) a *500* negativních. Trénovací množina je použita pro natrénování silného klasifikátoru a současně je také použita v GA ve fitness funkci pro natrénování a ohodnocení slabých klasifikátorů. Každý vzorek z trénovací množiny je upraven na šířku *40* pixelů a výšku *16* pixelů (velikost detekčního okna). Trénovací množina mimo jiné také slouží pro získání přesnosti slabých klasifikátorů ve fitness funkci.

Druhá část datasetu je určena pro testování. Ta se používá k otestování přesnosti výsledného AdaBoost klasifikátoru. Testovací dataset je dále rozdělen do dvou rozdílných částí. První část se skládá ze *169* obrázků, ve kterých mají všechny hledané objekty přibližně stejnou velikost. Na jednom testovacím obrázku se může vyskytovat i více detekovaných předmětů (aut). Druhá část testovacího datasetu se skládá ze *107* obrázků, na kterých mají objekty různé velikosti.

Výsledky silných klasifikátorů s evolučně navrženými příznaky jsou porovnávány se silnými klasifikátory, které využívají pouze standardní tvary příznaků. Jelikož jsou evolučně navržené příznaky získány na trénovací množině s iniciálním ohodnocením vzorků, nemůže být celý klasifikátor sestaven pouze z evolučně navržených tvarů příznaků. Takový klasifikátor by nepracoval dostatečně obecně a byl by přizpůsoben jen datům trénovací množiny s iniciálním ohodnocením (viz 4.3.3). Pro odstranění této vlastnosti je nutné po každém vybrání slabého klasifikátoru (dle AdaBoost principu) provést evoluční návrh nových příznaků s aktualizovanými vahami trénovacích dat. Vytvoření takového klasifikátoru je však velmi výpočetně náročné. Klasifikátory pro testování jsou proto sestavovány tak, že se evolučně navržené tvary příznaků používají jen na počátku klasifikační kaskády. Pro ilustraci uvedených tvrzení (o nefunkčnosti slabých klasifikátorů při jejich nevhodném použití) jsou uvedeny i klasifikátory, které využívají evolučně navržené tvary příznaků na konci klasifikační kaskády. U těchto klasifikátorů jsou získané výsledky horší než u klasifikátorů založených pouze na standardních tvarech příznaků.

Obrázek 33 zobrazuje výsledky silných klasifikátorů využívajících pouze standardní tvary příznaků a výsledky silných klasifikátorů využívajících evolučně navržené tvary ve spojení se standardními tvary příznaků. Všechny uvedené klasifikátory obsahují celkem *100* slabých klasifikátorů. Klasifikátor s označením „*Standard*“ obsahuje pouze konvenční tvary příznaků. Klasifikátor s označením *EVO 10 – 5* značí, že silný klasifikátor je sestaven ze tří částí. První číslice *10* v označení klasifikátoru značí, že úvodní část klasifikátoru je složena z *10* slabých klasifikátorů, jež využívají evolučně navržené tvary příznaků. Poslední číslice *5* značí, že závěrečná část klasifikátoru je složena z *5* slabých klasifikátorů, jež využívají evolučně navržené tvary příznaků. Zbylé slabé klasifikátory (*85*) jsou založeny na standardních tvarech příznaků.



Obrázek 33: Porovnání výsledků AdaBoost klasifikátorů založených na standardních a evolučních příznacích

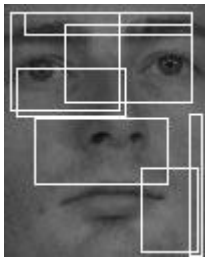
Obrázek 33 obsahuje tři skupiny sloupců – správná detekce, špatná detekce a F. míra. Všechny hodnoty mají procentuální vyjádření. F. míra je kombinace správné detekce a špatné detekce. Z grafů je možné vidět, že klasifikátor *EVO 10 – 0* vykazuje nejlepší výsledky. *EVO 10 – 0* je lepší než standardní přístup v přesnosti klasifikace o cca *4 %*, a přitom procento špatných klasifikací je na cca stejné úrovni v porovnání se standardním přístupem. Z grafu je také možné vidět, že klasifikátor *EVO 1 – 5* vykazuje nejhorší výsledky. Tyto výsledky demonstrují špatné použití slabých klasifikátorů založených na evolučně navržených tvarech příznaků. Ty totiž byly navrženy pro iniciální fázi klasifikátoru, ale v uvedeném případě byly použity převážně na konci klasifikátoru, a výsledná přesnost klasifikátoru tak vykazuje horší výsledky v porovnání s lidmi navrženými slabými klasifikátory.

4.5 Porovnání s ostatními pracemi

V literatuře neexistuje mnoho prací, které by se zabývaly podobným tématem. Jedinou prací, která se zabývá podobnou problematikou, je práce *Genetic Based LBP Feature Extraction and Selection for Facial Recognition* od autorů Joseph Shelton a kolektiv [70]. Tato práce však využívá LBP příznaky zcela jiným způsobem. Algoritmus detekce obličeje, kterým se práce zabývá, pracuje nad detekčním oknem, které rozděluje do několika políček (jak je naznačeno na obrázku 34).

Původní algoritmus pro detekci obličeje, který je uveden v článku, pracuje tak, že vypočte LBP funkce nad každým z políček. Tedy do každého políčka přiřadí jednu LBP funkci a nad tímto políčkem obrazu ji vyhodnotí. Autoři se pomocí genetického algoritmu snaží navrhnout nové rozložení políček nad obrazem. Možné nové rozložení je vyobrazeno na obrázku 35.

Obrázek 34: Rozdělení detekčního okna na políčka.



Obrázek 35: Geneticky navržené rozložení políček.

Autoři tak nevyužívají genetického algoritmu pro návrh každého z devíti tvarů konvoluce, ale využívají genetického algoritmu jen k návrhu tvaru a umístění políček, ve kterých se provede výpočet LBP funkce. Další obdobné práce se nepodařilo naleznout.

5 Jádro klasifikátoru

Kapitola představuje nový pohled na problematiku vyhodnocení AdaBoost klasifikátoru v FPGA. Výsledky uvedené v této kapitole využívají dříve dosažených výsledků autora v práci [37]. Tato kapitola uvádí rozšíření již dosažených výsledků a pro jejich důsledné pochopení je velmi důležité uvést rekapitulaci nejdůležitějších poznatků o způsobů vyhodnocení. V této kapitole ještě není představen kompletní klasifikátor, ale je představen unikátní způsob vyhodnocení jednoho detekčního okna, případně obrazu s jednou úrovní rozlišení.

Tato kapitola představuje jádro AdaBoost klasifikátoru – tzv. AC jednotku. Dříve představené architektury popsané v podkapitole 3.3, ale i jiné, se k vyhodnocení AdaBoost klasifikátoru staví konvenčním, tedy sekvenčním způsobem, jež je popsán v podkapitole 2.5.2 či případně v podkapitole 5.1.1. V této práci je však v podkapitole 5.1.2 popsán zcela nový způsob jak přistoupit k problematice vyhodnocení, který spočívá v neseřazeném vyhodnocení slabých klasifikátorů. Jak bude dále ukázáno, tento nový způsob vyhodnocení přináší výpočetní architektuře řadu důležitých výhod. Jednou z hlavních předností, vzhledem ke které byla tato architektura navrhována, je vysoká rychlost zpracování a také proudový charakter jednotky.

Nově představená architektura se od většiny doposud představených implementací liší také v tom ohledu, že nevyužívá velmi často používané Haarovy příznaky a integrální obraz, ale využívá slabé klasifikátory založené na LBP příznacích (viz podkapitola 2.4.2). Jejich použití však nesnižuje klasifikační přesnost, ale dokonce ji i zvyšuje, a způsob jejich zpracování výrazně napomohl vzniku této architektury (viz podkapitola 5.2).

5.1 Vyhodnocení detekčního okna AdaBoostu

5.1.1 Sekvenční zpracování detekčního okna

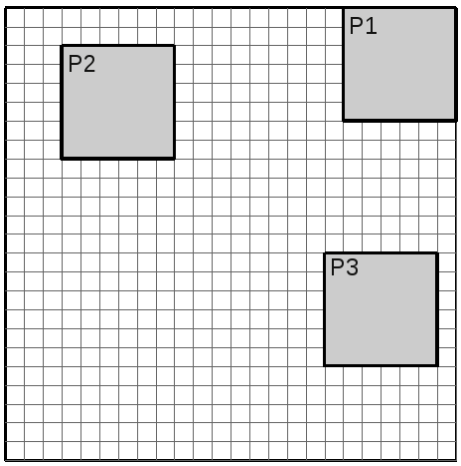
Pro připomenutí si nejdříve uveďme vyhodnocení detekčního okna sekvenčním způsobem. Tento způsob je odvozen převážně od sekvenční práce procesoru a také původní metody vyhodnocení AdaBoostu dle [7].

AdaBoost klasifikátor pracuje vždy v oblasti, která se nazývá detekční okno. Hledaný objekt se tedy musí nacházet v tomto okně, a posouzení, zda se objekt v okně nachází nebo ne, se děje pouze na základě obrazových dat, která spadají do aktuálního detekčního okna. Okolní pixely se ve vyhodnocení neuvažují. Například pro detekci obličejů se používá detekční okno o velikosti 24x24 pixelů [25]. Nevejde-li se objekt svou velikostí do detekčního okna nebo je naopak v detekčním okně příliš malý, provádí se změna rozlišení původního obrazu tak, aby měl hledaný objekt vhodnou velikost v rámci detekčního okna. Další možností je provádět změnu velikosti detekčního okna samotného. Obě možnosti se velmi často společně kombinují zvláště při zpracování na FPGA (viz podkapitola 3.3).

K rozpoznání objektu dochází na základě vyhodnocení množiny slabých klasifikátorů. Množina a parametry slabých klasifikátorů jsou určeny v procesu trénování, který určí jejich pozice v rámci detekčního okna, typ příznaku, případně velikost příznaku a další parametry slabého klasifikátoru. V jednom detekčním okně se tak může vyskytovat řádově tisíce slabých klasifikátorů, viz například tabulka 6. Pro implementaci nově navržené architektury byly vybrány slabé klasifikátory založené na LBP operátorech, které, jak již bylo ukázáno v práci [37], mají velmi výhodné vlastnosti vzhledem

k implementaci v FPGA. Pro jejich implementaci není třeba počítat složitý a paměťově náročný integrální obraz a jejich vyhodnocení se skládá převážně z jednoduchých operací, jako jsou logické posuny, sčítání, porovnávání a vyhledávání ve fixních tabulkách. LBP, nebo spíše MB-LBP příznaky (viz 2.4.2), mohou obecně nabývat libovolného obdélníkového tvaru a rozměru. Pro implementaci v FPGA je však tato vlastnost nežádoucí, a je mnohem vhodnější použít příznaky s omezenou velikostí. Empirickými testy bylo ověřeno, že omezí-li se velikost příznaků na maximální velikost *6 x 6* pixelů, tak nebude zásadně zhoršena klasifikační přesnost klasifikátoru [19]. Takové omezení velikosti samozřejmě předpokládá zapracování již do trénovacího algoritmu. Maximální rozměr příznaků *6 x 6* pixelů je velmi vhodný i vzhledem k implementaci v FPGA.

Na obrázku 36 je vyobrazeno detekční okno o velikosti *24 x 24* pixelů. V detekčním okně jsou znázorněny tři příznaky, jejichž pozice a velikosti jsou výsledkem trénovacího algoritmu. Příznaky mohou mít obecně libovolnou pozici v rámci detekčního okna a mohou se navzájem překrývat. Pořadí vyhodnocení příznaků je opět dáno trénovacím algoritmem. Pozice příznaku v rámci celé množiny slabých klasifikátorů mimo jiné určuje i významnost slabého klasifikátoru. Klasifikátor na pozici *N* tak má zpravidla větší váhu ve vyhodnocení než klasifikátor na pozici *N+1*. Tuto vlastnost však lze částečně opomenout, jak bude ukázáno v následující podkapitole.



Obrázek 36: Detekční okno 24 x 24 pixelů s 3 příznaky.

Je-li seřazená množina slabých klasifikátorů *{P2,P3,P1}*, tak vyhodnocení AdaBoost klasifikátoru klasickým sekvenčním způsobem dle autorů Freund a Schapire [7] probíhá tak, že se vyhodnotí nejprve slabý klasifikátor daný příznakem *P2*, následně se vyhodnotí slabý klasifikátor daný příznakem *P3*, jejich výsledky se sečtou do akumulátoru a následně se vyhodnotí i slabý klasifikátor na pozici *P1* a jeho výsledek se opět přičte do akumulátoru. Tímto způsobem dojde k vyhodnocení všech slabých klasifikátorů, ze kterých se AdaBoost klasifikátor skládá. V uvedeném případě jsou to jen tři slabé klasifikátory. Výsledek v akumulátoru se poté porovná s prahem vzešlým z trénování a tímto porovnáním se určí, zda v okně je, či není hledaný objekt. Při vyhodnocení detekčního okna pomocí metody WaldBoost [71] by se postupovalo mírně odlišněji. Po každém přičtení výsledku slabého klasifikátoru do akumulátoru by se mohlo provést porovnání s aktuálním prahem pro daný slabý klasifikátor, a pokud by byla hodnota nižší než první práh (práh pro ukončení detekce), tak by se zastavilo vyhodnocení a oblast detekčního okna by se prohlásila za negativní - neobsahuje hledaný předmět. Dále by se provedlo porovnání, zda nedošlo k překročení druhého definovaného prahu (práh pro detekci objektu), a pokud by se překročil, tak by se prohlásilo, že okno obsahuje hledaný předmět. V případě neuplatnění ani jednoho z prahů se pokračuje vyhodnocením dalšího slabého klasifikátoru v množině. Vyhodnocení na konci klasifikační kaskády, pokud se k ní klasifikátor dostane, je stejné jako u AdaBoostu.

Posledním významným způsobem, jak může být vyhodnoceno detekční okno, je AdaBoost dle autorů Viola a Johnes [82]. AdaBoost klasifikátor je dle jimi navrženého principu rozdělen na kaskády. Každá kaskáda se skládá z několika slabých klasifikátorů a po dokončení každé kaskády je provedeno prahování s cílem ukončení detekce. Prahování je obdobné jako v případě uvedeném u WaldBoost klasifikátoru.

Uvedené sekvenční vyhodnocení z hlediska FPGA implementace znamená, že pro vyhodnocení jednoho slabého klasifikátoru musí být načtena požadovaná data a následně provedeno vyhodnocení příznaku. Jelikož se příznak může vyskytovat kdekoliv v rámci detekčního okna, je nutné pro vysokou rychlost klasifikátoru jako celku zajistit vysokou rychlost přístupu ke zdrojovým obrazovým datům. Pro rychlou implementaci je tak třeba uložit v rychlé paměti FPGA (bloková paměť RAM -*BlockRAM*, nebo registry) oblast obrazu odpovídající minimálně velikosti detekčního okna. Dále je nutné vyřešit složitou logiku adresování. Mnoho operací je u sekvenčního vyhodnocení prováděno opakovaně a řada konvolucí, ze kterých se skládají slabé klasifikátory, je počítána opakovaně, jelikož je obsažena ve více slabých klasifikátorech.

U uvedeného způsobu vyhodnocení zpravidla nedochází k vyhodnocení slabých klasifikátorů mimo pořadí určené trénovacím procesem (pro WaldBoost je to nepřípustné). Všechny slabé klasifikátory tak jsou vyhodnoceny v přesně definovaném pořadí. Toto pořadí však není závazné a může být pro AdaBoost dle Freund a Schapire libovolně změněno. Proto mohou být dle tohoto původního algoritmu vyhodnoceny všechny slabé klasifikátory nezávisle a následně jen provedeno sečtení jejich výsledků a porovnání s prahem pro získání výsledku. U algoritmu AdaBoost dle autorů Viola a Johnes je možné provádět přeskládávání jen v rámci jednotlivých kaskád, jelikož pořadí jednotlivých kaskád je pro celkové vyhodnocení významné (po vyhodnocení každé kaskády se provádí prahování). Jak však bude ukázáno v kapitole 6, tak i tento princip lze upravit a s velkou výhodou využít.

Dle uvedených poznatků se tedy může nyní jevit jako nejvhodnější způsob pro čistě paralelní řešení v FPGA a dosažení maximální datové propustnosti právě originální AdaBoost dle Freund a Schapire i přesto, že počet operací nutný k jeho vyhodnocení je výrazně větší než v případě WaldBoost metody. Jelikož právě tento způsob dovoluje nezávislé vyhodnocení všech slabých klasifikátorů.

5.1.2 Neseřazené vyhodnocení detekčního okna

Předcházející kapitoly poukazují na nevýhody dosavadních známých implementací, které jsou založeny převážně na sekvenčním principu vyhodnocení. Architektura navržená v této práci se však snaží dosáhnout maximální možné propustnosti klasifikátoru. Toho však není možné dosáhnout používaným sekvenčním přístupem, ve kterém jsou paralelizovány jen vybrané části (např. výpočet příznaků), jak bylo ukázáno v podkapitole 3.3. Proto nyní představený způsob je založen na čistě paralelním řešení. Navíc nová architektura bude vystavěna tak, aby mohla nepřetržitě zpracovávat proud vstupních dat a nemusela do vyhodnocení vkládat žádné nežádoucí zpoždění a prodlužovat tak uměle vyhodnocení jednoho detekčního okna.

Prvním způsobem, kterým je možné realizovat paralelní implementaci AdaBoostu, je vyhodnocení všech slabých klasifikátorů v aktuálně zpracovávaném detekčním okně paralelně. To znamená, že musí být v jeden čas přístupny všechny pixely detekčního okna, které jsou použity v některém ze slabých klasifikátorů. Dále se musí nainstanciovat všechny slabé klasifikátory, u kterých je tak v podstatě znemožněno sdílené použití zdrojů v FPGA, jelikož všechny pracují v jeden čas, ale každý z nich pracuje na jiných datech. Po vyhodnocení všech slabých klasifikátorů je následně nutné provést sečtení jejich výsledků v jeden časový okamžik. To však vede k použití kaskády sčítaček

a registrů pro postupné sečtení výsledků za několik taktů a také mimo jiné i k mírnému zvýšení latence. Ta však není v tomto případě kritickým faktorem. Tento uvedený přímočarý způsob řešení je sice rychlý, ale z hlediska implementace v FPGA je velmi náročný na spotřebované výpočetní zdroje. Poznamenejme, že u tohoto způsobu vyhodnocení může být ohodnocení klasifikátoru spuštěno až v okamžiku, kdy jsou dostupná data z celého detekčního okna.

Jelikož přímočará jednoduchá paralelní implementace by nevedla k příliš efektivní implementaci v FPGA, byl navržen nový koncept zpracování kolektivem Filip Kadlček, Otto Fučík, Pavel Zemčík a Roman Juránek (FIT). Navržený koncept byl pak následně autorem práce zpracován, dále významně vylepšen a realizován do výsledné podoby (viz následující odstavce).

Jednou z nevýhod triviálního paralelního řešení je, že musí počkat, až je načteno celé detekční okno (nebo případně pixely obrazu spadajícího do nejpravějšího a nejspodnějšího slabého klasifikátoru – nejvyšší *X* a *Y* souřadnice). Pokud však provedeme úpravu vyhodnocení slabých klasifikátorů, tak není nutné čekat na data celého detekčního okna a jednotlivé klasifikátory mohou být vyhodnocovány již v čase, kdy jsou pro ně dostupná data. Takto lze zajistit velmi nízkou latenci klasifikátoru. Triviální řešení také využívá paměť pro uložení dat celého detekčního okna. Nové řešení se snaží paměť co nejvíce minimalizovat a ukládá jen obrazová data potřebná pro vyhodnocení největšího příznaku.

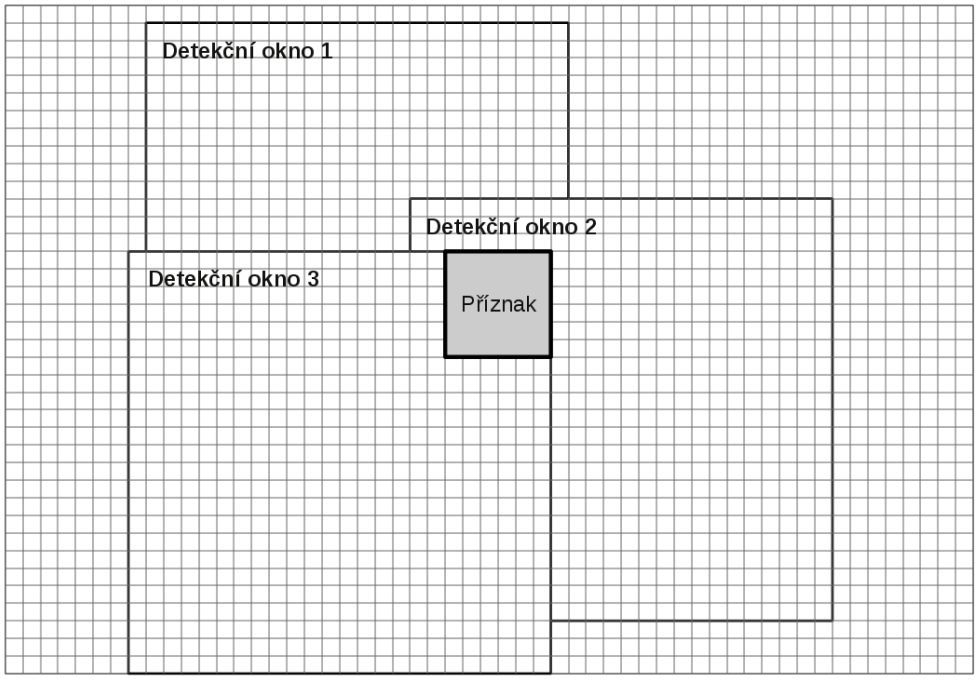
Nová architektura je založena na plně paralelním neseřazeném (ve smyslu výsledků trénování) vyhodnocení slabých klasifikátorů. Architektura tak zpracovává sice všechny slabé klasifikátory paralelně (v jednom kroku výpočtu), ale každý zpracovaný slabý klasifikátor může patřit do jiného detekčního okna. A jelikož architektura pracuje ihned, jakmile má dostupná data, tak je naprostá většina slabých klasifikátorů spočítána ještě dříve, než je načteno celé detekční okno. Latence tohoto přístupu je tak velmi malá.

V předchozích odstavcích byl naznačen princip vyhodnocení a nyní bude uveden detailně. Navržená architektura pracuje nad částí obrazu o velikosti největšího příznaku, který obsahuje silný klasifikátor. Pro případ této práce je to oblast o velikosti *6 x 6* pixelů. V této oblasti se provádí výpočet všech slabých klasifikátorů, které obsahuje zpracovávaný AdaBoost klasifikátor. Výsledky takto vyhodnocených slabých klasifikátorů však nepatří do jednoho detekčního okna, a nelze je tak jen jednoduše sečíst dle principu AdaBoost. Pokud bychom to udělali, znamenalo by to, že všechny slabé klasifikátory mají stejnou pozici, což je velmi nepravděpodobné vzhledem k principu AdaBoost klasifikátoru. Ve skutečnosti získané výsledky odpovídají několika detekčním oknům. Výsledky je proto nutné zpracovávat odlišným způsobem.

Nově uvedený způsob vyhodnocení pracuje tak, že má současně rozpracováno více detekčních oken a ke každému detekčnímu oknu udržuje akumulátor, ve kterém postupně provádí částečné součty dle AdaBoost metody. Počet současně rozpracovaných detekčních oken je roven počtu pixelů pruhu obrazu, který je určen výškou detekčního okna a šířkou vstupního obrazu po odečtení šířky detekčního okna. V každé pozici pásu obrazu začíná nové detekční okno. Na postupné sčítání výsledků lze také pohlédnout jako na přesně definované zpožděné sčítání výsledků slabých klasifikátorů do akumulátorů.

Předpokládejme, že máme silný klasifikátor se třemi slabými klasifikátory *P1*, *P2* a *P3*. Na obrázku 36 je znázorněno detekční okno a rozmístění jednotlivých slabých klasifikátorů. Ty jsou umístěny na následujících pozicích *x* a *y* (souřadnice jsou udány vzhledem k detekčnímu oknu):

- *P1 - x = 19, y = 0*
- *P2 - x = 4, y = 3*
- *P3 - x = 18, y = 14*



Obrázek 37: Zpracování příznaků v rámci několika detekčních oken.

Na obrázku 37 je znázorněno zpracování podoblasti detekčního okna o velikosti *6 x 6* pixelů dle principů nově uvedené architektury. Z obrázku lze vidět, že v jeden časový okamžik dochází ke zpracování všech příznaků paralelně, ale každý příznak patří do jiného detekčního okna. Na obrázku jsou pro ilustraci zobrazeny jen tři z celkového počtu *361* detekčních oken (*24 x 24* pixelů a velikost příznaku *6 x 6*). V jednom taktu tak dojde k vypracování částečných výsledků všech detekčních oken spadajících do této oblasti.

Vyhodnocovaná podoblast *6 x 6* pixelů se pixel po pixelu posouvá v obraze, a postupně tak dochází ke zpracování celého obrazu. Jakmile klasifikátor zpracuje oblast odpovídající jednomu detekčnímu oknu (případně poslednímu slabému klasifikátoru), tak bezprostředně pro něj vygeneruje výstup klasifikátoru. Tím je dosaženo velmi malé latence.

Pro správné sečtení výsledků všech slabých klasifikátorů je implementována v architektuře speciální lokální paměť pro ukládání částečných výsledků klasifikace. Pro každé z aktuálně vyhodnocovaných detekčních oken je v paměti vyhrazeno jedno paměťové místo. Po paralelním vyhodnocení všech slabých klasifikátorů dojde v jednom taktu k přičtení výsledků všech slabých klasifikátorů k příslušným paměťovým buňkám (akumulátorům). Poznamenejme, že uchování všech dílčích výsledků slabých klasifikátorů ze všech detekčních oken pro jejich následné sečtení a vyhodnocení by vedlo k potřebě velkého paměťového místa a implementaci velkých sčítaček, to však není nutné, jelikož dle principu vyhodnocení AdaBoost je možné produkovat částečné součty.

Jakmile je dostupný výsledek klasifikátoru *P2*, provede se sečtení jeho výstupu s výsledkem slabého klasifikátoru *P1* a hodnota se uloží do akumulátoru. Stejně tak i po získání příznaku *P3* se jeho hodnota přičte k aktuální hodnotě akumulátoru pro dané detekční okno. Pro uvedený klasifikátor tak dostaneme již celkový výsledek, se kterým provedeme prahování, a dostaneme tak požadovaný výsledek klasifikátoru. Přesné časy, kdy se mají provést součty jednolitých slabých klasifikátorů *P1*, *P2* a *P3*, jsou dány jejich pozicemi v detekčním okně a šířkou zpracovávaného obrazu. Pro uvedený

případ a za předpokladu, že se zpracovává obraz o šířce *512* pixelů, jsou zpoždění mezi jednotlivými slabými klasifikátory následující:

- mezi *P1* a *P2* = *1521*
- a mezi *P2* a *P3* = *5646* hodinových cyklů.

Pro stanovení zpoždění mezi jednotlivými slabými klasifikátory se nejprve provede jejich seřazení dle pozic v detekčním okně (první je nejhornější a nejlevější slabý klasifikátor). Zpoždění mezi klasifikátory se poté získá jednoduchým výpočtem:

$$delay(P1, P2) = (P2.y - P1.y) * IMGWidth + (P2.x - P1.y) \quad (5.1)$$

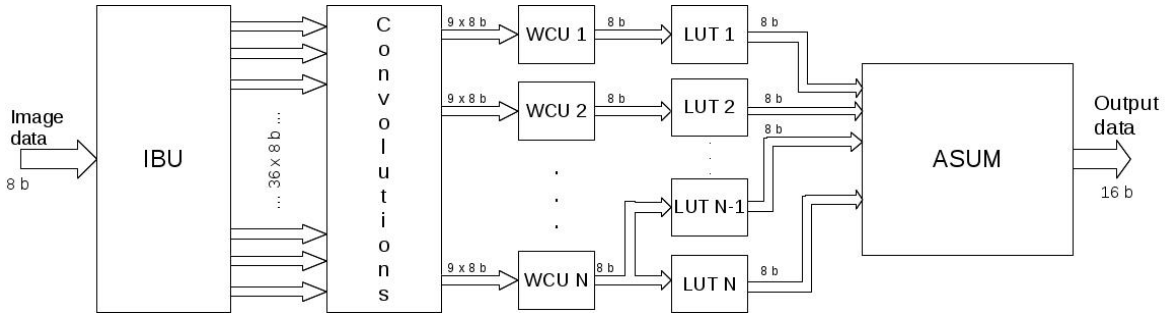
- Kde *P1, P2* představují slabé klasifikátory,
- zápis *P1.x* představuje pozici v horizontální ose obrazu a *P1.y* ve vertikální ose obrazu.
- *IMGWidth* představuje šířku obrazu.

Pro realizaci uvedeného příkladu je potřeba implementovat dvě zpožďovací linky. Počet zpožďovacích linek je vždy o jednu menší, než je počet slabých klasifikátorů (za posledním klasifikátorem není třeba generovat zpoždění – přímo se produkuje výsledek). Celková latence klasifikátoru je malá ve srovnání s ostatními řešeními (viz 5.2.5) a výsledky aktuálně zpracovaného detekčního okna jsou známy maximálně jen několik hodinových cyklů po dodání posledního pixelu obrazu do architektury. Pokud je však poslední slabý klasifikátor umístěn v dostatečné vzdálenosti od posledního pixelu detekčního okna, produkuje architektura výsledky klasifikace ještě dříve, než zná celý obraz (to je v případě, že poslední pixely v obraze nejsou použity žádným slabým klasifikátorem, a tedy pixely nejsou důležité pro detekci objetu).

Z uvedeného principu je vidět, že je založen na původním AdaBoost dle autorů Freund a Schapire, který umožňuje právě provádění neseřazeného vyhodnocení. Architektura se také snaží maximálně využít potenciálu FPGA, kdy za pomoci neuspořádaného vyhodnocení slabých klasifikátorů je dosaženo toho, že všechny slabé klasifikátory pracují paralelně a konečný výsledek klasifikace se získá vhodným sečtením dílčích výsledků se zpožděním. Dříve uvedená řešení (viz podkapitola 3.3) využívají velkých pamětí pro uložení dat celého detekčního okna. Uvedené řešení potřebu takové paměti eliminuje, ale na druhou stranu vytváří paměťové pole pro ukládání mezivýsledků klasifikátorů - akumulátorů. Vzhledem k dosažené úrovni paralelizace je ale tato transformace velmi prospěšná. Současně také bylo umožněno sdílení zdrojů mezi slabými klasifikátory tím, že pracují všechny na stejných datech, a tak velká část výpočtů, a tedy i výpočetních zdrojů, je sdílena. To v důsledku vede na významnou úsporu spotřebovaných zdrojů. Podrobnější informace lze nalézt v práci [37].

5.2 Architektura AdaBoost jádra (AC)

Na základě schématu vyhodnocení uvedeného v předchozí podkapitole byla navržena následující HW architektura klasifikátoru. V této práci bude uveden její zkrácený popis, kompletní popis architektury je dostupný v dřívější práci [37]. Klasifikátor je navržen jako zřetězená zpracovávající linka, která je rozdělena do pěti základních stupňů. Architektura navrženého klasifikátoru je zobrazena na obrázku 38. V následujících odstavcích je uveden popis jednotlivých částí, ze kterých se architektura skládá.



Obrázek 38: Architektura AdaBoost jádra klasifikátoru.

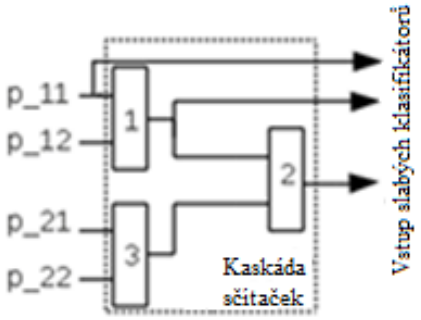
5.2.1 Obrazová paměť - *IBU*

Vstupní data architektury jsou tvořena proudem pixelů obrazu, který je zakódován v *256* stupních šedi. Pro uložení jednoho pixelu je tak zapotřebí 8 bitů. Vstupní obrazová data musí být v architektuře uložena, dokud nejsou zcela zpracována. Uložení celého obrazu v FPGA však není odůvodnitelné, a navíc je velmi drahé z hlediska spotřebovaných paměťových zdrojů. Proto *IBU* jednotka, jež je použita v architektuře, ukládá pouze tenký pásek vstupního obrazu. Minimální výška tohoto pásku obrazu je rovna výšce nejvyššího příznaku použitého ve slabých klasifikátorech, jež jsou použity v aktuálně konstruovaném silném klasifikátoru. Jelikož byla již dříve stanovena maximální velikost použitých LBP příznaků na *6 x 6* pixelů, je výška pásu obrazu právě *6* pixelů. Ve skutečnosti je však tento pásek ještě o jeden pixel zúžen na pouhých *5* pixelů, jelikož poslední načtený řádek se neukládá celý, ale jen jeho prvních *6* pixelů. Ukládání pouhého *5* pixelů vysokého pásku obrazu vede k výrazné úspoře paměťových zdrojů ve vstupní části architektury v porovnání s ostatními AdaBoost implementacemi, které velmi často potřebují uložit minimálně pás obrazu o výšce detekčního okna (což například pro detekci obličejů bývá velmi často *24* pixelů [25]). S ohledem na *ASUM* jednotu je ale nutné poznamenat, že v uvedené architektuře dochází v porovnání s ostatními architekturami, k ukládání mezivýsledků v *ASUM* jednotce, což snižuje celkovou dosaženou paměťovou úsporu. S každým hodinovým cyklem může jednotka číst ze svého vstupu jeden osmibitový pixel obrazu. Výstup jednotky je tvořen *36* osmibitovými hodnotami, které jsou produkovány každý hodinový cyklus. Výstup jednotky tak tvoří oblast pixelů, která odpovídá největšímu použitému LBP (případně MB-LBP) příznaku. S těmito daty je následně provedeno vyhodnocení všech slabých klasifikátorů paralelně. *IBU* se vnitřně skládá z několika registrů a řady zpožďujících linek implementovaných pomocí *BlockRAM* pamětí.

5.2.2 Blok konvolucí – *Convolutions*

Vyhodnocení slabého klasifikátoru je rozděleno do tří oddělených částí – výpočet konvolucí (*Convolutions*), výpočet samotného LBP operátoru (*WCU*) a tabulky s uloženými výsledky trénovacího procesu (*LUT*). Toto rozdělení bylo učiněno záměrně za účelem efektivní implementace a usnadnění možnosti sdílení FPGA zdrojů mezi jednotlivými slabými klasifikátory. *Convolutions* jednotka počítá konvoluce nad pixely vstupního obrazu (konvoluce představují sčítání pixelů jednotlivých oblastí MB-LBP a následně dělení vzniklého součtu příslušným počtem sčítanců). Konvoluční jednotka je uvnitř tvořena kaskádou sčítaček. Jelikož konvoluční jednotka počítá konvoluce všech slabých klasifikátoru současně, a to bez znalosti dalších souvislostí mezi slabými klasifikátory, tak může provádět sdílení částečných výpočtů konvolucí mezi různými slabými klasifikátory. Tímto způsobem je velmi efektivně proveden výpočet všech konvolucí všech slabých klasifikátorů s velmi malým počtem FPGA zdrojů.

Na obrázku 39 je zobrazena část konvoluční jednotky. Je zde ukázáno, jakým způsobem je prováděno sdílení částečných výsledků z jednotlivých částí konvolucí z různých slabých klasifikátorů. Některé výsledky konvolucí jsou získány až na konci sčítací kaskády, některé jsou získány již uprostřed sčítací kaskády a některé pixely jsou brány přímo ze vstupu sčítací kaskády (to znamená, že s nimi není nutné provádět výpočet konvoluce). Ve sčítací kaskádě nejsou žádné dvě operace sčítání prováděny na stejných datech. Množina konvolucí se samozřejmě liší pro každý klasifikátor, proto je sčítací kaskáda generována automaticky na základě popisu vstupního AdaBoost klasifikátoru. Výstup každé konvoluce je normalizován do 8bitové hodnoty. Normalizace je velmi jednoduchá, jelikož se jedná o dělení čísla mocninou dvou, kterou lze v FPGA provést za prakticky nulový čas a s prakticky nulovými požadavky na zdroje. Jedná se totiž o operaci logického posunu doprava (vynechání nejméně významných bitů). Uvedený způsob generování konvolucí redukuje nároky na FPGA zdroje (především *Slices*) až o *20 %* v porovnání se standardním přístupem uvedeným výše.



Obrázek 39:Kaskáda sčítaček.

5.2.3 Jednotka slabých klasifikátorů a vyhledávací tabulky

Vstup do jednotek slabých klasifikátorů (*WCU*) je tvořen konvolucemi z předchozí jednotky *Convolutions*. Architektura LBP, LRD nebo LRP operátorů je jednouchá a lze ji velmi efektivně implementovat v FPGA. Jednou z nejvýznamnějších výhod uvedené architektury je maximální počet instanciovaných LBP operátorů ve výsledném produktu, který není roven počtu slabých klasifikátorů, ale je roven počtu rozdílných LBP příznaků použitých v celém výsledném AdaBoost klasifikátoru. To je dáno tím, že všechny pracují na stejných vstupních datech. V této práci je většinou použito pouze několika málo tvarů příznaků (čtyři ručně navržené a jeden až dva evolučně navržené). Proto maximální počet instanciovaných operátorů v FPGA je právě *4-6* (avšak celkový počet operátorů použitých v klasifikátoru bývá několik set až několik tisíc).

Počet instanciovaných obrazových operátorů je tak redukován na minimální možný počet, který je nutný pro plně paralelní zpracování. Slabé klasifikátory se však skládají ještě i z vyhledávacích tabulek, které nesou výsledky trénování. Počet těchto tabulek však nemůže být redukován vzhledem k počtu slabých klasifikátorů obsažených v kompletním AdaBoost klasifikátoru. A zdroje potřebné pro implementaci *LUT* tak nemohou být redukovány nebo sdíleny – každý slabý klasifikátor potřebuje svou specifickou *LUT*. Ta se skládá v případě LBP operátoru z *256* osmibitových hodnot, které reprezentují odezvu slabého klasifikátoru na vstupní data. K datům všech *LUT* se přistupuje v každém hodinovém cyklu výpočtu a *LUT* musí okamžitě poskytovat výstupní hodnotu, která je adresována vstupní hodnotou. *LUT* je možné v FPGA implementovat jako distribuovanou paměť, která je pak složena z mnoha registrů (logických buněk) nebo z blokových pamětí (*BlockRAM*). Obsah všech *LUT* pro LBP operátory je však příliš velký na to, aby mohly být implementovány jen pomocí distribuované paměti, jelikož by se spotřebovalo neúměrné množství zdrojů v FPGA. Implementace pomocí

BlockRAM pamětí je mnohem vhodnější, ale i tato implementace má své nevýhody, které se však tato práce snaží eliminovat, či případně transformovat na výhodu. Jedna *LUT* potřebuje pro svoje uložení *2048* bitů paměti, avšak jedna *BlockRAM* má typicky *18 432* bitů (u novějších zařízení Xilinx Zynq UltraScale je to až *36 864* bitů [90]). Proto pokud by se pro implementaci jedné LUT použila jedna celá *BlockRAM,* docházelo by tak k masivnímu plýtvání drahého paměťového místa.

Výhoda *BlockRAM* pamětí, které využijeme, je, že poskytují dva zcela samostatné a také samostatně konfigurovatelné vstupy a výstupy. Avšak i přes to, pokud bychom do jedné *BlockRAM* umístili dvě *LUT*, tak by stále zůstávalo nevyužito *14 336* bitů paměti, což opět vede na velké plýtvání. Aby se zabránilo tak rozsáhlému plýtvání pamětí, byla představena následující optimalizační technika pro ukládání *LUT* do *BlockRAM*. Klíčová myšlenka optimalizace je založena na již dříve využitém a uvedeném faktu, že několik *LUT* má společné vstupy (adresy), které jsou generovány stejným LBP operátorem. Tato skutečnost vede k možnosti implementovat několik *LUT* v rámci jedné *BlockRAM* paměti. Obsah několika *LUT*, které jsou adresovány stejnou *WCU* jednotkou (stejný LBP operátor), může být spojen do jedné *BlockRAM*. Datová šířka výstupu *BlockRAM* může být nastavena několika způsoby, a to od šířky *8* bitů až do šířky *72* bitů (pokud využijeme i paritní bity pro ukládání dat). V jedné *BlockRAM* tak může být uložen obsah *1* až *9 LUT*. Maximální počet *9 LUT* však může být instanciován do jedné *BlockRAM* pouze za předpokladu, že všechny *LUT* používají jako svůj adresový vstup stejnou *WCU* jednotku. Je-li *BlockRAM* nakonfigurována na dvouportový režim, tak maximální počet instanciovaných LUT je 8, jelikož pak není možné využít datovou oblast pro ukládání parity. Jsou-li pak do jedné *BlockRAM* instanciovány dvě množiny *LUT,* tak každá z množin musí mít velikost menší než čtyři prvky. Poté je *BlockRAM* nakonfigurována tak, aby využívala dva samostatné vstupní a výstupní porty se šířkou *32* bitů (jiná šířka nemůže být použita vzhledem k adresování v *BlockRAM*; šířka adresy je pak *8* bitů). Jestliže je v této konfiguraci adresováno nějaké paměťové místo, tak jsou vždy navráceny čtyři hodnoty ze čtyř rozdílných *LUT* čtyř slabých klasifikátorů. Poznamenejme, že *BlockRAM* vždy zpřístupní na svém výstupu spojitou oblast své paměti. Proto, aby bylo možné dosáhnout požadovaného chování (uložení obsahu několika *LUT* do jedné *BlockRAM*), data jednotlivých *LUT* musí být v *BlockRAM* uložena s prokládáním. To znamená, že data jedné *LUT* jsou uložena vždy s offsetem *32* bitů. Offset pro ukládání musí být použit ve všech konfiguracích, ať už ukládáme obsah jedné, dvou, tří nebo čtyř *LUT*. Veškerý obsah a také propojení *BlockRAM* pamětí je generován automaticky pomocí speciálně vytvořeného automatického nástroje. Použitím tohoto přístupu ukládání obsahu *LUT* do *BlockRAM* je dosaženo nezanedbatelné úspory FPGA zdrojů, a to až *60 %* v porovnání s přístupem, který by tuto činnost neprováděl.

5.2.4 AdaBoost sčítací jednotka – *ASUM*

AdaBoost sčítací jednotka *ASUM* je klíčovou jednotkou celé architektury. Tato jednotka dovoluje provádět neseřazené vyhodnocení slabých klasifikátorů a následně zajišťuje správné sečtení výsledků slabých klasifikátorů. Výstup každé *LUT* je připojen na vstup této jednotky a výstup (výsledek) této jednotky je celkovým výsledkem klasifikátoru pro aktuálně zpracované detekční okno. Výstup jednotky lze chápat jako odhad pravděpodobnosti, že aktuálně zpracované detekční okno obsahuje hledaný objekt. *ASUM* je složena z mnoha *FIFO* (*First In First Out* - první dovnitř, první ven – zpožďovací linka) linek a sčítaček. Počet sčítaček a *FIFO* linek je dán počtem slabých klasifikátorů, které tvoří výsledný silný klasifikátor. Délka jednotlivých FIFO linek je dána pozicemi slabých klasifikátorů v detekčním okně a délka každé *FIFO* linky se určí pomocí následující rovnice *délka_akt_FIFO = pozice_akt_WC – pozice_předch_WC* (v případě první FIFO linky je vzdálenost brána k počátku detekčního okna). Kde *pozice_akt_WC* je pozice aktuálního slabého klasifikátoru v

rámci detekčního okna a *pozice_předch_WC* je pozice předcházejícího slabého klasifikátoru v detekčním okně.

V FPGA lze *FIFO* linky vytvořit jednak pomocí jednotlivých paměťových buněk (registrů) nebo použitím *BlockRAM* pamětí. První přístup je vhodný pro *FIFO* linky s malou délkou (to je například méně než deset). Použití *BlockRAM* pamětí je naopak výhodné pro delší *FIFO* linky (jelikož ke každému *FIFO* implementovanému pomocí *BlockRAM* je třeba vytvořit i čítač pozice ve *FIFO* lince a současně je zabrán jeden port *BlockRAM* paměti). Implementace, která je vytvořena s maximálním ohledem na úsporu paměťových zdrojů v FPGA, je založena na kombinaci obou zde uvedených principů. V případě použití *BlockRAM* pro sestrojení *FIFO*, jsou využity oba porty a každý z portů implementuje jedno nezávislé FIFO s různou délkou. Architektura se snaží maximálně využívat kapacitu *BlockRAM* pamětí tak, že generuje do jedné *BlockRAM* paměti *FIFO* linky s vhodnými délkami. Pro maximalizaci využití *BlockRAM* je implementován speciální algoritmus, který provádí uvedené mapování *FIFO* linek na *BlockRAM* paměti. Algoritmus je založen na principu kombinace dvou *FIFO* linek do jedné *BlockRAM* paměti, kdy se snaží vybrat z množiny *FIFO* linek, které mají být implementovány, jednu krátkou *FIFO* linku a do zbývající části *BlockRAM* paměti se snaží umístit *FIFO* linku, která ji kompletně zaplní. Některé *FIFO* linky mohou být rozděleny do dvou či více *BlockRAM* pamětí. Tento velmi efektivní způsob mapování *FIFO* linek do *BlockRAM* pamětí pomáhá uspořit až *50 % BlockRAM* pamětí na FPGA.

5.2.5 Latence AC jednotky

Z hlediska dalšího využití je důležité uvést latenci přestavené AC jednotky. Obvod je tvořen pomocí několika zřetězených stupňů a každý stupeň vytváří své zpoždění. Tabulka 7 uvádí zpoždění každého stupně zřetězené linky. Součet zpoždění (latencí) jednotlivých stupňů pak dává celkovou latenci AC jednotky. Jelikož je obvod generovaný z vyššího programovacího jazyka a každý silný klasifikátor má jiné požadavky, může se zpoždění pro každou implementaci mírně lišit. Největší změny jsou v jednotce provádějící výpočet konvolucí, kde je v závislosti na požadovaných konvolucích zpoždění od *1* do *4* hodinových cyklů. Celková latence AC jednotky je *5 – 9* hodinových cyklů (CLK).

Tabulka 7: Latence jednotlivých stupňů zřetězeného zpracování.

Název obvodu	**Zpoždění**
IBU	1CLK
Convolutions	1 – 4 CLK (dle použitých konvolucí)
WCU	1 nebo 2 CLK
LUT	1 CLK
ASUM	1 CLK
Celkem	**5– 9 CLK**

5.2.6 Propustnost

Nejvýznamnější předností AC jednotky je konstantní čas vyhodnocení jednoho detekčního okna (*1* hodinový cyklus) a nezávislost latence na počtu slabých klasifikátorů. To vytváří architekturu, která má garantovanou datovou propustnost a je necitlivá na obsah zpracovávaných dat na rozdíl od implementací založených na metodě WaldBoost. Tyto vlastnosti jsou získány použitím velkého množství *WCU* jednotek pracujících paralelně a také použitím velmi dlouhé zřetězené linky, která je primárně tvořena pomocí *ASUM* jednotky. Délka linky je dána velikostí detekčního okna a šířkou

vstupního obrazu. Pro vstupní obraz o šířce *1920* pixelů a detekční okno o velikosti *24 x 24* pixelů je délka linky *44 184* stupňů. Po naplnění linky jsou však výsledky klasifikace produkovány v každém hodinovém taktu.

5.3 Automatická syntéza klasifikátoru

Dříve představená řešení [47], [96] jsou postavena na typickém způsobu konstrukce AdaBoost klasifikátoru v FPGA. Ten spočívá v tom, že autoři uchopili princip funkce runtime části AdaBoost algoritmu a výstup z trénovacího procesu a navrhli univerzální jednotku pro FPGA, která dokáže pracovat s různě natrénovanými klasifikátory. Vstupem takové univerzální klasifikační jednotky je poté zpravidla klasifikátor, který je překódován do mikroinstrukcí, které řídí kompletní proces vyhodnocení klasifikátoru. Klasifikátor pak může být rekonfigurován na základě změny mikroprogramu klasifikační jednotky v FPGA. Takovýto přístup je logický a má řadu opodstatnění, mezi něž například patří: jedna jednotka pro různé typy klasifikačních úloh, není třeba rekonfigurovat FPGA při změně klasifikátoru, rychlá změna klasifikátoru a podobně. Avšak některé z uvedených předností takového řešení jsou omezené. Doposud implementované klasifikátory [96], [48] zpravidla dovolují změnu klasifikátoru jen za klasifikátor s podobnými vlastnostmi, to znamená omezený počet slabých klasifikátorů, velikost detekčního okna, omezená množina tvarů příznaků [96] a rozlišení vstupního obrazu. Jakákoliv z těchto uvedených změn (ale i jiné zde neuvedené změny) vedou k vytvoření nové implementace klasifikátoru v FPGA nebo minimálně ke změně generických parametrů architektury [47] a následné syntéze výsledného popisu klasifikátoru do FPGA. Vzhledem k tomu, že univerzální jednotka musí umět zpracovávat klasifikátor dle obecného popisu, tak zpravidla vyžaduje nemalé množství FPGA zdrojů na implementaci řídící logiky klasifikátoru v závislosti na výsledcích trénování. Pro řadu aplikací však není potřeba klasifikátor měnit nebo jeho rekonfigurace může být pomalá (několik sekund až minut). Takový klasifikátor je jednou vytvořen a poté je už jen používán po dlouhý čas bez jakékoliv změny.

Otázkou je, zda potřebujeme univerzální klasifikační jednotky nebo by bylo výhodnější použití aplikačně specializované klasifikační jednotky. Odpověď se zdá být jednoduchá, ve většině vestavěných systémů v průmyslu nebo dopravě není třeba používat univerzální jednotky, jelikož ke změně klasifikátoru dochází jen několikrát za životnost systému. Univerzální jednotka s možností rychlé změny tak není třeba. Uvážíme-li klasifikátor, který není možné za běhu rekonfigurovat, tak se otevírá možnost vytvoření aplikačně specifického klasifikátoru (ASC) pro aktuální úlohu. Taková klasifikační jednotka může být v porovnání s jednotkami vytvořenými na univerzálním přístupu vysoce optimalizována, což v důsledku znamená, že může konzumovat menší množství zdrojů FPGA, ale současně může být výrazně rychlejší při zpracování obrazu.

Jelikož struktura ASC je pro každou klasifikační úlohu odlišná, je nutné pro každý klasifikátor znovu sestavit kompletní FPGA popis. To je možné udělat buď manuálně, nebo lépe za pomoci automatizovaného nástroje. Manuální sestavení je příliš zdlouhavé a příliš nákladné na čas a lidské zdroje, proto jej dále nebudeme uvažovat. Kdežto automatizované sestavení dovoluje velmi efektivně reagovat na jakékoliv změny ve vstupním popisu klasifikátoru, ať už je to změna rozlišení vstupního obrazu, přidání dalších slabých klasifikátorů, změna velikosti detekčního okna a podobně. Automatizované sestavení klasifikátoru má také výhodu v tom, že budou instanciovány jenom ty jednotky a ty paměťové prvky, které budou opravdu využívány. Nemusí se tedy uvažovat možná změna rozlišení v budoucnu a podobně. Vstupem do nástroje pro automatizované sestavení klasifikátoru je zpravidla popis natrénovaného klasifikátoru. Výstupem je poté popis kompletního klasifikátoru v jazyce popisujícím hardware. V této práci byl zvolen výstup do VHDL (*Very-High*

Speed Integrated Circuits Hardware Description Language). Popis klasifikátoru ve VHDL je generován z vyššího programovacího jazyka, v této práci bylo použito jazyka C++. Sestavení klasifikátoru přestaveného v podkapitole 5.2 je založeno právě na automatizované syntéze. Ta také umožňuje provádět řadu dříve představených optimalizací vzhledem ke spotřebovaným zdrojům FGPA. Popis jednotlivých optimalizací pro architekturu byl již po částech uveden v podkapitole 5.2 a dále pak velmi detailní popis je uveden v práci [37]. Nejvýznamnější optimalizace spočívají v redukci zdrojů konvolučních jednotek, redukci počtu vyhodnocení operátorů slabých klasifikátorů, optimalizaci uložení tabulek s výsledky trénování a velmi důsledné práci s pamětí při vytváření *FIFO* linek. Veškeré zde uvedené výsledky pro klasifikátor z podkapitoly 5.2 jsou získány právě pomocí nástroje pro automatizované sestavení klasifikátoru. Hlavní nevýhodou uvedeného přístupu je nemožnost ovlivnit klasifikátor v FPGA beze změny FPGA designu, což však pro většinu uvažovaných aplikací není problémem. Pokud by však takový požadavek nastal, je možné nový klasifikátor znovu sestavit, syntetizovat a provést rekonfiguraci FPGA. Tím tak lze zajistit možnost pomalé změny klasifikátoru. Klasifikační přesnosti syntetizovaného a univerzálního klasifikátoru jsou si rovny, jelikož v obou případech může být použito naprosto stejných klasifikátorů a stejného typu vyhodnocení.

5.4 Dosažené výsledky AC jednotky

Tato podkapitola ukazuje výsledky dosahované AC jednotkou ze dvou úhlů pohledu. První je klasifikační přesnost implementace AdaBoost metody na FPGA v porovnání s implementací na CPU. A druhým úhlem pohledu, pro účely této práce mnohem důležitějším, jsou nároky na zdroje v FPGA a rychlost implementované architektury, které říkají, jak velké FPGA jednotka potřebuje a kolik obrazových dat bude schopna zpracovat.

5.4.1 Klasifikační přesnost

Na klasifikační přesnost má největší vliv použitá metoda. Ta je v našem případě pevně dána a je jí AdaBoost. Přesnost AdaBoost metody pak dále nejvíce ovlivňuje typ a počet použitých slabých klasifikátorů. Slabé klasifikátory v této práci jsou založeny na LBP operátorech vzhledem k jejich velmi dobrým výsledkům (viz sekce 2.4.2). Z hlediska finálního klasifikátoru tak nejvíce ovlivňuje přesnost počet slabých klasifikátorů. Přesnost klasifikátoru je dána procesem trénování a účelem této práce není prokázat, jaké přesnosti může AdaBoost klasifikátor dosáhnout. Jelikož je však přesnost klasifikace důležitá, tak jsou v tabulce 8 mimo jiné uvedeny i přibližné přesnosti klasifikátorů vzhledem k počtu slabých klasifikátorů. Klasifikační přesnost jednotky v FPGA však může být ovlivněna i jinými faktory než dříve uvedenými. Jsou jimi například způsob implementace matematických operací, aproximace některých výpočtů v FPGA, výpočet s menším počtem bitů, a tedy i snížení přesnosti a podobně.

Architektura AC jednotky má jen minimální vliv na přesnost výsledků AdaBoost klasifikátoru. Jediné zhoršení klasifikační přesnosti, které může nastat, je dáno použitím aritmetiky s pevnou desetinnou čárkou pro výpočet AdaBoost klasifikátoru na místo použití aritmetiky s plovoucí desetinnou čárkou. Aritmetika s pevnou desetinnou čárkou má v případě výpočtu s čísly s neukončeným nebo dlouhým desetinným rozvojem nižší přesnost než aritmetika s plovoucí desetinnou čárkou. Zhoršení přesnosti se může nejvíce projevit u výpočtu slabých klasifikátorů a u sčítání jejich výsledků (viz rovnice AdaBoostu 2.17). V původním AdaBoostu mají slabé klasifikátory data získaná v procesu trénování uložena v aritmetice s plovoucí desetinnou čárkou. To je však pro reálnou implementaci na FPGA velmi nevhodné. Proto se všechny FPGA implementace snaží tomuto

předejít. Jedním z možných řešení je převod výsledků získaných trénováním do aritmetiky s pevnou desetinnou čárkou (odkud je velmi snadné provést bezztrátový převod do celých čísel). Aby se však minimalizovalo zhoršení klasifikační přesnosti na minimum, byl upraven trénovací algoritmus tak, aby pracoval právě s čísly v aritmetice s pevnou desetinnou čárkou. Celkové negativní ovlivnění přesnosti použitím aritmetiky s pevnou desetinnou částkou je tak minimalizováno a ke zhoršení přesnosti tak nemusí nutně dojít. Autoři Hiromoto a kolektiv ve své práci [22] ukázali, že zhoršení přesnosti je menší než 1 %.

5.4.2 Nároky za zdroje v FPGA

V úvodní práci [37], která poprvé představovala AC jednotku, jsou dány výsledky syntézy a rychlosti zpracování vzhledem k dnes již poměrně zastaralým FPGA jednotkám Xilinx Spartan3 XC3S1000 a Xilinx Virtex-II 250 [88]. Dále pak byl klasifikátor syntetizován pro malá rozlišení obrazu (*512 x 512* pixelů). Nyní se již s nasazením klasifikátoru v takovýchto FPGA nepočítá, a proto byly provedeny nové experimenty, které ukazují možnosti architektury vzhledem k dnes moderním čipům rodiny Xilinx Zynq [89]. Stejně tak prvotní implementace předpokládaly pouze instanciování velmi malých klasifikátorů, avšak s příchodem nových FPGA technologií, tedy především dostupnosti většího počtu zdrojů na FPGA a propojení FPGA s univerzálním procesorem ARM, jako je například rodina čipů Xilinx Zynq [89], bylo umožněno uvažovat o implementaci klasifikátoru s velkým počtem slabých klasifikátorů.

AC jednotka je záměrně navržena tak, aby používala co nejmenší množství FPGA zdrojů [38]. Množství spotřebovaných FPGA zdrojů je nejvíce závislé na počtu slabých klasifikátorů, ze kterých je výsledný klasifikátor složen. Dalšími parametry, které významně ovlivňují množství spotřebovaných zdrojů, je šířka vstupního obrazu a výška detekčního okna. Tyto parametry nepřímo určují minimální velikost paměti, která je požadována pro zajištění funkčnosti klasifikátoru. Tabulka 8 ukazuje závislost spotřebovaných FPGA zdrojů AC jednotkou na počtu slabých klasifikátorů. Výsledky uvedené v tabulce jsou získány pro klasifikátor, který zpracovává *FullHD* obraz a využívá klasifikátor s velikostí detekčního okna *24 x 24* pixelů. Jako cílové FPGA byla zvolena rodina FGPA Xilinx Zynq. Například největší FPGA z této rodiny je XC7Z100, které má celkový počet dostupných zdrojů [89]:

- *Slice registry – 554 800,*
- *Slice LUT – 277 400,*
- *BlockRAM* paměti - *755* (*3* MB).

Z tabulky 8 je možné odvodit, že jádro AdaBoost jednotky obsahuje určitou konstantní část, která není závislá na počtu použitých slabých klasifikátorů. Tato část AC jednotky je použita především k ukládání velmi tenkého proužku obrazu a také pro vytváření zpožďovacích *FIFO* linek pro *ASUM* jednotku. Tato konstantní část klasifikátoru zkonzumuje přibližně *1400 Slice LUTs* a *20 BlockRAM* pamětí. Zbývající část zdrojů je konzumována převážně slabými klasifikátory. Pro implementaci jednoho slabého klasifikátoru v FPGA je třeba cca *10 Slice LUT* a *0,12 BlockRAM* paměti. Závislost spotřeby FPGA zdrojů v závislosti na počtu slabých klasifikátorů je pak téměř lineární. První sloupec tabulky udává počet slabých klasifikátorů. Druhý sloupec tabulky ukazuje přibližný *rejection rate* (procento úspěšně zamítnutých detekčních oken). Třetí sloupec ukazuje přibližnou klasifikační přesnost. Tyto hodnoty jsou experimentálně získány [26] a zpravidla se mírně liší pro každý klasifikátor. Zbývající sloupce definují spotřebu FPGA zdrojů pro implementaci daného klasifikátoru. Spotřeba zdrojů je uvedena jednak absolutně, ale také v procentuální míře využitelnosti daného FPGA. Z tabulky je tak velmi snadno vidět, že klasifikátor s malým počtem slabých klasifikátorů je možné implementovat i na poměrně malých FPGA, jako je Zynq 7020, ale naopak pro velké klasifikátory

s více než *1 000* slabých klasifikátorů (jen pro připomenutí práce [96] používá ve své práci pouhých *128* slabých klasifikátorů) je potřeba FPGA s velkým množstvím dostupných zdrojů.

Počet slabých klasifikátorů je nejdůležitější z hlediska spotřeby zdrojů, jelikož se může měnit ve velmi velkém rozmezí. Neméně důležitým parametrem je však také závislost spotřebovaných FPGA zdrojů na velikosti vstupního obrazu. Tato závislost je zobrazena v tabulce 9. Výsledky v této tabulce jsou uvedeny pro AdaBoost klasifikátor, který se skládá ze *100* slabých klasifikátorů založených na LBP operátorech a velikost detekčního okna je *24 x 24* pixelů. Z tabulky je vidět, že spotřeba FPGA zdrojů roste jen pozvolna vzhledem k velikosti vstupního obrazu. Díky tomu je poté možné klasifikátor použít i pro zpracování obrazu s velkým rozlišením a stále zachovat přijatelné požadavky na FPGA zdroje. První sloupec tabulky udává rozlišení obrazu. Zbývající sloupce stejně jako u předchozí tabulky udávají spotřebu zdrojů a procentuální využití jednotlivých FPGA.

K výsledkům uvedeným v tabulkách 8 a 9 je nutno podotknout, že jsou jen pro AC jednotku bez jakýchkoliv pomocných jednotek, jako je například jednotka pro změnu rozlišení vstupního obrazu, vizualizace výstupu a podobně. Výsledky jsou získány za pomoci syntetizačního nástroje Xilinx Vivado 2013.2. Uvedeny jsou výsledky po *Place & Route* operaci (v případě, že bylo FPGA příliš malé, jsou uvedeny výsledky po syntéze).

Tabulka 8: AC jednotka – Závislost spotřeby FPGA zdrojů na počtu slabých klasifikátorů pro vstupní obraz s FullHD rozlišením.

Poč. slabých klasifik.	Rejection rate Klasifikátoru	Přesnost Klasif.	Slice Reg.	Slice LUTs	BRAM	Xilinx Zynq 7020		Xilinx Zynq 7045		Xilinx Zynq 7100	
						LUT	BRAM	LUT	BRAM	LUT	BRAM
10	~75 %	~50 %	1323	1560	22	2.9 %	15.7 %	0.7 %	4,0 %	0.6 %	2.9 %
20	~90 %	~60 %	1578	1827	22	3.4 %	15,7 %	0.8 %	4,0 %	0.7 %	2.9 %
50	~98.5 %	~80 %	2119	2309	26	4.3 %	18.6 %	1.1 %	4.8 %	0.8 %	3.4 %
70	~99.2 %	~85 %	2423	2609	28	4.9 %	20.0 %	1.1 %	5.1 %	0.9 %	3.7 %
100	~99.6 %	~87 %	2490	2877	34	5.4 %	24.3 %	1.3 %	6.2 %	1.0 %	4.5 %
200	~99.9 %	~91 %	2498	3079	45	5.8 %	32.1 %	1.4 %	8.3 %	1.1 %	5.9 %
500	~99.95 %	~93 %	2502	6629	79	12.5 %	56.4 %	3.0 %	14.5 %	2.4 %	10.5 %
1000	~99.98 %	~94 %	2516	11658	133	21.9 %	95.7 %	5.3 %	24.6 %	4.2 %	17.7 %
1500	~99.999 %	~95 %	2510	16591	190	31.1 %	135.7 %	7.6 %	34.9 %	6.0 %	25.2 %

Tabulka 9: AC jednotka – Závislost spotřeby FPGA zdrojů na velikosti vstupního obrazu, počet použitých slabých klasifikátorů je 100.

Rozlišení	Slice Registers	Slice LUTs	BRAM	Xilinx Zynq 7020		Xilinx Zynq 7045		Xilinx Zynq 7100	
				LUT	BRAM	LUT	BRAM	LUT	BRAM
256 x 256	1970	2201	18	4.2 %	12.9 %	1.1 %	3.4 %	0.8 %	2.4 %
512 x 512	2022	2259	19	4.3 %	13.6 %	1.1 %	3.5 %	0.9 %	2.6 %
640 x 480	2050	2305	21	4.4 %	14.3 %	1.1 %	3.7 %	0.9 %	2.7 %
1024 x 768	2253	2499	24	4.7 %	16.5 %	1.2 %	4.3 %	1.0 %	3.1 %
1920 x 1200	2490	2877	34	5.5 %	24.3 %	1.4 %	6.3 %	1.1 %	4.6 %
3000 x 2000	2830	3319	46	6.3 %	32.9 %	1.6 %	8.5 %	1.2 %	6.1 %

5.5 Iterační proces návrhu AC jednotky

Proces automatizovaného sestavení AC jednotky byl představen již v podkapitole 5.2.6. Tento proces je však možné ještě dále optimalizovat a rozvinout za účelem nalezení co nejvhodnějšího klasifikátoru. V procesu automatizovaného generování klasifikátoru se uvažují optimalizace z hlediska FPGA kódu použitelného v libovolném FPGA. Tento kód tak vygeneruje co nejmenší klasifikátor s definovanými vlastnostmi. Z praktického hlediska však může být pohled na problematiku mírně odlišný. V praxi

velmi často potřebujeme do vybraného FPGA umístit co nejlepší klasifikátor. A právě technika uvedená v následujících odstavcích se zabývá touto problematikou.

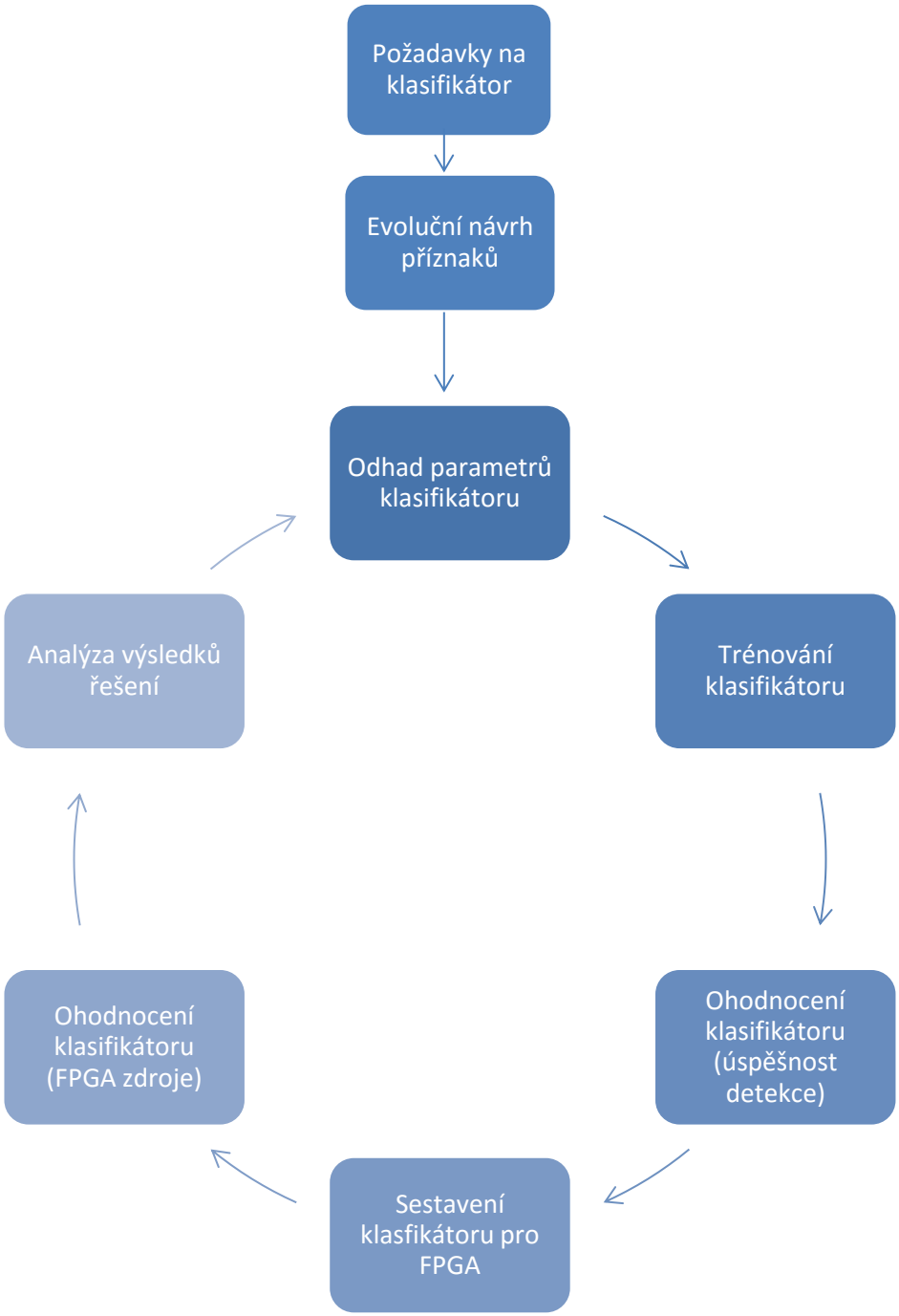
Tato podkapitola uvádí techniku, která je uvedena jako nadstavba nad automatizovanou syntézou klasifikátoru. Konvenční metody pro sestavení klasifikátoru jsou postaveny na znalostech experta. Expert určí všechny parametry klasifikátoru, ten se poté natrénuje a výstup se implementuje v FPGA. Proces návrhu je tak ve své podstatě rozdělen do dvou samostatných kroků. Prvním krokem je natrénování klasifikátoru do FPGA a získání popisu klasifikátoru. Druhým krokem je poté vytvoření klasifikátoru v FPGA na základě výsledků získaných v procesu trénování. Problémem této strategie ovšem velmi často bývá, že oba procesy jsou od sebe odděleny a proces trénování nebere ohledy na budoucí implementaci v FPGA. Dále pak při sestavování klasifikátoru v FPGA zpravidla již není možné měnit natrénovaný klasifikátor, a jsou tak sestavovány rozmanité architektury za účelem co nejefektivnější implementace klasifikátoru v FPGA. V této kapitole je představena technika, jak tuto bariéru mezi oběma uvedenými částmi efektivně odstranit.

5.5.1 Iterační schéma syntézy klasifikátoru

Pro navržení efektivního řešení klasifikátoru byla navržena iterační metoda, která v několika krocích automatizovaně nalezne vhodnou implementaci klasifikátoru v FPGA dle zadaných vstupních kritérií. Předchozí podkapitola 5.2 popisuje několik způsobů pro optimalizaci klasifikátoru již v procesu automatického sestavení. Při vytváření architektury a také následně při zkoumání vygenerovaných klasifikátorů a jejich výsledků, však bylo rozpoznáno, že je možné klasifikátor ještě více optimalizovat. Hlavním poznatkem, na kterém je postavena optimalizace, je, že slabé klasifikátory stejného typu mezi sebou sdílí velké množství zdrojů. V podstatě jediná část, kterou se liší, jsou *LUT* tabulky s výsledky trénování. Dále pak bylo detekováno, že vzhledem k implementaci v FPGA není cena jedné *LUT* vzhledem k zabraným zdrojům v FPGA konstantní. V podkapitole 5.2.3 je uvedeno, že cena jedné LUT může být od *1 BlockRAM* (nejdražší) až do *0,12 BlockRAM* (nejlevnější) v závislosti na tom, kolik LUT se podaří umístit do jedné *BlockRAM* paměti. Je-li implementován klasifikátor, který nemá plně využity všechny *BlockRAM* paměti, tak je možné jej rozšířit o další slabé klasifikátory vhodného typu (takový typ slabého klasifikátoru, pro který ještě nejsou *BlockRAM* paměti plně využity), a takové rozšíření bude ve výsledku velmi levné z hlediska spotřebovaných zdrojů FPGA a současně se zvýší klasifikační přesnost.

Na obrázku 40 je uvedeno schéma navržené iterační metody pro návrh klasifikátoru. Metoda se skládá ze dvou částí. První část se provádí jen v iniciálním cyklu a má za úkol získat základní parametry pro klasifikátor. Druhá část metody je již iterační cyklus, který se na základě uživatelem zadaných parametrů snaží naleznout co nejefektivnější implementaci klasifikátoru v FPGA. Jednotlivé části metody lze popsat následovně:

- *Požadavky na klasifikátor*: V této části určí expert základní požadavky na klasifikátor, kterými jsou požadovaná minimální klasifikační přesnost, požadovaná maximální spotřeba zdrojů klasifikátorem a cílové FPGA (má vyšší prioritu, než je přesnost klasifikátoru, a pokud nemůže být přesnost dosažena, tak je ignorována), velikost detekčního okna, data trénovací množiny (sada pozitivní a negativních trénovacích vzorků) a data pro testování klasifikátoru.
- *Evoluční návrh příznaků* je kompletně popsán v kapitole 4. V iteračním cyklu je prováděn jako druhá operace, která se provádí jen jedenkrát. Tato operace je však již plně automatizována a nepotřebuje zásah experta. Výstupem této operace jsou nové tvary příznaků pro LBP operátory, které jsou optimalizovány pro právě zpracovávanou úlohu



Obrázek 40: Schéma iteračního procesu návrh klasifikátoru.

(konkrétní typ předmětu). Jelikož se typ předmětu nemění, je možné tuto operaci provést jen jedenkrát na počátku procesu, a pak již výsledky v dalších iteracích jen opakovaně využívat.

- *Odhad parametrů klasifikátoru:* v tomto kroku se provádí odhad parametrů klasifikátoru a provádí se nastavení všech parametrů získaných v úvodních procesech *Požadavky na klasifikátor* a *Evoluční návrh příznaků*. Na základě těchto parametrů se provede odhad předpokládaného počtu slabých klasifikátorů. V úvodní iteraci jsou parametry počítány jen jako odhad dle kritérií a znalostí přednastavených expertem. V dalších kolech jsou odhady zpřesňovány na základě výsledků dosažených v přechozích iteracích. Tedy v *2.* až *N*-té

iteraci se provádí odhad s ohledem na typy slabých klasifikátorům, které jsou již použity. V těchto iteracích tak může docházet k omezení počtu slabých klasifikátorů, nebo případně k navýšení cílového počtu slabých klasifikátorů. V obou případech je klasifikátor znovu přetrénován, jelikož výsledků z předchozích iterací je možné použít jen velmi obtížně (každý krok trénování, včetně ohodnocení trénovací sady, by se musel zapamatovat – což vede k příliš velké spotřebě paměti). Uvedené požadavky vzešly především vzhledem k optimalizačním procesům pro umístění klasifikátorů v *BlockRAM*, které od určitých fází trénovacího procesu favorizují jen skupiny klasifikátorů vhodného typu.

- *Trénování klasifikátoru*: bylo pro účely iterativní metody návrhu klasifikátoru speciálně upraveno. Úprava se týká především vstupních parametrů, které určují, kolik slabých klasifikátorů má obsahovat výsledný klasifikátor, a také byl upraven způsob výběru slabých klasifikátorů do výsledné množiny. Po dosažení stanoveného počtu slabých klasifikátorů (tyto klasifikátory jsou vybírány standardním způsobem) se změní metodika pro jejich výběr. Hranice změny metody výběru se určuje dynamicky za běhu tak, že po přidání slabého klasifikátoru do výsledné množiny slabých klasifikátorů je provedena předběžná analýza, kolik bude klasifikátor v FPGA spotřebovávat zdrojů, a je-li překročen nastavený limit, provede se změna metodiky. Po změně metodiky se z dosavadních výsledků určí typy slabých klasifikátorů, jejichž implementace bude levná z hlediska počtu spotřebovaných zdrojů v FPGA (viz sekce 5.2.3). Tímto způsobem jsou poté favorizovány jen vybrané slabé klasifikátory a za minimální spotřebované zdroje je zvýšena účinnost výsledného klasifikátoru. Obvykle se jedná o přidání několika posledních klasifikátorů.
- *Ohodnocení klasifikátoru*: v tomto kroku se provede otestování klasifikátoru na testovací sadě dat. Tento krok určí, zda klasifikátor splňuje počáteční podmínky z hlediska přesnosti klasifikátoru.
- *Sestavení klasifikátoru pro FPGA* je založeno na procesu automatického generování klasifikátoru pomocí postupu představeného v podkapitole 5.2.6. Vybrané části tohoto algoritmu (jako je například generování *LUT* do *BlockRAM*) byly přeneseny již do předchozí části iteračního procesu, a to do části trénování klasifikátoru. Díky tomu do této sekce vstupuje již vhodně sestavený klasifikátor. Přenesením části algoritmu tak vzniklo v podstatě i propojení fáze generování s fází trénování, a byla tak odstraněna jedna z bariér konvenčního přístupu, o které je psáno v úvodu této podkapitoly, čímž tak bylo vytvořeno úzké propojení mezi oběma částmi.
- *Ohodnocení klasifikátoru (FPGA zdroje)*: se provádí za účelem zjištění reálných parametrů klasifikátoru pomocí nástroje Xilinx Vivado 2013.2 [93] (případně jinými obdobnými nástroji). Výstupem je poté počet spotřebovaných *Slice registrů*, *Slice LUT* a *BlockRAM* pamětí.
- *Analýza výsledků řešení*: je poslední částí iteračního procesu. V této části se provádí porovnání výsledků vytvořeného klasifikátoru (klasifikační přesnost, spotřeba zdrojů FPGA), a na jejich základě se určí, zda se iterační proces ukončí – to je v případě, že byly splněny vstupní požadavky nebo byl překročen maximální dovolený počet iterací, či zda se bude pokračovat další iterací. Při pokračování algoritmu další iterací můžou nastat dvě modelové situace. Natrénovaný klasifikátor je příliš velký – musí se provést redukce počtu slabých klasifikátorů. Nebo natrénovaný klasifikátor nesplňuje podmínky pro klasifikační přesnost a současně jeho velikost je menší, než dovolují uživatelem definovaná kritéria (musí být menší o takovou velikost, která zajistí, že je možné bezpečně implementovat další slabé

klasifikátory). V obou případech se provede na základě výsledků získaných v aktuální iteraci stanovení nového počtu předpokládaných klasifikátorů (dojde ke zpřesnění úvodního odhadu dle posledních výsledků trénování) a spustí se další kolo iterace.

Proces sestavení výsledného klasifikátoru probíhá v několika iteracích, přičemž s každou iterací se přibližuje k požadovanému výsledku. Experimentálními testy bylo zjištěno, že při správném nastavení vstupních parametrů může být počet iterací omezen na 10. Ve většině případů je řešení nalezeno do 10 iterací. Za správné nastavení algoritmu se považuje nastavení dosažitelné úspěšnosti a také použití rozumně velkého FPGA. Iterační proces návrhu klasifikátoru je však pomalý a jeho běh trvá v závislosti na velikosti trénovací množiny a velikosti požadovaného klasifikátoru až několik hodin. Jedná se však o strojový čas, přičemž se tento proces děje jen jednou při návrhu klasifikátoru, a tak je delší doba jeho běhu přijatelná. Iterační metoda pracuje s AdaBoost klasifikátorem a slabými klasifikátory založenými na LBP operátorech.

Využití iterační metody pro optimalizaci velikosti klasifikátoru má vliv především pro klasifikátory s malým počtem slabých klasifikátorů (méně než *100*), u kterých má i přidání několika málo (pěti) slabých klasifikátorů významný vliv na zvýšení klasifikační přesnosti. Takové klasifikátory se velmi často využívají jako malé, ale rychlé předzpracovávající jednotky. Významné využití však metoda najde i u velkého klasifikátoru, jehož popis je uveden v následující kapitole 6. V jeho případě bude dosažená úspora zdrojů ještě znásobena vzhledem k replikaci AC jednotek.

5.5.2 Dosažená úspora FPGA zdrojů

Maximální počet uspořených zdrojů daných touto metodou se dá stanovit dle následujících pravidel. Předpokládejme, že máme natrénován silný klasifikátor, který využívá čtyři základní druhy LBP operátorů (základ slabého klasifikátorů). V nejhorším případě je využití *BlockRAM* pamětí následující. Pro LBP příznaky typu *1-4* je vždy použita polovina *BlockRAM* paměti, což znamená, že je možné do jedné *BlockRAM* uložit vyhledávací tabulky čtyř slabých klasifikátorů. V každé je však instanciována jen jedna vyhledávací tabulka a pro každý ze tří uvedených typů zůstávají tři volné pozice v *BlockRAM* pamětech. Celkový počet zabraných *BlockRAM* pamětí uvedenými typy je dva. Poslední pátý typ klasifikátoru je lichý a zbyla pro něj tedy celá BlockRAM. Díky tomu je možno zde instaciovat až *9* vyhledávacích tabulek, což znamená, že je *8* volných pozic. Sečteme-li volné pozice pro všechny typy LBP příznaků, dostaneme, že je možné za velmi malé náklady implementovat až *20* dalších slabých klasifikátorů (*4·3 + 8*). Dodatečné náklady na implementaci takových slabých klasifikátorů budou jen na *Slice LUT* a *Slice registry*, kterých je však většinou dostatek a žádné další drahé *BlockRAM* paměti nebudou dále využity. Odhad nároků na *Slice LUT* pro přidané klasifikátory je cca *200* (*20·10,* viz sekce 5.4.2).

5.6 Pre-processingová klasifikační jednotka

Implementace kompletního klasifikátoru s několika sty až několika tisíci slabými klasifikátory je na FPGA velmi drahá z pohledu spotřebovaných zdrojů a pro řadu aplikací nemusí být ani opodstatněná. Pro vybrané aplikace postačuje pouze vytvoření jednotky, která provádí například jen předklasifikaci, tedy označení význačných objektů v obraze nebo na vyšší úrovni předvýběr zájmových snímků. Takto předzpracovaný obraz by se poté přenášel a následně i zpracovával v univerzálních výpočetních jednotkách.

Ceny elektrické energie neustále rostou a v poslední době se stalo velkým trendem porovnávat výkonnosti jednotlivých systému vzhledem ke spotřebované energii na počet provedených operací.

Systém pro detekci objektů založený na standardních procesorech může spotřebovávat enormní množství energie. V této podkapitole je prezentován způsob jak vytvořit AdaBoost klasifikátor tak, aby byl rychlý a současně byl energeticky velmi efektivní. Uvedený přístup spočívá v modifikaci AdaBoost algoritmu tak, že se rozdělí na dvě části: *pre-processingová* část (předzpracovávací) a *post-processingová* část (část dokončující výsledek zpracování). Výsledky tohoto přístupu byly prezentovány v práci [39]. V podkapitole je uvedeno srovnání energetické náročnosti výpočtu AdaBoost klasifikátoru na standardním CPU a na FPGA.

5.6.1 Dekompozice a profilace klasifikátoru

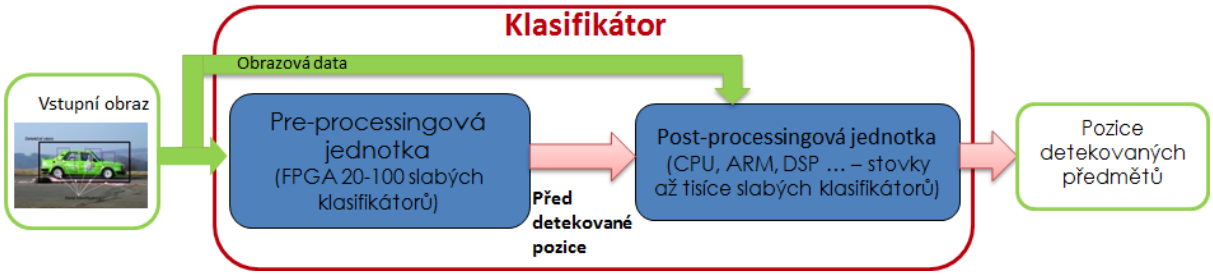
Současné metody, které se zabývají problematikou klasifikátorů obecně nebo přímo AdaBoost metodou, jsou zaměřeny především na vylepšení klasifikační přesnosti metody samotné nebo vylepšení principu trénování klasifikátoru. Tyto metody a přístupy jsou zajisté důležité a prospěšné, ale existují i jiné způsoby jak zvýšit rychlost klasifikátoru. Tato sekce přináší koncept pro urychlení zpracování klasifikátoru a také jak výrazně zredukovat nároky na energie pro klasifikátory založené na metodě AdaBoost.

Metody uvedené v kapitole 3 vykonávají AdaBoost klasifikátor na jedné výpočetní platformě. Velmi rozšířené je provádět výpočet na CPU, případně na CPU s podporou GPU [20] nebo na FPGA [47], [96]. Řešení uvedené v této kapitole je založeno na rozdělení algoritmu a následně na vykonávání algoritmu na dvou rozdílných výpočetních prostředcích. Uvedené řešení není založeno jen na pouhé akceleraci nejvíce používaných či nejpomalejších částí kódu, jako je to v případě GPU implementací, ale je založeno na téměř kompletní duplikaci výpočetních jednotek v každé z technologií. Proto *pre-processingová* jednotka a *post-processingová* jednotka mohou pracovat zcela samostatně.

V podkapitole 2.5.2 je uvedena rovnice 2.17 pro výpočet AdaBoost klasifikátoru. Klasifikátor lze v tomto případě chápat jako seřazený seznam slabých klasifikátorů. Vytvoření *pre-processingové* a *post-processingové* jednotky je založeno na tom, že *pre-processingová* jednotka implementuje jen první malou část ze seznamu slabých klasifikátorů (*20-100* slabých klasifikátorů) a zbývající část seznamu slabých klasifikátorů je zpracována v *post-processingové* jednotce (tato jednotka však může zpracovávat i celý klasifikátor, je-li to žádoucí).

Koncept rozdělení klasifikátoru je založen mimo jiné i na myšlence, že všechna detekční okna jsou zpracovávána v *pre-processingové* jednotce a *post-processingová* jednotka zpracovává jen malou část detekčních oken, které budou předvybrány právě pomocí *pre-processingové* jednotky. Poměr zpracovaných detekčních oken je okolo *100 000:1* a více [39]. Schéma zpracování vstupního obrazu je uvedeno na obrázku 41. Vstupní data jsou přivedena jak na vstup *pre-processingové* jednotky, tak i na vstup *post-processingové* jednotky. *Pre-processingová* jednotka provede velmi rychlé zpracování všech detekčních oken v obraze. Do *post-processingové* jednotky pak předá informace o oblastech, kde by se mohl nacházet hledaný objekt. *Post-Processingová* jednotka tak zpracovává pouze před vybrané oblasti obrazu. Aby bylo použití *pre-processingové* jednotky opodstatněné, tak musí mít přijatelnou velikost vzhledem ke zvolené technologii a musí provádět dostatečnou redukci datového toku (předvýběr). Pro významné snížení datového toku z *pre-processingové* jednotky je výhodné použít AdaBoost klasifikátor s *50 – 100* slabými klasifikátory [39].

Pro zpracování obrazu ve FullHD rozlišení je třeba zpracovat okolo deseti milionů detekčních oken v *pre-processingové* jednotce. *Post-processingová* jednotka pak zpracuje pouze desítky až stovky detekčních oken v závislosti na obsahu vstupního obrazu. Výstup celého systému je standardní a tvoří jej pozice a velikosti detekovaných objektů.



Obrázek 41: Blokové schéma zpracování vstupního obrazu pomocí dvou jednotek.

5.6.2 Dosažené výsledky

Jelikož je pro prezentovanou jednotku použit klasifikátor ze sekce 5.2, nebo případně klasifikátor z kapitoly 6, tak není třeba uvádět výsledky architektury z hlediska klasifikační přesnosti a také z hlediska spotřeby zdrojů v FPGA. Ty jsou uvedeny v příslušných kapitolách. Tato podkapitola je tak zaměřena na výsledky z hlediska spotřeby energie, jelikož uvedené dělení AdaBoostu na dvě výpočetní platformy přinese právě tuto výhodu.

Energetická náročnost výpočtu

Za účelem stanovení energetické efektivnosti a rychlosti výpočtu algoritmu na různých porovnávaných výpočetních platformách byly stanoveny nové metriky. První metrikou je počet zpracovaných detekčních oken na jeden watt spotřebované energie - DW/W, druhou metrikou je počet zpracovaných detekčních oken za jednu sekundu DW/s. Spotřeba samotného FPGA byla zjištěna pomocí nástroje Xilinx Power estimation tools (Pro Xilinx Virtex-II 500). Pro čip Xilinx Zynq byla spotřeba jednak určena nástrojem Xilinx Vivado (samostatný čip) a také reálně změřena pro celou vývojovou desku (AVNET ZedBoard) i se všemi podpůrnými obvody. Spotřeba CPU byla odhadnuta na základě řady měření spotřeby celého PC (osobní počítač) a na základě technických informací poskytnutých výrobcem procesoru. Použitý procesor byl (Intel i5 3570K s *4* jádry, AdaBoost algoritmus v průběhu měření běžel na všech 4 jádrech). V době měření byl použitý procesor z nejnovější série od Intelu a poskytoval vysoký výkon s relativně nízkou spotřebou energie (*55* W).

Tabulka 10 ukazuje srovnání rychlosti a energetické náročnosti klasifikačních systémů založených na CPU a FPGA. Ve všech experimentech jsou porovnávány výsledky pro stejnou aplikační úlohu. Porovnáme-li energetickou efektivitu výpočtu standardního CPU s FPGA, tak zjistíme, že na FPGA můžeme zpracovat přibližně $2 \cdot 10^4$krát více detekčních oken na jeden watt než na CPU. Srovnáme-li CPU s dosaženými výsledky Xilinx Zynq (bez podpůrných elektronických obvodů), tak poměr sice klesne, ale stále je FPGA schopno zpracovávat cca *2 207*krát více DW/W. Provedeme-li porovnání kompletního PC s kompletní vývojovou deskou Xilinx Zynq, tak zjistíme, že Xilinx Zynq je stále ještě *1 391*krát výhodnější. Přesné hodnoty porovnání uvedené v tabulce se mohou velmi jednoduše změnit, ať už v závislosti na použitém FPGA, CPU či technologii výroby FPGA nebo CPU, proto nemá smysl se zabývat přesnými čísly, a dále pak i přesnou konstrukcí vývojové desky. Na získané hodnoty se musíme podívat tak, že rozdíl mezi uvedenými architekturami je přibližně v řádu tisíců násobků, což je nejdůležitější ukazatel, který poskytuje tabulka.

Výsledky v tabulce 10 jsou získány pro AdaBoost klasifikátor, který se skládá ze *100* slabých klasifikátorů založených na LBP příznacích. AdaBoost klasifikátor však může obsahovat stovky až tisíce slabých klasifikátorů, ale pro objektivní porovnání byl tento klasifikátor použit na všech

výpočetních platformách. Pokud bychom v porovnání pro implementace na CPU použili klasifikátor založený na metodě WaldBoost, tak by dosažené výsledky DW/s a DW/W byly o cca 1 řád lepší. Avšak použití metody WaldBoost pro implementace na FPGA by vedlo ke snížení DW/W a DW/s (viz 5.1 a 6.2.3).

Výsledný systém pro zpracování AdaBoost klasifikátoru se může skládat například z FPGA (*pre-processingová* jednotka) a CPU (*post-processingová* jednotka) nebo z FPGA a ARM procesoru, jako nabízí například Xilinx Zynq. Budeme-li uvažovat systém skládající se z CPU ve spojení s FPGA, tak spotřeba celého PC se bude pohybovat okolo 50–60 W (klasifikátor poběží jen s velmi malým vytížením). Spotřeba celého systému pak bude cca 52–62 W. Výhoda takovéhoto uspořádání je, že ušetřená výpočetní kapacita CPU může být použita pro navázaný systém zpracovávající výsledky klasifikace (například dohledový systém) nebo může být využita ke zpracování výsledků klasifikace pro více *pre-processinogových* jednotek současně. Pomocí rozprostření výkonnosti mezi tyto dva uvedené výpočetní prostředky se zlepší ukazatel počtu zpracovaných detekčních oken za watt. Druhou možností jak vytvořit výslednou architekturu je využití ARM procesoru ve spojení s FPGA, které poskytuje například Xilinx Zynq. Toto spojení se jeví být velmi vhodným a bude poskytovat mnohem lepší výsledky z hlediska efektivity výpočtu než dříve uvedená kombinace. Parametry takového řešení jsou uvedeny v posledním řádku tabulky 10.

Poslední sloupec tabulky 10 uvádí rychlost zpracování na jednotlivých architekturách. Jednotky implementované v FPGA mohou zpracovávat mnohonásobně více detekčních oken než CPU. Pro implementaci na dnes již starém FPGA Virtex-II je to 41krát více, pro implementaci na Xilinx Zynq je to dokonce *82*krát více než v případě CPU. Na FPGA se tak může zpracovat mnoho násobně více dat.

Tabulka 10: Tabulka odhadnutých a změřených energetických nároků různých výpočetních platforem.

Výpočetní platforma	Spotřeba	DW/W	DW/s
CPU (pouze)	55 W (odhad)	61 032	3 356 786
Osobní počítač	104 W (změřeno)	32 276	3 356 786
FPGA Virtex-II 500 (pouze)	0,115 W (odhad)	1 352 015 732	137 500 000
Xilinx Zynq XC7020 (pouze)	2,041 W (odhad)	134 737 873	300 000 000
Xilinx Zynq XC7020(vývojová deska AVNET ZedBoard)	6.1 W (změřeno)	44 912 624	300 000 000

Využití v chytrých kamerách

Uvedený systém, ať už jako celek nebo případně jen *pre-processingová* část, nalezne uplatnění například v chytrých kamerách. Chytrá kamera vznikne spojením obrazového čipu s FPGA jednotkou, ve které je implementován například klasifikátor. Taková kamera může na svůj výstup produkovat snímaný obraz, ale současně může k datům poskytovat dodatečné informace, které budou říkat, co se vyskytuje v obraze a případně, kde se to vyskytuje. Jelikož ne všechny snímky z kamery je třeba zpracovávat komplexně v datových centrech, může chytrá kamera posílat ke zpracování jen zájmové snímky (viz podkapitola 6.2.6). Tím také významně klesne počet přenášených dat po síti, což je pro systémy se stovkami kamer velmi důležitý požadavek.

5.6.3 Hybridní klasifikační jednotka

Pro zlepšení dosahovaných výsledků je možné uvedenou *pre-processingovou* jednotku založenou na AdaBoost metodě kombinovat s *post-processingovou* jednotkou, která může být založena na zcela jiné klasifikační metodě. Vznikne tak hybridní klasifikační jednotka. Důvodem pro vznik takové jednotky je, že vyhodnocení AdaBoost na CPU nebo GPU je velmi výpočetně náročné, či případně AdaBoost nedosahuje požadované přesnosti.

V případě, že je třeba sestavit rychlou *post-processingovou* jednotku, je vhodné ji založit například na metodě WaldBoost, která je při zpracování na CPU či GPU o *1* až *2* řády rychlejší, jelikož nemusí zpracovávat celou klasifikační kaskádu. Dále pak může být použitím metody WaldBoost dosaženo i možného mírného zvýšení klasifikační přesnosti.

Je-li však cílem dosažení co nejlepší klasifikační přesnosti, může být jako *post-processingová* jednotka použit klasifikátor pracující na zcela odlišném principu. Příkladem takového jednotky může být například klasifikátor využívající konvoluční neuronovou síť (CNN) [98].

6 Klasifikátor

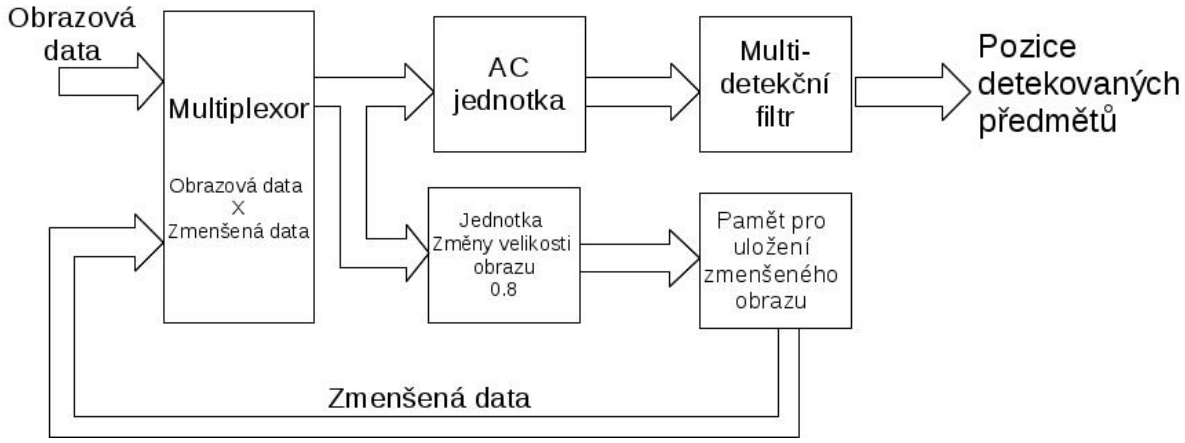
V kapitole 5 je popsáno jádro AdaBoost klasifikátoru – AC jednotka. Tato jednotka je navržena jen pro klasifikaci obrazu s jednou úrovní rozlišení a neuvažuje kompletní zpracování obrazu včetně změny rozlišení a ostatních operací nutných ke zpracování obrazu. Proto pro její reálné použití je třeba ji rozšířit o doplňkové jednotky, jako je například jednotka pro změnu rozlišení vstupního obrazu, řídící jednotka, jednotka pro shromažďování výsledků a také případně jednotka pro vykreslování výsledků do výstupního obrazu. Pro vytvoření takové nadstavby nad AC jednotkou existuje několik možných přístupů. V následujících kapitolách budou popsány dva hlavní přístupy. Zejména druhý uvedený způsob vyniká díky svým výhodným vlastnostem nad všechny doposud představené metody, a proto byl vybrán a následně experimentálně ověřen.

6.1 Sekvenčně paralelní klasifikátor

První základní způsob pro sestrojení kompletního klasifikátoru nad AC jednotkou je založen na sekvenčním přístupu ke zpracování obrazu. Zde se myslí sekvenční přístup z hlediska zpracování jednotlivých úrovní zmenšovaného vstupního obrazu. Pro zpracování vstupního obrazu je třeba provést několikanásobné zmenšení a zpracovat každý zmenšený stupeň. Na obrázku 42 je uvedeno zjednodušené schéma takového klasifikátoru. Základní funkční jednotkou tohoto klasifikátoru je AC jednotka. Vstupní data do klasifikátoru tvoří obrazová data, která jsou zpracovávána vstupní multiplexovací jednotkou. Tato jednotka v závislosti na tom, zda je zpracováván originální obraz nebo zmenšený vstupní obraz, přeposílá na svůj výstup právě vstupní obrazová data nebo data z pomocné paměti pro ukládání zmenšeného obrazu. Výstup multiplexovací jednotky je přiveden jednak ke zpracování AC jednotkou, která tvoří výsledky klasifikace, a dále také do jednotky pro změnu rozlišení obrazu, která pracuje paralelně a nezávisle na AC jednotce a provádí zmenšování obrazu se stanoveným faktorem - SF. Výstupní data z jednotky pro změnu velikosti obrazu jsou ukládána v paměti k tomu určené, která v tomto případě musí pojmout minimálně obrazová data pro první úroveň zmenšení obrazu. Pro FullHD obraz a zmenšení obrazu s poměrem *0,8* se jedná o uložení *1,32* megapixelů obrazových dat, což pro osmibitový obraz znamená potřebu datového úložiště o velikosti *1,32* MB. Pro *2.* až *N*-tou úroveň zpracování zmenšeného obrazu nejsou data čtena ze vstupního rozhraní, ale jsou multiplexovací jednotkou čtena právě z paměti pro ukládání zmenšeného obrazu. Data v paměti jsou v *2.* až *N-1* stupni zmenšování vstupního obrazu periodicky přepisována a objem ukládaných dat postupně klesá. V posledním kroku jsou uložena data, která jsou jen mírně obsáhlejší než je velikost detekčního okna. (Další operace zmenšení by vedla k obrazu s menší velikostí než je detekční okno.)

Výstupní data AC jednotky jsou tvořena souřadnicemi jednotlivých detekovaných objektů a také velikostmi detekovaných objektů. Velikost objektu je určena právě zpracovávaným stupněm zmenšeného obrazu. Jelikož detekční jednotka zpravidla pro jeden objekt v obraze produkuje pozitivní odezvu na více detekčních oknech, je třeba tyto několikanásobné odezvy filtrovat. O toto filtrování se stará právě multidetekční filtr. Základním typem filtrace může být například ponechání jen nejvýznamnější odezvy (maximum z N prvků). Taková operace však bude vhodná jen pro úzkou specifickou množinu aplikací, jako je například detekce RZ (registračních značek) vozidel v obraze u zařízení snímající jeden pruh vozovky. Pro řadu aplikací je však nutné produkovat na výstup veškeré detekce. Pro takové aplikace je poté nutné implementovat nad výstupními detekcemi například množinu morfologických filtrů, jako jsou erozní a dilatační filtry [15], pomocí kterých se provede

odfiltrování několikanásobných detekcí. Filtry pracují tak, že v prvním kroku se aplikuje dilatační filtr, který provede scelení oblastí s detekcemi (vygenerují více detekcí v zájmových oblastech) a následně se aplikuje několik erozních filtrů, které provedenou redukci detekovaných oken tak, aby pro každou oblast s více násobnou detekcí zůstala zachována jen právě jedna oblast. Erozních nebo dilatačních filtrů může být za sebou více, zpravidla však aplikace dvou nebo tří filtrů postačuje. Erozní a dilatační filtry jsou jednoduchými jednotkami, které pracují nad maticí výstupních dat (například *5 x 5*). Jejich implementace vyžaduje v podstatě jen uložení několika málo řádků obrazu s výsledky detekcí. (Detekce jsou v tomto případě ukládány jako kontinuální proud dat). Multidetekční filtr je aplikován nad každou úrovní zmenšeného obrazu samostatně. Po přefiltrování výsledků je tak na výstup poslána redukovaná množina pozic a velikostí zájmových detekčních oken.



Obrázek 42: Schéma sekvenčně-paralelního klasifikátoru.

Uvedený postup zpracování dat na obrázku 42 je velmi podobný způsobu, který se používá ke zpracování na CPU. Avšak tento uvedený způsob se odlišuje především tím, že zpracování jednoho detekčního okna je plně paralelizováno a trvá jeden hodinový cyklus (díky použití AC jednotky). Díky tomu je tato uvedená sekvenční jednotka velmi rychlá a dokáže zpracovávat až FullHD obraz s více než 30 snímky za sekundu v reálném čase (viz tabulka 11, poslední sloupec).

Architekturu této jednotky je možné optimalizovat vzhledem ke spotřebovaným výpočetním zdrojům stejným způsobem, který byl použit již v práci [95]. To spočívá v tom, že na FPGA je implementována jen AC jednotka a všechny ostatní operace jsou prováděny jinou výpočetní jednotkou. Takovou jednotkou může být například DSP (*Digital signal processor*), jako v případě [95], nebo v dnešní době se spíše bude jednat o CPU např. ARM, který je mimo jiné součástí jednotky Xilinx Zynq. Takové řešení však nepřinese zrychlení architektury vzhledem k počtu zpracovávaných snímků, spíše se dá očekávat zpomalení, a to vzhledem k rychlosti ARM procesoru a propustnosti rozhraní ARM a FPGA. Řešení je však zajímavé z hlediska akcelerace jen části výpočtu na FPGA.

Uvedený sekvenční způsob realizace je velmi podobný svou strukturou práce doposud známým architekturám uvedeným v podkapitole 3.3. Ač je tato architektura dle výpočtů rychlejší než dříve uvedená řešení, tak nebyla dále rozvíjena a další práce se zaměřily na vývoj mnohem propustnějšího a perspektivnějšího řešení. Vypočtená propustnost této architektury však bude sloužit jako reference pro porovnání výsledků nového přístupu.

6.2 Multiparalelní detektor objektů pracující v reálném čase - RT-MPOD

Detekce objektů hraje velmi důležitou roli v mnoha aplikacích zpracovávajících obraz a může významně ovlivnit efektivitu (spotřebu energií, spotřebované výpočetní prostředky) a výkonnost (latenci a propustnost) celého systému. Již bylo představeno mnoho různorodých architektur pro detekci objektů (viz kapitola 3) pracujících na různých principech a různých architekturách. Uvedené architektury však mají limitovanou rychlost zpracování obrazových dat na desítky megapixelů za sekundu (Mpps). Jsou tak schopny zpracovávat obraz s relativně malým rozlišením, ale vysokým počtem snímků (například *640 x 480* pixelů a 160 snímků za sekundu [96]), nebo obraz s vysokým rozlišením obrazu, ale nízkým počtem snímků za sekundu (*1920 x 1200* pixelů a *4* snímky za sekundu – odhad pro [96]). Ovšem dnešní obrazové senzory jsou schopny poskytovat obraz s velmi vysokým rozlišením (i více než *10* Mpx) a s velmi vysokým počtem snímků za sekundu (i více než *100* snímků, v realitě se však velmi často používá rychlost okolo *25 – 30* snímků, která zajišťuje plynulé přehrávání scény pro lidské oko). Uvedený tok obrazových dat vyžaduje rychlost zpracování v řádu stovek megapixelů za sekundu (*300* Mpps), což je však téměř o řád výše, než poskytují dosavadní FPGA implementace (*50* Mpps). Zpracování obrazu se stovkami Mpps vyžaduje velmi výpočetně rychlé řešení, které doposud nebylo realizováno. Nejvíce se mu prozatím blíží autor Chouchene [32] se svou implementací na GPU a CPU, jež dokáže zpracovat až *100* Mpps. Tato implementace je však z hlediska spotřeby energií mnohonásobně náročnější než právě FPGA technologie, a tak ji nelze považovat za přímého konkurenta. Na FPGA není známo řešení, které by bylo schopno zpracovat více než *100* Mpps v jedné instanci (viz sekce 3.3). Jelikož je detekce objektů zpravidla pouze jednou z prvotních operací v algoritmech pracujících s obrazem, tak její přenesení do specializované jednotky může mít velmi pozitivní vliv na celou vyhodnocovací architekturu. A to nejen z hlediska rychlosti zpracování, ale také z dnes velmi důležitého energetického hlediska, kdy se zpracování jednoho snímku v rámci ceny energie může snížit i v řádu tisíců, jak již bylo ukázáno v podkapitole 5.6.2.

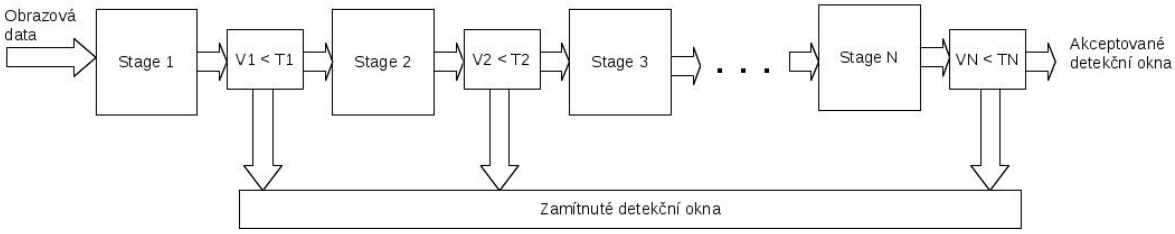
Potřeba velké datové propustnosti klasifikátoru přímo vybízí k použití řešení s FPGA nebo ASIC technologií. Současně se také ukazuje, že je výhodné použít koncept dělení klasifikátoru na dvě nebo i více částí, jak bylo uvedeno v podkapitole 5.6. Nově navržená architektura tyto dvě důležité věci nemůže přehlížet, a je proto navržena v souladu s oběma principy.

Dříve představené architektury pro zpracování AdaBoost, a z nich odvozených klasifikátorů na FPGA, jsou většinou založeny na různých vylepšeních sekvenčního přístupu a z jejich podstaty se dá stanovit, že jsou stále především sekvenční. Například jedna z nejrychlejších implementací, která byla představena v práci [96], je založena na mikroprogramovatelné jednotce, ale principiálně stále zachovává sekvenční zpracování AdaBoostu. Největší výhoda FPGA, kterou je možnost masivního paralelizmu, je tak využita jen částečně. Tato kapitola přestavuje zcela novou architekturu, která je založena na kompletním odstranění sekvenčního přístupu ze zpracování obrazu metodou AdaBoost. A to především tím, že přestavená architektura využívá několika stupňů paralelizace během zpracování obrazových dat. Nově uvedená architektura také povoluje rozdělení algoritmu mezi FPGA a CPU a přináší tak možnost výrazné úspory energií v porovnání se systémy založenými jen na CPU, případně na CPU a FPGA. Architektura dále přináší možnost opravdového zpracování obrazu s vysokým rozlišením a vysokým počtem snímků v reálném čase, čemuž výrazně dopomáhá proudový charakter zpracování dat.

6.2.1 AC jednotka - vylepšení

Architektura uvedená v této kapitole je založena na využití dříve představené vyhodnocovací jednotce AdaBoost jádra – AC jednotka (viz sekce 5.2). Tato jednotka sice detekuje pouze objekty s fixní velikostí, ale největší výhoda AC jednotky je v její velmi vysoké rychlosti zpracování s také velmi efektivní implementaci v FPGA, a to jak z hlediska spotřebovaných zdrojů, tak i z hlediska maximálních dosahovaných pracovních frekvencích. AC jednotka je založena na myšlence vyhodnocení všech slabých klasifikátorů, ze kterých se skládá výsledný AdaBoost klasifikátor, paralelně a dále také na proudovém charakteru práce této jednotky, která nepotřebuje vkládat žádné čekací stavy do procesu zpracování obrazových dat.

AC jednotka popsaná v podkapitole 5.2 představuje obecný klasifikátor a může být použita ke klasifikaci předmětů jakéhokoliv typu. Z implementačního hlediska má však AC jednotka několik omezení. Jedno z omezení vzniká v *ASUM* jednotce, která počítá výslednou hodnotu AdaBoost klasifikátoru ze všech zpožděných výsledků slabých klasifikátorů. *ASUM* jednotka se tak může stát úzkým hrdlem architektury při implementaci klasifikátorů obsahujících tisíce a více slabých klasifikátorů. Může tak být ovlivněna výpočetní výkonnost architektury díky tomu, že je prakticky nemožné v jeden hodinový cyklus provést sečtení několika (až desítek) výsledků slabých klasifikátorů. Pro provedení takového součtu by musela být implementována kaskáda sčítaček, ta by však způsobila výrazné zvýšení zpoždění, a v důsledku zvýšení zpoždění by také poklesla maximální dosahovaná pracovní frekvence pro FPGA. Jedním z možných řešení, jak překonat tento problém, je zavedení zřetězeného zpracování (*pipeliningu*) do *ASUM* jednotky. Dalším možným řešením je rozdělení AdaBoost klasifikátoru do oddělených částí (*stage*) [82], jak je ukázáno na obrázku 43.



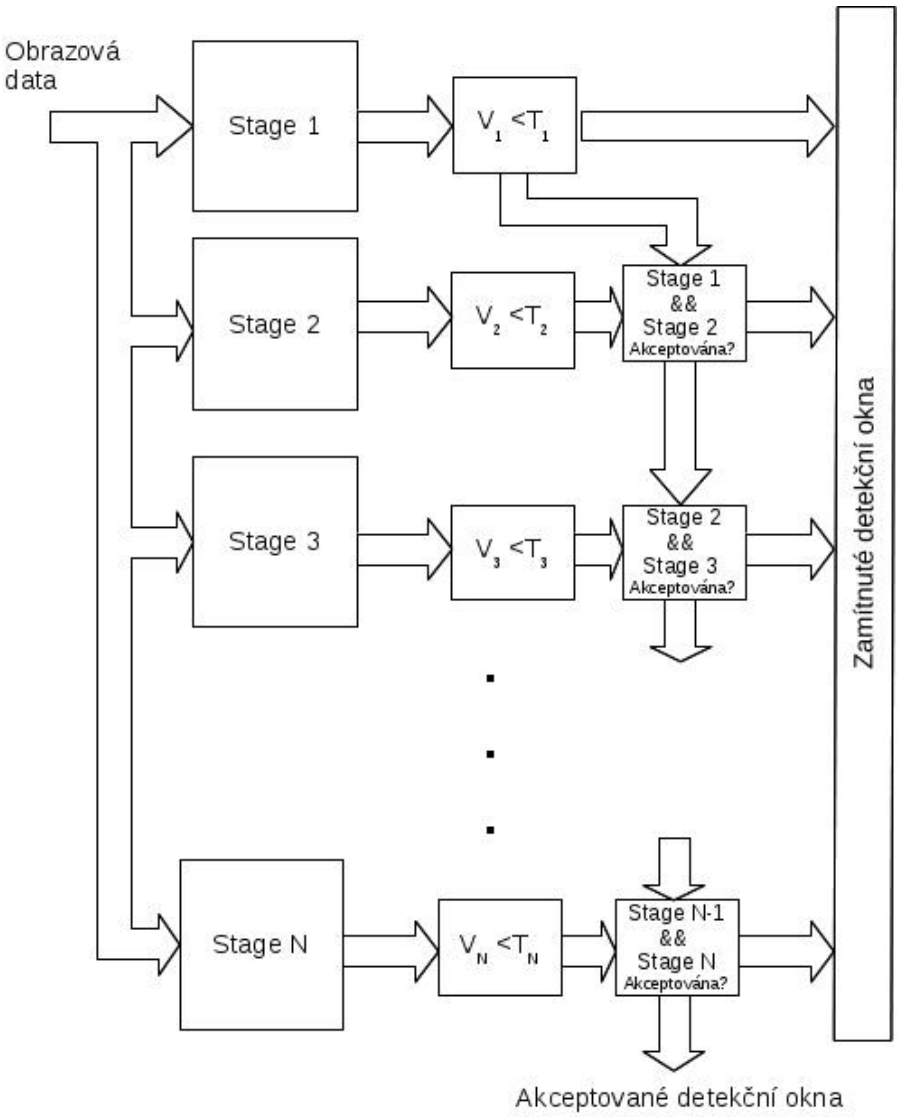
Obrázek 43: Rozdělení AdaBoost klasifikátoru do několika částí [82].

Každá *stage* zobrazená na obrázku 43 reprezentuje ve své podstatě jeden AdaBoost klasifikátor. Jednotlivé *stage* (*State N*) se od sebe liší především v počtu obsažených slabých klasifikátorů a také každá *stage* během vyhodnocování klasifikátoru pracuje s jinými individuálně získanými daty z procesu trénování. Výsledek každé *Stage N* je částečně závislý na výsledku všech předchozích *stages* (*Stage 1* až *Stage N-1*), takže výsledek žádné *Stage N* pro $N > 1$ nemůže být použit samostatně. Klasifikační prahy T_N jednotlivých *stage* jsou určeny v procesu trénování tak, aby první *stage* prováděla filtrování co největšího počtu negativních detekčních oken (detekční okna, které neobsahují hledaný předmět), a naopak *stage*, které jsou na konci kaskády, akceptují pouze detekční okna, jež obsahují hledané objekty.

Zpracování obrazu pomocí uvedeného schématu klasifikátoru je následující. Obrazová data jsou jako první vyhodnocena pomocí *Stage 1,* a pokud je výsledek porovnání kladný (je získán porovnávací operací v jednotce $V_N < T_N$), tak jsou data předána dále do vyhodnocení pomocí jednotky *Stage 2*. Pokud je výsledek ze *Stage 1* negativní, tak je další vyhodnocení ukončeno a aktuální detekční okno je označeno za negativní (neobsahuje hledaný objekt). Zpracování dat v dalších navázaných *stage* probíhá na obdobném principu. Popsaný způsob vyhodnocení je přínosný především pro sekvenční implementace klasifikátoru, kde je většina detekčních oken zpracována pouze pomocí *Stage 1,*

a zpracování dalších *stage*, které však mohou tvořit i více než *95 %* všech slabých klasifikátorů, je vynechána, protože *Stage 1* označí většinu DW negativně. Celkový počet *stage* je proměnný a závisí především na natrénovaném klasifikátoru a také na způsobu trénování. Detailní popis je uveden například v [82] nebo v [71].

Rozdělení AdaBoost klasifikátoru do samostatných částí (*stages*) může být jednoduše použito pro vytvoření klasifikátoru popsaného v podkapitole 5.2. Takové rozdělení dovolí efektivně odstranit limitaci architektury v *ASUM* jednotce uvedenou výše a vytvořit tak klasifikátor, který bude složen z tisíců slabých klasifikátorů, ale nyní však ve všech *stages* dohromady. Na obrázku 43 je ukázán původní sekvenční proces ohodnocení AdaBoost klasifikátoru, kde zpracování *Stage N* musí předcházet zpracování všech *Stage 1* až *Stage N-1*. Pro FPGA implementaci však takový sekvenční přístup nevyužije plně potenciál FPGA technologie. Za účelem využití potenciálu paralelnosti FPGA byly provedeny optimalizace v procesu vyhodnocení klasifikátoru, jak je uvedeno na obrázku 44. Uvedené optimalizace vycházejí z toho, že vyhodnocení každé ze *stage* může probíhat samostatně a až následné operace porovnání musí probíhat ve správném pořadí.



Obrázek 44: Paralelní vyhodnocení AdaBoost *stages*.

Každá *stage* na obrázku 44 je tvořena samostatnou AC jednotkou a všechny *stage* jsou vyhodnocovány paralelně na rozdíl od vyhodnocení v případě uvedeném na obrázku 43. Výsledky všech *stage* jsou tak vzhledem k principu AC jednotky získány ve stejný čas, a to nezávisle na počtu slabých klasifikátorů, které obsahují. Ovšem sekvenční přístup vyhodnocení výsledků jednotlivých *stage* musí zůstat zachován tak, aby se získal validní výsledek klasifikátoru. Proto pokud je výsledek detekčního okna zamítnut ve *Stage 1*, tak všechny ostatní výsledky *Stage 2 – Stage N* jsou zahozeny, i když jsou pozitivní, a celé detekční okno je označeno negativně. Sekvenční způsob vyhodnocení detekčního okna tak do celkového procesu vyhodnocení vkládá další latenci, ale celková propustnost architektury zůstává nezměněna. Propustnost je stále jedno detekční okno v každém hodinovém cyklu. Výstup každé $V_N<T_N$ jednotky je buď logická pravda (*DW* je přijato) nebo nepravda (DW je zmítnuto). Pro zachování sekvenčního principu při vyhodnocení AdaBoost metody je nutné přidat pouze jednu jednoduchou jednotku, která provádí operaci logický *AND* nad výsledky všech $V_N<T_N$ jednotek. Následně pokud je výsledek *AND* jednotky logická pravda, tak je detekční okno přijato. Tímto způsobem je možné vyhodnotit všechny *Stage* (*AC* jednotky) plně paralelně.

6.2.2 Popis kompletního klasifikátoru

AC jednotka sama o sobě není schopna samostatně provádět kompletní zpracování vstupního obrazu. Aby toho byla schopná, je třeba ji rozšířit o několik dalších jednotek, jak již bylo napsáno v podkapitole 6.1. Detekční okno AdaBoost klasifikátoru má konstantní velikost, avšak velikosti detekovaných předmětů v obraze se zpravidla liší. Proto je třeba v procesu detekce provádět několikanásobné zmenšení vstupního obrazu a zpracování každého zmenšeného obrazu pomocí AC jednotky. Každá oblast vstupního obrazu je proto zpracovávána několikrát, ale pokaždé s jinou velikostí (úrovní zmenšení).

Zpracování vstupního obrazu je tedy prováděno v několika iteracích, jak je ostatně ukázáno i na obrázku 45. V první iteraci je zpracováván obraz v rozlišení, které je dáno obrazovým senzorem, a současně je provedeno zmenšení těchto obrazových dat pro další iteraci. V dalších iteracích se tento proces neustále opakuje, dokud není dosaženo potřebného zmenšení obrazu. Zmenšováním obrazu se umožňuje detekce velkých předmětů v obraze. Počet stupňů zmenšení je závislý na velikosti detekčního okna, velikosti vstupního obrazu a zmenšovacím poměru (SF). Například pokud uvážíme zpracování obrazu ve FullHD rozlišení, což je *1920 x 1200* pixelů, velikost detekčního okna *24 x 24* pixelů a SF = *0,8* (tyto parametry se používají například v úlohách pro detekci obličejů), tak bude nutné provést zmenšení vstupního obrazu celkem *17*krát. Propustnost jednotky uvedené v podkapitole 6.1 musí být minimálně čtyřikrát vyšší (závisí na počtu zpracovávaných detekčních oken a počtu stupňů zmenšení obrazu), než je rychlost poskytování dat obrazovým senzorem. Stejně tak architektura z podkapitoly 6.1 by měla být vybavena vyrovnávací pamětí pro ukládání vstupních obrazových dat, které mohou do jednotky proudit v době, kdy zpracovává zmenšené snímky a není schopna zpracovávat data ze svého vstupu.

Zpracování dat v architektuře uvedené na obrázku 42 probíhá následovně. V první iteraci je vstupní obraz zmenšen s definovaným SF. Uvážíme-li SF = *0,8,* tak je vstupní FullHD obraz v tomto kroku zmenšen na rozlišení *1536 x 960* pixelů. Výstup jednotky pro změnu rozlišení je uložen v paměti, odkud jsou data pomocí multiplexoru vyčítána pro další iterace. Paměť je tak plněna v krocích *1* až *N-1*.

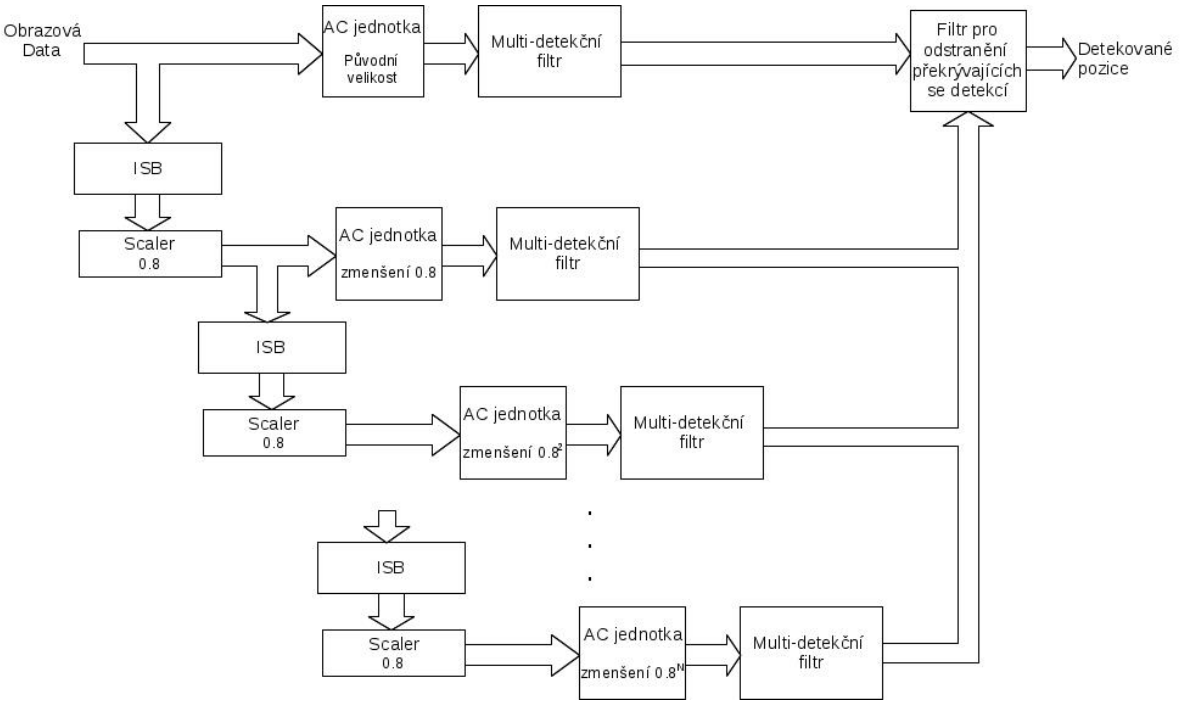
Architektura klasifikátoru popsaná v podkapitole 6.1 má výhody především v relativně nízkých nárocích na spotřebované hardwarové zdroje v FPGA nebo ASIC a také se vyznačuje velkou mírou znovuvyužití jednotek při zpracování jednoho obrazu. Každá jednotka je použita opakovaně během procesu vyhodnocení obrazu. Ovšem její hlavní nevýhoda tkví v limitované rychlosti zpracování

a poměrně malé možnosti rozšiřitelnosti vzhledem k rychlosti zpracování dat. Další nevýhodou je, že pokud bude pracovat na své maximální rychlosti, tak musí uložit téměř dva kompletní vstupní snímky. Jednotka tak potřebuje pro uložení dat 3,7 MB paměti, která by měla být velmi rychle přístupná a měla by se tak vyskytovat na FPGA nebo s ním být úzce spjata. Využití externí paměti k FPGA je možné, avšak poté se musí počítat s možnou sníženou propustností architektury, která může být zapříčiněna pomalým přístupem do externí paměti nebo její velkou latencí.

AC jednotka představená v podkapitole 5.2 je velmi kompaktní a konzumuje malé množství výpočetních zdrojů pro svoji implementaci. Tím tak otevírá možnost sestavit novou multiparalelní architekturu, která využívá vysokého stupně paralelizmu. Hlavní princip spočívá ve zpracování všech úrovní zmenšovaného obrazu v jednom čase a plně paralelně. V architektuře jsou navrženy dva hlavní stupně paralelizace. První stupeň je dán paralelním zpracováním všech úrovní zmenšených snímků najednou. Druhý stupeň paralelizace je pak dán zpracováním všech slabých klasifikátorů najednou pomocí AC jednotky.

Vstupní data do architektury jsou stejná jako v případě sekvenčního klasifikátoru uvedeného v podkapitole 6.1 – proud pixelů z obrazového senzoru. Nyní však mohou být obrazová data zpracovávána s každým hodinovým cyklem a není nutné vkládat mezi vstupní data žádné čekací cykly – data tak mohou být kontinuálně zpracovávána. Na obrázku 45 je vyobrazena nová architektura nazývaná Multiparalelní objektový detektor pracující v reálném čase (RT-MPOD). Postup zpracování dat v architektuře je následující. Obrazová data jsou přiváděna jednak na vstup AC jednotky na první úrovni zpracování (zpracování obrazu s originálním rozlišením obrazu) a současně jsou přiváděna do malé paměti (*ISB*) jednotky pro změnu rozlišení (*Scaler*), která provádí zmenšení obrazu pro následující, druhý stupeň zpracování obrazu. Výstup jednotky pro změnu rozlišení je přímo připojen na vstup následujícího stupně zpracování obrazu, který je opět tvořen samostatnou AC jednotkou, nyní však již pracující na zmenšeném vstupním obraze, a další jednotkou pro zmenšení obrazu. Tento (druhý) stupeň zpracování provádí vyhledávání o něco větších objektů v obraze, než provádí předchozí stupeň. Jelikož AC jednotka ve druhém stupni zpracovává obraz s nižším rozlišením, její implementace je mírně odlišná od implementace klasifikátoru v předchozím stupni. Implementace se liší především v délce *FIFO* linek v *ASUM* jednotce, kde pro klasifikátor pracující na nižším rozlišení již není nutné vytvářet tak dlouhé linky pro zpožděné sčítání výsledků slabých klasifikátorů, jako je tomu v případě prvního stupně zpracování. Výstup *Scaler* jednotky je připojen na vstup následující AC jednotky a také další *Scaler* jednotky. Tímto způsobem je obraz postupně zmenšován, až je dosaženo minimálního rozlišení obrazu, které je dáno velikostí detekčního okna (nebo případně maximální velikostí detekovaného předmětu v obraze). Typ použité metody pro změnu rozlišení záleží především na typu zpracovávané aplikace. V této práci bylo použito interpolace metodou nejbližšího souseda, jelikož tato metoda má žádané vlastnosti, jako je například zachování ostrých hran v obraze (cílová úloha je detekce RZ v obraze).

AC jednotka na prvním stupni zpracování vyhodnocuje největší množství dat (vstupní obraz v původním rozlišení). AC jednotka na druhém stupni zpracování může běžet na stejné rychlosti zpracování jako klasifikátor v první úrovni, avšak data dodávaná do této úrovně mají menší datový tok, jelikož se zpracovává obraz s menším rozlišením. Do druhého stupně zpracování je dodáváno o *20 %* méně dat než do jednotky v první úrovni. Pokles dat je dán poměrem zmenšení obrazu SF = *0,8*. Klasifikátor na druhém až *N*-tém stupni tak musí čekat na data a do jejich zpracování musí být vkládány zpožďovací cykly. Množství dat pro třetí a další úrovně zpracování postupně klesá, a klasifikátory proto mají menší vytížení. Toho lze využít a klasifikátory v posledních úrovních mohou běžet s nižší frekvencí, což bude mít za následek pokles spotřeby klasifikátoru.



Obrázek 45: Multi-Parallel architektura AdaBoostu

Výsledky následujících stupňů zpracování jsou získány vždy s jistým zpožděním od předchozího stupně zpracování. Proto musí být na výstupu všech jednotek zaveden jednoduchý synchronizační algoritmus. Každý stupeň zpracování přidává malou latenci (zvýší čas, kdy je obdržen finální výsledek), která je dána především distribucí dat přes *Scaler* jednotky (každá jednotka způsobí zpoždění dat o *2* hodinové cykly $T_{ScaleDelay}$). Každý stupeň zpracování pak způsobí zpoždění $T_{LevelDelay}$ (zpoždění AC jednotky – *5* hodinových cyklů + zpoždění *Scaler* jednotky - *2* hodinové cykly). Jestliže pak poslední pixel obrazu začne být zpracováván v čase T_0, tak poté je pixel kompletně zpracován posledním stupněm architektury v čase $T_{Final} = T_0 + T_{LevelDelay} + (N\text{-}1) \cdot T_{ScaleDelay}$. Výsledky všech zpracovávajících úrovní jsou shromažďovány a filtrovány v jednotkách pro odstranění vícenásobných detekcí a poté také v jednotce pro odstranění překrývajících se detekcí (viz podkapitola 6.1).

Aby bylo možné vytvořit *RT-MPOD,* jak je zobrazena na obrázku 45, tak musela být vytvořena řada specializovaných jednotek, které umožňují použití požadované úrovně paralelizmu. Hlavní výhodou uvedeného řešení je zpracování obrazu v reálném čase. Propustnost architektury je jeden pixel obrazových dat za jeden hodinový cyklus. Zde je však nutno podotknout, že jsou uvažovány pouze pixely originálního obrazu. V architektuře je zpracováváno několik detekčních oken ze všech úrovní zpracování v jednom hodinovém cyklu. Další výhodou je čistě proudový charakter uvedené architektury. Každý pixel obrazu je zpracován a následně je zahozen. Architektura neprovádí žádné ukládání vstupního obrazu tak, jako to velmi často musí provádět architektury založené na sekvenčním přístupu.

6.2.3 Výsledky

Klasifikační přesnost

Navržená architektura nemá žádný další vliv na klasifikační přesnost, než jaký je uveden v podkapitole 5.4.1.

Propustnost architektury

Nejdůležitější hledisko během návrhu nové architektury AdaBoost klasifikátoru bylo dosažení co největší datové propustnosti, jak je jen možné. V tabulce 11 je zobrazen počet detekčních oken, které je nutné zpracovat pro obraz s definovaným rozlišením. První sloupec tabulky definuje rozlišení obrazu v pixelech (šířka x výška obrazu). Druhý sloupec definuje požadovaný počet snímků za sekundu (*FPS*). Ve třetím sloupci je uveden počet detekčních oken, které je nutné zpracovat pro vyhodnocení obrazu pouze v originálním rozlišení (první stupeň zpracování v RT-MPOD). Ve čtvrtém sloupci je uveden počet všech detekčních oken (*DW*), která musí být zpracována pro vyhodnocení vstupního snímku ve všech úrovních zmenšení. Hodnoty ve třetím sloupci přímo korespondují s počtem operací, které je nutné provést v RT-MPOD (čtvrtý sloupec je pak obdobou pro sekvenční architekturu) pro zpracování obrazu. V pátém sloupci je dána minimální frekvence, na které musí běžet RT-MPOD klasifikátor, aby byl schopen zpracovat datový tok s požadovaným rozlišením a požadovaným počtem snímků. Šestý sloupec tabulky zobrazuje pro porovnání frekvenci, na které musí běžet sekvenční architektura tak, aby byla schopna vyhodnotit stejný datový tok. Vztah mezi požadovanou datovou propustností a odpovídající minimální pracovní frekvencí f RT-MPOD architektury může být vyjádřen pomocí následující rovnice:

$$f = FPS \cdot IMGWidth \cdot IMGHeight \tag{6.1}$$

Kde *FPS* je požadovaný počet snímků za sekundu, *IMGWidth* a *IMGHeight* jsou šířka a výška vstupního obrazu.

Poznamenejme však, že hodnoty frekvencí uvedené v šestém sloupci jsou získány pro AdaBoost klasifikátor, který využívá velmi rychlé AC jednotky, jež je použita i v RT-MPOD. Jiné architektury [96] však používají jiných přístupů, které nejsou zdaleka tak rychlé a pro dosažení uvedené propustnosti by musely pracovat na daleko vyšších frekvencích. Porovnáme-li například řešení navržené autory Zemčík a kolektiv [96] (nejrychlejší známé řešení), tak jejich architektura dokáže zpracovat *164* snímků za sekundu při rozlišení obrazu *640 x 480* pixelů a pracovní frekvenci *152* MHz. Sekvenční architektura uvedená ve sloupci šest dokáže při frekvenci *152* MHz zpracovat *180* snímků za sekundu při rozlišení *640 x 480* pixelů, z čehož vyplývá, že její rychlost je na velmi dobré úrovni v porovnání se Zemčíkem [96].

Tabulka 11: Závislost pracovní frekvence klasifikátoru na rozlišení obrazu a počtu snímků za sekundu pro RT-MPOD a sekvenční architekturu.

Rozlišení vstupního obrazu	FPS	DW ve snímku pouze pro původní rozlišení obrazu RT-MPOD	DW ve snímku sekvenční architektura	Frekvence RT-MPOD [MHz]	Frekvence sekvenční architektura [MHz]
640 x 480	25	307200	847968	7,7	21,2
640 x 480	50	307200	847968	15,4	42,4
640 x 480	100	307200	847968	30,8	84,8
1024 x 768	25	786432	2176914	19,7	54,5
1024 x 768	50	786432	2176914	39,4	108,9
1024 x 768	100	786432	2176914	78,7	217,7
1920 x 1200	25	2304000	6388045	57,6	159,8
1920 x 1200	50	2304000	6388045	115,2	319,5
1920 x 1200	100	2304000	6388045	230,4	638,9
3000 x 2000	25	6000000	16655567	150	416,4
3000 x 2000	50	6000000	16655567	300	832,8
3000 x 2000	100	6000000	16655567	600	1665,6

Z tabulky 11 lze odvodit, že výkonově nejnáročnější částí procesu vyhodnocení obrazu je právě vyhodnocení obrazu v původním plném rozlišení. Vyhodnocení tohoto snímku spotřebuje přibližně jednu třetinu celkového výpočetního výkonu potřebného pro vyhodnocení vstupního obrazu ve všech úrovních zmenšení obrazu. A naopak nejmenší snímky zase spotřebují nepatrné množství výpočetního výkonu. Vyhodnocení první třetiny zmenšených snímků (u FullHD obrazu se jedná o šest stupňů zmenšení) spotřebuje přibližně *90 %* celkového výpočetního výkonu, který je potřebný pro vyhodnocení všech úrovní zmenšení obrazu. Minimální pracovní frekvence pro vyhodnocení obrazu je pro RT-MPOD cca *3*krát menší než v případě sekvenční architektury, avšak RT-MPOD spotřebovává *18*krát více výpočetních jednotek. Uvedený rozdíl je dán především distribucí výpočetního výkonu, jak je uvedeno výše. Takové rozložení distribuce výkonu mezi jednotlivé stupně zmenšování obrazu otevírá možnost provést další optimalizace ve zpracování zmenšených snímků, a to buď pomocí sekvenční architektury, nebo pomocí jiného výpočetního prostředku, jako je například CPU nebo ARM.

Výpočetně nejpomalejším prvkem v celé architektuře je AC jednotka. Maximální pracovní frekvence pro danou jednotku byla syntézním nástrojem stanovena na 302,5 MHz pro FPGA Xilinx Zynq XC7Z045 [91] (*Place and Route*). S touto architekturou je tak možné zpracovávat vstupní obraz s rozlišením až *6* megapixelů a 50 snímky za sekundu. Poznamenejme, že parametry obrazu uvedeného v posledním řádku tabulky 11 jsou za hranicí možností RT-MPOD architektury a současných FPGA, jelikož frekvence potřebná pro zpracování je *600* MHz. Zpracování takového množství obrazových dat (*300* Mpps) je však i na dnešních CPU (Intel Core i5-3570K čtyři jádra, pracovní frekvence *4,1* GHz [35]) daleko za hranicí jejich limitů, jelikož rychlost zpracování AdaBoostu na uvedeném CPU je přibližně *3,3* Mpps [39]. Rychlost zpracování však závisí na mnoha kritériích, jako je například délka klasifikační kaskády a podobně. Poznamenejme však, že pokud by se při zpracování na CPU nahradil AdaBoost za WaldBoost, tak by díky vlastnostem WaldBoost vzrostla propustnost zpracování, ale ani tak by to nebylo dostatečné (při použití dříve uvedeného CPU). Stejně tak i zpracování na CPU ve spojení s GPU, které je výrazně rychlejší než jen samotné CPU, není proveditelné, jak bylo ukázáno v [32]. Při práci s takto velkým datovým tokem však vznikají i jiné problémy, jako je například jen samotný přenos obrazových dat z kamery do zpracovávající jednotky. Přenos uvedeného datového toku již nemůže být prováděn např. pomocí v průmyslu široce rozšířeného a levného 1 Gbit ethernetu, jelikož datový tok uvedeného obrazu je 2,4 Gbit/s. Pro přenos by se musela použít linka s vyšší kapacitou (10 Gbit/s) nebo provádět ztrátovou kompresi dat, která je však pro řadu aplikací nevhodná.

Tabulka 12 ukazuje, že architektura RT-MPOD dokáže zpracovat mnohem větší množství dat, než ostatní doposud představené architektury. Nevýhodou RT-MPOD je však počet spotřebovaných zdrojů FPGA, a tedy především velká spotřeba *BlockRAM* pamětí. Architektury srovnávané v tabulce 12 jsou implementovány většinou na starších FPGA a jejich implementace na dnešních FPGA není známa. Tento fakt může vést k obtížnému srovnání výsledků. Hlavní rozdíl, který se práce snaží prezentovat, však spočívá především v úrovni paralelizmu RT-MPOD. Implementace architektur uvedených v tabulce 12 na dnešních moderních FPGA by zřejmě přinesla zvýšení pracovní frekvence jednotlivých architektur a následně i zvýšení datové propustnosti jednotlivých implementací. Implementace RT-MPOD na starších FPGA není možná, jelikož neposkytují dostatečné množství FPGA zdrojů. Aby však bylo možné provést více objektivní porovnání, byla provedena experimentální implementace nejpomalejší části RT-MPOD pro FPGA Xilinx Spartan 6 LX150T (FPGA stejné rodiny, jako používá nejrychlejší známé řešení v práci [96]). Dosažená pracovní frekvence *AC* jednotky pro toto FPGA je *160,5* MHz (*Place and Route*). Tato frekvence je o trochu větší než v případě práce [96]. Pokud by byla celá RT-MPOD implementována na FPGA Spartan 6, tak by

rychlost zpracování byla 519 snímků za sekundu při rozlišení *640 x 480* pixelů. Propustnost architektury je možné stanovit pouze na základě dosažené pracovní frekvence a parametrů vstupního obrazu, jak ukazuje vzorec (6.1) a tabulka 12. Uvedené řešení je tak stále významně rychlejší než jiná známá řešení. Výsledky architektury [96] pro novější FPGA jako je například Xilinx Zynq doposud nejsou známy, proto nemůže být provedeno objektivní srovnání datových propustností.

Tabulka 12: Porovnání klasifikační rychlosti RT-MPOD se známými architekturami. FPS jsou získány pro vstupní obraz s rozlišením *640 x 480* pixelů. RT-MPOD je konfigurována s 500, případně 128 slabými klasifikátory.

	FPS	**SF**	**Frek. [MHz]**	**BRAMs**	**LUTs**	**FPGA**
Zemčík [95]	22	None	-	14	2980	Virtex II 250
Kim [44]	50	-	106	18	133000	Virtex5 LX330T
Kyrkou [47]	40	0.67	100	24	25800	Virtex2 XC2VP30
Lai [48]	143	0.75	126	44	20900	Virtex2 XC2VP30
Zemčík [96]	164	0.8	152	31	7373	Spartan6 LX45T
RT-MPOD (500 LBP)	**974**	**0.8**	**300**	**425**	**95200**	**Zynq XC7Z045**
RT-MPOD (100 LBP)	**974**	**0.8**	**300**	**131,5**	**30850**	**Zynq XC7Z045**

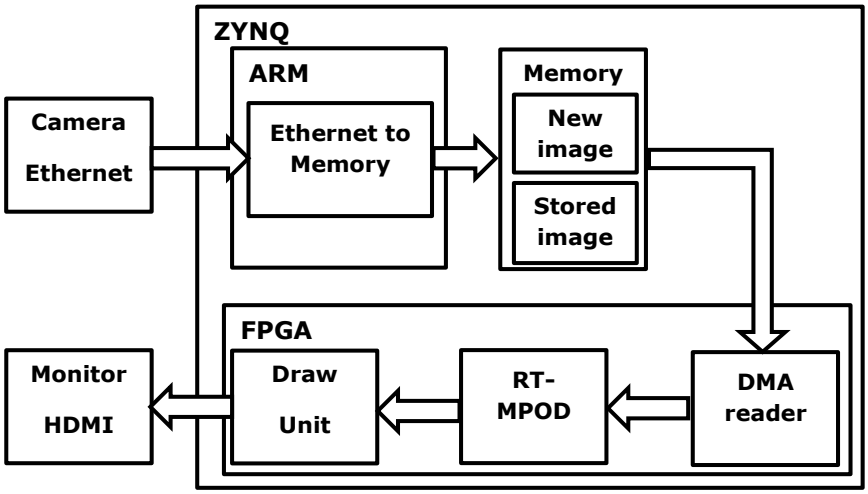
Nejvýznamnější rozdíl mezi RT-MPOD a řešením uvedeným v [96] je však ve způsobu zpracovávání obrazového toku. RT-MPOD využívá několika stupňového paralelizmu tak, aby se zpracovalo jedno detekční okno za jeden hodinový cyklus. RT-MPOD se nikdy nevrací k již jednou zpracovanému pixelu. Na druhou stranu řešení [96] je založeno na vysoce optimalizovaném mikroprogramovatelném stroji, který je však ze své podstaty sekvenční. To znamená, že řešení [96] zpracovává slabé klasifikátory postupně, a ne všechny najednou, jako je tomu u RT-MPOD. Navíc jejich architektura není založena na AdaBoost, ale na WaldBoost, který je právě pro sekvenční zpracování mnohem vhodnější, jelikož má WaldBoost proměnnou délku zpracování [71] detekčního okna, která výrazně zkracuje sekvenční vyhodnocení. Na řešení [96] lze pohlížet jako na specializovaný jednoúčelový obrazový procesor, který pro zajištění vysoké rychlosti provádí spekulativní rozpracování několika instrukcí (slabých klasifikátorů) za účelem zavedení jakéhosi paralelizmu. Vzhledem k vlastnostem WaldBoost není čas zpracování jednoho vstupního snímku konstantní, jako je tomu v případě RT-MPOD. Tato vlastnost sice při sekvenčním zpracování výrazně zvyšuje rychlost a snižuje počet provedených výpočetních operací, ale při zpracování obrazu s více objekty zájmu se tak u implementace [96] může prodloužit doba zpracování, a při práci na maximální rychlosti je tak třeba vytvářet vyrovnávací paměti pro příchozí snímky. U implementace [96] tudíž není možné analyticky stanovit datovou propustnost.

Testovací klasifikátor

Za účelem prokázání výhod uvedeného řešení před sekvenčním přístupem a stejně tak, aby se ověřily vlastnosti navržené multiparalelní architektury, byla jednotka úspěšně implementována na Xilinx Zynq platformě. Ačkoliv cílem této práce je představit architekturu, která dokáže zpracovat obrazová data s vysokým rozlišením (*1920 x 1200* pixelů) a vysokou propustností (více než 100 Mpps), pro účely porovnání s existujícími řešeními (viz tabulka 12) a ověření vlastností byly vytvořeny dvě testovací implementace, z nichž první byla konstruována jako předzpracovávající jednotka s omezeným počtem slabých klasifikátorů tak, aby ji bylo možné umístit do FPGA Xilinx Zynq XC7Z020. Druhá implementace tvoří kompletní klasifikátor, který zpracovává vstupní obraz o rozlišení *640 x 480* pixelů. Ve druhém případě RT-MPOD využívá klasifikátor složený ze 100 slabých klasifikátorů založených na LBP příznacích, a je tak velmi blízký klasifikátoru použitém v práci [96], který využívá

128 slabých klasifikátorů. Poměr pro postupné zmenšování obrazu byl pro obě implementace nastaven na *SF = 0,8* a použitá velikost detekčního okna je *60 x 15* pixelů. Tyto parametry vedou k RT-MPOD, která má celkem *9* stupňů (*9 AC* jednotek a zpracovává *9* úrovní zmenšeného obrazu). *AC* jednotka na prvním stupni zpracovává obraz v originálním rozlišení. *AC* jednotka na druhém stupni zpracovává obraz s rozlišením *512 x 384* pixelům a tak dále. A poslední AC jednotka zpracovává obraz o rozlišení *107 x 81* pixelů. Jako klasifikační úloha byla v obou případech zvolena detekce registračních značek vozidel.

Na obrázku 46 je uvedeno celkové schéma architektury, které bylo použito pro oba případy. Vstupní data do architektury jsou generována pomocí kamery připojené přes ethernetové rozhraní. Toto rozhraní bylo zvoleno, jelikož je velmi často vyžíváno v reálných aplikacích a je možné pomocí něj snadno připojit kamery mnoha výrobců. Snímek obrazu přijatý z kamery je ukládán v externí paměti RAM na použité platformě pomocí ARM procesoru. K této paměti má současně přístup ARM i FPGA. Tato obrazová data jsou pak následně z RAM do FPGA vyčítána pomocí jednotky přímého přístupu do paměti (DMA – *Direct Memory Acces*) a zasílána na vstup RT-MPOD jednotky. Jelikož použitá kamera je mnohem pomalejší než RT-MPOD, tak jeden dodaný snímek může být čten z paměti několikrát. Obrazová data jsou ze vstupu RT-MPOD přenášena do AC jednotky a současně také do jednotky pro ukládání úzkého proužku obrazu (první část jednotky pro změnu velikosti obrazu) v první úrovni zpracování obrazu. Obrazová data jsou postupně zmenšována a zpracovávána pomocí AC jednotek ve všech úrovních. Výsledkem RT-MPOD jsou nalezené pozice a velikosti detekovaných předmětů. Tyto objekty jsou vizualizovány pomocí vykreslovací jednotky a zobrazovány na monitoru. Vzhledem k úspoře zdrojů na FPGA (především pamětí) bylo vykreslování detekovaných předmětů prováděno vždy do následujícího snímku. Příklad aplikace byl prezentován v práci [28].



Obrázek 46: Testovací RT-MPOD klasifikátor na platformě Xilinx Zynq.

První experiment byl proveden na vývojové desce Xilinx SOC ZC702 [92], která je osazena jen velmi malým FPGA Zynq XC7Z020. Řešená úloha detekce RZ v obraze je specifická tím, že velikost SPZ v obraze pro účely dopravních aplikací, jako je například dokumentování přestupků u měření rychlosti a podobně, má omezenou velikost, a tak se nemusí provádět zpracování ve všech úrovních zmenšeného obrazu. Tento fakt také dopomohl k tomu, aby bylo možné klasifikátor umístit do vybraného FPGA. Pro úsporu zdrojů byla délka klasifikační kaskády 50 slabých klasifikátorů, což postačuje pro detekci SPZ ve funkci předzpracovávající jednotky. Na implementaci tohoto řešení

spolupracoval Jiří Husák[4]. Tímto experimentem byla ověřena funkčnost architektury a také dříve uvedené parametry pro zpracování obrazu (viz příloha B). Druhým experimentem, jehož výsledky syntézy jsou uvedeny v posledním řádku tabulky 12, bylo ověření implementace kompletního klasifikátoru. K tomuto účelu bylo vybráno FPGA Xilinx Zynq XC7Z045. Maximální dosažitelná pracovní frekvence pro toto řešení je *300 MHz*, což vede na rychlost zpracování 974 snímků za sekundu pro obraz o vstupním rozlišení *640 x 480* bodů.

6.2.4 Nároky na FPGA zdroje

Ačkoliv je AC jednotka konstruována tak, aby měla co nejmenší spotřebu zdrojů, tak implementace kompletního RT-MPOD klasifikátoru s tisíci slabými klasifikátory, jak je uvedeno na obrázku 45, spotřebovává velké množství FPGA zdrojů. Tato velká spotřeba je dána především použitím samostatné AC jednotky pro každý stupeň zpracování zmenšeného obrazu. Ovšem pokud v FPGA nebudeme implementovat kompletní klasifikátor, ale budeme chtít implementovat jen *pre-processingovou* nebo případně filtrační jednotku, tak spotřeba FPGA zdrojů může být výrazně snížena. Pro účely implementování *pre-processingové* jednotky je dostatečný počet slabých klasifikátorů *50* až *100*, aby bylo dosaženo významného snížení výstupního datového toku z jednotky. Klasifikační přesnost takového řešení je stále na velmi dobré úrovni a je více než *90%* v mnoha typech úloh, jak bylo ukázáno v práci [26].

Tabulka 13 ukazuje porovnání dvou případů implementace RT-MPOD klasifikátorů. V prvním případě (druhý řádek tabulky) je implementována architektura, která obsahuje AdaBoost klasifikátor s *1 000* slabými klasifikátory využívající LBP příznaky. Klasifikátor zpracovává vstupní obraz v rozlišení FullHD se *100* snímky za sekundu. AC jednotka v prvním stupni architektury spotřebovává *46* KB (datové buffery) + *256* KB (data slabých klasifikátorů) paměti a přibližně *12 000* S*lices LUT*. Pro AC jednotku na následujícím stupni spotřeba prostředků mírně klesne, jelikož se zpracovává obraz s menším rozlišením. Celkový klasifikátor však potřebuje pro svoji implementaci celkem *4,9* MB paměti a cca *220 000* S*lice LUT*. Jak je ukázáno v tabulce 13, tyto požadavky jsou za hranicí možností dnes dostupných FPGA, avšak s příchodem nových FPGA Xilinx UltraScale+ [90] nebo Altera Stratix 10 [2] již implementace bude možná, jelikož tyto čipy obsahují dostatek prostředků.

Tabulka 13: Závislost spotřeby FPGA zdrojů na počtu slabých klasifikátorů – RT-MPOD klasifikátoru pro vstupní obrazový s FullHD rozlišením.

Počet slabých klasifikátorů	Slice LUTs	Paměť	BRAM	Xilinx Zynq 7020		Xilinx Zynq 7045		Xilinx Zynq 7100	
				LUT	BRAM	LUT	BRAM	LUT	BRAM
100	54000	771 KB	193	101,5 %	151,4 %	24,7 %	38,9 %	19,4 %	28,0 %
1000	220000	4.9 MB	1348	413,5 %	962,5 %	100,6 %	247,2 %	79,3 %	178,4 %

První řádek tabulky 13 ukazuje odlišný případ klasifikátoru s menší spotřebou výpočetních zdrojů. V tomto případě není architektura využita k implementaci kompletního klasifikátoru, ale implementuje pouze malý klasifikátor, který obsahuje *100* slabých klasifikátorů (opět připomeňme pro srovnání, že v díle [96] byl použit klasifikátor se *128* slabými klasifikátory). Použití takovéto jednotky je primárně jako *pre-processingová* jednotka, která vykonává předzpracování (filtraci) obrazu, a finální výsledek klasifikace je pak získán v navázané jednotce, která však zpracovává jen zlomek dat. Xilinx Zynq [91] nebo podobné čipy, které obsahují jednak programovatelnou logiku ale také univerzální procesor jsou velmi vhodným kandidátem pro implementaci takového řešení. ARM procesor v Zynq

[4] FIT VUT v Brně

poskytuje dostatečný výkon pro dokončení klasifikace ve smyslu *post-processingové* jednotky. Proto může být celý systém implementován na jednom čipu. První stupeň uvedené *pre-processingové* jednotky spotřebovává cca *46 + 26* KB paměti a *2 900* S*lice LUT*. Kompletní klasifikátor pak spotřebovává přibližně *771* KB paměti a přibližně *54 000* S*lices LUT*. Jak ukazuje tabulka 13, tak takovou jednotku je možné implementovat například s použitím FPGA Xilinx Zynq XC7Z100 nebo XC7Z045.

RT-MPOD architektura má velkou redundanci ve smyslu použitých jednotek. Kompletní klasifikátor pro zpracování FullHD obrazu se SF = *0,8* obsahuje *18* stupňů (*17* + originální) zpracování. A pro každou úroveň zpracování (zmenšený obraz) je instanciována samostatná AC jednotka. Každá však pracuje na odlišných vstupních datech. Největší potenciál, jak odstranit replikaci jednotek a tím i snížit spotřebu zdrojů, je v *LUT* tabulkách slabých klasifikátorů (pozor nejedná se o *Slice LUT* v FPGA). Tyto tabulky obsahují data z trénování a jsou použity jako paměti, ze které se pouze čte. Navíc data všech *LUT* na všech úrovních zpracování jsou stejná, ale prozatím je technologicky nerealizovatelné uložit data *LUT* tabulek pro všechny AC jednotky jen jedenkrát, jelikož v každém hodinovém cyklu může každá AC jednotka přistupovat k těmto datům. A každá AC jednotka však může přistupovat na jinou adresu v paměti. V uvedené architektuře tak k datům přistupuje v jeden čas až *18* jednotek (při zpracování FullHD obrazu). Pro takovou implementaci by byla třeba paměť, která může vykonávat *18* samostatných adresovacích a čtecích operací současně v jednom hodinovém cyklu. Realizace takové paměti na FPGA však není známa.

6.2.5 Dekompozice klasifikátoru

V podkapitole 5.6 byl uveden princip dělení AC jednotky na dvě a více částí. Tento koncept dělení zůstal zcela zachován i pro RT-MPOD jednotku a je možné jej nadále využít. Jak je ukázáno v tabulce 8, spotřeba zdrojů klasifikátorem je nejvíce závislá na počtu slabých klasifikátorů. Vhodným přizpůsobením jejich počtu je možné efektivně řídit množství spotřebovaných zdrojů, avšak za cenu snížení klasifikační přesnosti. Vlastnosti RT-MPOD tak mohou být účinně přizpůsobovány vzhledem k požadované aplikaci (propustnost, spotřeba zdrojů).

6.2.6 Aplikace klasifikátoru a budoucí vývoj

Existuje mnoho aplikací, které potřebují detekovat objekty s pevnou velikostí v obraze (například s šířkou *100–150* pixelů). Příkladem takové aplikace je detekce RZ (registrační značka) v obraze, která se používá v dopravních aplikacích. Použitím RT-MPOD je možné implementovat velmi rychlý, ale překvapivě i velmi malý a energeticky nenáročný klasifikátor, který bude detekovat RZ určené velikosti, což znamená, že bude zpracovávat jen *2–3* úrovně zmenšeného obrazu. Na základě údajů z tabulky 8 lze odvodit, že takový klasifikátor bude možné implementovat i ve velmi malém FPGA jako je Xilinx Zynq XC7Z020. Takový klasifikátor byl rovněž použit pro ověření implementace v podkapitole 6.2.3.

Navrženou architekturu je vzhledem k její velké rychlosti také možné použít pro zpracování obrazu z několika kamer. Taková jednotka však musí být rozšířena o externí paměť pro ukládání příchozích snímků z jednotlivých kamer a multiplexor pro přepínání jednotlivých zdrojů obrazových dat. Stejně tak bude potřebovat na svém výstupu jednotky pro třídění výstupních dat. Použitím toho přístupu je však možné za pomocí jedné jednotky zpracovat obraz až z pěti kamer s *FullHD* rozlišením a *25* snímky za sekundu.

Propustnost RT-MPOD je velmi vysoká v porovnání s jinými známými řešeními, jak je ukázáno v tabulce 12. Avšak i zde ještě existují možnosti jak dále zvýšit rychlost zpracování. Jednou z možností

je, že se nebude zpracovávat každé detekční okno, ale každé *N*-té detekční okno. Pokud by se zavedlo takovéto zpracování, pak by byla rychlost architektury po několika úpravách ještě *N*-krát navýšena, ale spotřeba zdrojů by zůstala téměř zachována. Vzhledem k principu zpracování AdaBoost metodě, pokud bude *N* rozumně malé, tak by nebyla ani zásadně ovlivněna přesnost metody a zůstala by téměř na stejné úrovni. Ostatně tuto techniku používají především sekvenční implementace, jako je například [16].

Přímým propojením FPGA a kamery může vzniknout takzvaná chytrá kamera, která bude přímo provádět zpracování obrazu. Výstupem takové kamery je poté i informace o tom, co se v obraze vyskytuje. V takovém případě může být RT-MPOD jednotka použita jako filtrační nástroj ve smyslu speciálního obrazového kodéru. Komprese obrazu bude v tomto případě založena na přenosu pouze zájmových snímků, které obsahují hledaný objekt. Takový typ komprese je vhodný především pro aplikace, pro něž je nepřijatelné použití ztrátové komprese a potřebují mít originální data z obrazového senzoru. Kompresní poměr takového řešení může být v závislosti na aplikační úloze mnohem lepší, než jakého se dosahuje použitím velmi vyspělých, avšak univerzálních kodeků H.264 [85] a H.265 [34].

Značnou nevýhodou RT-MPOD oproti jiným řešením uvedeným v podkapitole 3.3 jsou velké požadavky na FPGA výpočetní zdroje. Ty je však možné snížit za předpokladu, že by se architektura upravila tak, aby se část zpracování prováděla sekvenčně. Na obrázku 45 je uvedeno schéma RT-MPOD, ze kterého je vidět replikace AC jednotek na *N* stupních (zpracování zmenšenin vstupního obrazu). S každým stupněm zpracování však klesá objem zpracovávaných dat a proto AC jednotky v *2.* až *N*-té stupni nejsou plně vytíženy a do zpracování vkládají čekací cykly. Na základě tohoto poznatku je možné architekturu upravit tak, že stupně *M* až *N*, kde $1 < M < N$, budou nahrazeny jednou sekvenční jednotkou, která zpracuje všechny příslušné zmenšeniny vstupního obrazu. *M* musí být zvoleno tak, aby sekvenční jednotka zpracovala všechny odpovídající úrovně zmenšeného obrazu dříve, než dorazí data následujícího snímku. *M* je vhodné zvolit v cca jedné třetině počtu stupňů zpracování, jelikož zpracování cca první třetiny zmenšených obrazů spotřebuje cca *90 %* všech výpočetních operací. Při vhodné volbě *M* zůstane propustnost nezměněna. Případná sekvenční jednotka může být založena buď na AC jednotce, případně je možné použít i jednotky založené na odlišném způsobu vyhodnocení (například na metodě WaldBoost). Tímto přístupem je možné snížit spotřebu zdrojů o cca *45 %*.

6.2.7 Sestavení klasifikátoru

Architektura RT-MPOD je poměrně komplexní a obsahuje velké množství vyhodnocovacích jednotek, které mají však unikátní strukturu pro každou implementaci klasifikátoru. Implementace AC jednotky je plně automatizována pomocí automatického syntézního nástroje, který je uveden v podkapitole 5.2.6. Ten je pak dále vylepšen iterační technikou uvedenou v podkapitole 5.5. Tyto nástroje je možné rozšířit tak, aby automaticky generovaly celou architekturu pro RT-MPOD klasifikátor. Vstupem do takového generátoru pak bude jen množina vstupních dat, cílové FPGA a další parametry (velikost detekčního okna, přesnost a podobně) a výstupem bude jednak klasifikátor v jazyce pro popis hardware, ale také přímo konfigurační soubor pro FPGA. Tímto řešením tak bude možné velmi rychle a operativně vytvářet nové implementace klasifikátorů.

6.2.8 Shrnutí výsledků

RT-MPOD architektura dovoluje zpracovávat obraz s velmi vysokým datovým tokem (až *300* Mpps). Od dříve představených architektur uvedených v podkapitole 3.3 se liší tím, že již od samotného

začátku je koncipována pro zpracování obrazů s velkým datovým tokem. Jednou z pozitivních vlastností RT-MPOD je proto její malá citlivost na změnu rozlišení. S rostoucím rozlišením obrazu klesá propustnost architektury pomaleji, než je tomu u sekvenčních implementací. Těm propustnost klesá rychleji, jelikož s narůstajícím rozlišením zpracovávaného obrazu je nutno zpracovávat více úrovní zmenšeného obrazu, a tím pádem i mnohem více detekčních oken. RT-MPOD má lineární závislost propustnosti na rozlišení vstupního obrazu. Propustnost je závislá jen na počtu pixelů vstupního obrazu a nepromítá se do ní počet úrovní zpracování zmenšeného obrazu jako v případě sekvenčních architektur, které musí každý zmenšený obraz vyhodnotit zvlášť a každá úroveň zmenšení tedy dále prodlouží dobu jejich zpracování. Proto propustnost RT-MPOD s rostoucím rozlišením klesá s cca 3krát menší směrnicí (strmostí) přímky než u sekvenčních metod. Autoři prací uvedených v podkapitole 3.3 pracují jen s obrazem o rozlišení *640 x 480* pixelů, dá se předpokládat na základě tabulky 11, že bude rychlost těchto řešení s narůstajícím rozlišením obrazu výrazně klesat a zároveň porostou nároky na implementaci v FPGA.

RT-MPOD vyniká svou velmi vysokou datovou propustností a také proudovým charakterem zpracovávání dat. Tato architektura potřebuje jen jeden hodinový cyklus pro zpracování jednoho pixelu vstupního obrazu a není třeba se nikdy vracet k již zpracovanému pixelu nebo jej ukládat pro další zpracování. Proto není potřeba v architektuře vytvářet vyrovnávací paměti pro ukládání příchozích snímků nebo vkládání zpožďovacích cyklů do vstupních dat. RT-MPOD (implementované v Xilinx ZYNQ XC7Z045) může zpracovat obraz s až *300* Mpps, což je skoro o řád výše, než dokážou jiná známá řešení z tabulky 12 v jedné instanci (jejich paralelizované varianty nebyly představeny). Tato rychlost je dostatečná například pro zpracování obrazu o rozlišení *3000 x 2000* pixelů s *50* snímky za sekundu. Hlavní přínos metody je však především v představeném vícestupňovém paralelním zpracování dat. Tedy zpracování všech úrovní zmenšených obrázků najednou (první stupeň paralelizmu) a také zpracování všech slabých klasifikátorů paralelně (druhý stupeň paralelizmu). Architektura je navržena tak, aby mohla být velmi jednoduše rozdělena pro vytvoření hybridních zpracovávajících jednotek (to je využití například FPGA a CPU pro výpočet jedné úlohy).

7 Závěr

Detekce objektů je velmi důležitý úkon při zpracování obrazu. O jejím významu svědčí velký počet prací, které se stejně jako tato práce zabývají danou problematikou. Tato operace je velmi náročná na výpočetní výkon, a jelikož se považuje jen za iniciální operaci při zpracování obrazu, klade se velký důraz na její efektivitu při zpracování tak, aby výkon CPU zůstal dostupný pro další navázané aplikace. Řada prací se zabývá optimalizací klasifikačních algoritmů na CPU. Tato práce se však zaměřuje na zpracování této úlohy na jiných výpočetních platformách, či případně pomocí předzpracovávajících jednotek.

První část práce je zaměřena na obecný popis algoritmů pro detekci objektů. V této části byly pro řešení problematiky vybrány následující kandidátní algoritmy AdaBoost [7], AdaBoost (Viola a Johnes) [82] a WaldBoost [71]. Poté byla vybrána metoda AdaBoost vzhledem k jejím dobrým klasifikačním vlastnostem a také vzhledem k možnosti ji velmi efektivně implementovat v FPGA, jak bylo následně prokázáno v této práci. AdaBoost klasifikátor v této práci pracuje především ve spojení se slabými klasifikátory založenými na LBP, LRP a LRD obrazových operátorech.

Ve třetí kapitole je popsána a diskutována řada implementací AdaBoost algoritmu na různých výpočetních platformách. V této kapitole je detekován problém rychlosti zpracování dat pomocí doposud představených implementací a architektur. Na základě tohoto zjištění byla navržena zcela nová architektura, která odstraňuje sekvenční zpracování z procesu AdaBoost klasifikátoru.

Ve čtvrté kapitole je představena technika, pomocí níž je možné navrhovat nové tvary LBP operátorů. Ty jsou nově navrhovány pomocí genetického algoritmu na míru konkrétní aplikační úlohy. Navržená technika je založena na hypotéze, že pro detekci různých typů objektů jsou vhodné různé tvary obrazových operátorů, které je mohou vhodně popisovat. Genetický algoritmus tak na základě vstupních dat navrhne vhodný tvar nového příznaku. Pomocí experimentů bylo ukázáno, že přesnost klasifikátoru může být zvýšena o *3* až *4* procenta. Vzhledem k již tak vysoké přesnosti klasifikátorů je tento výsledek velmi dobrým a lze jej využít hned několika způsoby. Hlavní využití je především u klasifikátorů obsahujících jen malý počet slabých klasifikátorů, u kterých je velmi důležitá přesnost klasifikace každého ze slabých klasifikátorů. Použití lepšího slabého klasifikátoru vede k tomu, že je možné pro dosažení stejné klasifikační přesnosti použít menšího počtu slabých klasifikátorů, a následně tak například zkrátit čas vyhodnocení nebo ušetřit zdroje potřebné pro implementaci klasifikátoru, což je velmi žádané při implementaci klasifikátoru v FPGA. Výsledek genetického algoritmu je nejvíce ovlivněn správnou volbou fitness funkce, proto byly v rámci této práce provedeny experimenty s řadou fitness funkcí. Implementované fitness funkce se dělí do dvou kategorií. Fitness funkce zaměřené na klasifikační přesnosti a fitness funkce zaměřené na implementaci v FPGA. Výsledná fitness funkce však kombinuje obě tyto hlediska za účelem dosažení co nejvhodnějších výsledků při implementaci v FPGA. Nové tvary příznaků navržené genetickým algoritmem jsou využívány pro sestavení nově navrženého klasifikátoru v této práci.

Zbylá část práce se zabývá problematikou návrhu nové architektury, neboli přesněji metodiky pro automatizovaný návrh aplikačně specifických architektur klasifikátorů. Jako první je představeno jádro AdaBoost klasifikátoru, které zcela odstraňuje sekvenční charakter zpracování z vyhodnocení AdaBoost klasifikátoru. Odstranění sekvenčního zpracování je dosaženo za pomoci neuspořádaného vyhodnocení slabých klasifikátorů. Ty tak mohou být vyhodnoceny paralelně a jejich výsledky jsou až následně přeuspořádány dle principu AdaBoost. V podkapitole 5.2 je uveden popis jednotek navrhované architektury a je diskutována latence architektury, která dosahuje jen několika málo (*5-9*)

hodinových cyklů. Avšak rychlost klasifikátoru je na hranici možností a jednotka dokáže zpracovávat jeden pixel obrazu za jeden hodinový cyklus. Jednou z hlavních předností architektury, na které byl vystavěn kompletní klasifikátor, je proudový charakter zpracování dat. To znamená, že v architektuře se není třeba nikdy vracet k již jednou zpracovaným pixelům. Klasifikační přesnost uvedené jednotky není téměř nijak ovlivněna. (Pouhý zanedbatelný negativní vliv má použití aritmetiky s pevnou desetinnou čárkou.) Navržené jádro klasifikátoru (AC) má velmi kompaktní rozměry a pro jeho implementaci v FPGA je potřeba jen velmi málo FPGA zdrojů.

Pro návrh AC jednotky byl sestaven iterační algoritmus, který v několika iteracích nalezne nejvhodnější klasifikátor vzhledem k zadaným vstupním požadavkům. Iterační algoritmus je významný tím, že má na svém vstupu data pro trénování a popis parametrů výsledného klasifikátoru (rozlišení obrazu, přesnost klasifikátoru, požadavky na výsledné FPGA a tak dále). A jeho výstupem je již implementovaný design pro FPGA. Algoritmus tak zcela sám provádí proces trénování klasifikátoru, nalezení vhodných tvarů příznaků pomocí genetického algoritmu, sestavení klasifikátoru do FPGA, ohodnocení klasifikátoru dle klasifikační přesnosti ale také i dle velikosti v FPGA. Navržený algoritmus najde a vytvoří vhodný klasifikátor během několika málo iterací (obvykle méně než 10).

V šesté kapitole práce je využito všech doposud prezentovaných dílčích výsledků k vytvoření výsledného AdaBoost klasifikátoru. Ten je navržen s ohledem na maximální rychlost práce, jelikož obrazové senzory posledních generací produkují výstupní obraz ve vysokém rozlišení (až desítky Mpx) při velmi vysokém počtu snímků za sekundu (desítky až stovky), a doposud představené architektury téměř nejsou schopny zpracovávat takový obraz v reálném čase. Architektury uvedené v tabulce 6 jsou schopny v reálném čase zpracovávat obrazový tok, který má pouze desítky milionů pixelů za sekundu, a navíc autoři poskytují svá řešení jen pro velmi malá rozlišení obrazu. Hlavní nevýhodnou řešení uvedených v tabulce 6 je vysoká míra sekvenčního zpracování. Neméně důležitým problémem je i přenos nezpracovaných dat z kamery do výpočetní jednotky, jelikož pro takové datové toky je již potřeba minimálně *10* Gbit přenosové linky.

Tato práce však přestavuje novou multiparalelní architekturu pro detekci objektů, jež je založena na AdaBoost metodě – RT-MPOD. Tato architektura se liší od doposud představených řešení extrémně vysokou datovou propustností a proudovým charakterem zpracování dat. Výhodou RT-MPOD je necitlivost vůči složitosti zpracovávané scény, což nesplňují implementace založené na metodě WaldBoost. U architektury byly zcela zachovány veškeré výhody AC jednotky a zůstala zachována i rychlost zpracování jeden pixel za jeden hodinový cyklus a také to, že se není třeba vracet k již jednou zpracovanému pixelu. Díky tomu není nutné v architektuře implementovat velké vstupní vyrovnávací paměti pro uložení obrazu, či dokonce provádět vkládání zpožďovacích cyklů do procesu zpracování. RT-MPOD (implementovaná na Xilinx ZYNQ XC7Z045) může zpracovat obrazový tok s až *300* miliony pixelů za sekundu, což je téměř o řád výše než dokážou ostatní řešení uvedené v tabulce 6 (poznamenejme, že uvedená řešení jsou jen pro zpracování obrazu s velmi malým rozlišením *640 x 480* pixelů a zpracování obrazu s velkým rozlišením by je mohlo vzhledem k jejich sekvenčnímu charakteru znatelně zpomalit a zvýšit jejich nároky na výpočetní zdroje). Pro ilustraci uveďme, že rychlost zpracování RT-MPOD je dostatečná pro zpracování obrazového toku s rozlišením *3000 x 2000* pixelů a *50* snímky za sekundu. Novost architektury je především v multiparalelním zpracování dat. Všechny úrovně zmenšovaného obrazu jsou zpracovávány najednou (první základní úroveň paralelizmu) a také všechny slabé klasifikátory jsou zpracovávány paralelně (druhá základní úroveň paralelizmu). Výsledky zpracování ze všech úrovní jsou tak získány ve stejný čas. Umožnění vytvoření takového řešení je především díky AC jednotce, která má konstantní rychlost zpracování detekčního okna o délce jeden hodinový cyklus. Bez ní by byl uvedený princip zpracování neefektivní.

RT-MPOD, stejně jako AC jednotka, je navržena tak, aby ji bylo možné velmi snadno a efektivně rozdělit na několik výpočetních jednotek a provádět hybridní zpracování dat (tím se myslí například použití FPGA a CPU nebo FPGA a ARM). Podkapitola 6.2.5 ukazuje jakým způsobem je možné vytvářet malé předzpracovávající jednotky založené na RT-MPOD architektuře. Taková jednotka poté může být použita například jako filtrační jednotka, která bude sloužit pro výběr zájmových snímků. Filtrace obrazu se dá poté považovat za specializovaný obrazový kompresor, který může být pro řadu aplikací mnohem vhodnější než použití obecných videokodeků, které způsobují ztrátu informace. Aplikace takového kodeku vede k výraznému snížení požadavků na přenosovou linku a nákladnou komunikační infrastrukturu. Takovou malou předzpracovávající jednotku je možné implementovat s malými požadavky na zdroje v FPGA a může tak být implementován i v malém a levném FPGA, jako je například Xilinx Zynq XC7Z020. Vlastnosti architektury byly experimentálně ověřeny právě na takové předzpracovávající jednotce.

Rozdělení detektoru objektů na dvě výpočetní architektury s sebou může přinést také velmi výrazné úspory z hlediska spotřeby energií, pokud je zvoleno rozdělení na správné výpočetní architektury. Bude-li se většina operací provádět na FPGA, které má velmi nízkou spotřebu energií na vykonanou operaci, tak se celková spotřeba energií pro výpočet algoritmu výrazně sníží. Vhodným návrhem architektury bylo umožněno takové dělení a v závislosti na aplikaci tak může být dosažena úspora energií na jeden zpracovaný snímek až v řádu tisíců, jak bylo ukázáno i v autorově práci [39].

7.1 Přínos práce

Hlavní přínosy této práce lze shrnout do následujících bodů.

- Byla vytvořena metoda pro návrh nových aplikačně-specifických tvarů příznaků pomocí genetického algoritmu, které jsou navrženy s ohledem na FPGA implementaci, a bylo dosaženo zvýšení přesnosti klasifikace [40].
- Byla navržena nová architektura aplikačně-specifického klasifikátoru pro AdaBoost metodu a bylo ukázáno, že tato architektura má srovnatelnou klasifikační přesnost jako jiné architektury, ale poskytuje velmi vysokou rychlost zpracování v porovnání s doposud představenými řešeními. Architektura je založena na multiparalelním zpracování dat a dovoluje rozdělení úlohy na více výpočetních prostředků. Výsledky byly prezentovány postupně v pracích [41], [38].
- U navržené architektury bylo zcela odstraněno sekvenční zpracování především za pomoci neuspořádaného vyhodnocení slabých klasifikátorů a byla navržena plně proudová architektura, která má konstantní rychlost zpracování dat [41], [38]. Tedy propustnost architektury je necitlivá na obsah dat, a je tak garantována na rozdíl od architektur založených na metodě WaldBoost.
- Byla navržena metoda pro automatizované vytváření klasifikátoru na základě vstupních kritérií. Tato metoda umožňuje provádět přizpůsobení výsledného klasifikátoru vzhledem k několika požadavkům, jako je klasifikační přesnost, výkonnost a spotřeba energií. Generičnost algoritmu tak byla přenesena do jazyků vyšší úrovně abstrakce [38].

Práce představila a ověřila ucelenou metodiku pro automatizovaný návrh výkonných aplikačně-specifických klasifikátorů, ale současně zachovává pro návrháře možnost jednoduše měnit požadavky na výsledný klasifikátor, a uchovat tak řešení v obecné rovině. Uvedené řešení nemá v dostupné literatuře obdobu.

7.2 Pokračování práce

Největší slabinou RT-MPOD jednotky jsou její velké požadavky na FPGA výpočetní zdroje. Proto by se další výzkum měl zabývat metodou jak snížit požadavky na tyto zdroje. Možností jejich redukce je několik. První je založena na faktu, že intenzita výpočetních operací v RT-MPOD jednotce s každým stupněm zmenšení obrazu klesá, a je tak možné od určitého stupně provádět již sekvenční vyhodnocení, ale současně zachovat všechny výhody architektury. Vzhledem k rozložení operací do jednotlivých úrovní se dá usoudit, že již celá druhá polovina zpracování zmenšených snímků by mohla být zpracovávána sekvenčně a dosáhnout tak téměř *45%* úspory zdrojů.

Druhá možnost je založena na faktu, že RT-MPOD spotřebovává velmi vysoké množství *BlockRAM* pamětí pro uložení dat slabých klasifikátorů. Obsah *BlockRAM* pamětí pro všechny úrovně klasifikátoru je však stejný. Klasifikátory z nejvyšší úrovně přistupují do paměti (téměř) každý hodinový cyklus, a tak se mezi nimi jen velmi špatně vytváří možnost sdílení paměťových prvků. Avšak klasifikátory v posledních stupních zpracování do paměti přistupují jen „občas“ (vzhledem k nižší intenzitě výpočtů), a tak je možné vhodným prokládaným přístupem do paměti vytvořit sdílenou paměť pro větší skupinu AC jednotek. Na základě znalosti intenzity výpočetních operací se předpokládá, že by se touto technikou mohlo uspořit i více než *50* % *BlockRAM* pamětí.

Rychlost uvedené architektury je v porovnání s ostatními implementacemi velmi vysoká, avšak i tak je možnost ji ještě zvýšit. Konkurenční metody pro zvýšení rychlosti zpracování přeskakují při procházení obrazu detekčním oknem vybraný počet sloupců. Tuto techniku je možné vzhledem k principu AdaBoost aplikovat, jelikož objekty jsou zpravidla detekovány na několika sousedních pozicích. Tím je poté možné zvýšit rychlost algoritmu až čtyřikrát při zpracování každého čtvrtého sloupce. Rychlost zpracování by tak teoreticky mohla narůst na více než *1* miliardu pixelů za sekundu.

8 Příloha A

8.1 Příklad evolučně navržených tvarů příznaku

Níže jsou uvedeny dva příklady návrhu nových tvarů příznaků pro dvě různé klasifikační úlohy.

8.1.1 Boční pohled na automobil

Pro úlohu rozpoznání bočního pohledu auta byl na datasetu UIUC [1] proveden evoluční návrh nových tvarů pro LBP operátor. Parametry GA byly nastaveny následovně:

- Velikost trénovací množiny 10 000 vzorků.
- Použitý konfigurační soubor:

```
<evolution>
        <sizeOfGeneration val="100" />
        <sizeOfSelectionPop val="99" />
        <maxNumberOfGeneration val="4000" />
        <endingErrRate val="0.00000010" />
        <searchWindowSize width="6" height="6" />
        <initChromozomeDistribution val="5" />
        <selection type="tournament">
             <ChromosomeToTournament val="2" />
        </selection>
        <countOfMutation val="1" />
        <mutationProbability val="70" />
        <crosOverProbability val="80" />
        <populationBloatLimit val="250" />
        <fitnessFunction type="bestNAverageAndFPGA" FPGAinfluence="0.4" />
  </evolution>
```

Dosažený výsledný operátor má míru chyb *0,1283* a výsledný chromozom má tvar:

```
1: 000000 000000 000000 001010 000000 000000
2: 000000 000000 000000 000001 000000 000000
3: 000000 000000 000000 000000 000000 010000
4: 000000 000000 000000 000000 000010 000000
5: 000000 000000 000000 011111 000000 000111
6: 000000 000000 000000 000100 000000 000000
7: 000000 000001 000000 010111 000000 000111
8: 000000 011000 000000 001000 000000 000100
9: 000000 000000 000000 010000 000000 000000
```

Po překódování chromozomu do obrazového tvaru dostaneme:

	8	8			7
	5,7,9	1,5,8	5,6,7	1,5,7	2,5,7
				4	
	3		5,7,8	5,7	5,7

Obrázek 47: Evolučně navržený tvar příznaku pro úlohu rozpoznání bočního pohledu na automobil.

Na obrázku 47 je vyobrazena matice o rozměru *6 x 6* pixelů a tvar LBP operátoru získaného evolučním návrhem. LBP operátor se skládá z 9 oblastí a každou z oblastí lze interpretovat následovně. Patří-li pixel do dané oblasti, tak je v něm uvedeno číslo příslušné oblasti. Například oblast 1 se skládá ze dvou pixelů na pozicích *x = 3, y = 4* a *x = 5, y = 4*. Není-li uvedeno číslo oblasti na obrázku, tak je oblast prázdná.

Z příznaku je vidět několik důležitých poznatků:

- oblasti se výrazně překrývají,
- nejsou využity všechny pixely vymezeného obrazu,
- oblasti jsou nespojité,
- velikosti příznaků se velmi liší od 1 do 8 pixelů,
- velikosti příznaků mají mocninu dvou

8.1.2 Registrační značka

Pro úlohu rozpoznání RZ byl proveden evoluční návrh nových tvarů pro LBP operátor. Parametry GA byly nastaveny následovně:

- Použitý konfigurační soubor:

```
<evolution>
        <sizeOfGeneration val="50" />
        <sizeOfSelectionPop val="49" />
        <maxNumberOfGeneration val="2500" />
        <endingErrRate val="0.00000010" />
        <searchWindowSize width="6" height="6" />
        <initChromozomeDistribution val="5" />
        <selection type="tournament">
             <ChromosomeToTournament val="2" />
        </selection>
```

```
        <countOfMutation val="1" />
        <mutationProbability val="70" />
        <crosOverProbability val="80" />
        <populationBloatLimit val="250" />
        <fitnessFunction type="bestNAverageAndFPGA" FPGAinfluence="0.4" />
    </evolution>
```

Dosažený výsledný operátor má na trénovací množině míru chyb *0,025* a výsledný chromozom má tvar:

```
1: 010000 000000 000000 000000 000000 000010
2: 000000 000001 000000 000000 000000 000000
3: 001101 000000 000000 000001 001100 001100
4: 101000 001000 000100 000000 000000 000000
5: 000100 010000 000000 000000 000000 000000
6: 000100 011101 010000 010000 000000 000010
7: 011100 111111 000010 000010 001000 110110
8: 000011 100001 000000 000000 000000 000000
9: 100100 111100 000000 000000 000001 000100
```

Po překódování chromozomu do obrazového tvaru dostaneme:

4,9	1,7	3,4,7	3,5,6,7,9	8	3,8
7,8,9	5,6,7,9	4,6,7,9	6,7,9	7	2,6,7,8
	6		4	7	
	6			7	3
		3,7	3		9
7	7	3	3,7,9	1,6,7	

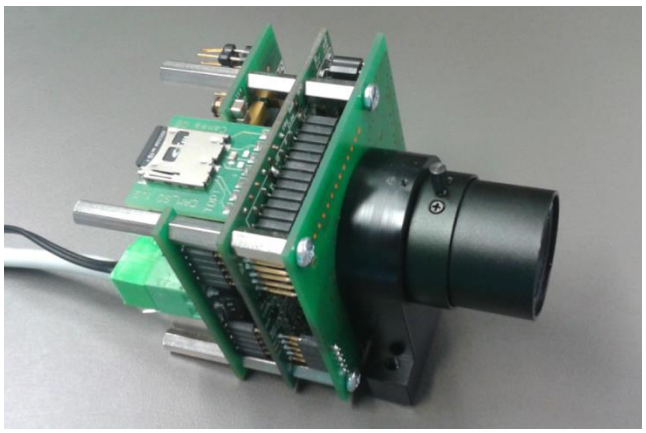
Obrázek 48: Evolučně navržený tvar příznaku pro úlohu rozpoznání SPZ.

Pro nově navržený tvar příznaků platí obdobné závěry jako pro příznak navržený pro boční pohled automobilu. Dalším zkoumáním bylo zjištěno, že příznak se adaptuje pro rozpoznávání horní hrany RZ.

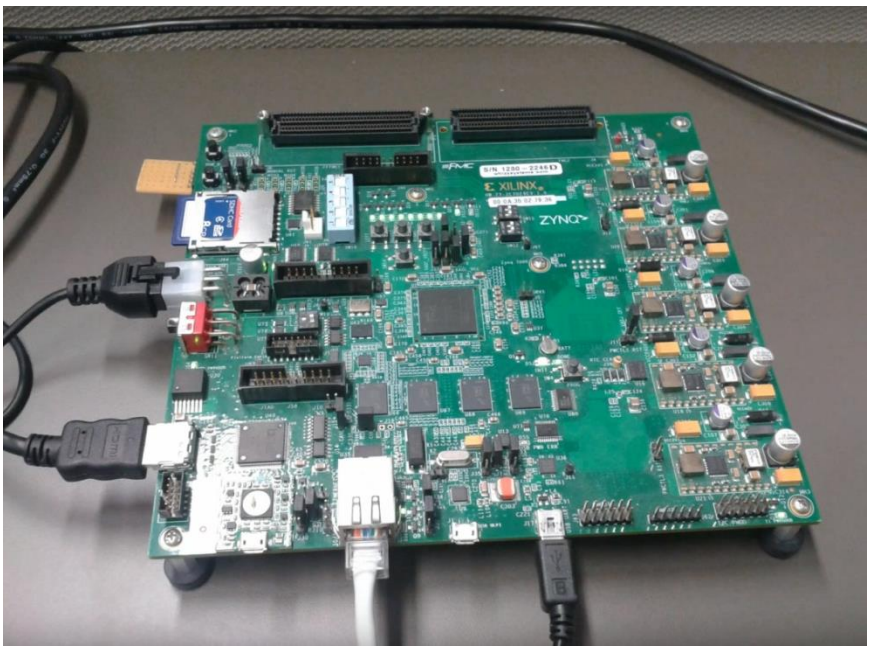
9 Příloha B

9.1 Realizace RT-MPOD klasifikátoru

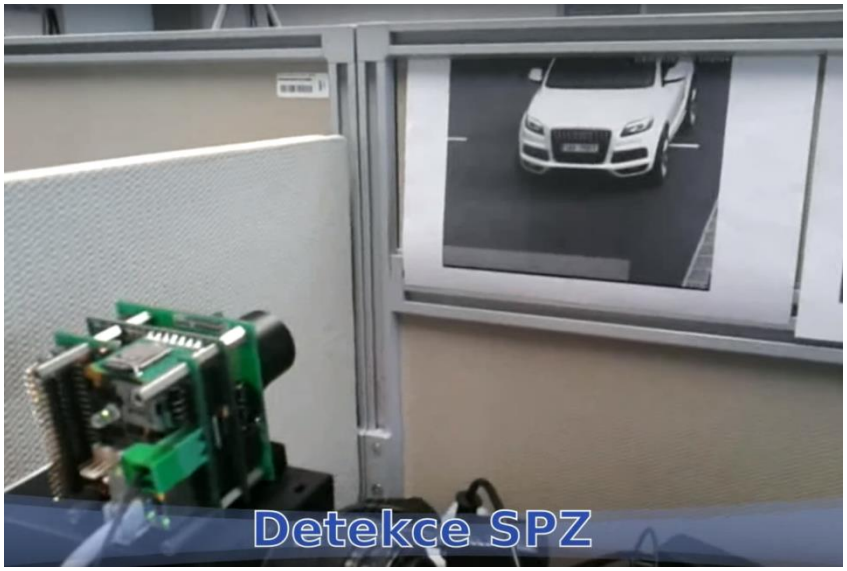
Ve spolupráci s Jiřím Husákem byla implementována testovací architektura popsaná v podkapitole 6.2.3 a vznikly následující obrázky. Na obrázku 49 je zobrazena použitá kamera, která byla zapůjčena firmou Camea, spol. s r. o. Na obrázku 50 je zobrazena vývojová deska Xilinx ZC 702, jež obsahuje FPGA Xilinx Zynq 7020. Obrázek 51 zobrazuje scénu s kamerou a se snímaným obrázkem. Na obrázku 52 je uvedena obrazovka monitoru s výstupem RT-MPOD architektury. V bílém rámečku je zobrazena detekovaná SPZ. Snímky jsou převzaty z demonstračního videa vytvořeného Jiřím Husákem [28].



Obrázek 49: Použitá kamera M621.



Obrázek 50: Použitá vývojová deska Xilinx ZC 702 [92].



Obrázek 51: Obrázek s kamerou snímající testovanou scénu.

Obrázek 52: Vykreslovaná scéna na monitoru připojeného PC. V bílém rámečku je vyznačena detekce SPZ.

Literatura

[1] Agarwal, S., Awan, A., Roth, D.: *UIUC Image Database for Car Detection.*

[2] Altera: HyperFlex Architecture, Stratix 10 Product Table, 2015.

[3] Barczak, A., Dadgostar, F.: Real-time hand tracking using a set of cooperative classifiers based on haar-like features. In *Research Letters in the Information and Mathematical Sciences*.2005.

[4] Benenson, R., Omran, M., Hosang, J., Schiele. B.: Ten years of pedestrian detection, what have we learned? *arXiv preprint arXiv*:1411.4304, 2014.

[5] Bourdev, L., Brandt, J. (2005). Robust object detection via soft cascade, *CVPR*.

[6] Drucker, H., Schapire, R., Simard, P.: Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 4, pp.705–719, 1993.

[7] Freund. Y., Schapire, R.: *A short introduction to boosting*. *Soc. for Artif*, no. 5, 1999: pp. 771-780.

[8] Freund, Y., Schapire, R. E.: *A decision-theoretic generalization of on-line learning and an application to boosting*. In *EuroCOLT '95: Proceedings of the Second European conference on Computational Learning Theory*, London, UK: Springer-Verlag, 1995, ISBN 3-540-59119-2, pp. 23-37.

[9] Freund. Y.: Boosting a weak learning algorithm by majority. I*nformation and Computation*, vol. 121, no. 2, pp. 256–285, 1995.

[10] Gao, C., Lu, S. L.: Novel FPGA based haar classifier face detection algorithm acceleration, in: *Proceedings of International Conference on Field Programmable Logic and Applications*, 2008, pp 373–378.

[11] Goldberg, D., Kalyanmoy, D.: A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In *Foundation of Genetic algorithms*, 1991, pp. 69-93.

[12] Green 500: online `www.green500.org`.

[13] Haar, A.*: Zur Theorie der orthogonalen Funktionensysteme*, chapter Mathematische Annalen. 1910, s. 331-371.

[14] Hanai, Y., Hori, Y., Nishimura, J., and Kuroda T.: A versatile recognition processor employing Haar-like feature and cascaded classifier, in *ISSCC, Dig. Tech. Papers*, Feb. 2009, pp. 148–149.

[15] Haralick, R. M., Sternberg, S. R., & Zhuang, X. (1987). Image analysis using mathematical morphology. Pattern Analysis and Machine Intelligence, IEEE Transactions on, (4), 532-550.

[16] He, C., Papakonstantinou, A., & Chen, D.: A novel SoC architecture on FPGA for ultra fast face detection. In *Computer Design*, 2009.

[17] Herout, A., Juránek, R., Zemčík, P.: Implementing the Local Binary Patterns with SIMD Instrictions of CPU. In *proceedings of WSCG 2010 Plzeň*, CZ, ZČU v Plzni, 2010, ISBN 978-80-86943-86-2 pp. 39-42.

[18] Herout, A., Zemcik, P., Juranek, R., Hradis, M.: Implementation of the "Local Rank Differences" Image Feature Using SIMD Instructions of CPU. In *Proceedings of Sixth Indian Conference on Computer Vision*, Graphics and Image Processing, 2008. ICVGIP '08. Sixth Indian Conference on.

[19] Herout, A., Zemčík, P., Hradis, M., Juránek, R., Havel, J., Jošth, R. a Žádník, M.: Low-Level Image Features for Real-Time Object Detection, *IN-TECH Education and Publishing*, pp. 25 2009.

[20] Herout,A., Jošth, R., Juránek, R., Havel, J., Hradiš, M., Zemčík, P.: Real-time object detection on CUDA. *In Journal of Real-Time Image Processing*, September 2011, vol. 6, no. 3, pp 159-170

[21] Hjelmås, E., Low, B. K.: "Face detection: A survey," *Comput. Vision Image Understanding*, vol. 83, no. 3, pp. 236–274, Sep. 2001.

[22] Hiromoto, M., Sugano, H., Miyamoto, R.: Partially parallel architecture for Adaboost-based detection with Haar-like features, *IEEE Trans. Circuits Syst. for Video Technol.*, vol. 19, no. 1, pp. 41–52, Jan. 2009.

[23] Holland, J.: *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[24] Hradiš, M., Herout, A., Zemčík, P.: Local Rank Patterns - Novel Features for Rapid Object Detection. In *Proceedings of International Conference on Computer Vision and Graphics 2008*, vol. 12 in Lecture Notes in Computer Science, Springer Verlag, 2008, ISSN 0302-9743, s. 1-12.

[25] Hradiš, M.: *AdaBoost v počítačovém vidění*. Diplomová práce, Fakulta informačních technologií, VUT v Brně, 2007.

[26] Hradis., M.: Framework for research on detection classifiers. In *Proceedings of the 24th Spring Conference on Computer Graphics (SCCG '08).* ACM, New York, NY, USA, pp. 155-161, 2008.

[27] Huang, C., Vahid, F.: Scalable object detection accelerators on fpgas using custom design space exploration, *SASP*, 2011.

[28] Husak, J.: *Using high-level synthesis for ZYNQ platform applications*. Diplomová práce. Faulta informačních technologií VUT v Brně, 2015.

[29] Chapman, B., Jost, G., Pas, R.: *Using OpenMP Portable Shared Memory Parallel Programming*. Scientific and Engineering Computation, 2007, ISBN: 9780262533027, pp. 384.

[30] Choi, W., Tse, S., Wong, K., Lam, K.: Simplified Gabor wavelets for human face recognition. *Pattern Recognition*, ročník 41, č. 3, 2008: s. 1186-1199, ISSN 0031-3203, part Special issue: Feature Generation and Machine Learning for Robust Multimodal Biometrics.

[31] Cho, J., Mirzaei, S., Oberg, J., Kastner, R.: "Fpga-based face detection system using haar classifiers," in Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays, New York, 2009, pp. 103–112.

[32] Chouchene, M., Sayadi, F., Bahri H., Dubois, J., Miteran J., Atri, M.: Optimized parallel implementation of face detection based on GPU component. *Microprocessors and Microsystems*, vol. 39, no. 6, August 2015, Pages 393–404

[33] Chen, Y., Li, W., Tong, X.: Parallelization od AdaBoost algorithm on Multi-Core processors. *Corporate Technology Group*, Intel Corporation

[34] ITU-T H.265 *Hight efficiency video codding*, 2013.

[35] Intel: Online: ark.intel.com/products/80807/Intel-Core-i5-3570K-Processor-6M-Cache-up-to-3_80-GHz.

[36] Juránek, R., Zemčík, P., Hradiš, M.: Real-Time Algorithms of Object Detection Using Classifiers. *InTech - Open Access Publisher*, 2012, pp. 1–22.

[37] Kadlček. F.: *Implementace obrazových klasifikátorů v FPGA*. Diplomová práce. Faulta informačních technologií VUT v Brně, 2010.

[38] Kadlcek, F., Fucik, O.: Automatic synthesis of small AdaBoost Classifier in FPGA, In: *IEEE Design and Diagnostics of Electronic Circuits and Systems DDECS'2013*. Brno: IEEE Computer Society, 2013, pp. 1-6. ISBN 978-1-4673-6133-0.

[39] Kadlček, F., Fučík, O: Fast and Energy Efficient AdaBoost Classifier, *FPGAWorld'13*, September 10-12, 2013, Copenhagen and Stockholm. Copyright 2013 ACM 978-1-4503-2496-0/13/09 .

[40] Kadlček, F., Fučík, O.: *Evolutionary design of Local Binary Pattern feature shapes for object detection.* In *NASA/ESA Conference on Adaptive Hardware Systems (AHS-2012)*, Nuremberg, Germany, 2012, s. 8.

[41] Kadlček, F., Zemčík, P., Juránek, R.: *Automatic synthesis of classifiers in FPGA.* In *International Bata conference for Ph.D. Students and Young Researchers*, Tomas Bata University in Zlin, 2011, ISBN 978-80-7454-013-4.

[42] Kearns, M. J.: *The computational complexity of machine learning*. MIT press. 1990.

[43] Kearns, M., Valiant, L.:. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the Association for Computing Machinery*, 41(1):67–95,January 1994.

[44] Kim, D., K., Jung, J., H., Nguyen, T., T., Kim, D. J., Kim, M., S., Kwon, K., H., Jeon, J., W.: An fpga-based parallel hardware architecture for real-time eye detection, *JSTS*, 2012.

[45] Kong, J., Deng, Y.: GPU accelerated face detection. In: *International Conference on Intelligent Control and Information Processing*, 2010, pp. 584–588.

[46] Krpec, J., Němec, M.: Face detection CUDA accelerating. In: *ACHI 2012: The Fifth International Conference on Advances in Computer–Human Interactions*, 2012, pp. 155-160.

[47] Kyrkou, C., Theocharides, T.: A flexible parallel hardware architecture for adaboost-based real-time object detection, In *VLSI Systems*, 2011.

[48] Lai, H. C., Savvides, M., Chen, T.: "Proposed FPGA hardware architecture for high frame rate (>100 fps) face detection using feature cascade classifiers, in *Proc. 1st IEEE Int. Conf. Biometrics: Theory, Appl., Syst.*, Sep. 2007, pp. 1–6.

[49] Lee, T.: *Image representation using 2D Gabor wavelets*, vol. 18 no. 10, 1996: pp. 959-971.

[50] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86 (11)*, 2278–2324. doi:10.1109/5.726791. Retrieved 16 November 2013.

[51] Li, X., Kin-Man, L., Shen, L., Zhou, J.: Face detection using simplified Gabor features and hierarchical regions in a cascade of classifiers. *Pattern Recognition Letters*, vol. 30, no. 8, 2009: pp. 717-728, ISSN 0167-8655.

[52] Li, S., Zhang, Z., Shum, H. & Zhang, H.: Floatboost learning for classification, The Conference on Advances in Neural Information Processing Systems (NIPS), 2002.

[53] Liao, S., Zhu, X., Lei, Z., et al.: Learning Multi-scale Block Local Binary Patterns for Face Recognition. *Chinese Academy of Sciences*, China, 2007, pp. 828-837.

[54] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision*, 1999. *The proceedings of the seventh IEEE international conference on*, vol. 2, pp.1150–1157.

[55] Mäenpää, T.: *The Local Binary Pattern approach to texture analysis - extensions and applications*. Dizertační práce, Faculty of Technology, University of Oulu, 2003.

[56] Manning, Ch., Raghavan, P., Schutze, H.: Introduction to Information Retrieval. *Cambridge University Press*, 2008, 978-0-521-86571-5.

[57] McCready, R.: "Real-time face detection on a configurable hardware system," presented at the Int. Symp. Field Program. Gate Arrays, Monterey, CA, 2000.

[58] Nanni, L., Lumini, A.: Local binary patterns for a hybrid fingerprint matcher. *Pattern Recognition*, vol. 41, no. 11, 2008: s. 3461-3466, ISSN 0031-3203.

[59] NVIDIA CUDA C Programming Guide Version 4.0 [online]. http://developer.download.nvidia.com/compute/cuda/4_0/toolkit/docs/CUDA_C_Programming_Guide.pdf

[60] Ojala, T., Pietikäinen, M., Harwood, D.: A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, vol. 29, no. 1, 1996: s. 51-59, ISSN 0031-3203.

[61] Pertsau, D., Uvarov, A.: Optimal structure of face detection algorithm using GPU architecture, In *XIV International PhD Workshop OWD*, 2012, pp. 20–23.

[62] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[63] Pietikäinen, M.: Image Analysis with Local Binary Patterns. In *Image Analysis*, University of Oulu, Finland, 2005, ISBN 978-3-540-26320-3, pp. 115-118.

[64] Polok, L., Herout, A., Zemčík, P., Hradiš, M., Juránek, R., Jošth, R.: "Local Rank Differences" Image Feature Implemented on GPU. In *ACIVS '08: Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, Berlin, Heidelberg: Springer-Verlag, 2008, ISBN 978-3-540-88457-6, pp. 170-181.

[65] Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach (2nd ed.)*, Upper Saddle River, New Jersey: Prentice Hall, pp. 111–114, ISBN 0-13-790395-2.

[66] Simard, P., Steinkraus, D., Platt. J.: Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *ICDAR*, vol. 3, pp. 958-962. 2003.

[67] Shi, Y., Zhao, F., Zhang, Z.: Hardware implementation of adaboost algorithm and verification, in *Proc. 22nd Int. Conf. Adv. Inf. Netw. Appl.—Workshops (AINAW)*, 2008, pp. 343–346.

[68] Schapire, R.: The strength of weak learnability. *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990.

[69] Sharma, B., Thota, R., Vydyanathan, N., Kale, A.: Towards a robust, real-time face processing system using CUDA-enabled GPUs, in: *International Conference on High Performance Computing*, 2009, pp 368–377.

[70] Shelton, J. a kol.: Genetic Based LBP Feature Extraction ans Selection for Facial Recognition. In proceeding *ACM-SE'11 Proceedings of the 49th Annual Souteast Regional Conference*, ACM New York, USA, 2011, ISBN 978-1-4503-0686-7.

[71] Sochman, J., Matas, J.: Waldboost - learning for time constrained sequential detection, *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* - Volume 2, IEEE Computer Society, Washington, DC, USA, 2005, pp. 150–156..

[72] Sochman, J., Matas, J.: Adaboost with totally corrective updates for fast face detection, *FGR*, pp. 445–450, 2004.

[73] Sochman, J., Matas, J.: Learning fast emulators of binary decision processes, *International Journal of Computer Vision*, 2009, 83(2): pp. 149–163.

[74] Stone, J. E., Gohara, D.,Shi, G.: .OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 2010, vol. 12, no. 1-3, pp. 66-73.

[75] Tajeripour, F., Kabir, E. and Sheikhi, A.: Fabric defect detection using modified local binary patterns. *EURASIP J. Adv. Signal Process*, 2008, pp. 1-12, ISSN 1110-8657.

[76] Theocharides, T., Link, G., Vijaykrishnan, N., Irwin, M. J., Wolf, W.: Embedded hardware face detection, presented at the *17th Int. Conf. VLSI Des.*, Mumbai, India, Jan. 2004.

[77] Tang, X., Ou, Z., Su, T., Zhao, P.: Cascade AdaBoost classifiers with stage features optimization for cellular phone embedded face detection system, in *Proc. ICNC*, 2005, pp. 688–697.

[78] Tatourian, A.: NVIDIA GPU Architecture & CUDA Programming environment, In *Computer Science*, CUDA, GPGPU, Parallel Programming, Software Architecture, 2013.

[79] Theocharides, T., Vijaykrishnan, N., Irwin, M. J.: A parallel architecture for hardware face detection, in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI Des*. (ISVLSI), Karlsruhe, Germany, pp. 452–453.

[80] Valiant, L., G.: A theory of the learnable. *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142,November 1984.

[81] Viola, P., and Johnes, M.: Robust Real-Time Object detection, Cambridge research laboratory, 2001.

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

[83] Wei, Y., Bing, X., Chareonsak, C.: FPGA implementation of AdaBoost algorithm for detection of face biometrics, in *Proc. IEEE Int. Workshop Biomed. Circuits Syst.*, 2004, pp. S1/6-17–S1/6-20.

[84] Wald, A.: *Sequential Analysis*, JohnWiley and Sons, Inc, 1947.

[85] Wiegand, T., Sullivan, G.J. Bjontegaard, G., Luthra, A.: Overview of the H.264/AVC video coding standard,Circuits and Systems for Video Technology, IEEE Transactions, vol.13, no. 7, ISSN 1051-8215, pp. 560-576, 2003.

[86] Wilson, P. I., Fernandez, J.: Facial feature detection using Haar classifiers. *J. Comput. Small Coll.*, vol. 21, no. 4, 2006: s. 127-133, ISSN 1937-4771.

[87] Xilinx Inc.: San Jose, CA, Xilinx University Program, Jan. 2009. Available: http://www.xilinx.com/univ/.

[88] Xilinx Inc.: *Virtex-II Pro and Virtex-II Pro X FPGA User Guide* [online]. http://www.xilinx.com/support/documentation/user_guides/ug012.pdf (prosinec 2011).

[89] Xilinx Inc.: Zynq-7000 All Programmable SoC Overview, 2008.

[90] Xilinx Inc.: UltraScale Architecture and Product Overview, DS890, December 2015.

[91] Xilinx Inc.: Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit. 2015. http://www.xilinx.com/products/boards-and-kits/ek-z7-zc702-g.html

[92] Xilinx Inc.: ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide. 2014. http://www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/ug850-zc702-eval-bd.pdf.

[93] Xilinx Inc.: Vivado Design Suite User Guide. 2014. Online http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2013-4.html.

[94] Yang, G., Huang, T. S.: Human face detection in a complex background. In Patter Recognition, vol. 27, Elsevier, 1994, pp. 53 – 63.

[95] Zemčík, P., Žádnik, M.: AdaBoost Engine. In *Field Programmable Logic and Applications*, Aug. 2007, p. 656-660.

[96] Zemčík, P., Juránek, R., Musil, P., Musil, M., Hradiš, M.: High performance architecture for object detection in streamed videos. IEEE. 2013.

[97] Zhang, L., Chu, R., Xiang, S., et al.: Face Detection Based on Multi-Block LBP Representation. In *Advances in Biometrics, Chinese Academy of Sciences*, 2007, ISBN 978-3-540-74548-8, s. 11-18.

[98] Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., Cong, J.: Optimizing fpga-based accelerator design for deep convolutional neural networks. In Proceedings of the *2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. s. 161-170. February 2015. ACM.