

Detekce obličejů z obrazu neuronovou sítí

Bc. Martin Hozík

Diplomová práce
2014



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2013/2014

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Martin Hozík**
Osobní číslo: **A12389**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Informační technologie**
Forma studia: **prezenční**

Téma práce: **Detekce obličejů z obrazu neuronovou sítí**
Téma anglicky: **Face Detection in Image by Means of Artificial Neural Networks**

Zásady pro vypracování:

1. Seznamte se se způsoby programové detekce obličejů, případně obecně libovolných charakteristických tvarů ze statického i dynamického obrazu v reálném čase.
2. Proveďte možnosti a efektivitu programové detekce obličejů.
3. Využijte umělé neuronové sítě pro detekci.
4. Vytvořte multiplatformní aplikaci, která bude detekovat obličeje z obrazových dat a vyřešte kalibraci nastavení jejích algoritmů.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. ZELINKA, Ivan.: Umělá inteligence I. Volume 1. Zlín: Vutium, Brno, 1998, 126 p. ISBN 80-214-1163-5.
2. ŠNOREK M., JIŘINA M.: Neuronové sítě a neuropočítače, ČVUT, 1996, ISBN 80-01-01455-X.
3. BÍLA J.: Umělá inteligence a neuronové sítě v aplikacích, ČVUT, 1996, ISBN 80-01-01275-1.
4. BOSE N.K., LIANG P.: Neural Network Fundamentals with Graphs, Algorithms, and Applications, McGraw-Hill Series in Electrical and Computer Engineering, 1996, ISBN 0-07-006618-3.
5. ŠÍMA J., NERUDA R.: Teoretické otázky neuronových sítí. Vyd. 1. Praha, 1996: Matfyzpress, 390 s., ISBN 80-85863-18-9
6. BURGER W., BURGE M.: Digital Image Processing: An Algorithmic Introduction using Java, Springer Science and Business Media, 2008, ISBN 978-1-84628-379-6.
7. SCHILDT H.: Java The Complete Reference, 8th Edition, McGraw-Hill, 2011, ISBN 978-0-07-160630-1

Vedoucí diplomové práce:

doc. Ing. Zuzana Komínková Oplatková, Ph.D.
Ústav informatiky a umělé inteligence

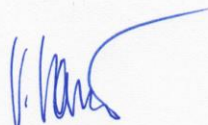
Datum zadání diplomové práce:

21. února 2014

Termín odevzdání diplomové práce:

20. května 2014

Ve Zlíně dne 21. února 2014


prof. Ing. Vladimír Vašek, CSc.
děkan




doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl jsem seznámen s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

ABSTRAKT

Tato diplomová práce popisuje v teoretické části problematiku umělých neuronových sítí a jejich adaptaci pro řešení problému počítačového vidění. Zabývá se také optimalizací obrazu a metodami heuristické segmentace.

Praktická část popisuje tvorbu aplikace pro detekci obličejů z obrazu v jazyce JavaScript, použité technologie, jednotlivé algoritmy a jejich účinnost a výpočetní náročnost.

Klíčová slova: Umělá inteligence, konvoluční neuronová síť, počítačové vidění, detekce obličejů, Neocognitron, LeNet, JavaScript.

ABSTRACT

Theoretical part of this master's thesis describes the use of artificial neural networks and their adaptation for computer vision implementation. It also discusses image optimization and heuristic segmentation.

The practical part describes the creation of application for face detection from an image in JavaScript language, the technology and algorithms used and their efficiency and computational complexity.

Keywords: Artificial intelligence, convolutional neural network, computer vision, face detection, Neocognitron, LeNet, JavaScript.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Děkuji své vedoucí doc. Ing. Zuzaně Komínkové Oplatkové, Ph.D. za odbornou pomoc a asistenci při tvorbě této práce.

OBSAH

ÚVOD	9
I TEORETICKÁ ČÁST	10
1 NEURONOVÉ SÍTĚ	11
1.1 ÚVOD DO NEURONOVÝCH SÍTÍ	11
1.1.1 Biologické neuronové sítě.....	11
1.1.2 Umělé neuronové sítě.....	12
1.2 USPOŘÁDÁNÍ UMĚLÝCH NEURONOVÝCH SÍTÍ	13
1.2.1 Paměť neuronové sítě	13
1.2.2 Stavové a nestavové sítě.....	14
1.2.3 Učení neuronových sítí.....	14
2 IDENTIFIKACE A KLASIFIKACE OBRAZOVÝCH RASTRŮ NEURONOVOU SÍTÍ	15
2.1 ZPRACOVÁNÍ OBRAZOVÝCH VJEMŮ LIDSKÝM ZRAKEM.....	15
2.2 HLUBOKÁ UMĚLÁ NEURONOVÁ SÍŤ	16
2.2.1 Konvoluční neuronová síť	16
2.3 COGNITRON A NEOCOGNITRON.....	17
2.3.1 Učení sítě Neocognitron.....	21
2.4 KONVOLUČNÍ SÍŤ LENET	21
2.5 KLASIFIKACE VĚTŠÍHO POČTU OBJEKTŮ.....	23
3 BAREVNÁ A JASOVÁ ANALÝZA A OPTIMALIZACE OBRAZU.....	24
3.1 BIOLOGICKÝ ASPEKT VNÍMÁNÍ JASU A BAREVNOSTI.....	24
3.2 BAREVNÉ MODELY SE ZÁVISLOU A NEZÁVISLOU SLOŽKOU INTENZITY JASU	25
3.3 LIDSKÉ PLEŤOVÉ BARVY	26
3.4 NORMALIZACE OBRAZU FREKVENČNÍ ANALÝZOU SPEKTRA JASU	28
3.5 IZOLACE HOMOGENNÍCH OBLASTÍ ALGORITMEM FLOOD FILL.....	28
II PRAKTICKÁ ČÁST	30
4 KLÍČOVÉ TECHNOLOGIE APLIKACE.....	31
4.1 WORLD WIDE WEB (WWW).....	32
4.2 HYPERTEXTOVÝ ZNAČKOVACÍ JAZYK (HTML) A KASKÁDOVÉ STYLY (CSS)	32
4.3 ECMAScript A JavaScript V PROSTŘEDÍ WEBOVÉHO PROHLÍŽEČE.....	32
4.4 INICIATIVA HTML5	33
4.4.1 Objekt Canvas a nízko-úrovňová manipulace s obrazovými daty.....	33
4.4.2 Nativní přístup k multimédiím a systému souborů	34
5 JEDNOTLIVÉ FÁZE OBRAZOVÉ ANALÝZY APLIKACE	35
5.1 UČENÍ NEURONOVÉ SÍTĚ TRÉNOVACÍ MNOŽINOU OBRAZŮ	36
5.2 NALEZENÍ OPTIMÁLNÍCH POČÁTEČNÍCH PODMÍNEK	38
5.3 JASOVÁ A BAREVNÁ OPTIMALIZACE	39
5.4 FILTR PLEŤOVÝCH BAREV.....	39
5.5 IZOLACE A SELEKCE HOMOGENNÍCH PLOCH	40
5.6 KLASIFIKACE VYBRANÝCH OBLASTÍ NEURONOVOU SÍTÍ.....	40
6 ZHODNOCENÍ VÝPOČETNÍ NÁROČNOSTI A TECHNOLOGICKÝCH OMEZENÍ.....	43

6.1	SROVNÁNÍ VÝPOČETNÍ NÁROČNOSTI JEDNOTLIVÝCH OPERACÍ	43
6.2	SROVNÁNÍ VÝKONNOSTI BĚŽNÝCH WEBOVÝCH PROHLÍŽEČŮ	45
6.3	Hlavní úskalí analýzy monoskopického počítačového vidění	46
ZÁVĚR.....		47
SEZNAM POUŽITÉ LITERATURY		49
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....		51
SEZNAM OBRÁZKŮ		52
SEZNAM PŘÍLOH.....		53

ÚVOD

Prvky umělé inteligence, umožňující strojům řešit reálné komplexní problémy, jsou dnes klíčovým faktorem akademického i komerčního vývoje informačních technologií.

Ukazuje se, že, pokud jde o umělou inteligenci, většina řešení vzešlých z přirozené evoluce je mnohdy překvapivě optimálních – co se kvality i výkonosti týče. Biologicky inspirované algoritmy tak dnes často tvoří páteř systémů vykazujících inteligentní chování nebo pracujících s daty z okolního světa.

Zrak je dominantní lidský smysl. Až 80% veškerých okolních informací vnímá člověk právě svým zrakem. Doplnit informační systémy o stejný druh datového vstupu je tedy pochopitelná vývojová adaptace – především v případě, kdy systémy musí samy interagovat s okolním světem. Hlavní úkol takzvaného *počítačového vidění* je tedy stejný, jako úkol zrakového centra mozkové kůry – identifikovat a rozeznat objekty na základě intenzity a vlnové délky od nich odraženého světla.

Existuje celá řada přístupů, jak řešit obrazovou analýzu. Jeden z nich se inspiruje biologií člověka a využívá k řešení stejného problému stejný mechanismus – síť neuronů. Konvoluční neuronové sítě, těsně inspirované uspořádáním sítnice a zrakového mozkového centra člověka a některých savců, dnes vykazují nejlepší výsledky při klasifikaci objektů v rastrovém dvourozměrném poli.

Už ze samotného principu věcí, kdy člověk je typický objekt, se kterým stroj nebo informační systém běžně interaguje, je jednou z klasických aplikací *počítačového vidění* rozpoznávání lidí. Pro drtivou většinu účelů, ať už jde o identifikaci, autentizaci nebo rozpoznání emocí, je z celého lidského těla obličej klíčovým zdrojem informací. Detekce obličejů z obrazu je tak první a nezbytný krok jakékoliv další analýzy.

Tato práce tedy za svůj cíl volí detekci obličejů z obrazu, přestože postup, ekvivalentní ke zde popisovanému, zvládne s drobnými změnami detekovat a klasifikovat jakékoliv objekty, které jsou charakteristické svým tvarem a rozsahem barev.

A protože web, respektive Internet, je nejdůležitější informační médium současného světa, použije i aplikace pro detekci obličejů jeho přirozených prostředků – webového prohlížeče a jeho nativních schopností a technologií, které dosáhly úrovně, kdy už je takový cíl možné realizovat.

I. TEORETICKÁ ČÁST

1 NEURONOVÉ SÍTĚ

1.1 Úvod do neuronových sítí

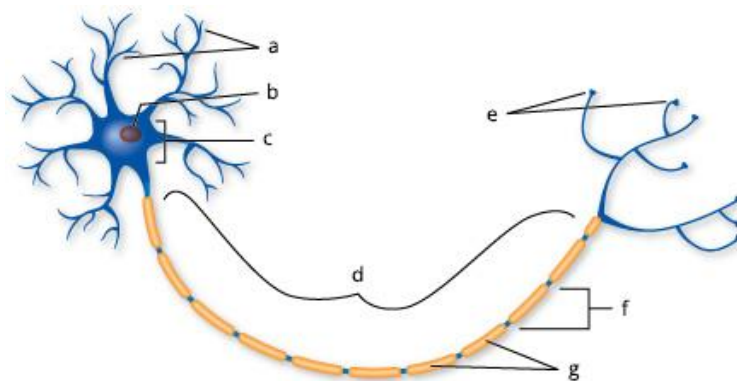
Ve většině aplikací reálného světa musí počítače respektive počítačové programy řešit problémy týkající se identifikace a analýzy komplexních datových vzorů. Protože konvenční algoritmy nejsou příliš vhodné pro řešení takových úloh, propůjčily si moderní informační technologie určité vlastnosti z fyziologie lidského mozku.

1.1.1 Biologické neuronové sítě

Umělé neurony a neuronové sítě napodobují některé klíčové atributy svých biologických protějšků.

Obrázek 1 znázorňuje přibližnou podobu lidského neuronu. Jeho základní část je tělo, jehož buněčný obal funguje jako sodno-draselná pumpa, která propouští sodíkové ionty vně a draslíkové ionty dovnitř buňky a udržuje tak rozdíl elektrického potenciálu buňky a okolí na přibližně 100 mV.

Okolí těla buňky tvoří výběžky zvané dendrity, plnící roli signálových vstupů, a jeden výstup – axon, sloužící jako vodič akčního potenciálu. Vzhledem k obecně špatné elektrické vodivosti nervových vláken, je signál opakovaně zesilován pomocí Myelinových buněk respektive Ranvierových zářezů a přiveden na jednotlivá zakončení axonů do případných synaptických spojení. [1]

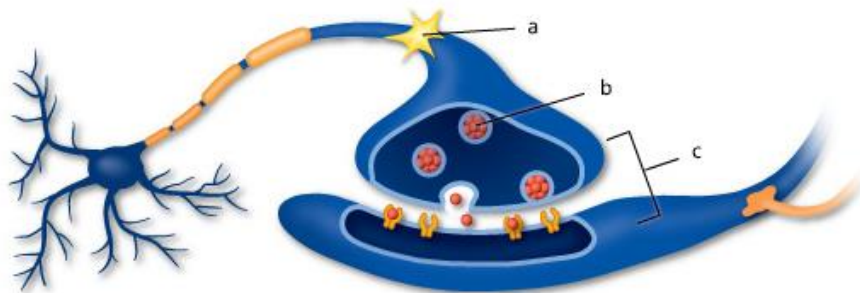


Obrázek 1: Lidský neuron

*Dendrity (a), jádro (b), tělo neuronu (c), axon (d), zakončení axonu (e),
Ranvierovy zářezy (f), Myelinové buňky (g) – (zdroj wakeneurology.com)*

Proces aktivace neuronu začíná excitací potenciálu neuronu v místě synapsí připojených na dendrity nad určitou mez, které způsobí vyslání akčního potenciálu axonem směrem k axonálním zakončením.

Obrázek 2 ukazuje synaptické spojení mezi axonálním zakončením a dendritem jiného neuronu. Akční potenciál vzniklý aktivací neuronu zvýší tok vápníkových iontů, depolarizaci potenciálu, uvolnění membrány a vypuštění chemických neurotransmiterů. [1]



Obrázek 2: Synapse

Akční potenciál (a), neurotransmiter (b), synapse (c) – (zdroj wakeneurology.com)

V závislosti na nastavení a druhu synaptického spojení respektive k množství uvolněného neurotransmiteru je pak potenciál cílového neuronu zvýšen nebo snížen. V případě, že zvýšení potenciálu překročí určitou mez, dojde k aktivaci neuronu a signál se přesouvá do další části sítě.

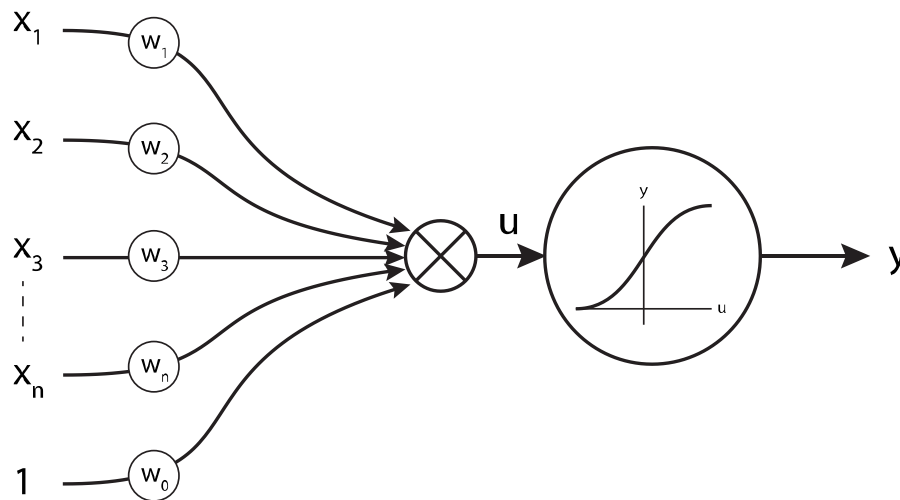
Potenciálová aktivační mez, množství neurotransmiteru i nastavení synaptických spojení se v mozku opakovaným používáním mění – síť je tak schopná optimalizovat svou funkci – učit se. [1]

1.1.2 Umělé neuronové sítě

Umělé neurony a neuronové sítě v základních rysech napodobují uspořádání a chování biologických neuronů.

Obrázek 3, umělý neuron, obsahuje obdobné části jako jeho biologický protějšek. Dendritické synapse jsou nahrazeny vstupními veličinami. Jak budou ovlivňovat akční potenciál neuronu, pak určují váhy jednotlivých vstupů. Aktivační potenciálový práh reprezentuje aktivační funkce se sumou vstupních potenciálů jako nezávislou proměnnou a výstupem jako proměnnou závislou.

Mezi vstupy může být také posun (*bias*) s konstantní hodnotou a vlastní vahou.



Obrázek 3: Umělý neuron – Perceptron

Vstupy (x), váhy vstupů (w), vstup (u) a výstup (y) aktivační funkce

Váhy a podoba aktivační funkce tedy reprezentují nastavení neuronu. Podobně jako u lidského mozku, aby neuronová síť mohla správně fungovat, je nutné optimalizovat nastavení jednotlivých neuronů – naučit síť reagovat na podněty potřebným způsobem.

1.2 Uspořádání umělých neuronových sítí

Samotný neuron nebo skupina neuronů pochopitelně už z principu své jednoduché podoby nemůže řešit jiné než lineárně separabilní problémy. Pokročilá praktická činnost neuronové sítě tedy vyžaduje propojení neuronů do více vrstev. V závislosti na typu sítě mohou být, vzhledem ke schopnosti synapse akční potenciál zvyšovat i snižovat, tvořeny komplexní dopředné i zpětné vazby.

1.2.1 Paměť neuronové sítě

Dvě hlavní skupiny architektur neuronových sítí lze rozdělit podle typu paměťové asociace, kterou reprezentují.

Autoasociativní síť má shodný počet vstupů i výstupů a má obecnou vlastnost rekonstruovat vstupní data podle nejbližšího naučeného vzoru. To ji předurčuje k použití k rekonstrukci zkreslených dat nebo přibližnému řešení NPC problémů. Typickým představitelem je *Hopfieldova síť*. [2]

Heteroasociativní síť má menší počet výstupů a principiálně má tendenci identifikovat a přiřadit vstupní data k jednomu z naučených vzorů. Na rozdíl od asociativní sítě, může heteroasociativní paměť přiřadit vstup odlišnému výstupu, je tak ideální pro rozpoznávání nebo třídění datových vzorů. Typickými představiteli jsou *vícevrstvý perceptron*, *síť BAM* nebo *Kohonenova síť*. [2]

1.2.2 Stavové a nestavové sítě

Podle přítomnosti nebo absence zpětných vazeb rozlišujeme dvě základní architektury neuronové sítě.

Dopředná (Feedforward) architektura neobsahuje žádné zpětné vazby z výstupních do vstupních neuronů, předchozí stavy sítě tak neovlivňují následující. Typickými představiteli jsou všechny perceptronové sítě. [2]

Zpětnovazební (Feedback) architektura obsahuje zpětné vazby a výstup tak závisí nejen na aktuálních vstupních datech, ale i na předchozím stavu sítě. Typickým představitelem je *Hopfieldova síť*. [2]

1.2.3 Učení neuronových sítí

Jednou z nejdůležitějších vlastností neuronových sítí je schopnost učit se – přizpůsobovat svou vnitřní konfiguraci, aby vstupním datům odpovídaly požadované (nebo alespoň konzistentní) výstupní data.

K učení sítě slouží takzvaný *trénovací algoritmus* využívající *trénovací množinu vstupních dat*. Rozlišujeme trojici trénovacích algoritmů.

Učení s učitelem probíhá iteračním přibližováním požadovanému výsledku vždy po dvojicích kroků – aktivace a adaptace. Každý cyklus (epocha) vyhodnocuje odchylku od požadovaného výstupu (chybu) a upraví váhy sítě pro její zmenšení typicky postupně od výstupní vrstvy zpětně ke vstupní vrstvě. Tato metoda se označuje jako *Backpropagation*. [2]

Učení bez učitele naopak využívá jen vstupních dat bez jakékoliv klasifikace. Proces učení hledá charakteristické rysy jednotlivých prvků a optimalizuje váhy sítě tak, aby konzistentní vstupy poskytovaly konzistentní výstupy. [2]

Kombinací předchozího pak může být i *Zpětnovazební učení*, kdy jsou sledovány výstupy sítě a vhodné výsledky jsou schvalovány a nevhodné zamítány. V závislosti na tom jsou pak posilovány nebo oslabovány váhy patřičných spojení. [2]

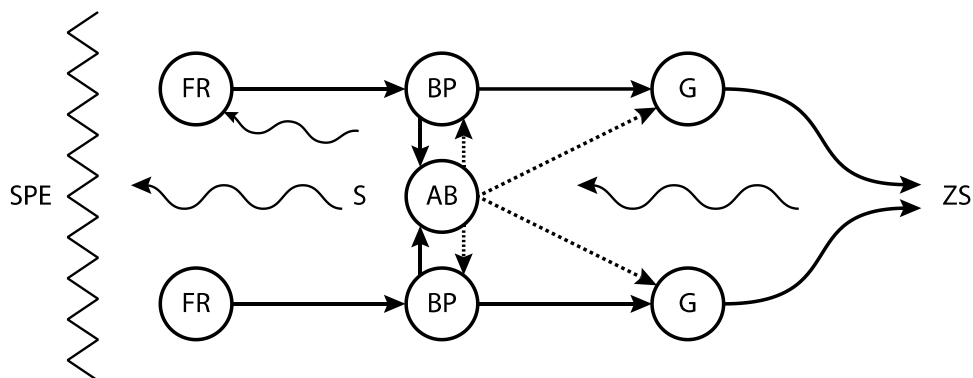
2 IDENTIFIKACE A KLASIFIKACE OBRAZOVÝCH RASTRŮ NEURONOVOU SÍŤÍ

Jedno z klasických využití neuronových sítí je pro zajištění takzvaného *počítačového vidění*. To zahrnuje především identifikaci a klasifikaci objektů z obrazových rastrů.

2.1 Zpracování obrazových vjemů lidským zrakem

Velká část moderních implementací neuronových sítí pro analýzu obrazu se přímo nebo přibližně inspiruje dispozicemi vzešlými z evoluce zrakového smyslu u člověka nebo obratlovců obecně.

Obrázek 4 zjednodušeně znázorňuje neuronovou strukturu sítnice lidského oka.



Obrázek 4: Neuronová struktura sítnice lidského oka
Fotoreceptory (FR), bipolární (BP) a amakrinní (AB) buňky, gangliony (G),
zrakový svazek (ZS), síťnicový pigmentový epitel (SPE) a světlo (S).

Dopadající světlo excituje fotoreceptory absorpcí fotonů pigmentem *rodopsinem*. Nezachycené světlo pak dopadne na síťnicový pigmentový epitel, na kterém, na rozdíl od některých jiných obratlovců, není odraženo zpět, ale pohlceno. Člověk tak evolučně zvolil nižší citlivost, ale vyšší přesnost zraku. [3] [4]

Z fotorecepčního neuronu se akční potenciál šíří dále excitačním glutaminovým synaptickým spojením do bipolárních buněk, zlepšujících kontrast a do amakrinních buněk, snižujících, inhibičními GABA synapsi, šum zrakového signálu. [3] [4]

Hlavní část předzpracování zrakových vjemů na sítnici nakonec provádí gangliony. Ty agregují signál z většího počtu sousedních bipolárních a amakrinních buněk a podle svého typu reagují na lokální hrany obrazu a detekují jednoduché tvary. Agregace signálu zde způsobuje i

významnou kompresi, kdy neuronová síť sítnice reaguje především na klíčové prvky obrazu a informace o souvislých plochách potlačuje. Tohoto faktu také využívá většina optických klamů. [3] [4]

Prodloužené axony ganglionů pak tvoří zrakový svazek, spojení se zrakovým centrem v mozkové kůře, často nepřesně označované jako zrakový nerv.

Důvod, proč je sítnice obratlovců uspořádána inverzně vzhledem k dopadajícímu světlu, tedy s fotoreceptory pod vlastní neuronovou infrastrukturou, není zatím spolehlivě objasněn. Jedna z možných příčin je, že v rané fázi vývoje plodu jsou obě sítnice přímou součástí mozku. [5]

2.2 Hluboká umělá neuronová síť

Hierarchické uspořádání biologických neuronových sítí, s postupnou degradací komplexity vstupních dat až na úroveň lineární separability, inspirovalo celou řadu umělých neuronových struktur.

Hluboká síť je obecně pravděpodobnostní model sestavený z několika vrstev skrytých stochastických prvků, někdy označovaných jako *detektory vlastností*, které jsou schopny reagovat na naučenou vlastnost a zbytek dat delegovat dále do nižších vrstev. V případě neuronových sítí jsou prvky typicky neurony nebo skupiny neuronů, které na danou vlastnost reagují změnou potenciálu.

Klíčový problém u dopředných hlubokých sítí představuje jejich učení, kdy u většího počtu vrstev významně roste výpočetní náročnost metodiky *backpropagation* a projevují se nedostatky ve správném šíření chyb, kdy má síť tendenci uváznout v sub-optimálním stavu při takzvaném *zmizení gradientu* chybové funkce. [6]

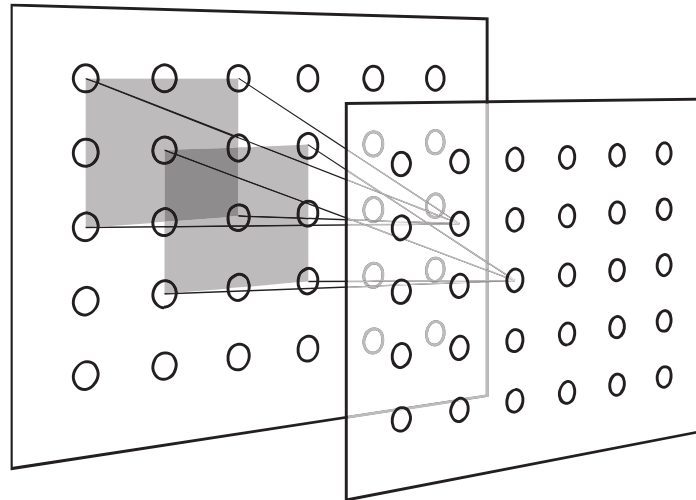
Jedno z řešení tohoto problému představuje použití dvoufázového učení. V první fázi je využito dopředného učení *bez učitele*, postupujícího sítí vrstvu po vrstvě, kdy jsou posilovány váhy nejvíce reagujících neuronů. Druhá fáze má pak za cíl kompenzovat *hladovost* předchozí optimalizace jemnějším doladěním vah klasickou metodou *backpropagation*. [7]

2.2.1 Konvoluční neuronová síť

Při zpracování vizuálních vjemů neuronovou sítí je velmi často využito konvolučního uspořádání prvků uvnitř jednotlivých vrstev. Konvoluční síť na rozdíl od konvenční sítě neignoruje topologii vstupujících dat a bere v potaz jejich dvourozměrný charakter.

Obrázek 5 znázorňuje typické uspořádání konvoluční sítě, kdy jednotlivé jednotky zpracovávají výstupy z blízkého okolí nadřazené vrstvy v oblastech, které se navzájem

překrývají. Jednotlivé neurony uvnitř každé z vrstev také typicky sdílejí stejnou sestavu vstupních vah. [8]



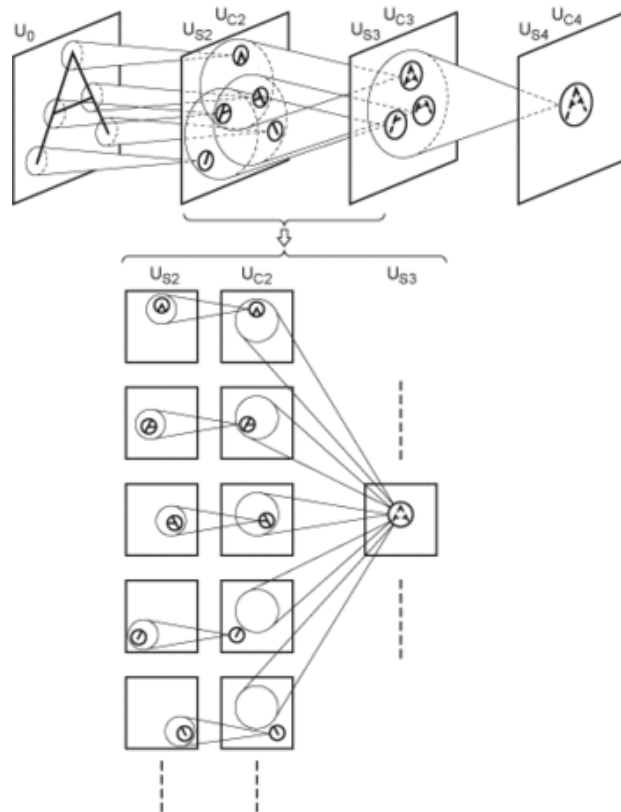
Obrázek 5: Ukázka konvolučního uspořádání synapsí vrstev neuronové sítě

Síť je takto částečně schopna kompenzovat posuny a deformace obrazu tím, že neurony mohou být aktivovány naučeným vzorem z jiné oblasti, než byla původní pozice vzoru.

Konektivita konvoluční sítě však nesmí být příliš hustá, aby byla degradace topologie dat postupná a síť si zachovala potřebné rozlišovací schopnosti. [7]

2.3 Cognitron a neocognitron

Cognitron a *Neocognitron* jsou architektury neuronových sítí specializované pro identifikaci obrazových objektů. Využívají specializované jednotky rozpoznávající menší části obrazu a výsledky jsou pak agregovány v následujících vrstvách sítě až do finální klasifikace objektu. Obrázek 6 zjednodušeně ukazuje tuto sekvenci na písmenu A. Proces se tak podobá zpracování vjemů ze sítnice a ve zrakové části mozku, kdy jsou přednostně identifikovány charakteristické části obrazu jako hrany nebo spojnice a zbylé jednodušší plochy jsou zanedbány.

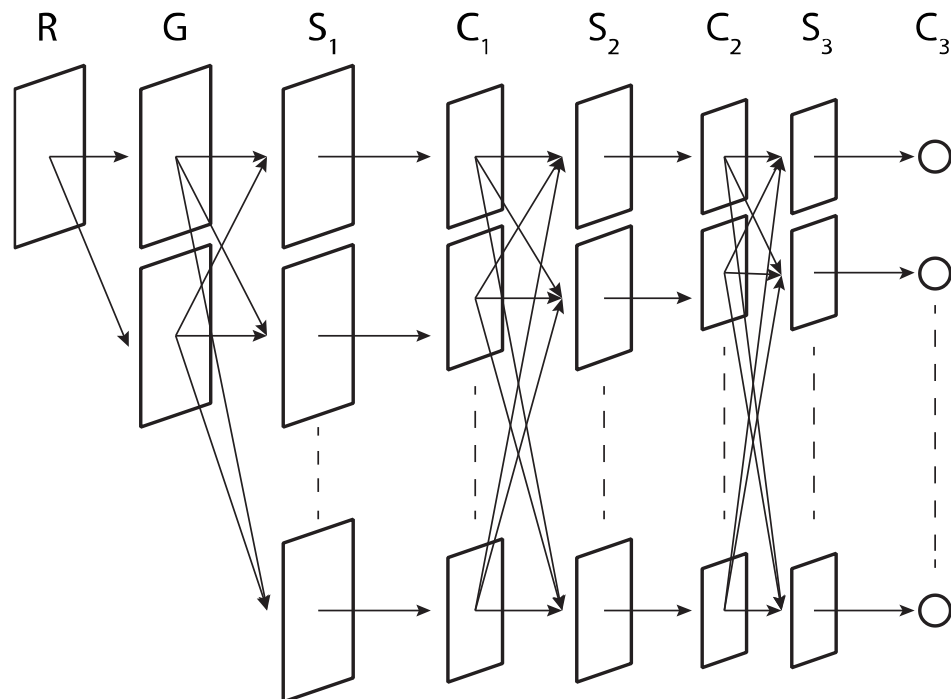


Obrázek 6: Ukázka rozpoznání písmene A
síti neocognitron (zdroj [9])

Hierarchie sítě je složena ze střídajících se takzvaných *jednoduchých* a *komplexních* vrstev. Toto uspořádání napodobuje jednoduché a komplexní buňky zrakové mozkové kůry. Každá vrstva je dále rozdělena do takzvaných *rovin*, které odpovídají rozlišovací schopnosti sítě v dané vrstvě – reprezentují její asociační paměť.

Samotné skupině vrstev předchází ještě vrstva zvýrazňující hrany na tmavém a světlém podkladě. Je zde opět patrná inspirace z „*on-center*“ a „*off-center*“ ganglionů lidské sítnice.

Obrázek 7 ukazuje celkovou strukturu sítě se třemi jednoduchými a komplexními vrstvami. Poslední vrstva plní klasifikační účel.

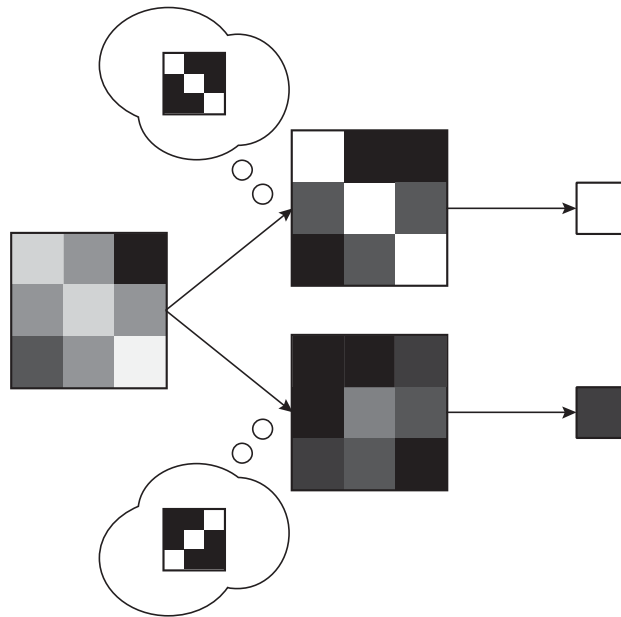


Obrázek 7: Struktura vrstev sítě Neocognitron
 Vstupní vrstva – sítnice (R), nalezení hran On/Off-Center (G),
 jednoduchá (S) a komplexní (C) vrstva.

Neurony *jednoduchých* vrstev reagují na naučené vzory v nadřazené komplexní vrstvě ve svém blízkém okolí, označovaném také jako *vnímaná oblast*. Oblasti mají konvoluční uspořádání, překrývají se. Váhy každého neuronu jsou v rámci dané roviny *jednoduché vrstvy* shodné, vzor tedy může být nalezen nezávisle na umístění nebo počtu výskytů.

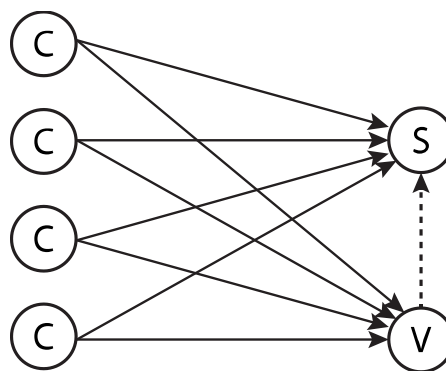
Neurony *komplexní* vrstvy agregují signály z neuronů jednoduchých vrstev opět v určitém blízkém okolí. Jejich počet je na rozdíl od jednoduché vrstvy již menší, dochází ke zmenšení komplexnosti, agregaci charakteristických prvků obrazu a jeho převzorkování. Váhy vstupů neuronů komplexních vrstev jsou neměnné, neurony těchto vrstev se tedy na samotné klasifikaci nepodílejí. [9]

Obrázek 8 zjednodušeně ilustruje rozpoznání jedné z dvojice naučených vzorů jednoduchou vrstvou a následnou agregaci výsledku vrstvou komplexní.



Obrázek 8: Jednoduché a komplexní vrstvy při rozpoznávání tvarů

Obrázek 9 dále znázorňuje důležitý prvek na vstupní části neuronů *jednoduché vrstvy* – *inhibitor*. Ten má stejné vstupy jako přidružený neuron *jednoduché vrstvy*, výstup je pak průměr těchto hodnot. Synapse je inhibiční, zajišťující deaktivaci připojeného neuronu, který nemá na vstupu vzor s dostatečně velkou vahou – tedy vzor, který je neuron naučen rozpoznávat. Takto je zajištěn průchod pouze relevantních dat vrstvami sítě.



Obrázek 9: Inhibitor neuronu jednoduché vrstvy
Neurony komplexní (C) a jednoduché (S) vrstvy
s připojeným inhibičním neuronem (V)

Díky postupné agregaci rozpoznávaných vzorů je síť schopna identifikovat naučené vzory i v částečně deformovaném obrazu. [9]

2.3.1 Učení sítě Neocognitron

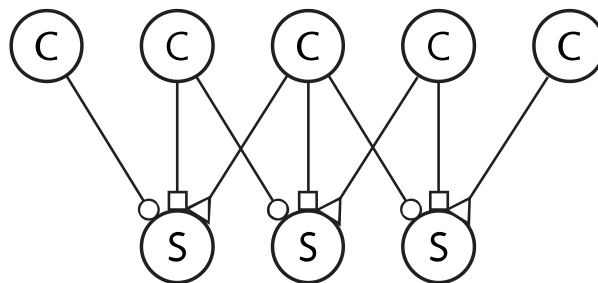
Základní varianta *Neocognitionu* využívá dopředného učení *bez učitele*, kdy se síť sama organizuje průchodem obrazu z *trénovací množiny* dat. Proces upravuje pouze váhy neuronů v *jednoduchých* vrstvách, neurony *komplexních* vrstev slouží pouze k agregaci nalezených vzorů. Roviny *jednoduchých* vrstev sdílejí vždy stejnou sadu vah u všech neuronů, vrstva tak může identifikovat vzor bez ohledu na jeho pozici a počet výskytů.

Počáteční hodnoty vah jsou voleny velmi nízké, průchodem jednotlivých obrazů jsou pak posilovány způsobem „*vítěz bere vše*“, tedy jen u nejméně reagující synapse. Váhy spojení od inhibitorů jsou vždy upraveny na průměrnou hodnotu vah zbylých excitačních synapsí. [9]

Důležitý aspekt učení neuronů *jednoduchých* vrstev je jejich postupná diferenciací – učením neurony postupně ztrácí schopnost reagovat na odlišné podněty a dochází tak k samovolné dělbě práce jednotlivých rovin uvnitř vrstev. [7]

Posilování synapsí také dodržuje určitou prostorovou symetrii, kdy jsou posilovány nejen váhy dotčeného neuronu, ale také jeho sousedů ve shodné prostorové distribuci. To dále rozšiřuje odolnost sítě vůči deformacím obrazu. [7]

Obrázek 10 ukazuje prostorovou symetrii synapsí mezi jednoduchými a komplexními neurony.



Obrázek 10: Prostorová symetrie synapsí

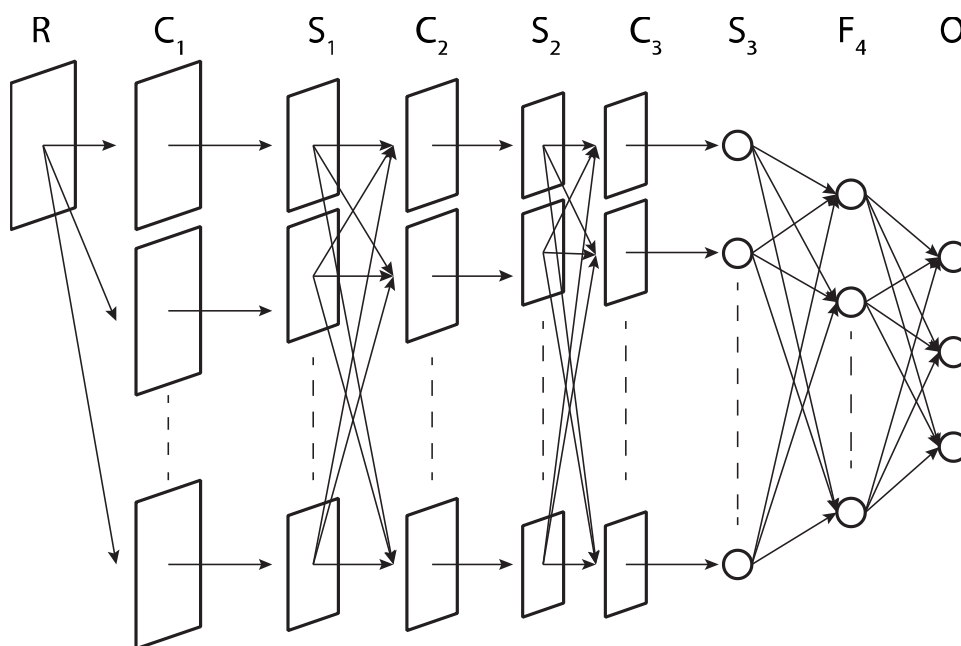
2.4 Konvoluční síť LeNet

Jednou z pozdějších adaptací Neocognitronu je konvoluční síť *LeNet* od francouzského vědce Yanna LeCuna.

LeNet je velmi podobný architektuře Neocognitronu: extrakce vlastností je prováděna analýzou obrazu neurony s lokální *vnímanou oblastí*, produkující konvoluční mapu, která je poté rozměrově redukována vzorkovací vrstvou do hlubších částí sítě pro extrakci vlastností

vyššího řádu. Ekvivalentem *jednoduchých* a *komplexních* vrstev jsou zde *konvoluční* a *vzorkovací* vrstvy.

Jak ukazuje obrázek 11, LeNet přidává oproti Neocognitronu na konec sítě konvenční, hustě propojenou, dvojici vrstev neuronů pro konečnou klasifikaci, využívajících pro svou aktivaci radiální bázové funkce. Ta určuje Eukleidovskou vzdálenost vstupů oproti naučeným vzorům. [8]



Obrázek 11: Struktura sítě LeNet

Vstupní vrstva – sítnice (R), konvoluční (C),
vzorkovací (S) a plně propojená (F) vrstva a výstupní klasifikace (O)

Druhý důležitý rozdíl je způsob učení – zatímco Neocognitron se při učení organizuje sám, LeNet používá *backpropagation*, konkrétně *stochastický sestup gradientem* chybové funkce s bezprostřední korekcí vah jednotlivých neuronů. Řízené učení, s *učitelem*, tak umožňuje síti přesnější klasifikaci nalezených vzorů. [8]

Nevýhoda učení *backpropagation* u hlubokých sítí byla již zmiňována. LeNet má ovšem, podobně jako ostatní konvoluční sítě, díky řídkému propojení konvolučních vrstev a váhové symetrii, mnohem menší počet stupňů volnosti než konvenční hustě propojené hluboké sítě. I tak je učení náročnější a vyžaduje větší trénovací množinu dat.

2.5 Klasifikace většího počtu objektů

Klasický problém počítačového vidění je rozlišení samostatných objektů, které nelze jednoduše oddělit od jejich okolí.

Běžně používané konvenční řešení je takzvaná *heuristická segmentace*, tedy rozdělení obrazu a výběr potenciálních oblastí heuristickou analýzou. Získané kandidáty pak postupně klasifikuje a hodnotí rozpoznávací jednotka.

Prostorová invariance konvolučních neuronových sítí ovšem nabízí i zajímavou možnost využít jedinou *prostorově dislokovanou neuronovou síť* pro přímou detekci a klasifikaci objektů z heterogenního obrazu. Takové řešení pak ale klade řádově vyšší nároky na konečnou klasifikaci, kdy neurony mohou agregovat reakce na charakteristické tvary nesouvisejících objektů a docházet tak k chybným závěrům. Klasifikační vrstvy tak v tomto případě obsahují často lexikální a kompoziční analyzátory, které jsou schopny falešně pozitivní výsledky eliminovat. [8]

Tato práce se přidrží, vzhledem k charakteristickému rozsahu barev i úzkému rozsahu orientací obličejů v obraze, konvenčních heuristických metod, které budou popsány blíže v následující části.

3 BAREVNÁ A JASOVÁ ANALÝZA A OPTIMALIZACE OBRAZU

Samotné klasifikaci obrazu v neuronové síti předchází takzvaná *segmentace*, tedy nalezení relevantních segmentů, které potenciálně mohou obsahovat právě jeden hledaný objekt.

V našem případě je segmentace dosaženo sérií *heuristických* metod, které mají za cíl normalizovat vlivy prostředí a separovat oblasti, které svým přibližným tvarem a barevným spektrem odpovídají hledanému objektu – obličej.

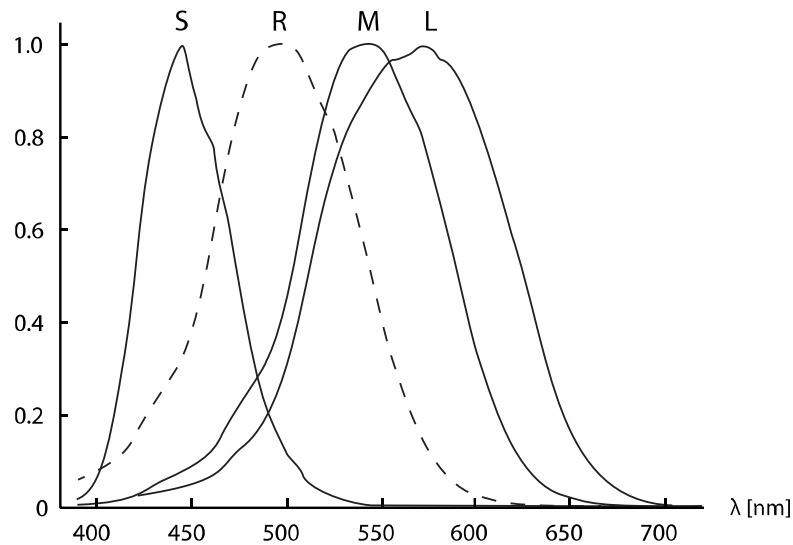
3.1 Biologický aspekt vnímání jasu a barevnosti

Jakým způsobem zpracovávají neurony lidské sítnice obrazové vzory již bylo zmiňováno v kapitole 2.1. Lidský zrak je také schopen normalizovat zrakové vjemy kompenzací vnějších optických podmínek – je schopen chromatické a jasové adaptace.

Kapitola 2.1 ukazovala funkci neuronových klasifikátorů, ganglionů. Jeden z typů ganglionů má však vyhrazený jiný účel, obsahuje pigment *melanopsin* a může tak být aktivován vnějším absorbovaným světlem, podobně jako fotoreceptory sítnice. Na rozdíl od nich ale fotosenzorický ganglion aktivuje papilární reflexní oblouk, ovládající svaly zornice. Oko je takto schopno samo regulovat množství světla dopadajícího na sítnici bez nutnosti zapojení vyšších mozkových funkcí. [4]

Lidský zrak disponuje takzvaným *trichromatickým* viděním, kdy jsou fotoreceptory diferenciovány na evolučně starší *tyčinky* s velkým rozsahem citlivosti a pomalou reakční dobou a evolučně mladší *čípky* s rychlou reakční dobou, menší citlivostí a specifickým aktivačním rozsahem vlnové délky. Typická diferenciací čípků obratlovců na dvojici vlnových délek je u člověka a některých dalších savců doplněna i třetím typem pro střední vlnovou délku. Podrobněji to ukazuje obrázek 12. Vnímaná barva obrazu je pak iluze vytvořená zrakovým centrem mozkové kůry na základě intenzity potenciálů jednotlivých typů receptorů, která však umožňuje rozlišovat tvary nejen podle intenzity, ale i podle vlnové délky odraženého světla. [4]

Důležitou vlastností neuronů sítnice je v tomto případě schopnost inhibičními vazbami kompenzovat dlouhodobou zvýšenou saturaci receptorů určitého typu a korigovat tak například barvu osvětlení scény. Toho často využívají barevné optické klamy. [4]



Obrázek 12: Reaktivita fotoreceptorů sítnice na vlnovou délku světla

Čípky diferenciované na krátkou (S), střední (M) a dlouhou (L) vlnovou délku a tyčinky (R)

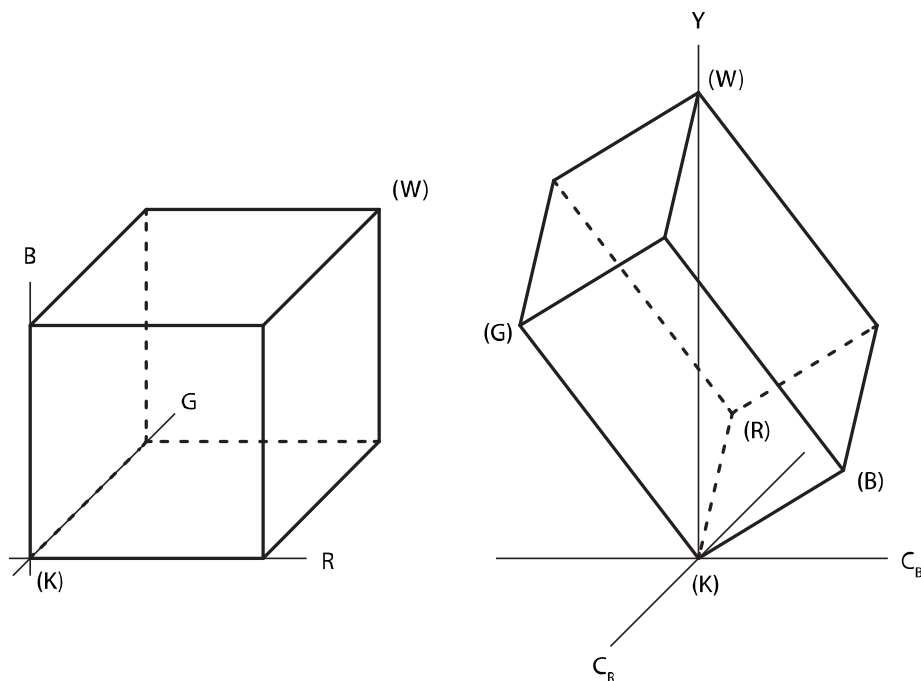
3.2 Barevné modely se závislou a nezávislou složkou intenzity jasu

Mezi klasické barevné modely počítačové a vizuální techniky patří aditivní model *RGB*, tedy červená, zelená a modrá intenzita. Kopíruje tak vrcholy citlivosti fotoreceptorů lidské sítnice pro krátkou, střední a dlouhou vlnovou délku.

Pro účely barevné analýzy je ovšem model *RGB* značně nevýhodný. Jeho jas je kontrolován všemi třemi složkami, trpí významnou redundancí barevných definic a pro potřeby výběru oblasti se vyznačuje nízkou prostorovou koherencí příbuzných barevných tónů.

Tato práce tedy volí pro barevnou selekci model $Y C_B C_R$ s nezávislou složkou jasu a složkami modré a červené chrominance vyjadřujících rozdíl oproti referenční hodnotě. Obrázek 13 srovnává modely *RGB* a $Y C_B C_R$.

Model vznikl pro potřeby přenosu obrazového signálu s nízkou mírou redundance a možností nezávisle komprimovat chromatické komponenty, na které je lidský zrak méně citlivý než na jas samotný. Nezávislá intenzita jasu také umožňuje velmi snadno lineárně separovat tmavé a světlé odstíny pleťových barev, jak blíže demonstruje obrázek 15 v kapitole 3.3.



Obrázek 13: Barevné modely RGB a $YC_B C_R$

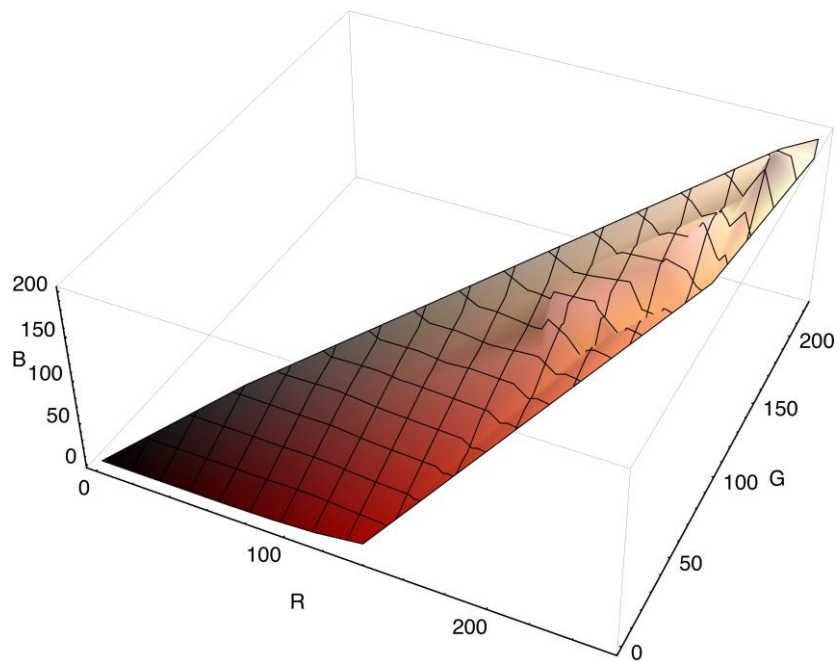
Červená (R), zelená (G), modrá (B), černá (K) a bílá (W) barva.
Složky jasu (Y), modrý (C_B) a červený (C_R) chrominanční komponent.

3.3 Lidské pleťové barvy

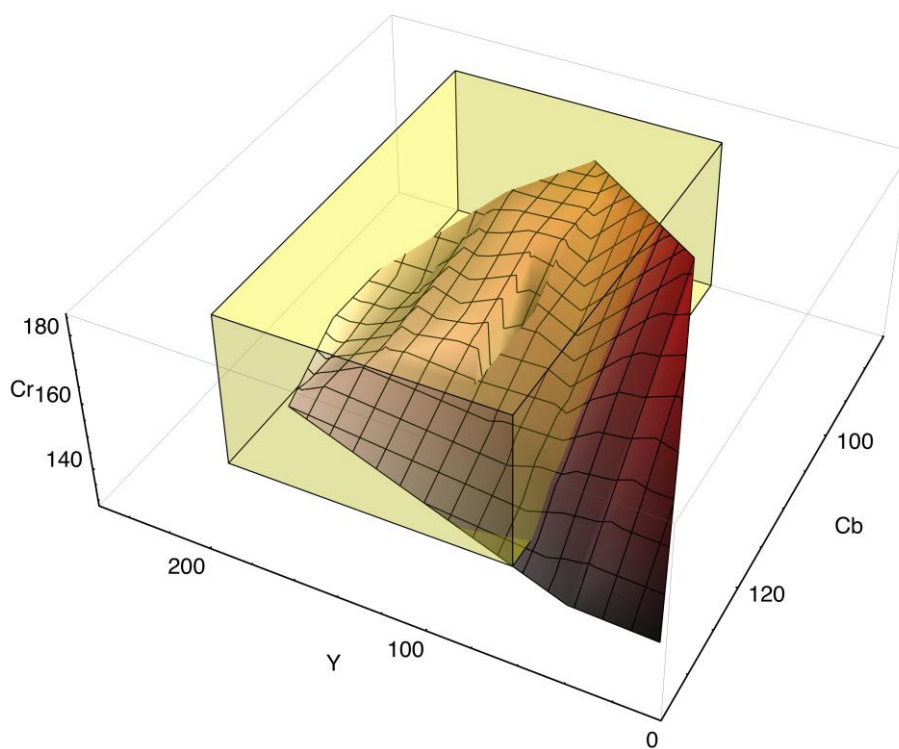
Barva lidské pleti je významně ovlivněna množstvím pigmentu *melaninu* především u populace s tmavou pokožkou. U světlejší pleti barvu určuje i cirkulující hemoglobin respektive prokrvení hlubších vrstev kůže.

Pro účely selekce pleťových barev bylo u této práce vybráno třicet pleťových odstínů vzorníku PANTONE® [10], které jsou znázorněny v prostoru RGB (obrázek 14) a $YC_B C_R$ (obrázek 15).

Vhodnost použití barevného modelu s nezávislou jasovou složkou byla zmiňována v kapitole 3.2. Pro následující analýzu jsou vybrány pleťové barvy s hodnotou jasu alespoň 80 bodů. Tmavší odstíny již kolidují s barvami jiných, nepleťových objektů. Separaci a analýzu velmi tmavých pleťových oblastí je vhodné provést odděleně s jinou sadou heuristických metod a ideálně i s odlišně trénovanou neuronovou sítí pro samotnou klasifikaci.



Obrázek 14: Plet'ové barvy Pantone® v prostoru RGB

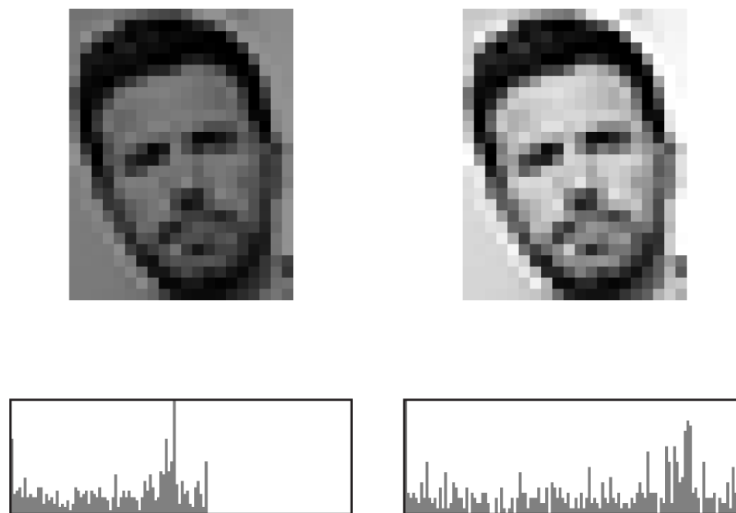


Obrázek 15: Plet'ové barvy Pantone® v prostoru $Y C_B C_R$
Kvádr znázorňuje použitý rozsah pro světlé a středně tmavé typy pleti

3.4 Normalizace obrazu frekvenční analýzou spektra jasu

V reálných podmínkách může být rozsah jasu obrazu snížen nesprávně zvolenými parametry expozice. Rozšířením dynamického rozsahu jasu je možné jednoduše kompenzovat takové zkreslení.

Obrázek 16 ilustruje korekci podexponovaného obrazu roztažením frekvenčního jasového spektra na plný rozsah.



Obrázek 16: Korekce podexpozice normalizací jasového histogramu

V případě barevného obrazu je vhodné před optimalizací separovat jasovou komponentu například převodem na barevný model HSL nebo $YCbCr$, protože normalizace samotných RGB kanálů by mohla způsobit nežádoucí zkreslení barev. [11]

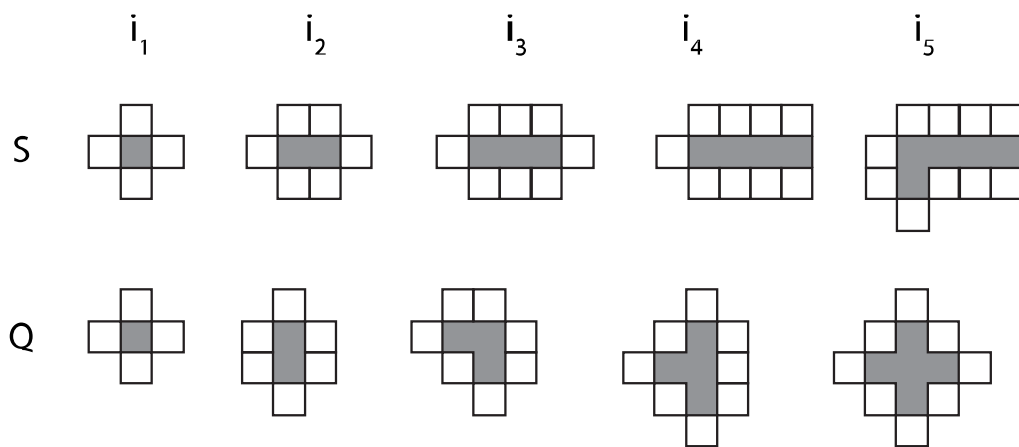
3.5 Izolace homogenních oblastí algoritmem Flood fill

Plochu vzešlou z barevné analýzy je třeba segmentovat na uzavřené oblasti. U každé ohraničené oblasti je dále vhodné určit její obsah, hrubý poměr stran a případně i orientaci. Tyto informace pak mohou posloužit k odstranění zjevně nevyhovujících oblastí z uvažované množiny testovacích dat – typicky příliš malých, velkých nebo s příliš malým poměrem stran.

Tuto úlohu velmi dobře řeší rekurzivní výplňový algoritmus, často označovaný jako *flood fill* – „rozlití výplně“.

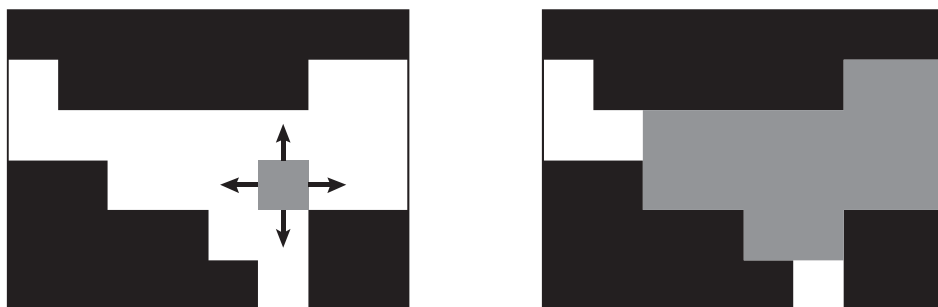
V našem případě je algoritmus opakován v určitém kroku napříč obrazem, vždy jeden cyklus za druhým. Každý cyklus vyplňuje plochu jinou hodnotou, výsledkem je matice hodnot, kdy shodné hodnoty tvoří vždy jednu ohraničenou plochu.

V závislosti na počtu rekurzí, který může být u větších ploch značný, je vhodné hierarchickou rekurzi se zásobníkem rozbalit do ploché iterační fronty. To podstatným způsobem sníží paměťovou náročnost algoritmu a změní pořadí iterací na přirozenější průběh, jak blíže ukazuje obrázek 17.



Obrázek 17: Algoritmus flood fill se zásobníkem (S) nebo frontou (Q)

Vzhledem k tomu, že v reálném obraze jsou jednotlivé objekty velmi často v blízkosti, případně se i překrývají, je vhodné algoritmus *flood fill* vylepšit o kontrolu okolí vyplňovaných buněk tak, aby toleroval jen dostatečně velké spojnice ploch a částečně tak kompenzoval případné nevýrazné hrany objektů obrazu. Obrázek 18 ukazuje algoritmus s tolerancí nejméně dvou pixelů pro pokračování rozlévání výplně. Reálný přírůstek tohoto parametru by byla zvýšená viskozita rozlévané barvy uzavírající drobné netěsnosti oblastí.



Obrázek 18: Algoritmus flood fill uzavírající příliš malé spojnice

II. PRAKTICKÁ ČÁST

4 KLÍČOVÉ TECHNOLOGIE APLIKACE

Detekci obličejů z obrazových dat implementuje tato práce v prostředí HTML a JavaScript, respektive pomocí prostředků a rozhraní obecného moderního prohlížeče. Využívá tak rozsáhlého vývoje, který tato skupina informačních technologií prodělala za posledních několik let a který webové technologie posunul do oblasti plnohodnotných aplikačních prostředí.

Mezi důvody, proč volit webovou platformu, patří:

- Velmi rychlý vývoj jejich prostředků a schopností díky velké exponovanosti a poptávce uživatelů.
- Téměř bezprecedentní přenositelnost mezi platformami i zařízeními. Webové technologie jsou dnes integrální součástí celé řady stolních i mobilních běhových prostředí.
- Decentralizovaný a z podstatné části licenčně nezatížený vývoj platformy.
- Bohatá rozšiřitelnost a interoperabilita s lokálními i vzdálenými aplikacemi a službami nativními prostředky nebo prostřednictvím zásuvných modulů.

Kromě výhod existují také určité nevýhody, plynoucí ze způsobu, jakým webové technologie fungují:

- Přes obecnou a dlouhodobou snahu optimalizace webových prostředí, mají webové technologie vždy větší režii, než nízko-úrovňové kompilované programovací jazyky přistupující bezprostředně k prostředkům systému.
- Webové aplikace pracují typicky uvnitř *sandboxu* svého běhového prostředí a jejich možnosti interagovat se systémem jsou tak velmi omezené. Nižší svoboda ovšem vždy znamená i nižší odpovědnost – z hlediska bezpečnosti a kontroly kvality lze tedy toto omezení chápat i pozitivně.
- Stejně jako u ostatních interpretovaných jazyků, jsou i v případě webových technologií zdrojové součásti programů k dispozici komukoliv z uživatelů. Nelze tedy spoléhat na jakoukoliv formu *bezpečnosti skrze utajení*.

4.1 World Wide Web (WWW)

World Wide Web, je souhrn propojených (často proto označovaných jako *hypertextových*) dokumentů v síti Internet, které jsou jednoznačně identifikovány pomocí identifikátorů *URI*. Ty typicky plní i funkci jednoznačných lokalizátorů, tedy *URL*, směřujících na konkrétní webové servery. [12]

Strukturu těchto dokumentů tvoří běžně, nikoliv však výhradně, značkovací jazyk *HTML*, doplněný často o vizuální definici stylů *CSS* a případně také o soubor programových instrukcí – nejčastěji v jazyce *JavaScript*.

Komunikaci s webovým serverem, grafické zobrazení dokumentů webu a interpretaci skriptů obstarává program – *webový prohlížeč*. Komunikace probíhá prostřednictvím protokolu *HTTP* nad protokolem *TCP* s případnými dalšími zabezpečenými vrstvami. [13]

4.2 Hypertextový značkovací jazyk (HTML) a kaskádové styly (CSS)

HTML je hlavním kódovacím jazykem webových dokumentů a aplikací. Je odvozen z obecnějšího *SGML* a jeho strukturu představuje obyčejný čistý text, jehož části jsou ohraničeny značkami, které jim dávají sémantický význam. [14]

Vzhledem k tomu, že *HTML* je jazyk *hypertextový*, umožňuje připojovat nebo odkazovat na další dokumenty a prostředky.

Prvotní verze *HTML* umožňovaly kromě sémantického významu definovat také vizuální podobu dat. S rozvojem grafických schopností prohlížečů se postupně prosadila oddělená definice vizuálních stylů v samostatném formátu *kaskádových stylů*.

Soubor *CSS* direktiv pak může být jak součástí *HTML* dokumentu, tak i jako dokument s vlastním *URI*. Charakteristickou vlastností *CSS* direktiv je jejich vzájemná *kaskádová dědičnost*, kdy jedna direktiva může doplňovat druhou. [15]

4.3 ECMAScript a JavaScript v prostředí webového prohlížeče

JavaScript je skriptovací, objektově-prototypově orientovaný programovací jazyk s dynamickou typovou disciplínou. Jeho mutace jsou nyní formálně standardizovány jako implementace jazyka *ECMAScript*.

Z hlediska zpracování je *JavaScript* standardně interpretován, novější běhová prostředí v současnosti pro zvýšení výkonnosti volí také běžně *JIT* kompilaci.

Použití JavaScriptu je nejčastěji určeno pro validaci a zpracování interakcí uživatele a manipulaci s prvky *objektového modelu dokumentu (DOM)* v prostředí webového prohlížeče. Vývoj schopností moderních webových prohlížečů dnes dále přináší možnosti asynchronní komunikace se serverem a přístup k některým prostředkům operačního systému, jako jsou multimediální zařízení, systém souborů nebo geolokace. [16]

Značná rozšířenost JavaScriptu inspirovala vznik celé řady projektů, které jeho jazykové i funkční schopnosti rozšiřují, například serverová platforma *Node.js* nebo běhové prostředí *Dart*.

Přestože vývoj jazyka zastřešuje organizace *ECMA*, nemá vliv na podobu a funkčnost *API*, se kterými jazyk pracuje, ani na jeho proprietární rozšíření, které některé webové prohlížeče implementují. To klade zvýšené nároky na optimalizaci programů a programovou kontrolu dostupných schopností před jejich využitím.

Přestože nomenklatura syntaxe i struktura některých objektů JavaScriptu byla inspirována Javou, není použití ani vývoj těchto dvou jazyků nijak provázán.

4.4 Iniciativa HTML5

V roce 2004 založilo několik členů zainteresovaných do vývoje majoritních webových prohlížečů pracovní skupinu pro webové aplikační technologie (*WHATWG*) s cílem urychlit vývoj moderních webových prostředků a především je zasadit do jednotného standardu. Tato iniciativa byla označena jako *HTML5*, na základě tehdy aktuální verze HTML – 4.01, přestože klíčovou součástí standardu bylo spíše sjednocení *API* moderních multimediálních a komunikačních prostředků a dotýkalo se tak více programové, než prezenční části webových aplikací a dokumentů. [17]

Pátou verzí jazyka HTML je tedy třeba chápat v širších souvislostech, především co se jejího vztahu k JavaScriptu týče.

4.4.1 Objekt Canvas a nízko-úrovňová manipulace s obrazovými daty

Jedním z klíčových prvků, které HTML5 standardizuje, je objekt *Canvas*. Ten navrhla v roce 2004 společnost Apple pro potřeby některých webových nástrojů systému Mac OS X verze 10.4. Později byl postupně implementován i ostatními prohlížeči a standardizován.

Objekt *Canvas* slouží jako dvourozměrné plátno pro programovou nízko-úrovňovou manipulaci s rastrovou grafikou. Umožňuje kreslení bitmap, křivek, textů i výplní a následné transformace i extrakce rastrových oblastí.

Metody Canvas API důležité pro tuto práci jsou především ty, které pracují s objektem *ImageData* – bitovou mapou vybrané rastrové oblasti, obsahující pole osmibitových hodnot kanálů *RGBA*. Surové hodnoty barevných kanálů jednotlivých pixelů obrazu pak zpracovávají optimalizační a analytické části aplikace. [18]

4.4.2 Nativní přístup k multimédiím a systému souborů

Mezi důležité schopnosti moderních webových prohlížečů patří rozhraní, které umožňují skriptům webu přistupovat k některým prostředkům vně *sandboxu* prohlížeče.

Možnost vybrat lokální soubor k odeslání na server prostřednictvím webového formuláře implementoval již v roce 1995 Netscape Navigator 2. HTML5 přidává rozhraní *File API*, umožňující přečtení i manipulaci dat takto připojených souborů ještě na straně klienta pomocí JavaScriptu. Mimo to jsou jeho metody také schopny zachytit a zpracovat soubory, které byly dovnitř webového dokumentu přetaženy myší z vnějšího systému. [18]

Moderní prohlížeče jsou rovněž schopny komunikovat s některými multimediálními periferiemi systému. Metoda *getUserMedia* dává webu možnost vyžádat si data z kamery či mikrofону počítače. Ty jsou pak skriptům k dispozici jako standardní obrazový či zvukový datový proud. V případě obrazu jsou pak metody Canvas API schopny snímat vybrané snímky videa v reálném čase a pracovat tak s nimi v jejich surové rastrové formě. [18]

5 JEDNOTLIVÉ FÁZE OBRAZOVÉ ANALÝZY APLIKACE

Webová aplikace, která je součástí této práce, demonstruje jednotlivé fáze optimalizace a analýzy obrazu pro detekci obličejů.

Pro svůj běh vyžaduje aplikace některou z aktuálních verzí majoritních prohlížečů a předpokládá, především v případě trénování neuronové sítě, i výpočetní výkon, který odpovídá současné úrovni počítačové techniky. Výpočetní režii jednotlivých kroků algoritmu se blíže věnuje kapitola 6.

Přestože jádro aplikace může fungovat i v mobilním prostředí, ovládací prvky této konkrétní demonstrace předpokládají klasický stolní počítač. Připojení k internetu není vzhledem k charakteru aplikace třeba.

Konkrétní schopnosti, které aplikace před startem testuje a vyžaduje, jsou:

- Podpora objektu *Canvas* a existence jeho 2D kontextu,
- přístup k poli bytů bitmapy, respektive existence objektu *Uint8ClampedArray*,
- podpora *File API*, respektive objektu *FileReader* pro manipulaci se soubory,
- podpora rozhraní *URL*, respektive metody *createObjectURL* pro vytváření referencí na datové objekty
- a podpora *LocalStorage* pro ukládání dat v prohlížeči.

Konvoluční neuronová síť pro finální klasifikaci obrazů je v této aplikaci sestavena pomocí JavaScriptové knihovny *ConvNetJS* [19], umožňující tvorbu a učení hlubokých neuronových sítí.

Obrázek 19 ukazuje první krok algoritmu, načtení rastrových dat ze souboru nebo datového proudu webkamery.

Velikost obrazu je v našem případě zmenšena do oblasti 800×480px (*WVGA*). Převzorkování provádí API objektu *Canvas*, konkrétně metoda *drawImage*, lineární interpolací „nejbližší soused“ – nižší optická kvalita pro účely následující analýzy ovšem není podstatná.

Detekce obličejů z obrazu neuronovou sítí
 Demonstrace algoritmu diplomové práce
 (online/1.0)

načíst obrázek trénovat síť načíst uloženou síť uložit aktuální síť do prohlížeče



zdrojová data Vylepšení obrazu pleťové barvy uzavřené oblasti testovaná množina výsledek

Surová data načtená z obrazu zmenšená do oblasti 800×480px.

Minimální velikost mezery pro výplň oblasti	<input type="text" value="0.02"/>	% plochy obrazu
Povolená plocha oblasti	<input type="text" value="1"/>	až <input type="text" value="250"/> % plochy obrazu
Minimální poměr stran oblasti	<input type="text" value="50"/>	%
Práh úspěšnosti detekce	<input type="text" value="70"/>	%
Úprava jasů	<input type="text" value="10"/>	

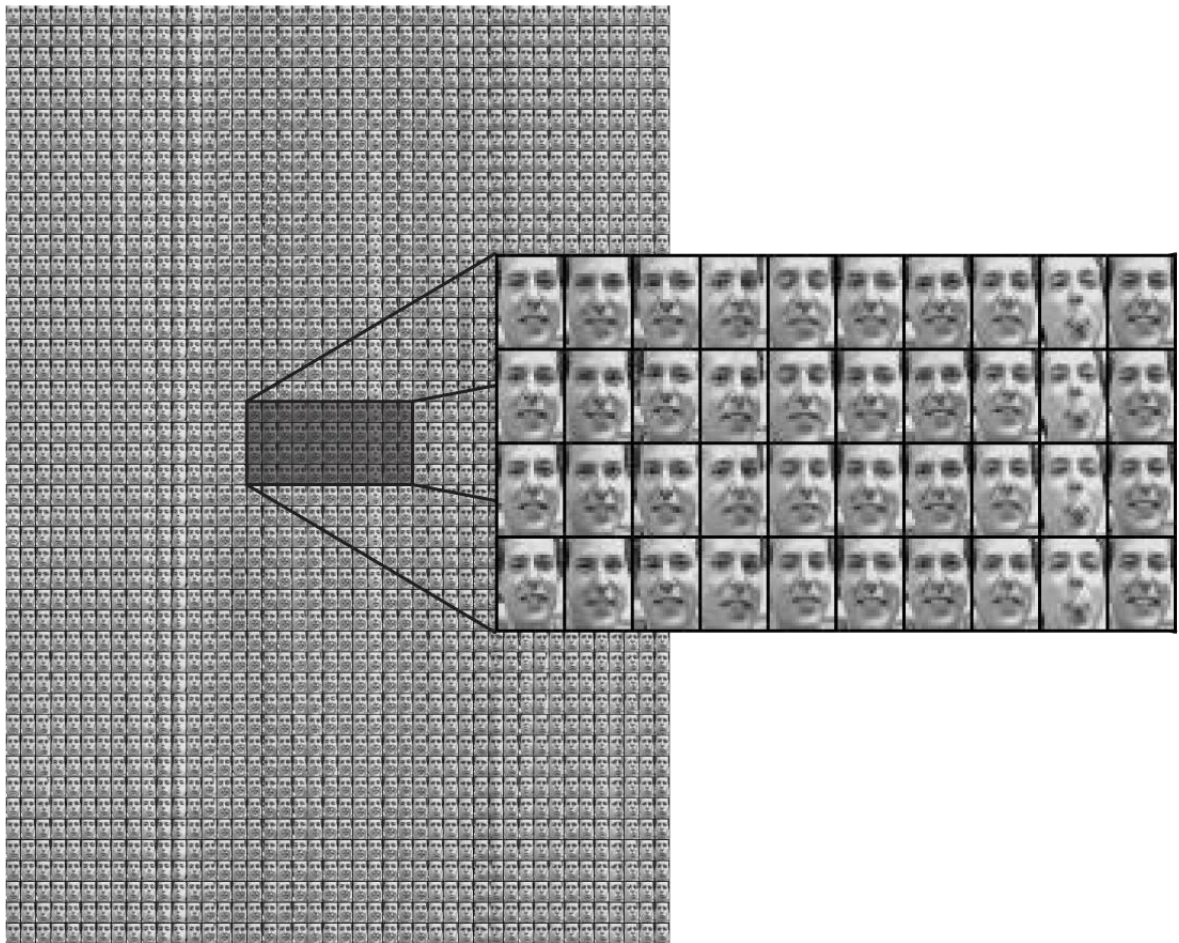
Obrázek 19: Webová aplikace pro demonstraci algoritmů

5.1 Učení neuronové sítě trénovací množinou obrazů

Součástí demonstrace je možnost provést nové trénování sítě. Takto vzniklé parametry sítě umožňuje aplikace uložit do dlouhodobé paměti prohlížeče – prostřednictvím rozhraní *LocalStorage*.

Vzhledem k určité úrovni stochastičnosti, je vhodné trénování opakovat vícekrát a volit takový výsledek uspořádání sítě, který bude vykazovat nejlepší výsledky.

Obrázek 20 ukazuje zvolenou trénovací množinu různých výrazů obličejů. Důležitá je zde určitá obrazová diverzita, která zaručuje správné rozdělení charakteristických rysů obrazu do jednotlivých ploch sítě.

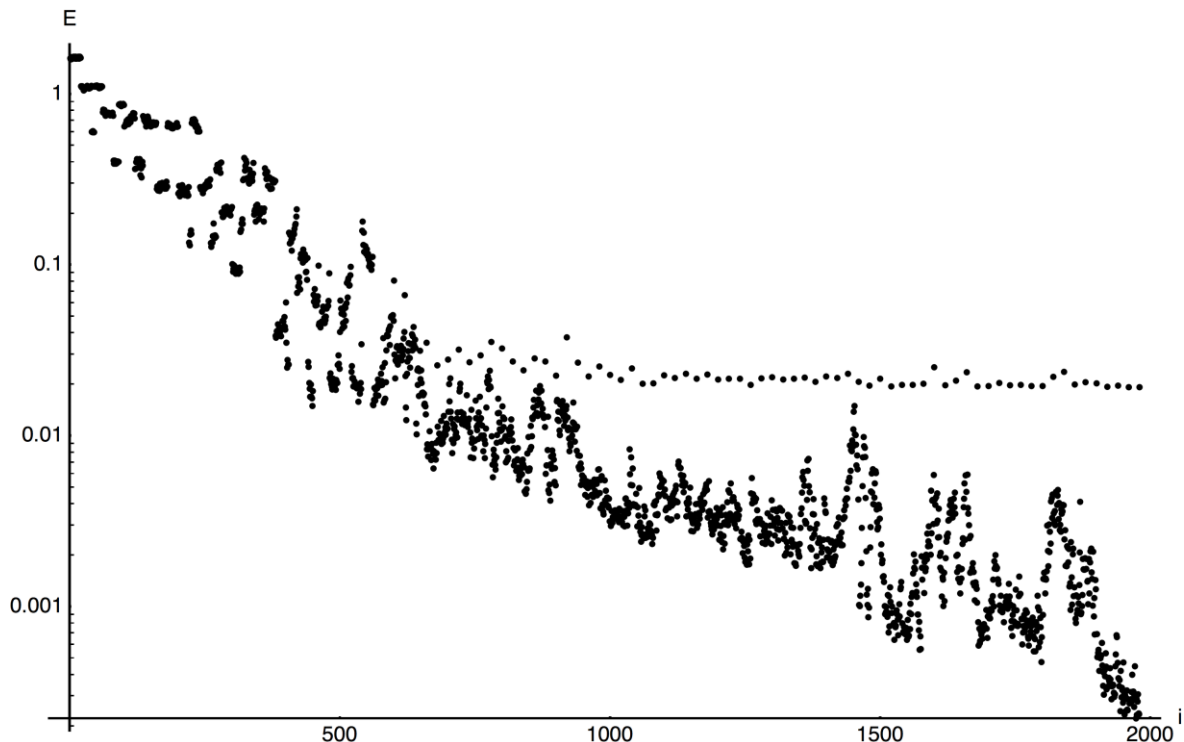


Obrázek 20: Trénovací množina obličejů Brendana Freye [20]

Kromě obličejových klasifikačních tříd je nutné síti poskytnout i třídy, do kterých mohou „propadnout“ zbylé, nevyhovující, obrazy. V našem případě je při trénování síť připojena trojice tříd, pro které jsou vygenerována souvislá data s různou intenzitou jasu.

Učení probíhá zpětným gradientním algoritmem *Adadelta* [21] z implementace knihovny *ConvNetJS*. Jako hlavní přednost algoritmu *Adadelta* je, oproti konvenčním stochastickým gradientním sestupným metodám, uváděna adaptace optimalizačního kroku na základě předchozí strmosti gradientu a tedy určitá robustnost, nižší výpočetní náročnost a rychlejší optimalizace.

Obrázek 21 ukazuje průběh jedné optimalizace neuronové sítě. Můžeme vidět, že přibližně 700 trénovacích epoch stačí k přijatelnému nastavení vah sítě.



Obrázek 21: Průběh optimalizace sítě
Hodnota chybové funkce (E) v jednotlivých epochách trénování (i).

5.2 Nalezení optimálních počátečních podmínek

Soubor heuristických optimalizací a analýz, které popisovala kapitola 3, obsahuje celou řadu počátečních podmínek, které je třeba definovat. Konkrétně je to v případě demonstrační aplikace:

- Rozsah povolené plochy analyzovaných oblastí v procentu celkové plochy obrazu, který vybírá pouze relevantně velké výřezy pro klasifikaci.
- Citlivost výplňového algoritmu *flood fill* na „netěsnosti“ homogenních oblastí.
- Minimální poměr stran oblastí, který odstraní příliš úzké nebo příliš široké výřezy.
- Celkový posun jasu a kontrastu.

Pro hodnoty parametrů první dvojice podmínek, ovlivňujících převážně segmentaci obrazu, existuje určitá korelace s reálnou vzdáleností snímané scény od objektivu. To jest, objekty ve vzdálenější rovině budou evidentně menší než ty v rovině bližší. Vzhledem k očekávané velikosti lidského obličeje je možné otestovat poměrně úzký rozsah hodnot.

Optimalizační algoritmus pak vyhledává parametry jejich postupným otestováním a volí takové, které poskytnou nejlepší průměrné výsledky klasifikace.

5.3 Jasová a barevná optimalizace

Optimalizace barevných a jasových parametrů obrazu se skládá ze dvou nezávislých částí. První částí je normalizace histogramů, popsaná v kapitole 3.4. Ta kompenzuje posuvy spektra dané expozicí.

Druhá část je absolutní korekce jasu a kontrastu, vzešlá z konfigurace nebo automatické kalibrace počátečních podmínek algoritmů. Ta má za cíl částečně kompenzovat barevné a jasové deformace ovlivňující finální klasifikaci, jako je příliš slabé či silné osvětlení, či příliš malou nebo velkou odrazivost a případně i zmírnit vliv přítomnosti výrazných stínů.

5.4 Filtr plet'ových barev

První fází segmentace je vyhledání vyhovujících plet'ových odstínů. Vhodnost volby barevného modelu s nezávislou jasovou složkou pro výběr oblasti byla zmiňována v kapitole 3.2.

Demonstrační aplikace volí rozsah kanálů Y : 80 – 213, C_B : 88 – 128, C_R : 135 – 180. Ten znázorňuje také obrázek 15.

Obrázek 22 ukazuje výsledek izolace plet'ových barev.



Obrázek 22: Ukázka izolace plet'ových barev

5.5 Izolace a selekce homogenních ploch

Oblast pleťových barev je nyní segmentována na souvislé oblasti algoritmem *flood fill*, podrobněji popsáným v kapitole 3.5.

Obrázek 23 ukazuje výběr relevantních oblastí a zamítnutí těch proporcčně neodpovídajících zvoleným počátečním podmínkám analýzy.

Algoritmus *flood fill* v této konkrétní ukázce pracuje s tolerancí 0,02 ‰ plochy obrazu, což odpovídá přibližně pěti pixelům.

Barvy jednotlivých oblastí jsou voleny náhodně a pouze pro účely demonstrace.



Obrázek 23: Ukázka izolace a selekce homogenních ploch

5.6 Klasifikace vybraných oblastí neuronovou sítí

Konečnou fází analýzy je klasifikace oblastí neuronovou sítí, vybraných předchozími kroky.

Obrázek 24 ukazuje výsledek klasifikace a procentuální shodu testované množiny dat s obličejovou třídou neuronové sítě.

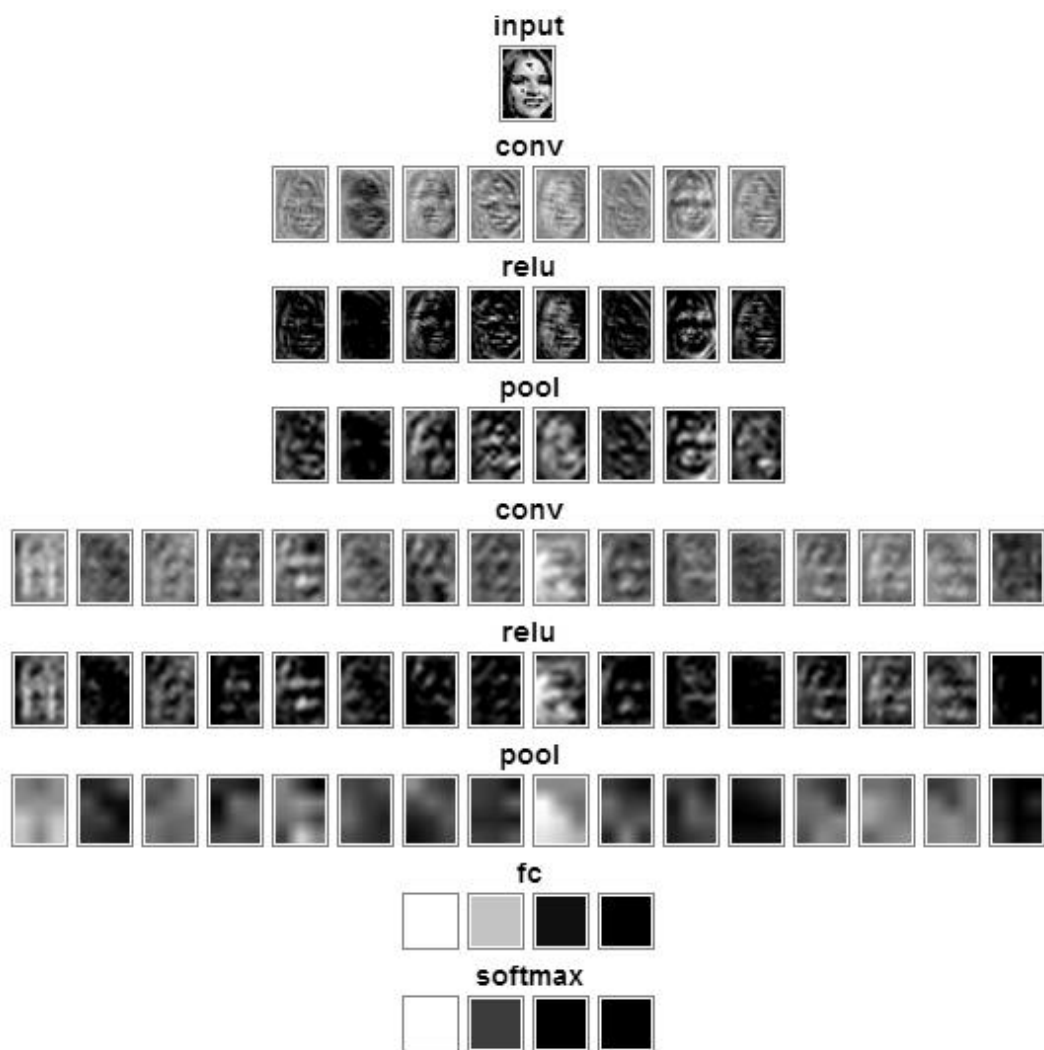


Obrázek 24: Ukázka výsledku klasifikace
Procentuální údaj vyjadřuje shodu s obličejovou třídou.

Průchod dat testovací množiny vrstvami sítě je možné zobrazit kliknutím na patřičnou oblast.

Obrázek 25 ukazuje data druhého obličeje při průchodu sítí konvolučními (*conv*) vrstvami s rektifikačními lineárními jednotkami (*ReLU*) a vzorkovacími vrstvami (*pool*). Sít' je ukončena dvojicí plně propojených (*fc*) vrstev. Neurony poslední vrstvy používají aktivační funkci *softmax* pro finální klasifikaci tříd.

Můžeme dále vidět, že extrakce vlastností je rozdělena na 8 a 16 rovin v první a druhé konvoluci.



Obrázek 25: Ukázka vstupních dat vybraných vrstev konvoluční sítě

6 ZHODNOCENÍ VÝPOČETNÍ NÁROČNOSTI A TECHNOLOGICKÝCH OMEZENÍ

Zásadní výhoda neuronových sítí je jejich schopnost řešit náročné problémy s minimální výpočetní reží. V typickém případě, kdy není nutné síť trénovat během samotného použití, je výkon potřebný pro její provoz zanedbatelný.

Vyšší výpočetní náročnost v našem případě vykazují kroky předcházející samotné klasifikaci a pochopitelně nejvyšší režii vyžaduje samotný trénink neuronové sítě.

Optimalizace JavaScriptových běhových prostředí dnes tvoří prioritu číslo jedna vývoje všech majoritních webových prohlížečů. Výkonnost algoritmu se tedy může výrazně lišit jak v různých vykreslovacích jádrech, tak dokonce i v různých jejich verzích.

Následující výkonnostní údaje byly naměřeny s pomocí profilerů webových prohlížečů Chrome/34, Firefox/29 a Internet Explorer/11 v prostředí Windows 8.1 s procesorem Intel Core i7-3820 @ 3,6 GHz.

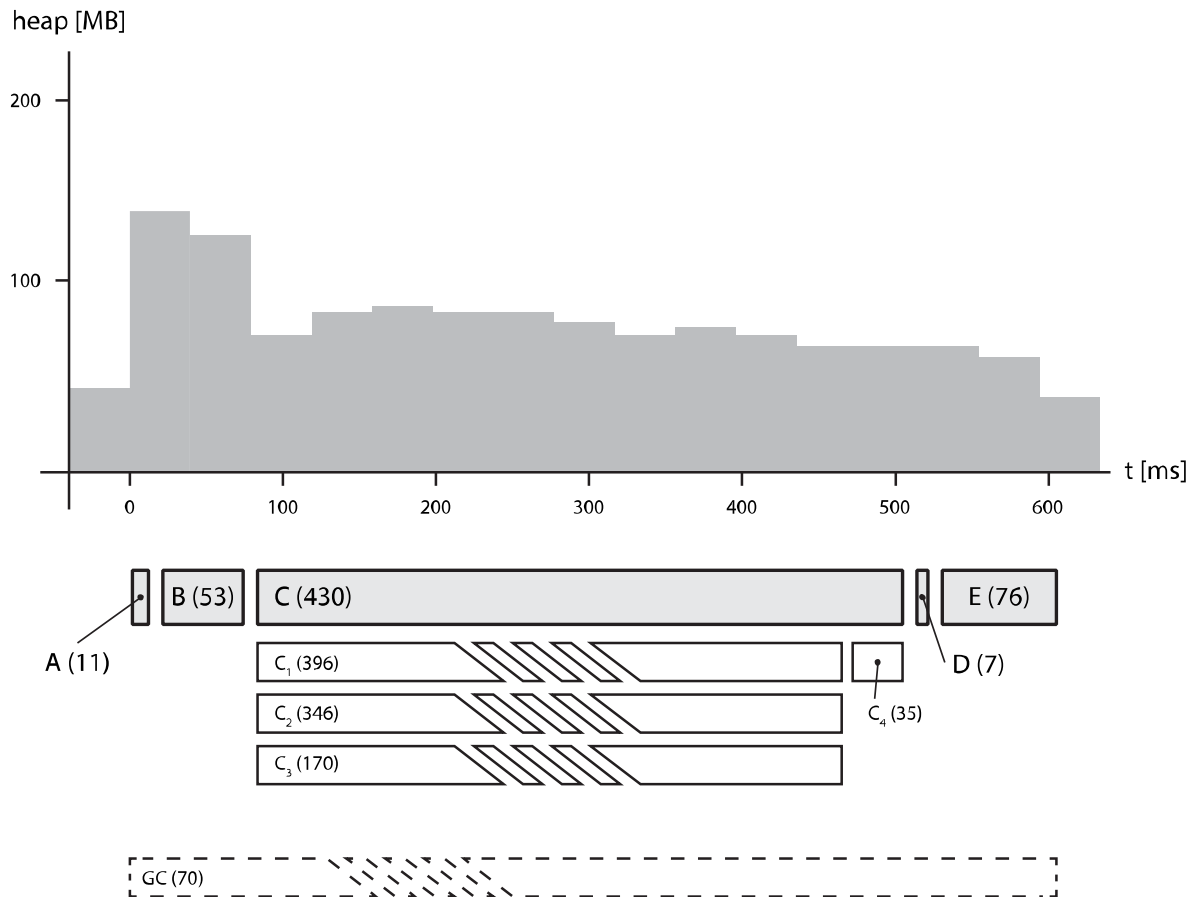
6.1 Srovnání výpočetní náročnosti jednotlivých operací

Na základě údajů z profileru prohlížeče Chrome/34 můžeme posoudit časovou a paměťovou náročnost jednotlivých fází analýzy.

Obrázek 26 ukazuje obsazení paměti v jednotlivých částech algoritmu a časy jednotlivých operací. Je třeba zdůraznit, že údaje o paměťové náročnosti jsou jen přibližné, v závislosti na míře optimalizace a způsobech, jakými JavaScriptové běhové prostředí hospodaří s přidělenými prostředky, jak častý je *garbage collecting* nebo v jaké míře jsou prostředky sdíleny.

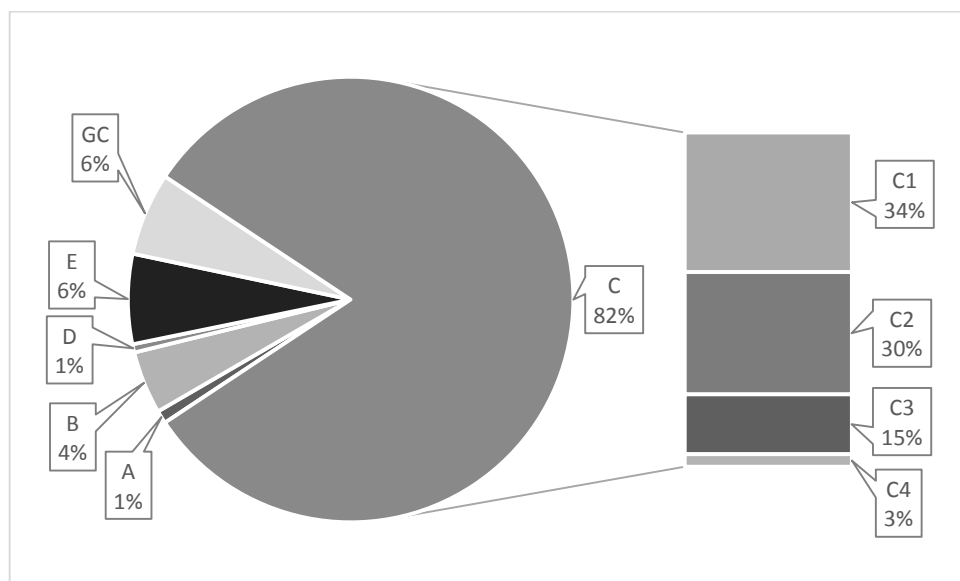
Podle očekávání patří mezi paměťově nejnáročnější první dvojice kroků, tedy barevná a jasová optimalizace (A) a filtr pleťových barev (B). V obou případech proces pracuje se všemi body obrazu a kopíruje dlouhá pole dat.

Časově nejnáročnější je heuristická segmentace (C). V závislosti na rozsahu pleťové plochy a zvoleným parametřům algoritmu *flood fill*, jde o proces s největším počtem iterací. Kapitola 3.5 zmiňovala vhodnost rozbalení hierarchické rekurze se zásobníkem do ploché iterační fronty. V tomto případě je takové uspořádání již nezbytné, protože zásobník JavaScriptových běhových prostředí je běžně omezen na několik set úrovní, což pro potřeby aplikace už nestačí.



Obrázek 26: Výpočetní a paměťová náročnost analýzy

Jasová a barevná optimalizace (A), filtr pleťových barev (B), segmentace (C₁), flood fill (C₂), ověření tolerance flood fill (C₃), výpočet hranic (C₄), izolace výřezů (D), klasifikace neuronovou sítí (E) a garbage collector (GC).

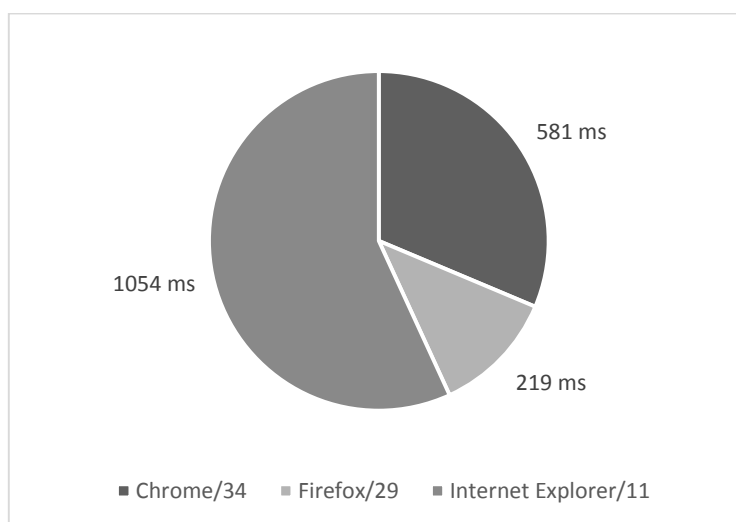


Obrázek 27: Procentuální zastoupení výpočetního času operací analýzy

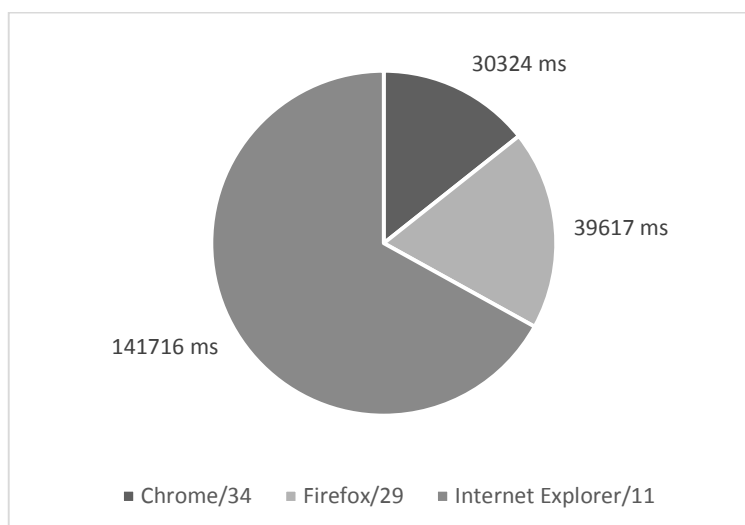
6.2 Srovnání výkonnosti běžných webových prohlížečů

Jak již bylo zmíněno, JavaScriptová běhová prostředí zaznamenávají pravděpodobně nejdynamičtější vývoj, co se programových prostředí obecně týče. Rozdíly ve výkonu stejného algoritmu napříč prohlížeči jsou tedy značné.

Jak můžeme vidět z časových výsledků analýzy obrazu (obrázek 28) a samotného tréninku neuronové sítě (obrázek 29), v případě neakcelerovaných výpočetních operací je výkon prostředí *V8* (Chrome) a *SpiderMonkey* (Firefox) srovnatelný. Pro prostředí *Chakra* Internet Exploreru je přibližně 2-3krát pomalejší.



Obrázek 28: Výpočetní doba analýzy



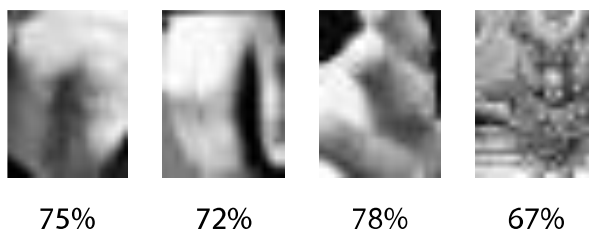
Obrázek 29: Výpočetní doba trénování neuronové sítě

6.3 Hlavní úskalí analýzy monoskopického počítačového vidění

Přesnost detekce tváře v běžných optických podmínkách a se standardní snímací technikou může kolísat mezi 90% pro ideální světelné podmínky, kolmý a přímý úhel snímání až po 40% pro špatné světelné podmínky, nepřímý úhel snímání a objekty zakrývající obličej, jako vlasy nebo brýle.

Podle subjektivních výsledků testování detekce v této práci by se dala heuristická segmentace označit za slabou stránku algoritmu. Především pak v případech, kdy jsou obličeje v blízkosti, nebo jsou z velké části zakryty brýlemi a vlasy.

Konvoluční neuronová síť pak chybí spíše směrem k falešně-positivní obličejové klasifikaci – především pokud má na vstupu příliš malý výřez obrazu nebo výřez s velkým množstvím hran a tvarů, které jsou schopny falešně saturovat detektory vlastností konvolučních vrstev. Obrázek 30 ukazuje některé příklady falešné positivity.



*Obrázek 30: Ukázka falešně pozitivních klasifikací
Zleva sevřená pěst, hřbet ruky, prsty, ornament dekorace.*

Pro podstatné zvýšení spolehlivosti detekce by se nabízelo posunout hardwarové prostředky směrem k pokročilým, nicméně dnes již ne tak nedostupným, technikám snímání obrazu, mezi které patří:

- Nasvícení scény IR světlem a podrobnější analýza odrazivosti a barvy pleti,
- trojrozměrné snímání dvojicí kamer nebo IR laserem,
- snímání obrazu v IR spektru termo-kamerou.

ZÁVĚR

Tato práce měla za cíl zmapovat možnosti a realizovat detekci obličejů z obecných rastrových obrazových dat. Pro tento úkol tak zvolila typický nástroj – hlubokou konvoluční neuronovou síť – pracující v poněkud netypickém prostředí – uvnitř webového prohlížeče.

Přestože taková adaptace není příliš obvyklá, tak vzhledem k masivnímu rozvoji webových a cloudových technologií a služeb, kdy se na web, potažmo do webového prohlížeče, stěhují stále více komplexní aplikace, dává smysl implementovat některé náročné algoritmy už i na straně klienta. Pokud počátek 21. století patřil „*Webu 2.0*“, kdy se z konzumentů stali uživatelé a z uživatelů autoři obsahu, druhé desetiletí posouvá web na úroveň ostatních aplikačních prostředí. A v některých ekosystémech dokonce prostředí první volby.

Praktickým výsledkem této práce je demonstrační aplikace, která ukazuje všechny důležité kroky obrazové analýzy a umožňuje tak rychle prověřit účinnost zvoleného postupu, vhodnost nastavených parametrů a správnost uspořádání neuronové sítě.

Z testování aplikace plynou také některé důležité závěry a doporučení:

- Rychlost a schopnosti JavaScriptových prostředí moderních prohlížečů a střední výkon běžných počítačů a mobilních zařízení dosáhl úrovně, kdy už jsou schůdné i pokročilé datové analýzy na straně klienta v reálném čase.
- Vzhledem k tomu, že nevyužívá žádných nestandardních nástrojů, je s minimálními úpravami jádro aplikace přenositelné do mobilních telefonů a tabletů. Více než jinde zde ovšem bude důležitý jejich výpočetní výkon a tak i pravděpodobně potřebná optimalizace některých kroků algoritmu.
- Hluboká konvoluční neuronová síť je ideálním prostředkem pro klasifikaci vzorů v obrazových datech a poskytuje velmi slušné výsledky s minimální výpočetní režíí.
- Neuronová síť má tendenci chybovat na částech lidského těla s podobnými proporcemi a tvary jako má obličej. Je to například sevřená pěst nebo hřbet ruky. Vhodným opatřením by mohlo být naučit síť na tyto tvary reagovat, vytvořením další klasifikační třídy pro neobličejové části těla se svou vlastní trénovací množinou dat.
- Problematika heuristické segmentace obrazu před samotnou klasifikací má velký potenciál k dalšímu rozvoji. Kromě barevné analýzy by bylo vhodné doplnit další metody pro zpřesnění vyhledávání potenciálních obličejových oblastí. Alternativně by šel tento problém uchopit i implementací jediné *prostorově dislokované neuronové sítě*, kdy by segmentaci nahradila až analýza jejích výstupů.

- Segmentace se ukazuje být časově kritickou operací. Především proto, že její výpočet obnáší značný počet neparalelizovaných iterací. Nabízí se zde možnost algoritmus zcela přepsat a upravit jej pro běh pomocí takzvaných *Web Workers*, což jsou paralelní výpočetní vlákna podporovaná moderními prohlížeči.

Pro případ nasazení algoritmu do reálného použití jsou připraveny a zdokumentovány všechny vytvořené objekty, které aplikace používá. Demonstrační aplikace pak může sloužit jako nástroj pro konfiguraci a testování neuronové sítě a přidružených algoritmů před jejich nasazením do ostrého provozu.

SEZNAM POUŽITÉ LITERATURY

1. **Freeman, James A. a Skapura, David M.** *Neural networks: algorithms, applications, and programming techniques*. Mishawaka, IN, U.S.A. : Addison-Wesley Pub, 1991. ISBN 0-201-51376-5.
2. **Kasabov, Nikola K.** *Foundations of neural networks, fuzzy systems, and knowledge engineering*. Cambridge, MA, U.S.A. : Massachusetts Institute of Technology, 1998. ISBN 0-262-11212-4.
3. **Georgeson, Mark A., Green, Patrick R. a Bruce, Vicki.** *Visual Perception: Physiology, Psychology and Ecology*. New York, USA : Psychology Press, 2013. ISBN: 1841692387 .
4. **Kolb, Helga, a další.** Webvision. *The Organization of the Retina and Visual System*. [Online] 2014. [Citace: 2. květen 2014.] <http://webvision.med.utah.edu/>.
5. **Kröger, Ronald H.H. a Biehlmaier, Oliver.** Space-saving advantage of an inverted retina. *Vision Research*. [Online] 9. únor 2009. [Citace: 2. květen 2014.] <http://www.sciencedirect.com/science/article/pii/S0042698909003162>. ISSN: 0042-6989.
6. **Hochreiter, Sepp.** *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies*. Munchen, Germany : Fakultat fur Informatik, Technische Universitat Munchen, 2001.
7. **Bengio, Yoshua.** *Learning Deep Architectures for AI (Foundations and Trends in Machine Learning)*. Montreal, Canada : Université de Montréal, 2009. ISBN: 1601982941.
8. **LeCun, Yann, a další.** *Object Recognition with Gradient-Based Learning*. Red Bank, NJ, USA : Feature Grouping, Springer, 1999.
9. **Fukushima, Kunihiro a Miyake, Sei.** Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*. 6, 1982, 15.
10. **Pantone LLC.** PANTONE. *SkinTone™ Guide*. [Online] 2014. [Citace: 2. duben 2014.] <http://pantone.com/pages/pantone.aspx?pg=21046>.
11. *Fast implementation of color constancy algorithms.* **Morel, Jean-Michel, Petro, Ana B. a Sbert, Catalina.** San Jose : Proc. SPIE 7241, Color Imaging XIV: Displaying, Processing, Hardcopy, and Applications, 724106, 2009. doi:10.1117/12.805474.
12. **W3C Technical Architecture Group.** Architecture of the World Wide Web, Volume One. *World Wide Web Consortium*. [Online] 15. prosince 2004. [Citace: 18. května 2012.] <http://www.w3.org/TR/2004/REC-webarch-20041215/>.

13. **The Internet Society.** Hypertext Transfer Protocol -- HTTP/1.1. *The Internet Engineering Task Force*. [Online] Červen 1999. [Citace: 15. května 2012.] <http://tools.ietf.org/html/rfc2616>. RFC 2616.
14. **HTML Working Group.** HTML 4.01 Specification. *World Wide Web Consortium*. [Online] 24. prosince 1999. [Citace: 12. května 2012.] <http://www.w3.org/TR/html4/>.
15. **CSS Working Group.** Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. *World Wide Web Consortium*. [Online] 7. června 2011. [Citace: 12. května 2012.] <http://www.w3.org/TR/CSS2/>.
16. **Nixon, Robin.** *Learning PHP, MySQL, and JavaScript*. Sebastopol, CA, USA : O'Reilly Media, 2009. ISBN 978-0-596-15713-5.
17. **WHATWG Group.** Specifications. *Web Hypertext Application Technology Working Group*. [Online] 2014. [Citace: 15. květen 2014.] <http://www.whatwg.org/>.
18. **Mozilla Developer Network and individual contributors.** Web API Interfaces. *Mozilla Developer Network*. [Online] 2014. [Citace: 10. květen 2014.] <https://developer.mozilla.org/en-US/docs/Web/API>.
19. **Karpathy, Andrej.** Javascript library for training Deep Learning models. *ConvNetJS*. [Online] Stanford University. [Citace: 25. leden 2014.] <http://cs.stanford.edu/people/karpathy/convnetjs/>.
20. **Roweis, Sam.** Data for MATLAB hackers. *Sam Roweis, Associate Professor*. [Online] Courant Institute of Mathematical Sciences. [Citace: 20. únor 2014.] <http://cs.nyu.edu/~roweis/data.html>.
21. **Zeiler, Matthew D.** *ADADELTA: An Adaptive Learning Rate Method*. Ithaca, NY, USA : Cornell University, 2012. arXiv:1212.5701.
22. **Basilio, Jorge, a další.** Explicit Image Detection using YCbCr Space Color Model as Skin Detection. [autor knihy] Alexander Zemliak a Nikos Mastorakis. *APPLICATIONS of MATHEMATICS and COMPUTER ENGINEERING*. Mexico City : Sección de Estudios de Posgrados e Investigación, 2011.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application programming interface
Backpropagation	Back propagation of error
BAM	Bidirectional associative memory
CSS	Cascading Style Sheet
DOM	Document Object Model
ECMA	European Computer Manufacturers Association
FC	Fully connected
GABA	Kyselina gama-aminomáselná
HSL	Hue, saturation, lightness
HTML	Hypertext markup language
HTTP	Hypertext Transfer Protocol
IR	Infračervený
JIT	Just In Time
NPC	Non-deterministic polynomial-time complete
ReLU	Rectified linear unit
RGB	Red, green, blue
RGBA	Red, green, blue, alpha
SGML	Standard Generalized Markup Language
TCP	Transmission control protocol
YCbCr	Barevný model jas, modrá a červená chrominance
URI	Uniform resource identifier
URL	Uniform resource locator
WHATWG	Web Hypertext Application Technology Working Group
WVGA	Wide Video Graphics Array
WWW	World Wide Web

SEZNAM OBRÁZKŮ

Obrázek 1: Lidský neuron.....	11
Obrázek 2: Synapse.....	12
Obrázek 3: Umělý neuron – Perceptron.....	13
Obrázek 4: Neuronová struktura sítnice lidského oka	15
Obrázek 5: Ukázka konvolučního uspořádání synapsí vrstev neuronové sítě	17
Obrázek 6: Ukázka rozpoznání písmene A sítí neocognitron (zdroj [9])	18
Obrázek 7: Struktura vrstev sítě Neocognitron	19
Obrázek 8: Jednoduché a komplexní vrstvy při rozpoznávání tvarů.....	20
Obrázek 9: Inhibitor neuronu jednoduché vrstvy	20
Obrázek 10: Prostorová symetrie synapsí	21
Obrázek 11: Struktura sítě LeNet	22
Obrázek 12: Reaktivita fotoreceptorů sítnice na vlnovou délku světla	25
Obrázek 13: Barevné modely RGB a $Y_{CB}C_R$	26
Obrázek 14: Pleťové barvy Pantone® v prostoru RGB.....	27
Obrázek 15: Pleťové barvy Pantone® v prostoru $Y_{CB}C_R$	27
Obrázek 16: Korekce podexpozice normalizací jasového histogramu.....	28
Obrázek 17: Algoritmus flood fill se zásobníkem (S) nebo frontou (Q).....	29
Obrázek 18: Algoritmus flood fill uzavírající příliš malé spojnice	29
Obrázek 19: Webová aplikace pro demonstraci algoritmů.....	36
Obrázek 20: Trénovací množina obličejů Brendana Freye [20]	37
Obrázek 21: Průběh optimalizace sítě	38
Obrázek 22: Ukázka izolace pleťových barev.....	39
Obrázek 23: Ukázka izolace a selekce homogenních ploch.....	40
Obrázek 24: Ukázka výsledku klasifikace.....	41
Obrázek 25: Ukázka vstupních dat vybraných vrstev konvoluční sítě.....	42
Obrázek 26: Výpočetní a paměťová náročnost analýzy	44
Obrázek 27: Procentuální zastoupení výpočetního času operací analýzy.....	44
Obrázek 28: Výpočetní doba analýzy	45
Obrázek 29: Výpočetní doba trénování neuronové sítě.....	45
Obrázek 30: Ukázka falešně pozitivních klasifikací	46

SEZNAM PŘÍLOH

- P1** Dokumentace API objektů demonstrační aplikace
- P2** Demonstrační aplikace na CD

PŘÍLOHA P1: DOKUMENTACE API OBJEKTŮ DEMONSTRAČNÍ APLIKACE

CLASS: FACEDETECTION

FaceDetection

new FaceDetection(neuralNet *opt*)

Base face detection object

This:

- {FaceDetection}

Parameters:

Name	Type	Attributes	Description
neuralNet	convnetjs.Net	<optional>	Neural net

Version: 1.0.0
Author: Martin Hozík

Members

(private, inner) **net** :convnetjs.Net

Type:

- convnetjs.Net

(private, inner) **trainer** :convnetjs.SGDTrainer

Type:

- convnetjs.SGDTrainer

Methods

classify(imageData, verbose *opt*) → {Object}

Classify image data

Parameters:

Name	Type	Attributes	Default	Description
imageData	ImageData			Image data
verbose	Boolean	<optional>	false	Verbose output

Returns:

Classification results

Type
Object

getBlankVolume(brightness) → {convnetjs.Vol}

Get blank volume

Parameters:

Name	Type	Description
brightness	Number	Brightness level 0-1 (white)

Returns:

Type
convnetjs.Vol

getInputDimensions() → {Object}

Get input layer dimensions

Throws:

Error

Returns:

Type
Object

getLayersStats() → {Array}

Get stats for neural net layers

Returns:

Type
Array

getLayerWeights(layer, filters *opt*, gradient *opt*) → {Array.<ImageData>}

Get layer weights visualisation

Parameters:

Name	Type	Attributes	Default	Description
layer	number			
filters	boolean	<optional>	false	
gradient	boolean	<optional>	false	

Throws:

Error

Returns:

layer weights visualisation

Type
Array.<ImageData>

getVolume(imageData) → {convnetjs.Vol}

Get volume from image data

Parameters:

Name	Type	Description
imageData	ImageData	

Returns:

Type
convnetjs.Vol

initNet(width, height, classes)

Initialize neural net

Parameters:

Name	Type	Description
width	Number	
height	Number	
classes	Number	

loadNet(jsonString) → {undefined}

Load net from JSON

Parameters:

Name	Type	Description
jsonString	String	

Returns:

Type
undefined

toString() → {String}

Returns:

JSON of Net

Type String

trainNet(trainSet, classNo) → {Object}

Train net

Parameters:

Name	Type	Description
trainSet	Array	Set of training image data
classNo	Number	Class number

Returns:

Statistics

Type Object

trainNetBlankTrainSet(brightness, length, classNo) → {Object}

Train net to blank input

Parameters:

Name	Type	Description
brightness	Number	Brightness level 0-1 (white)
length	Number	
classNo	Number	

Returns:

Type Object

CLASS: IMAGEPREPROCESSOR

ImagePreprocessor

new ImagePreprocessor(imageData *opt*)

Image preprocessor

This:

- {ImagePreprocessor}

Parameters:

Name	Type	Attributes	Description
imageData	ImageData	<optional>	Source image data

Version: 1.0.0
Author: Martin Hozík

Members

(static) colors :Array

Array of random colors

Type:

- Array

(private, inner) **src** :ImageData

Type:

- ImageData

Methods

(static) ColorHSLtoRGB(h, s, l) → {Array}

Converts an HSL color value to RGB.

Parameters:

Name	Type	Description
h	Number	The hue
s	Number	The saturation
l	Number	The lightness

Returns:

The RGB representation

Type
Array

(static) ColorRGBtoHSL(r, g, b) → {Array}

Converts an RGB color value to HSL.

Parameters:

Name	Type	Description
r	Number	The red color value
g	Number	The green color value
b	Number	The blue color value

Returns:

The HSL representation

Type
Array

(static) colorRGBtoYCbCr(r, g, b) → {Array}

Convert RGB to YCbCr space via CCIR 601 recommendations

Parameters:

Name	Type	Description
r	Number	Red
g	Number	Green
b	Number	Blue

Returns:

YCbCr color components

Type
Array

(static) colorYCbCrtoRGB(y, cb, cr) → {Array.<Number>}

Convert YCbCr to RGB space via CCIR 601 recommendations

Parameters:

Name	Type	Description
y	Number	Luma
cb	Number	Blue chroma difference
cr	Number	Red chroma difference

Returns:

RGB color components

Type
Array.<Number>

(static) randomColor() → {Array.<Number>}

Get random RGB color

Returns:

RGB Color

Type

Array.<Number>

(static) resizeImageData(imagedata, width *opt*, height *opt*, keepAspectRatio *opt*) → {ImageData}

Resize Image Data

Parameters:

Name	Type	Attributes	Default	Description
imagedata	ImageData			
width	Number	<optional>		
height	Number	<optional>		
keepAspectRatio	Boolean	<optional>	true	

Throws:

Error

Returns:

Type

ImageData

(private, inner) detectGap(space, row, col, gap, id, direction) → {Number}

Detect gap in given direction

Parameters:

Name	Type	Description
space	Array	
row	Number	
col	Number	
gap	Number	
id	Number	
direction	String	

Throws:

Error

Returns:

Type

Number

(private, inner) equalizeHistogram(src)

Equalizes the histogram of an unsigned 1-channel image

Parameters:

Name	Type	Description
src	Array	1-channel source image

(private, inner) floodFill(space, row, col, areaId, gap, stack)

Flood fill chunks in image space

Parameters:

Name	Type	Description
space	Array	Image space
row	Number	
col	Number	
areaId	Object	
gap	Number	
stack	Array	call stack

(private, inner) getChunks(colorRGBA, gap) → {Array}

Compute chunks

Parameters:

Name	Type	Description
colorRGBA	Array.<Number>	Matching color (0-255 or -1 as wildcard)
gap	Number	Min gap size ratio

Returns:

Type
Array

(private, inner) resolveChunkCoords(space, areaRange, areaRatio) → {Array}

Resolve chunk coordinates

Parameters:

Name	Type	Description
space	Array	
areaRange	Array.<Number>	
areaRatio	Number	Minimal area ratio

Returns:

Array of coordinates

Type
Array

filterColorRangeYCbCr(y, cr, cb, binary ^{opt}) → {ImageData}

Filter image color using YCbCr color space

Parameters:

Name	Type	Attributes	Default	Description
y	Number Array.<Number>			Y value or min,max range
cr	Number Array.<Number>			Cr value or min,max range
cb	Number Array.<Number>			Cb value or min,max range
binary	boolean	<optional>	false	Return black/white image instead of transparent/opaque

Returns:

Type
ImageData

getChunkCoords(colorRGBA, areaRange, gap, areaRatio) → {Array}

Get solid chunks of matching color

Parameters:

Name	Type	Description
colorRGBA	Array.<Number>	Matching color (0-255 or -1 as wildcard)
areaRange	Array.<Number>	Min and max area ratio
gap	Number	Min gap size ratio
areaRatio	Number	

Returns:

Type
Array

getChunkShowcaseImageData(colorRGBA, gap) → {ImageData}

Showcase solid chunks of matching color

Parameters:

Name	Type	Description
colorRGBA	Array.<Number>	Matching color (0-255 or -1 as wildcard)
gap	Number	Min gap size ratio

Returns:

Type
 ImageData

getCutouts(coords, width *opt*, height *opt*) → {Array.<ImageData>}

Get cutouts by coordinates

Parameters:

Name	Type	Attributes	Description
coords	Array.<Object>		Array of coordinates
width	Number	<optional>	Output width
height	Number	<optional>	Output height

Returns:

Type
 Array.<ImageData>

getNormalizedLightness() → {ImageData}

Get ImageData with normalized lightness channel

Returns:

Type
 ImageData

setImageData(imageData)

Set image data

Parameters:

Name	Type	Description
imageData	ImageData	

CLASS: MEDIAIO

MediaIO

new MediaIO(imagedata *opt*)

Media Input/Output

This:

- {MediaIO}

Parameters:

Name	Type	Attributes	Description
imagedata	ImageData	<optional>	Image data

Version: 1.0.0
Author: Martin Hozík

Members

(private, inner) **imageData** :ImageData

Type:

- ImageData

Methods

(private, inner) **getCanvas()** → {HTMLElement}

Get image data as Canvas

Returns:

Type
HTMLElement

(private) **getCanvas()** → {HTMLElement}

Get image data as Canvas

Returns:

Type
HTMLElement

getImageData() → {ImageData}

Get Image data

Returns:

Type
ImageData

getSlices(row *opt*, col *opt*, width, height, transpose *opt*, flatten *opt*) → {Array|ImageData}

Get image slices

Parameters:

Name	Type	Attributes	Default	Description
row	Number	<optional>		Row index
col	Number	<optional>		Column index
width	Number			Slice width
height	Number			Slice height
transpose	Boolean	<optional>	false	Transpose multidimensional array
flatten	Boolean	<optional>	false	Flatten array

Returns:

array of ImageData or ImageData

Type
Array | ImageData

loadImage(src, width *opt*, height *opt*)

Load image from source (URL)

Parameters:

Name	Type	Attributes	Description
src	String		
width	Number	<optional>	
height	Number	<optional>	

onerror(error;)

On error handler

Parameters:

Name	Type	Description
error;	*	

onload()

On image load handler

toString() → {String}

Get current Image data as URL string

Returns:

Image DataURL

Type

String

CLASS: MAIN

Main

new Main()

Main class for face detection demonstration

Version:	1.0.0
Author:	Martin Hozík

Throws:

Error

Members

(inner) areaRange :Array.<number>

Allowed area range in percent

Type:

- Array.<number>

(inner) areaRatio :number

Allowed minimal aspect ratio of area

Type:

- number

(inner) brightness :number

Brightness correction

Type:

- number

(inner) contrast :number

Contrast correction

Type:

- number

(private, inner) facedetection :FaceDetection

Face detection class

Type:

- FaceDetection

(private, inner) gap :number

Flood fill close gap ratio

Type:

- number

(inner) sourceImageData :ImageData

Type:

- ImageData

(inner) threshold :number

Success/Fail classification treshold

Type:

- number

(private, inner) YCbCrRange :Array.<Number>

Skin color range

Type:

- Array.<Number>

Methods

(static) selftest() → {Object}

Get compatibility selftest result

Returns:

Type
Object

classifyChunks() → {Array}

Get final classification result

Returns:

Type
Array

getAugmentedImage() → {ImageData}

Get image data with brightness and contrast augmentation

Returns:

Type
ImageData

getChunkCoords() → {Array}

Get chunk coords

Returns:

Type
Array

getChunkShowcase() → {ImageData}

Get chunk showcase image data

Returns:

Type
ImageData

getSkinColors() → {ImageData}

Get skin colors

Returns:

Type
ImageData

getSourceData() → {ImageData}

Get raw Image Data

Returns:

Type
ImageData

initNeuralNet(width, height, classCount)

Initialize neural net

Parameters:

Name	Type	Description
width	number	
height	number	
classCount	number	

loadNeuralNet(jsonString)

Load neural net from JSON string

Parameters:

Name	Type	Description
jsonString	String	

loadSourceFromURL(url, width *opt*, height *opt*)

Load source image data from URL

Parameters:

Name	Type	Attributes	Description
url	String		
width	Number	<optional>	
height	Number	<optional>	

neuralNetReady() → {boolean}

Test neural net readiness

Returns:

Type
boolean

onerror(error)

On error handler

Parameters:

Name	Type	Description
error	*	

onsourceready()

On source image data ready handler

optimizeParams()

Try to find optimal environmental parameters values for current image

Throws:

Error

restoreNeuralNet(name)

Restore neural net from local storage

Parameters:

Name	Type	Description
name	String	

setParams(gapValue, areaRangeMin, areaRangeMax, areaRatioValue, thresholdValue, contrastValue, brightnessValue)

Set environmental parameters

Parameters:

Name	Type	Description
gapValue	number	
areaRangeMin	number	
areaRangeMax	number	
areaRatioValue	number	
thresholdValue	number	
contrastValue	number	
brightnessValue	number	

storeNeuralNet(name)

Store current neural net into local storage

Parameters:

Name	Type	Description
name	String	

traceNetWeight(area) → {Array}

Trace neural net internal weights

Parameters:

Name	Type	Description
area	number	

Returns:

Type
Array

trainNeuralNet(trainSet, classId) → {*}

Train neural net using train set of images or brightness value

Parameters:

Name	Type	Description
trainSet	Array.<ImageData> Number	
classId	Number	

Throws:

Error

Returns:

statistics

Type
*