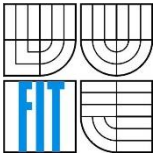
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH A OPTIMALIZACE OBRAZOVÝCH KLASIFIKTÁRŮ

DESIGN AND OPTIMALISATION OF IMAGE CLASSIFIERS

DISERTAČNÍ PRÁCE
PHD THESIS

AUTOR PRÁCE — Ing. FILIP KADLČEK
AUTHOR

VEDOUCÍ PRÁCE — doc. Dr. Ing. OTTO FUČÍK
SUPERVISOR

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.

Abstrakt

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.

Klíčová slova

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

Obsah

1 Úvod

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. 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.

Detekce objektů se provádí pomocí obrazových klasifikátorů. Jednou z velmi dobře známých a také velmi často používaných metod je AdaBoost [6]. Tato univerzální klasifikační metoda však vyžaduje 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 [40]) 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.

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. Možnost měnit vlastnosti operativně a rychle mezi uvedenými kritérii je velmi žádoucí. Takový požadavek však univerzální implementace splňují jen velmi obtížně. Aplikačně specifický přístup k implementaci prezentovaný v této práci však ukazuje techniku, jak tyto potřeby velmi efektivně splnit.

Práce [3] 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.

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.

2 Detekce 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 detektory objektů, 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.

2.1 Vektor příznaků

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 2.2. 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.

2.2 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 2.3).

2.2.1 Local Binary Pattern (LBP)

Local Binary Pattern (LBP) je diskrétní obrazový operátor, který byl představen v prací [28] a později také v [26]. 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 [9]:

$$LBP(x_c, y_c) = \sum_{n=0}^{7} 2^n \, s\left(x_{i_n} - y_c\right) \tag{2.1}$$

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:

$$s(x) = \begin{cases} 1, if\ x \geq 0 \\ 0, if\ x < 0 \end{cases} \tag{2.2}$$

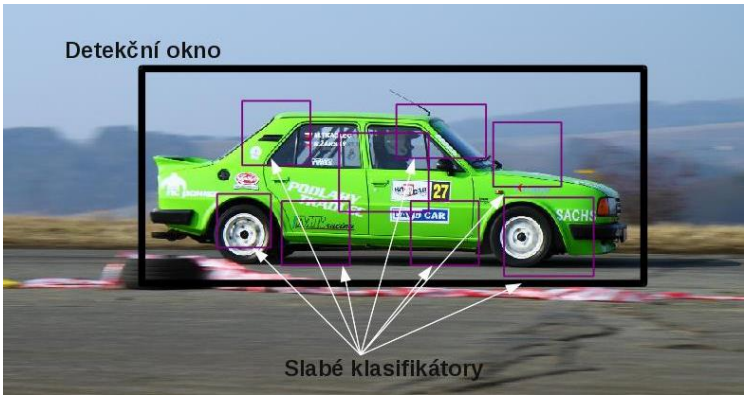
LBP je velmi robustní operátor, který je invariantní vůči změně intenzity světla v obraze. Pro získání větší robustnosti však bylo vyvinuto jeho rozšíření na MB-LBP - *Multi-scale Blok Local Binary Pattern* [25]. Rozšíření spočívá v nahrazení jednoho pixelu v mřížce *3 x 3* pixelů původního LBP za regiony. 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. Vyhodnocení MB-LBP je velmi podobné vyhodnocení LBP. Pouze jednotlivé pixely jsou nahrazeny hodnotou regionů. 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.

2.2.2 Slabý klasifikátor

Slabé klasifikátory jsou založeny například na dříve uvedených obrazových operátorech. Slabý klasifikátor je vytvořen tak, že je operátor instanciován na definované místo v obraze a je mu dána velikost. Jeden operátor tak může být instanciován 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.

2.3 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 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.



Obrázek 1: Detekční okno se sedmi slabými klasifikátory. Výsledné vyhodnocení je založeného na metodě AdaBoost.

Princip funkce klasifikátoru (rozpoznání předmětu, ne 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). Jakýkoliv klasifikovaný předmět musí být celý v detekčním okně. Obrázek 1 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 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.

2.3.1 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 [5]. Algoritmus odstraňuje řadu nevýhod původní *boostingové* metody.

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:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{2.3}$$

Kde funkce *sign* je definována jako:

$$sign(x) = \begin{cases} 1 \; když \; x > 0 \\ -1 \; jinak \end{cases} \tag{2.4}$$

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 lze však 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, je možné toto násobení následně vypustit (předpočítat na konci trénovacího procesu).

3 Implementace boostingových metod

Od prvního představení adaptivní *boostingové* metody autory Freund a Schapire [5], [4] v roce 1995 bylo publikováno mnoho prací, které se snaží efektivně tuto metodu implementovat. První práce se zabývaly implementací metody na standardních CPU [5], 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.

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 [31], [27], [11]. Autoři v těchto pracích jen zřídkakdy zvažovali AdaBoost za alternativu k jimi uvedeným řešením. Ačkoliv je AdaBoost univerzální detektor, tak se první implementace na FPGA velmi často objevovali 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 [32], [31], [12], [14], [34], [29], [24] a [7]. Většina uvedených řešení je založena na základních principech, jež byly představeny v práci Theocharidese [33]. 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 [12] 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. Propustnost řešení navrženého Hiromotem je *30* snímků za sekundu při rozlišení *640 x 480* pixelů.

Autor Cho [14] 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.

Shi ve své práci [29] 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.

Zemčík a Žádník představili první implementaci AdaBoost engine již v roce 2007 v práci [39]. Engine je založen na využití principu jemného multi-threadingu 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 [35], 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.

Později Zemčík ve spolupráci s dalšími autory svůj engine přepracoval do nové podoby, kterou prezentoval v práci [40]. 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 [39]. 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. Engine pracuje jako programovatelný automat, jež je řízen *64*bitovou instrukční sadou. Engine implementuje zřetězené zpracování za účelem paralelizování zpracování jednotlivých slabých klasifikátorů. 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 a zpracovával by obraz o rozlišení minimálně FullHD, by se významně změnila spotřeba zdrojů na FPGA a zřejmě by i výrazně poklesla propustnost architektury. Výsledky této práce je tak třeba posuzovat v omezeních, které si stanovili autoři, a nelze je přímo srovnávat například s řešením vytvořeným Kyrkou [23].

3.1 Porovnání dosažených výsledků známých architektur

V tabulce 1 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 1: Dosažené výsledky jednotlivých implementací pro rozlišení obrazu 640x 480 pixelů.

	Zemcik [39]	**Hiromoto [12]**	**Cho [14]**	**Wei [34]**	**Shi [29]**	**Lai [24]**	**Kykou [23]**	**Zemcik [40]**	**He [8]**
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

V práci [3] bylo ukázáno, že největšího pokroku z hlediska klasifikace bylo dosaženo vytvořením nových typů slabých klasifikátorů. 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. A s návrhem nových příznaků se tato množina rozroste.

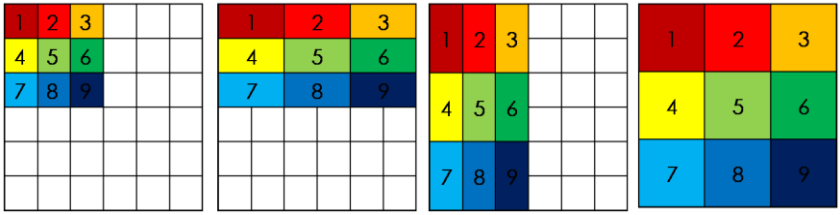
4.1 Tvar operátoru

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 [10]. Uveďme, že pro detekci obličeje se běžně používá velikost detekčního okna *24 x 24* pixelů [10] 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 2 ukazuje ručně stanovené a v současnosti používané tvary operátorů [9]. 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.

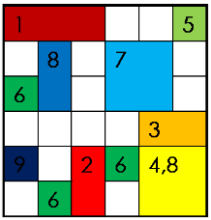
Každá oblast (jedna barva v obrázku) tvoří vždy jen jednu spojitou obdélníkovou oblast, stejně tak i všechny příznaky dohromady tvoří další spojitou oblast. Žádné dvě oblasti se nepřekrývají a žádný pixel není součástí více než jedné oblasti v jednom tvaru příznaku.



Obrázek 2:Standardní tvary příznaků [9].

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 3 ukazuje možný tvar nově navržených příznaků. 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 výsledky jako u Haarových vlnek.



Obrázek 3: 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 [28] víme, že se při výpočtu LBP příznaku se provádí porovnání všech okrajových oblastí (jejich konvolucí) s konvolucí 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. Proto je nutné provést úpravu dynamických rozsahů u obou hodnot do stejného „normalizovaného“ rozsahu. Převod do normovaného 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.

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 také které mají velký stavový prostor. GA je vhodný pro řešení uvedeného problému, jelikož není třeba naleznout globální maximum, ale postačuje naleznout 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.

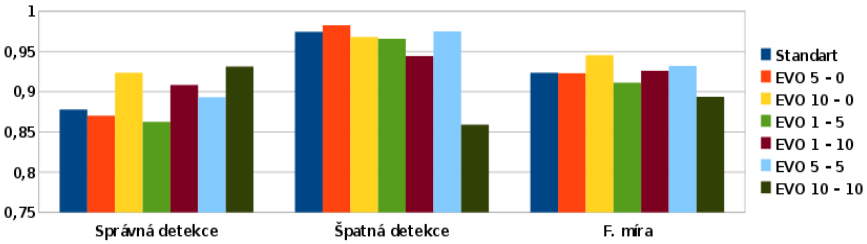
4.3 Dosažené výsledky

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.

Výsledky silných klasifikátorů s evolučně navrženými příznaky jsou porovnávány se silnými klasifikátory, které využívají pouze standardní tvary příznaků. Celý klasifikátor nemůže být sestaven pouze z evolučně navržených tvarů příznaků, jelikož by nepracoval dostatečně obecně a přizpůsobil by se jen datům trénovací množiny. Proto jsou klasifikátory pro testování 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 na standardních tvarech příznaků.

Obrázek 4 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 4: Porovnání výsledků AdaBoost klasifikátorů založených na standardních a evolučních příznacích

Obrázek 4 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.

5 Jádro klasifikátoru

Kapitola představuje nový pohled na problematiku vyhodnocení AdaBoost klasifikátoru v FPGA. 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 kapitole 3, ale i jiné, se k vyhodnocení AdaBoost klasifikátoru staví konvenčním, tedy sekvenčním způsobem, jež je popsán v podkapitole. V této práci je však v podkapitole 5.1 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 2.2.1).

5.1 Neseřazené vyhodnocení detekčního okna

Architektura navržená v této práci se 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 kapitole 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.

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 (FIT) a Roman Juránek (FIT). Navržený koncept byl pak následně autorem práce zpracován, dále významně vylepšen a realizován do výsledné podoby (viz následující odstavce).

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.

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 5 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):

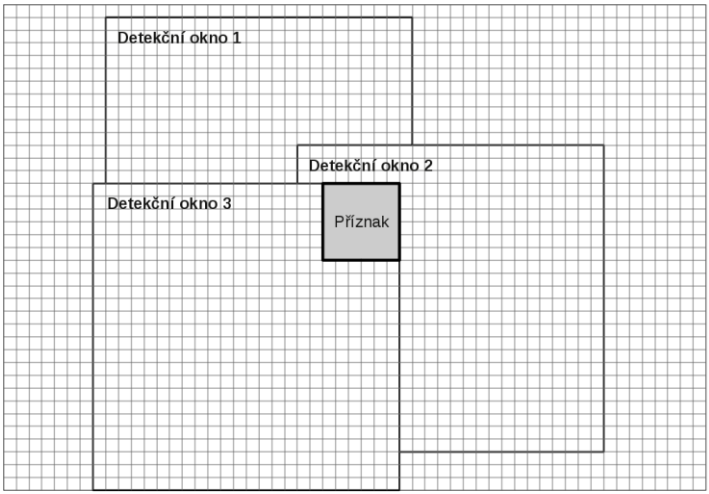
- *P1 - x = 19, y = 0*
- *P2 - x = 4, y = 3*
- *P3 - x = 18, y = 14*

Na obrázku 5 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.



Obrázek 5: Zpracování příznaků v rámci několika detekčních oken.

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 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:

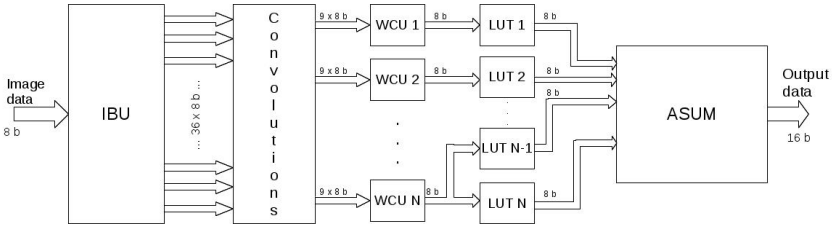
$$delay(P1, P2) = (P2.y - P1.y) * IMGWidth + (P2.x - P1.y) \tag{5.1}$$

- 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.

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 kapitola 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.

5.2 Architektura AdaBoost jádra (AC)

Na základě schématu vyhodnocení uvedeného v předchozí podkapitole byla navržena následující HW architektura klasifikátoru. Ten je navržen jako zřetězená zpracovávající linka, která je rozdělena do pěti základních stupňů. Architektura navrženého klasifikátoru je zobrazena na obrázku 6. V následujících odstavcích je uveden popis jednotlivých částí, ze kterých se architektura skládá.



Obrázek 6: Architektura AdaBoost jádra klasifikátoru.

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.

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).

Jednotka slabých klasifikátorů a vyhledávací tabulky

Vstup do jednotek slabých klasifikátorů (*WCU*) je tvořen konvolucemi z předchozí jednotky *Convolutions*. Architektura LBP operátoru 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.

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 pravděpodobnost, 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.

5.3 Automatická syntéza klasifikátoru

Jelikož struktura AC 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. 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.

5.4 Nároky za zdroje v FPGA

AC jednotka je záměrně navržena tak, aby používala co nejmenší množství FPGA zdrojů [18]. 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 2 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.

Z tabulky 2 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 [13] 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 je potřeba FPGA s velkým množstvím dostupných zdrojů.

Tabulka 2: AC jednotka – Závislost spotřeby FPGA zdrojů na počtu slabých klasifikátorů pro vstupní obraz s FullHD rozlišením.

Poč. sl. klasif.	Rejection rate Klasif.	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 %

5.5 Iterační proces návrhu AC jednotky

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 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.

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. Výstupem této operace jsou nové tvary příznaků pro LBP operátory, které jsou optimalizovány pro právě zpracovávanou úlohu (konkrétní typ předmětu).

- *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*.

- *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.

- *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 5.3.

- *Ohodnocení klasifikátoru (FPGA zdroje)*: se provádí za účelem zjištění reálných parametrů klasifikátoru pomocí nástroje Xilinx Vivado 2013.2 [38] (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.

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.

6 Multiparalelní detektor objektů pracující v reálném čase

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.

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 [40], 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.

6.1 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.

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).

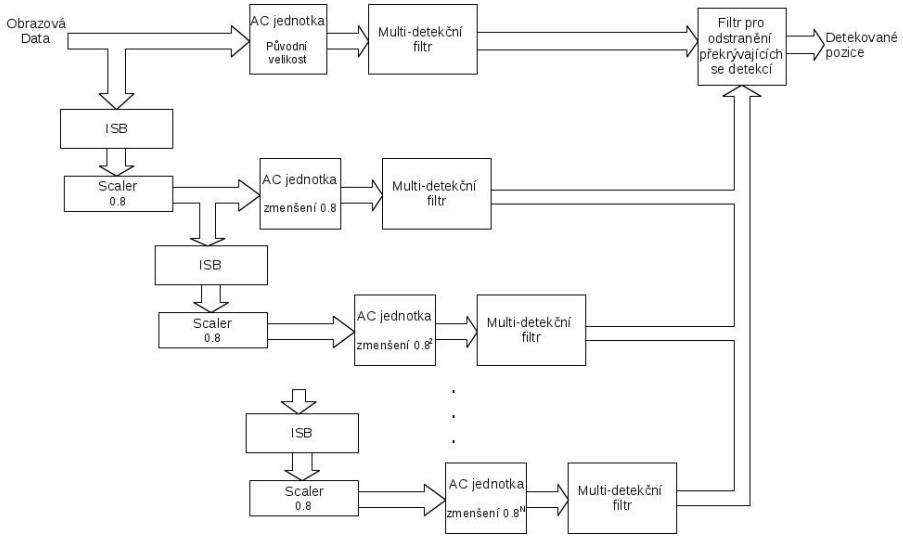
Zpracování vstupního obrazu je tedy prováděno v několika krocích, jak je ostatně ukázáno i na obrázku 7. V první iteraci je zpracováván obraz v rozlišení, které je dáno obrazovým senzorem a současně je provedeno zmenšení těchto obrazových dat pro další iteraci. V dalších iteracích se tento proces neustále 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 sekvenční jednotky uvedené 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 by sekvenční architektura 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 postavené na sekvenčním způsobu vydnocení 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*.

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 tvoří proud pixelů z obrazového senzoru. Data jsou 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 7: 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.

Aby bylo možné vytvořit *RT-MPOD*, jak je zobrazena na obrázku 7, 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 pouze pixely originálního obrazu jsou uvažovány. 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 Propustnost architektury

Nejdůležitější hledisko během návrhu nové architektury AdaBoost klasifikátoru bylo dosažení co největší datové propustnosti, jak je jen možné. 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. Ze vzorce 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.

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 [37] (*Place and Route*). S touto architekturou je tak možné zpracovávat vstupní obraz s rozlišením až *6* megapixelů a 50 snímky za sekundu. Zpracování takového množství obrazových dat (*300* Mpps) je však i na dnešních CPU (Intel Core i5-3570K čtyři jádra, pracovní frekvence *4,1* GHz [16]) daleko za hranicí jejich limitů, jelikož rychlost zpracování AdaBoostu na uvedeném CPU je přibližně *3,3* Mpps [19]. Rychlost zpracování však závisí na mnoha kritériích, jako je například délka klasifikační kaskády a podobně.

Tabulka 4 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 4 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 4 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. Propustnost architektury je možné stanovit pouze na základě dosažené pracovní frekvence a parametrů vstupního obrazu, jak ukazuje vzorec (6.1). Uvedené řešení je tak stále významně rychlejší než jiná známá řešení.

Tabulka 3: 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 [39]	22	None	-	14	2980	Virtex II 250
Kim [22]	50	-	106	18	133000	Virtex5 LX330T
Kyrkou [23]	40	0.67	100	24	25800	Virtex2 XC2VP30
Lai [24]	143	0.75	126	44	20900	Virtex2 XC2VP30
Zemčík [40]	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 [40] 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 [40] je založeno na vysoce optimalizovaném mikroprogramovatelném stroji, který je však ze své podstaty sekvenční. To znamená, že řešení [40] 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 [30] detekčního okna, která výrazně zkracuje sekvenční vyhodnocení. Na řešení [40] 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. Zpracování obrazu s více objekty zájmu se tak u implementace [40] může prodloužit, a při práci na maximální rychlosti je tak třeba vytvářet vyrovnávací paměti pro příchozí snímky. U implementace [40] 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 4) 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 [40], 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.

6.3 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 7, 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*. Klasifikační přesnost takového řešení je stále na velmi dobré úrovni a je více než *90%* v mnoha typech úloh, jak bylo ukázáno v práci [13].

Tabulka 5 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 5, tyto požadavky jsou za hranicí možností dnes dostupných FPGA, avšak s příchodem nových FPGA Xilinx UltraScale+ [36] nebo Altera Stratix 10 [2] již implementace bude možná, jelikož tyto čipy obsahují dostatek prostředků.

Tabulka 4: Závislost spotřeby FPGA zdrojů na počtu slabých klasifikátorů – RT-MPOD klasifikátoru pro vstupní obrazový s FullHD rozlišením.

Počet slabých klasif.	**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 5 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 [40] 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 [37] nebo podobné čipy, které obsahují jednak programovatelnou logiku ale také univerzální procesor jsou velmi vhodným kandidátem pro implementaci takového řešení. ARM procesor v Zynq 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 5, 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.

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.

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 jednou již zpracovaným pixelům. Klasifikační přesnost uvedené jednotky není téměř nijak ovlivněna. (Pouhý malý 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 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 nejsou téměř schopny zpracovávat takový obraz v reálném čase. Architektury uvedené v tabulce 1 jsou schopny v reálném čase zpracovávat obrazový tok, který má pouze desítky Mpps, a navíc autoři poskytují svá řešení jen pro velmi malá rozlišení obrazu. Hlavní nevýhodnou řešení uvedených v tabulce 1 je vysoká míra sekvenčního zpracování.

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. RT-MPOD (implementovaná na Xilinx ZYNQ XC7Z045) může zpracovat obrazový tok s až *300* Mpps, což je téměř o řád výše než dokážou ostatní řešení uvedené v tabulce 1. 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.

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). Vlastnosti architektury byly experimentálně ověřeny právě na takové předzpracovávající jednotce.

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 [20].
- 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 [21], [18].
- 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 [21], [18]. 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 [18].

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.

Literatura

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

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

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

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

[5] 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.

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

[7] 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.

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

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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 .

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

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

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

[28] 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.

[29] 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.

[30] 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..

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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).

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

[37] 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

[38] 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.

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

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

Publikace autora

Kadlček, F.: Zarovnávání paralelních textů. Bakalářská práce, FIT VUT v Brně, Brno, 2008.

Kadlček, F.: Implementace obrazových klasifikátorů v FPGA. Diplomová práce, FIT VUT v Brne, Brno, 2010.

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

Kadlček, F.: Platforma pro akceleraci obrazových klasifikátorů. In Počítačové architerktury a diagnostika 2011. Česko-slovenský seminář pre študentov doktorandského štúdia, Stará Lesná, 2011, s. 32-37.

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.

Kadlček, F.: Návrh a optimalizace obrazových klasifikátorů, In: Počítačové architektury & diagnostika 2012, Milovy, CZ, VCVUT, 2012, s. 91-96, ISBN 978-80-01-05106-1.

Kadlček, F., Fučík, O.: Fast and Energy Efficient AdaBoost Classifier, In: FPGAWorld'13, Copenhagen and Stockholm, SE, ACM, 2013, s. 1-5, ISBN 978-1-4503-2496-0.

Kadlček, F., Fučík, O.: Automatic synthesis of small AdaBoost Classifier in FPGA, In: IEEE Design and Diagnostics of Electronic Circuits and Systems DDECS'2013, Brno, CZ, IEEE CS, 2013, s. 1-6, ISBN 978-1-4673-6133-0.

Tobola, P., Kadlček, F.: EMC Model Building Using ANSA, In: 4th ANSA & µETA International Conference, 2 June 2011, Thessaloniki, Greece, Volume: Session 8B.

Životopis

Osobní údaje	
Příjmení, Jméno	**Kadlček Filip**
Adresa	U stadionu 761, 687 62 Dolní Němčí, Česká Republika.
Telefon	+420776251098
E-mail	kadlcek.filip@centrum.cz
Web	http://www.fit.vutbr.cz/~ikadlcek/
	http://www.linkedin.com/pub/filip-kadlček/42/44a/376
Státní příslušnost	česká
Datum narození	8.7.1986
Pohlaví	mužské
Vzdělání	
Období	2010 - nyní
Dosažená klasifikace	Doktorské studium
Obor / Zaměření	Výpočetní technika a informatika
Název organizace	Fakulta Informačních Technologií, VUT v Brně
Období	2008 - 2010
Dosažená klasifikace	Ing. (s vyznamenáním)
Obor / Zaměření	Informační technologie - Počítačové systémy a sítě
Název organizace	Fakulta Informačních Technologií, VUT v Brně
Období	2005 - 2008
Dosažená klasifikace	Bc. (s vyznamenáním)
Obor / Zaměření	Informační technologie
Název organizace	Fakulta Informačních Technologií, VUT v Brně
Období	2001 - 2005
Dosažená klasifikace	Maturita (s vyznamenáním)
Obor / Zaměření	Elektrotechnika a informační technologie
Název organizace	Střední průmyslová škola, Uherské Hradiště
Pracovní zkušenosti	
Období	2014 - nyní
Vykonávaná funkce	Vývojář dopravních aplikací
Hlavní pracovní náplň a oblasti	Vývoj a výzkum v oblasti dopravních zařízení pro monitorování a sledování silničního provozu a parkovacích oblastí
Název organizace	CAMEA, spol. s r. o.
Období	2007 - 2014
Vykonávaná funkce	Senior SW developer
Hlavní pracovní náplň	Vývoj zakázkového SW, vedení projektů, analýza problémů, vývoj SW pro zpracování výsledků z CFD analýz a strukturálních analýz, vývoj uživatelských modulů pro SW ANSA a META, vývoj v rámci evropského projekt HIRF-SE; spolupráce na projektech s českými i zahraničními partnery
Název organizace	Evektor, spol. s r. o.

Období	2007 - 2014
Vykonávaná funkce	Externí spolupracovník
Hlavní pracovní náplň	TKF – předvývoj, TFA – metodický tým, vývoj SW, pro zpracování výsledků z CFD analýz a strukturálních analýz, vývoj uživatelských modulů pro SW ANSA a META
Název organizace	Škoda Auto, a. s.
Období	2010 - nyní
Vykonávaná funkce	Doktorské studium
Hlavní pracovní náplň	Vývoj nástroje pro automatizovanou syntézu malých obrazových klasifikátorů v FPGA. Vývoj architektury klasifikátoru v FPGA.
Název organizace	Fakulta Informačních Technologií, VUT v Brně
Období	2006
Vykonávaná funkce	SW developer - young researcher
Hlavní pracovní náplň	Vývoj systému pro automatický překlad matematických vzorců do brailova písma
Název organizace	Fakulta Informačních Technologií, VUT v Brně.

Výzkumné a rozvojové projekty

2014-2016	Architektury paralelních a vestavěných počítačových systémů, VUT v Brně, FIT-S-14-2297.
2011-2013	Pokročilé bezpečné, spolehlivé a adaptivní IT, VUT v Brně, FIT-S-11-1.