VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Analýza zornic v obrazech

Pupil Analysis in Images

2020 Markéta Hrabánková

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Markéta Hrabánková**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Analýza zornic v obrazech
Pupil Analysis in Images

Jazyk vypracování: čeština

Zásady pro vypracování:

Sledování pohybu a polohy zornic v obrazech je v posledních letech hodně rozvíjené téma. Aplikace tohoto druhu může být použita například v oblasti samořiditelných vozidel k detekci únavy řidiče.

1. Popište základní pojmy a metody v oblasti analýzy zornic (duhovek) v obrazech.
2. Seznamte se s volně dostupnými knihovnami a popište jaké možnosti nabízí v této oblasti (například OpenCV, Dlib, TensorFlow).
3. S pomocí knihoven vytvořte vybraný detektor (rozpoznávač).
4. Experimentálně ověřte funkčnost, přesnost a rychlost navrženého řešení na renomovaných datasetech.
5. Své závěry řádně zdokumentujte v textu práce.

Seznam doporučené odborné literatury:

[1] Wolfgang Fuhl, Thiago Santini, Gjergji Kasneci, Wolfgang Rosenstiel, and Enkelejda Kasneci. 2017. PupilNet v2.0: Convolutional Neural Networks for CPU based real time Robust Pupil Detection. CoRR abs/1711.00112 (2017)
[2] Wolfgang Fuhl, Thomas Kübler, Katrin Sippel, Wolfgang Rosenstiel, and Enkelejda Kasneci. 2015. ExCuSe: Robust Pupil Detection in RealWorld Scenarios. In Computer Analysis of Images and Patterns, George Azzopardi and Nicolai Petkov (Eds.). Springer International Publishing, Cham, 39--51.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radovan Fusek, Ph.D.**

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 14. května 2020

Kvabánková

Ráda bych na tomto místě poděkovala panu doktoru Radovanu Fuskovi za vedení a pomoc při tvorbě práce.

Abstrakt

Cílem práce bylo seznámit se s metodami pro detekci zornic v obrazech a následně s pomocí volně dostupných knihoven naimplementovat detektor zornic. V této práci je navrženo několik způsobů řešení a při implementaci byly použity knihovny OpenCV a dlib. Základem řešení je kombinace evolučních algoritmů s metodami pro detekci zornic. Představené metody používají různé způsoby detekce, některé jsou založené na základních algoritmech z oblasti analýzy obrazu, kterým je například detekce hran, v jiných řešeních jsou použity metody strojového učení. Přesnost a rychlost navrhovaných způsobů řešení při použití různých parametrů pro nastavení algoritmů je ověřena na renomovaných datasetech.

Klíčová slova: detekce zornic, evoluční algoritmy, detekce hran, HOG deskriptor, SVM klasifikátor, konvoluční neuronová síť

Abstract

The aim of this thesis was to get familiar with pupils detection methods and then implement a pupil detector with usage of available libraries. There are several solutions proposed in this thesis and libraries used in the implementation are OpenCV and dlib library. A combination of evolutionary algorithms with pupils detection methods is used in the solution. Presented methods use different kinds of detection, some of them are based on image analysis primary algorithms such as edge detection, the other solutions use machine learning methods. Precision and speed of the proposed solutions with usage of different algorithms parameters settings is verified on renowned data sets.

Keywords: pupils detection, evolutionary algorithms, edges detection, HOG descriptor, SVM classifier, convolutional neural network

Obsah

Seznam použitých zkratek a symbolů

AIPF	–	Angular Integral Projection Function
CNN	–	Convolutional neural network
DE	–	Diferenciální evoluce
ElSe	–	Ellipse Selector
ExCuSe	–	Exclusive Curve Selector
GI4E	–	Gaze Interaction for Everybody
HOG	–	Histogram of Oriented Gradients
LPW	–	Labelled pupils in the wild
OpenEDS	–	Open Eye Dataset
RAM	–	Random access memory
ReLU	–	Rectified Linear Unit
SGD	–	Stochastic gradient descent
SOMA	–	Samoorganizující se migrační algoritmus
SVM	–	Support vector machine

Seznam obrázků

Seznam tabulek

1 Úvod

Detekce a analýza obličeje za pomocí obrazů nalézá v dnešní době spoustu různých využití. Jedním z nich může být anonymizace fotografií či videa, kdy je detekovaný obličej rozmazán nebo upraven, aby nebylo možné danou osobu poznat. Detekce obličeje je také prvním krokem k rozpoznání obličeje z obrazů, což mohou používat například policejní složky pro nalezení osoby pomocí videí z veřejných kamer. Další oblasti využití nabízí detekce a analýza částí obličeje. Sledováním pohybu a polohy očí, úst či obočí lze určit emoce nebo chování osoby, což může být použito v automobilovém odvětví pro určení únavy nebo zdravotního stavu řidiče. V rámci analýzy obličeje jsou vyvíjeny metody pro zjištění pozice zornic či duhovek a informace o poloze středu zornice se dá dále použít k určení směru pohledu, čehož se dá využít také v automobilu pro sledování pozornosti řidiče nebo k ovládání palubní desky pomocí očí. Směr pohledu může být použit i k ovládání úplně jiného stroje, například upraveného invalidního vozíku [1]. Přestože má dnes detekce zornic již různá využití, je to stále náročný úkol, jelikož v reálném prostředí přesnost detekce zhoršují odlišné úrovně osvětlení, líčení očí nebo různé předměty jako sluneční či dioptrické brýle. Proto jsou hledány nové a vhodnější způsoby detekce, přičemž díky rostoucímu výkonu počítačů jsou stále více rozvíjeny metody založené na strojovém učení.

Tato práce se zabývá detekcí zornic v obrazech. Nejprve jsou popsány některé již známé metody detekce založené na různých principech, dále jsou představeny veřejné datasety obrázků pro otestování řešení a následně jsou popsány různé způsoby vlastního řešení detektoru společně s výsledky testování těchto metod.

V první kapitole práce jsou blíže popsány metody detekce zornic v obraze včetně Houghovy transformace, která není primárně určena k detekci zornic, ale může být použita pro detekci kruhu. Další kapitola je věnována datasetům obrázků a jsou zde představeny datasety GI4E a BioID, které byly použity při testování vlastní implementace metod pro detekci zornic. V následující kapitole jsou popsány postupy navržených metod, přičemž důležitou součástí řešení je použití evolučních algoritmů v kombinaci s metodami z oblasti analýzy obrazu. Poslední kapitoly jsou věnovány výsledkům testování jednotlivých způsobů řešení, problémům daných metod a návrhům zlepšení.

2 Metody detekce zornic v obraze

Metod pro detekci zornic existuje v dnešní době spousta a v následujících kapitolách jsou popsána některá ze současných řešení. Nejprve je část věnována Houghově transformaci [2], což není algoritmus přímo pro detekci zornic, ale může být v upravené verzi použit pro detekci kruhu. Další metoda [4] je založena na prahování, detekci hran a následném hledání vhodného tvaru hrany. Nadcházející tři metody mají společného spoluautora, jímž je W. Fuhl. Algoritmy ExCuSe [5] a ElSe [6] využívají detekce hran, které poté upravují a filtrují pro nalezení nejvhodnějšího obrysu zornice. Následující popsaná metoda z článku PupilNet v2.0 [7] dosahuje díky použití konvolučních neuronových sítí velice dobrých výsledků i při testování na obrazech z reálného prostředí.

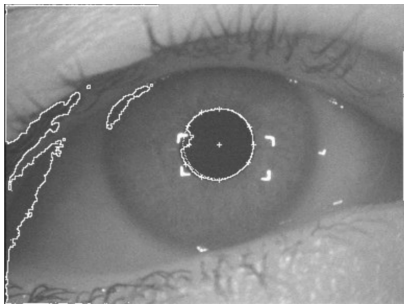
2.1 Houghova transformace

Houghova transformace byla navržena především pro detekci přímek [2], ovšem v upravené podobě může být použita pro identifikaci kruhů v obrazech. Tento způsob detekce kruhu je implementován i v knihovně OpenCV (knihovna pro práci z obrazy) [3]. Předpokladem využití této metody je obraz, ve kterém jsou detekovány hrany, jelikož v postupu se pracuje pouze s body, které jsou určeny jako hrany. Základem řešení je vyjádření kruhu pomocí tří parametrů: pozice středu(x, y) a poloměr (r). Postupně jsou procházeny všechny body, které byly detekovány jako hrany a spočítají se všechny možné středy kružnic, na kterých by mohl ležet daný bod. Pro uložení těchto výpočtů slouží třírozměrný čítač, kde každá buňka odpovídá kružnici s různými parametry x, y a r. Hodnota v čítači určuje počet bodů v obraze, které leží na kružnici s danými parametry. V průběhu výpočtu se hodnoty v čítači inkrementují a podle výsledného počtu bodů pro každou kružnici se určí, zda se daná kružnice v obraze nachází nebo ne. Popsaná metoda není primárně určena k detekci zornic, avšak může být v některých případech použita. Ve vlastním řešení jsem se částečně inspirovala a využila počítání bodů, které leží na kružnici s danými parametry.

2.2 Metoda založená na prahování a detekci hran

V metodě [4] navrhli autoři způsob detekce směru pohledu, jehož součástí je určení pozice zornice, která musí být velmi přesná, aby chyba detekce nebyla propagována v dalším výpočtu pro určení směru pohledu. K detekci zornice využili prahování podle jasu, což znamená, že v prvním kroku této metody jsou z obrázku vybrány jen pixely, které mají hodnotu jasu pod určitým prahem (blízkou černé) a je spočítáno těžiště těchto pixelů, které určuje střed zornice. Po prvním určení středu je iterativně prohledáváno okolí a střed je přepočítáván dokud nedojde k ustálení. Dalším krokem je určení obrysu zornice, k čemuž je použit Laplacianův operátor (obrázek 1).

Následně je ověřován tvar obrysu, aby odpovídal kruhu, což je určeno pomocí diagonál procházejícími body obrysu a nalezeným středem. Aby byl správný tvar ověřen, musí být vzdálenosti

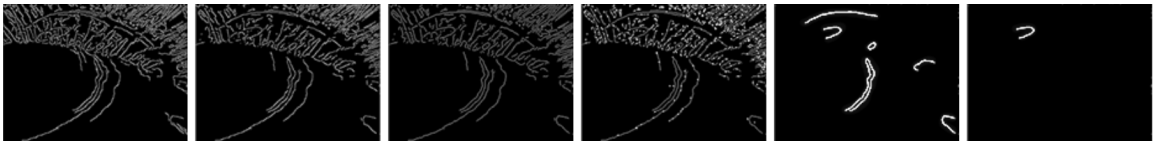


Obrázek 1: Nalezené ohraničení a střed zornice [4].

bodů, kterými prošla diagonála, ke středu zornice shodné. Pokud jsou nalezeny některé příliš vzdálené body, jsou odstraněny z výpočtu a je přepočítán poloměr zornice. Středy stejných diagonál jsou nakonec použity i pro přesnější určení výsledného středu zornice.

2.3 ExCuSe

Autoři metody nazvané ExCuSe (Exclusive Curve Selector) [5] měli za cíl vylepšit detekci zornic v reálném prostředí, kde je identifikace ztížena mimo jiné velmi proměnlivým prostředím (odlesky, změna osvětlení) nebo šumem z líčení či kontaktních čoček. Společně s novým algoritmem vytvořili i dataset ručně označených obrázků z reálného prostředí. Metoda pracuje s obrázkem ve stupních šedi a nejprve je jas znormalizován přes celý obraz a je spočítán histogram jasu. Pokud se v histogramu vyskytuje vrchol pro světlou oblast, je dále zornice hledána pomocí nalezení hran s využitím Cannyho detektoru. Nalezené hrany jsou následně filtrovány tak, aby byla nalezena zakřivená hrana, která ohraničuje nejtmavší jas v obraze (obrázek 2).



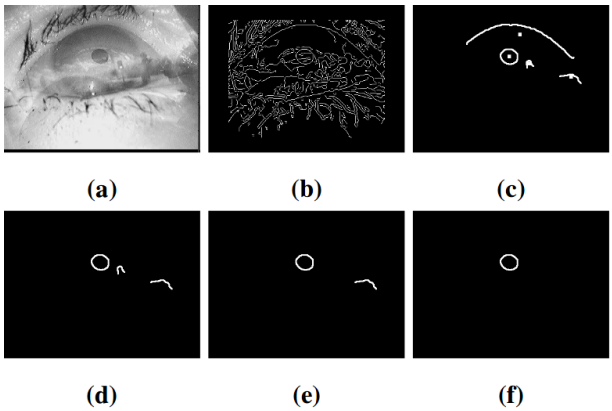
Obrázek 2: Postupná filtrace hran pro nalezení ohraničení zornice v metodě ExCuSe [5].

Pokud v prvním vypočteném histogramu obrázku není nalezen vrchol, je hrubá pozice zornice odhadnuta pomocí AIPF (Angular Integral Projection Function), kde je počítán součet hodnot pixelů ve směru osy x nebo y. Hrubý odhad pozice je následně opravován podle jasu v okolí nelezeného středu a pomocí obrazu s hranami a vyfiltrovaného obrazu podle prahu.

2.4 ElSe

ElSe (Ellipse Selector) metoda [6] je stejně jako ExCuSe metoda založena na detekci hran pomocí Cannyho detektoru a součástí práce bylo také ruční označení nových obrazů z reálného prostředí. Po detekci hran, která je prvním krokem algoritmu, jsou z obrazu odstraněny hrany, které by mohli narušovat ohraničení zornice (obrázek 3). Zbývající hrany jsou ohodnoceny na zá-

kladě jejich křivosti, hodnoty jasu a možnosti, jak lze hranou proložit elipsu. Pokud je nalezena vhodná elipsa, která sedí k některé z hran, je její popis vrácen jako výsledek. V případě, že není žádná elipsa nalezena, následuje další analýza. Aby byly urychleny výpočty konvolucí, které následují v dalších krocích, je nejprve sníženo rozlišení obrazu, poté jsou provedeny konvoluce s dvěma různými filtry a po přenásobení těchto dvou výsledků je vybrána pozice maximální hodnoty v obraze a určena jako hrubý střed zornice. Po převedení této pozice do původního obrazu ve větším rozlišení je pro určení přesnějšího středu nutná optimalizace pozice podle okolí v původním obraze.



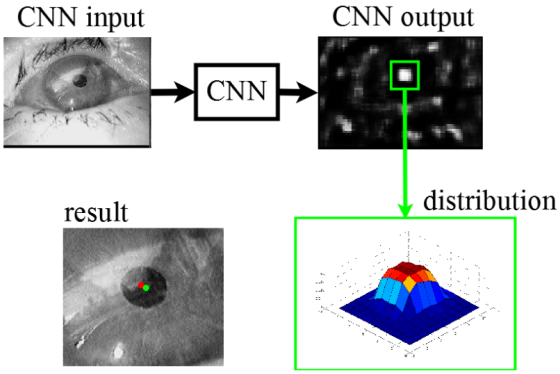
Obrázek 3: Postupná filtrace hran pro nalezení ohraničení zornice v metodě ElSe [6].

2.5 PupilNet v2.0

Metoda popsaná v článku PupilNet v2.0 [7] je oproti předchozím algoritmům založena na strojovém učení, přesněji jsou v ní využity konvoluční neuronové sítě. Přesnost této metody je velice dobrá a řešení není příliš časově náročné, výpočet probíhá v reálném čase bez použití grafické jednotky.

Autoři navrhují dvě konvoluční neuronové sítě za sebou, kdy je nejprve na obraze v menším rozlišení použita konvoluční neuronová síť s méně vrstvami, aby byl rychle získán hrubý odhad polohy zornice. Jelikož je v první fázi použit obraz s menším rozlišením, sníží se vliv šumu na výslednou detekci a díky hrubému odhadu je ve druhé fázi použita pouze oblast původního obrazu okolo dočasné pozice, což snižuje výpočetní náročnost sítě. Druhá konvoluční neuronová síť má za úkol zpřesnit nalezenou pozici a pracuje již s obrazy ve větším rozlišení. Cíl obou sítí je podobný, proto mají i stejnou architekturu. Jako první je provedena konvoluce, následuje podvzorkování podle průměru, poté druhá konvoluční vrstva a nakonec plně propojená neuronová síť. V konvolučních vrstvách první neuronové sítě autoři otestovali použití osmi a více filtrů, přičemž zvýšení počtu filterů mělo za následek drobné zlepšení, ale navýšilo výpočetní náročnost. U druhé neuronové sítě je použito v obou konvolučních fázích osm filterů.

Kromě řešení se dvěma sítěmi autoři navrhují z důvodu zrychlení výpočtu i postup s jednou konvoluční neuronovou sítí, jejímž vstupem jsou obrazy v menším rozlišení. Pro zpřesnění pozice u tohoto řešení je okolí výsledku převedeno na rozdělení pravděpodobnosti, podle kterého je následně vypočítána nová přesnější pozice. Znázornění tohoto postupu je na obrázku 4. Za cenu vyšší výpočetní náročnosti může být toto řešení spojeno s další neuronovou sítí podobně jako u postupu se dvěma sítěmi.



Obrázek 4: Postup pro zpřesnění pozice [7].

3 Datasety

Při vývoji metod pro analýzu obrazu je potřeba mít soubor obrázků, nad kterými je metoda testována. Pro detekci zornic jsou vytvářeny datasety z různých prostředí a obrázky mají odlišná rozlišení, mohou být barevné či ve stupních šedi a každý soubor obrázků může být vhodný pro otestování jiných vlastností testovaných metod. Některé z datasetů nemusí obsahovat referenční pozice zornic, pokud nejsou primárně určeny pro detekci zornic, ale například jen očí či částí obličeje, avšak i tyto datasety lze po dodatečné anotaci použít. Tvůrci datasetů jsou především sami autoři algoritmů a důležité jsou veřejné datasety, nad kterými může spouštět algoritmus kdokoli. Díky tomu se mohou mezi sebou porovnat přesnosti detekce různých metod na stejném souboru obrázků. V následujících kapitolách jsou popsány některé z dostupných datasetů pro detekci zornic.

3.1 BioID

BioID dataset [8] byl vytvořen pro všechny, co se zabývají detekcí obličeje v obraze, aby mohli porovnat kvalitu svých algoritmů s ostatními. Při tvorbě datasetu byl kladen důraz na přirozené podmínky, tudíž je v obrazech různé osvětlení, pozadí i velikost obličeje. Soubor se skládá z 1 521 obrázků, na kterých je vždy obličej jednoho z 23 různých osob. Příklady snímků z tohoto datasetu jsou na obrázku 5.

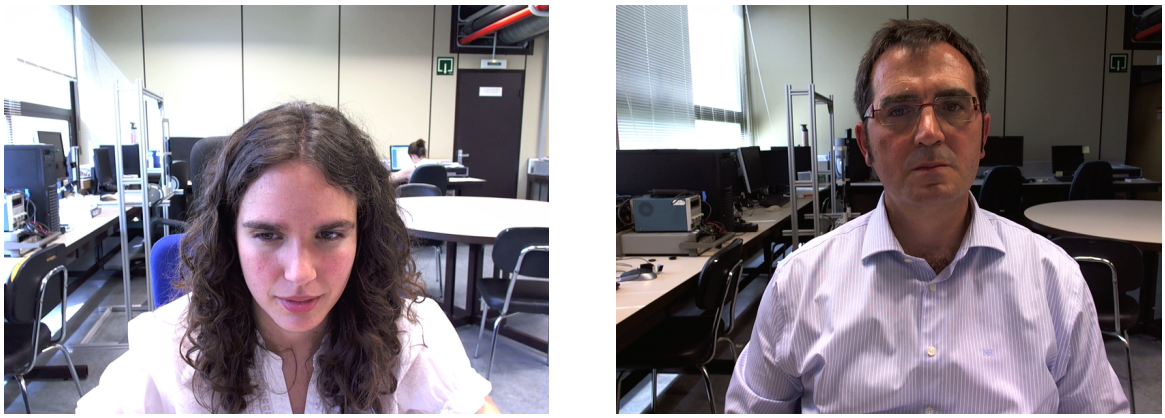
(a) (b)

Obrázek 5: Příklady obrázků z BioID datasetu [8].

Obrazy jsou ve stupních šedi a mají rozlišení 384x286 pixelů. K datasetu také patří ručně určené pozice oka a dostupné je i označení důležitých bodů na obličeji pro rozpoznání výrazu a analýzu obličeje.

3.2 GI4E

Dataset obrázků GI4E (Gaze Interaction For Everybody) [9] je veřejný dataset vytvořený pro detekci středu duhovky a rohů oka. Celkem je v datasetu 1 339 obrazů s 103 odlišnými osobami. Snímky jsou barevné s rozlišením 800x600 pixelů a byly zachyceny webovou kamerou. Ručně označené pozice, které jsou součástí datasetu, obsahují střed duhovky a rohy oka. Ukázky z tohoto datasetu jsou na obrázku 6. Jelikož je tento dataset určen pro detekci středu duhovky, mají osoby na všech obrazech alespoň částečně otevřené oči.

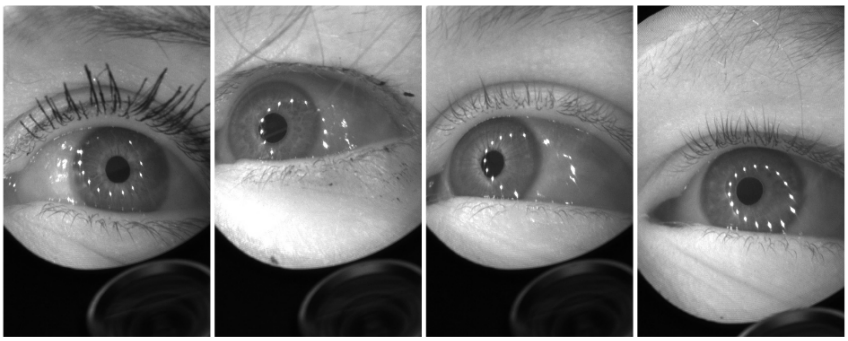


(a) (b)

Obrázek 6: Příklady obrázků z GI4E datasetu [9].

3.3 OpenEDS

OpenEDS (Open Eye Dataset) [10] je dataset obrázků získaných pomocí snímků z kamer připevněných na brýle pro virtuální realitu. Na snímcích je 152 různých osob a dataset obsahuje 12 759 obrázků, pro které jsou označené pozice důležitých částí oka.



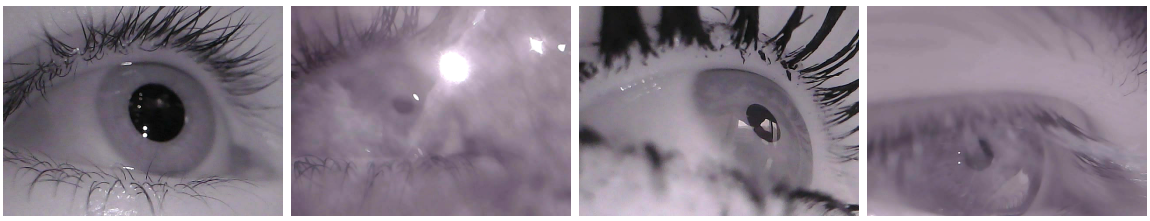
Obrázek 7: Příklady obrázků z OpenEDS datasetu [10].

Dále je součástí datasetu 252 690 neoznačených obrázků oka, 91 200 snímků z krátkých videí a data obsahující mračna bodů pro 143 párů očí. Ukázka obrázků z OpenEDS datasetu je

na obrázku 7. Oproti předchozím datasetům, které byly vytvořeny především pro ohodnocení metod detekující pozice částí oka ze snímků z reálného prostředí, je tento soubor obrázků určen pro otestování metod sledující směr pohledu u snímků oka s vyšším rozlišením. Takové metody mohou být využity například v aplikacích pro virtuální realitu.

3.4 LPW

Dataset LPW (labelled pupils in the wild) [11] obsahuje 66 videí s oblastí oka a je určen pro vývoj a ohodnocení metod pro detekci zornice u snímků s vysokým rozlišením. Videa byla natočena s cílem získat snímky z různého prostředí a s odlišným vzhledem osob. Na obrázcích je 22 různých osob a videa byla vytvořena pomocí kamery sledující oko, která byla připevněna vždy na hlavě dané osoby. Dataset obsahuje obrázky s odlišným osvětlením (venkovním i vnitřním), přičemž osoby na snímcích jsou různé národnosti a pohlaví, nosí brýle či kontaktní čočky nebo jsou silně nalíčené. Ukázky obrázků z LPW datasetu jsou na obrázku 8.

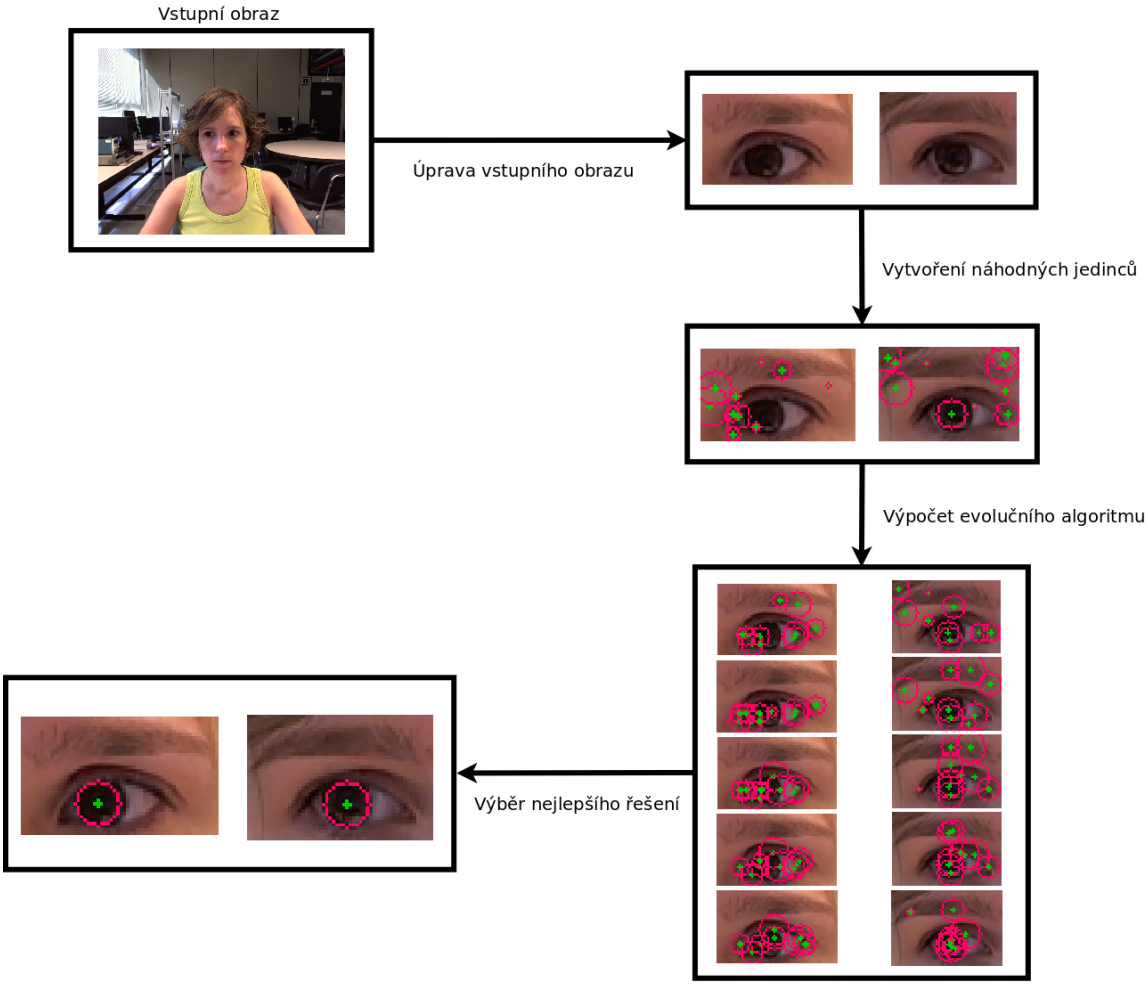


(a) (b) (c) (d)

Obrázek 8: Příklady obrázků z LPW datasetu [11].

4 Implementace detekce zornic v obraze

Cílem vlastní implementace algoritmu byla detekce pozice středu zornice v obraze. V následujících kapitolách jsou představeny navrhované způsoby řešení, jejichž základem je použití evolučních algoritmů s různými účelovými funkcemi. Detekce zornice v obraze spočívá v nalezení nejvhodnější pozice v obraze a evoluční algoritmy slouží k řešení optimalizačních úloh, proto jsem se rozhodla tento druh algoritmů ve vlastním řešení použít.



Obrázek 9: Znázornění postupu řešení. V prvním kroku je vstupní obraz oříznut tak, aby vznikly dva obrazy pro detekci zornice v pravém a v levém oku. V následujících krocích je hledáno co nejlepší řešení v každém obraze zvlášť. Nejprve jsou jedinci v obraze vygenerováni náhodně a následně jsou provedeny výpočty evolučního algoritmu, ve kterých se postupně mění parametry jedinců v populaci. Posledním krokem je výběr nejlepšího jedince, jehož pozice je výsledným řešením.

V první kapitole je popsán společný postup implementovaných řešení, další kapitola je věnována obecným principům evolučních algoritmů a popisu diferenciální evoluce a SOMA, což jsou dva evoluční algoritmy, které jsem ve vlastním řešení použila. V následující kapitole jsou před-

staveny účelové funkce použité v implementaci, přičemž jednotlivé funkce odpovídají postupu pro nalezení zornice v obraze. Poslední kapitola je věnována popisu jednotlivých způsobů řešení, které kombinují evoluční algoritmy diferenciální evoluce a SOMA s různými účelovými funkcemi.

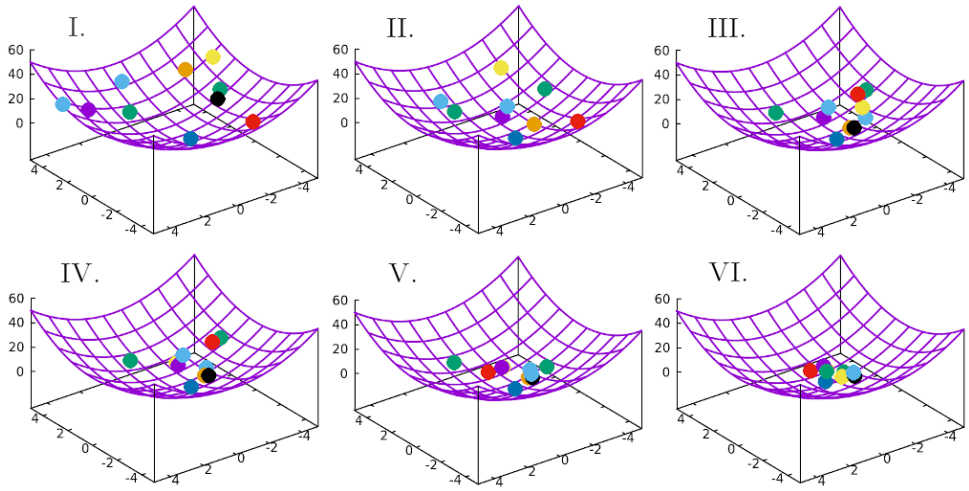
4.1 Postup řešení

Součástí vlastní implementace detekce zornic v obraze je více způsobů řešení, přičemž všechny jsou založené na stejném postupu s využitím evolučních algoritmů a liší se v použitých účelových funkcích a evolučních algoritmech. V této části je popsán společný postup těchto řešení.

Jednotlivé kroky, které jsou společné pro všechny metody, jsou znázorněny na obrázku 9. U každé metody je vstupní obraz vždy nejprve upraven, aby nebyl prohledáván celý, ale jen v oblasti oka. K tomuto účelu jsem vyžila detektor zájmových bodů obličeje z knihovny dlib [12], což mi umožnilo nalézt důležité body v okolí očí a vytvořit tak dva menší obrazy. Každý obraz je následně převeden do stupňů šedi a je proveden výpočet pro detekci zornice. K nalezení pozice zornice je využit evoluční algoritmus, jehož cílem je najít nejvhodnějšího jedince, který u tohoto řešení odpovídá části vstupního obrazu a parametry takového jedince jsou pozice v původním obraze a velikost. Pomocí těchto hodnot lze z parametrů jedince určit danou oblast ve vstupním obraze. V implementovaných řešeních jsou použity evoluční algoritmy SOMA a diferenciální evoluce, které jsou blíže popsány v části 4.2. Na začátku evolučního algoritmu jsou pozice v obraze vygenerovány náhodně a následně jsou v několika krocích podle daných pravidel, které jsou určeny zvoleným algoritmem, vybírány různé oblasti v obraze s cílem nalézt část obrazu, kde se vyskytuje zornice. K určení, jak moc vhodná je daná oblast obrazu, zda se na ní zornice vyskytuje či ne, slouží účelová funkce. V řešení jsem využila různé účelové funkce a jejich principy jsou popsány v části 4.3. Znázornění změn parametrů jedinců při běhu evolučního algoritmu je na obrázku 9. Na konci výpočtu evolučního algoritmu je vybráno nejvhodnější řešení ze všech prohledaných oblastí, přičemž pozice této oblasti v původním vstupním obraze je výslednou nalezenou pozicí zornice. Pokud je vstupní obraz součástí nějakého renomovaného datasetu, je v rámci testování tato pozice porovnána s referenční pozicí z daného datasetu a je určena přesnost nalezeného řešení.

4.2 Evoluční algoritmy

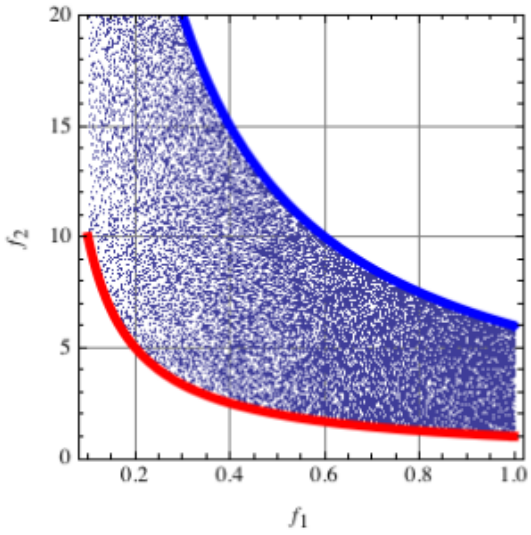
Evoluční algoritmy [14] jsou využívány k řešení různých optimalizačních úloh a ve vlastním řešení jsem je použila k nalezení pozice v obraze, která se co nejvíce blíží pozici zornice v obraze. Metod označovaných jako evoluční algoritmy je mnoho, přičemž některé se hodí jen pro specifické druhy úloh a jejich použití není univerzální. Příkladem je metoda optimalizace pomocí mravenčí kolonie, která je vhodná pro řešení problému obchodního cestujícího, avšak u jiných úloh nedosahuje tak dobrých výsledků. Algoritmů, které neřeší pouze specifické optimalizační problémy je dnes již spousta a mezi nejlépe hodnocené patří algoritmy diferenciální evoluce a SOMA [14], proto jsem tyto dva algoritmy zvolila pro vlastní řešení. V této kapitole jsou popsány obecné principy evolučních algoritmů a na závěr jsou představeny algoritmy diferenciální evoluce a SOMA.



Obrázek 10: Znázornění změny parametrů jedinců v jednotlivých generacích při běhu evolučního algoritmu.

Přesná implementace evolučních algoritmů se liší, všechny jsou však založeny na stejných principech, které jsou inspirovány evolucí v přírodě. Základem algoritmů jsou jedinci s různými hodnotami parametrů, kteří společně tvoří populaci, a cílem je nalézt takové jedince, kteří budou co nejvíce vyhovovat našim požadavkům. Na počátku je populace vygenerována náhodně a následně jsou podle různých pravidel, které se liší od použitého algoritmu, vytvořeni noví jedinci. Dále jsou jedinci ohodnoceni, k čemuž slouží účelová funkce. Tvorba této funkce je zásadním krokem, odvíjí se od řešené úlohy a nalezením minima či maxima této funkce jsou zjištěny optimální parametry k řešení dané úlohy. Vstupem účelové funkce jsou atributy jedinců a výstupní hodnota odpovídá ohodnocení jedince, z čehož vyplývá, že pokud je hledáno minimum této funkce, nejvhodnějším řešením jsou parametry jedince s co nejnižším ohodnocením. Jakmile jsou všichni jedinci v populaci ohodnoceni, je dalším krokem algoritmu výběr nejvhodnějších jedinců do další populace. Tato populace tvoří novou generaci, ze které jsou znovu tvořeni noví jedinci. Pro dosažení optimálního výsledku je obvykle vypočteno několik generací a nakonec je vybrán jeden nejvhodnější jedinec, jehož parametry jsou řešením dané optimalizační úlohy. Na obrázku 10 je

znázorněna postupná změna populace v jednotlivých generacích při hledání minima a z obrázku je viditelné, že se jedinci postupně přibližují nejvhodnějšímu řešení (globálnímu minimu).



Obrázek 11: Vyznačené Paretovy linie na množině možných řešení [14]. Červená křivka odpovídá linii při hledání minimálních hodnot, modrá křivka odpovídá hledání maximálních hodnot.

V některých případech může být potřeba použít pro ohodnocení jedinců více účelových funkcí a taková úloha se nazývá víceúčelová optimalizace. Při takové optimalizaci jsou porovnávány hodnoty více účelových funkcí najednou, což znamená, že pro každého jedince existuje více hodnot, podle kterých musí být porovnán s ostatními. Jelikož není porovnávána jen jedna hodnota, i výsledných řešení je nakonec více a odpovídají tzv. Paretově linii [14], což je množina řešení, které jsou všechny nejlepší a zároveň neporovnatelné (při porovnání jedinců může být sice hodnota jedné účelové funkce lepší, ale hodnota druhé funkce bude horší). Vyobrazení takové linie je na obrázku 11, kde je červenou křivkou označena množina nejvhodnějších řešení, pokud je hledáno minimum u obou funkcí, a modrá křivka odpovídá nejvhodnějším řešením při hledání maximálních hodnot.

4.2.1 Diferenciální evoluce

Algoritmus diferenciální evoluce [14] je prvním ze dvou evolučních algoritmů, které jsem ve svém řešení použila. Princip diferenciální evoluce je velice podobný genetickým algoritmům, které jsou inspirovány evolucí v přírodě. Na začátku algoritmu je populace vygenerována náhodně a tvorba nových jedinců je založena na tzv. mutaci a křížení. Nový jedinec je ze starého vytvořen vždy pomocí tří jiných náhodně vybraných jedinců, jejichž parametry jsou zkombinovány a dochází tak k mutaci. Následně je provedeno křížení, kdy je náhodně vybráno, zda bude mít nový jedinec hodnotu parametru stejnou jako původní jedinec, nebo převezme hodnotu vzniklou při mutaci. Nakonec je provedeno ohodnocení nového jedince pomocí účelové funkce a pokud je nový jedinec lepší než ten původní, je zařazen do nové populace. V případě, že má nový jedinec horší ohodno-

cení, stane se součástí nové generace původní jedinec. Při využití víceúčelové optimalizace jsou nejprve vytvořeni všichni noví jedinci, kteří společně s původními jedinci tvoří velkou populaci, ze které jsou následně vybráni nejvhodnější jedinci pro vytvoření nové generace. Po výpočtu několika generací je z poslední populace vybrán jedinec s nejlepším ohodnocením, jehož parametry jsou výsledným řešením úlohy.

4.2.2 SOMA

Evoluční algoritmus zvaný SOMA (samoorganizující se migrační algoritmus) [14] je druhým algoritmem použitým v implementaci. Tato metoda není založena na principech evoluce, jako je tomu u algoritmu diferenciální evoluce, je však inspirována jiným přírodním jevem. Průběh algoritmu má napodobovat pohyb inteligentních jedinců, kteří vzájemně spolupracují při řešení společného úkolu, například při hledání potravy. V reálném světě mohou být takoví jedinci mravenci, včely nebo smečka vlků. Na začátku algoritmu je jako u většiny metod náhodně vygenerována populace, ze které je na základě ohodnocení pomocí účelové funkce vybrán vedoucí jedinec. Poté následuje úprava parametrů ostatních jedinců z populace, která je přirovnávána k pohybu (migraci) jedinců a je nazývána migrační kolo (obdoba generací v metodě diferenciální evoluce). V migračním kole je pro všechny jedince vypočítán směr k vedoucímu jedinci a v tomto vypočteném směru provede každý jedinec vždy několik kroků, přičemž při každém kroku je provedeno ohodnocení a po vykonání všech kroků se jedinec přesune na pozici, kde bylo ohodnocení nejlepší. Po přesunu všech jedinců je vybrán nový vedoucí jedinec pro další migrační kolo. Aby nebyl pohyb jedinců příliš deterministický, je vždy náhodně vybráno, zda se daný parametr jedince při každém kroku změní podle vypočteného směru, nebo parametr zůstane nezměněn. Takto se jedinci nepohybují vždy přímo k vedoucímu jedinci, ale je prohledáván větší prostor a algoritmus tak lépe najde globální extrém. Po ukončení posledního migračního kola je opět zvolen vedoucí jedinec, jehož parametry jsou řešením dané úlohy.

4.3 Účelové funkce

Nezbytnou součástí evolučních algoritmů je účelová funkce, která je vždy vytvořena na základě optimalizační úlohy a nalezením globálního extrému (minima či maxima) této funkce je získáno řešení daného problému. Ve vlastním řešení jsem zvolila několik účelových funkcí, jejichž cílem je vyhodnotit, zda se v dané části obrazu vyskytuje zornice nebo ne. V tomto případě tak účelová funkce odpovídá metodě pro detekci zornice v obraze. Některé z navrhovaných metod jsou založené na základních principech bez použití strojového učení, jiné využívají složitějších algoritmů, jejichž součástí je učení pomocí trénovacího datasetu. Účelové funkce jsou obecně určeny k ohodnocení jedinců a cílem evolučního algoritmu je nalézt jedince s nejlepším ohodnocením. V případě navrhovaného řešení je cílem nalézt oblast ve vstupním obraze, kde se vyskytuje zornice. Jedince si proto lze představit jako oblasti obrazu (čtverec, kruh) a parametry takového jedince jsou pozice (souřadnice x a y odpovídající středu čtverce nebo kruhu) a velikost oblasti (velikost hrany čtverce či průměr kruhu). Znázornění náhodně vygenerovaných jedinců s odlišnými hodnotami parametrů (různé pozice a velikosti) je na obrázku 12.

Obrázek 12: Vyobrazení náhodně vygenerovaných jedinců ve vstupním obraze.

V následujících kapitolách budou blíže popsány metody detekce zornice použité ve vlastním řešení jako účelové funkce.

4.3.1 Jas a poloměr

Tato účelová funkce je základním řešením z navrhovaných metod pro detekci zornice v obraze a není v ní využito strojového učení. Jedincem, jehož parametry jsou optimalizovány, je kruh představující zornici v obraze, přičemž cílem je nalézt v obraze kruh s co nejtmavší barvou. Nejprve jsem pro ohodnocení jedince zvolila variantu, kdy je počítán pouze průměrný jas v kruhové oblasti obrazu dané parametry jedince (pozice v obraze, poloměr), přičemž čím menší je výsledná průměrná hodnota jasu, tím lepší je daný jedinec (je hledáno minimum účelové funkce). U takového řešení při běhu evolučního algoritmu konverguje parametr poloměru k jedničce, jelikož se tak nejjednodušeji zajistí nejmenší průměrný jas v dané kruhové oblasti. Proto jsem se rozhodla pro použití dvou účelových funkcí (společně s využitím principů víceúčelové optimalizace) a jako druhou účelovou funkci jsem zvolila hodnotu poloměru kruhu. Hodnota této účelové funkce by

měla být naopak maximální, čímž je dosaženo výsledků, které odpovídají kruhovým oblastem v obraze s co nejmenším průměrným jasem, ale s co největším poloměrem.

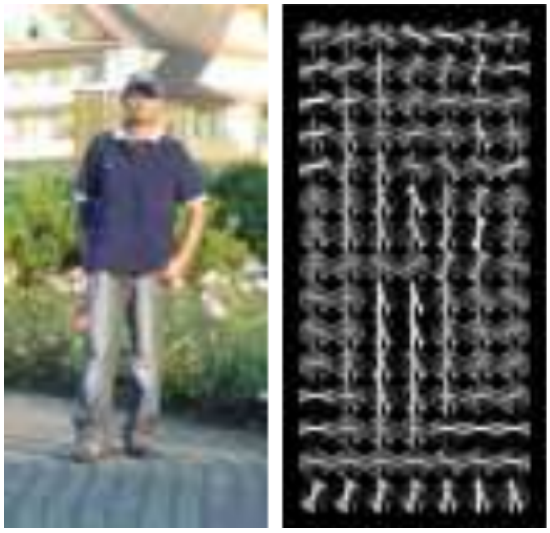
4.3.2 Jas a hrany

Předchozí účelová funkce 4.3.1 nebere v potaz rozdíl jasu (hranu) mezi zorničkou a duhovkou, proto jsem se rozhodla v dalším způsobu využít detekce hran pomocí Cannyho detektoru, který je implementovaný v knihovně OpenCV [13]. Tento detektor využívá principu dvojího prahování a je použit i v metodách pro detekci zornic nazvaných ExCuSe [5] a ElSe [6], které jsou popsané v částech 2.3 a 2.4. Cílem účelové funkce založené na detekci hran je najít kruh v obraze, na jehož obvodu bude co nejvíce bodů patřících do detekovaných hran. Při výpočtu jsou procházeny všechny detekované hrany a s určitou odchylkou je podle vzdálenosti od středu dané kružnice určeno, zda bod na kružnici patří nebo ne. Body ležící na kružnici jsou sčítány obdobně jako u Houghovy transformace kruhu popsané v části 2.1 a jejich počet má být co nejvyšší (je hledáno maximum funkce). Nalezení kruhu pomocí hran by mohlo být použito jako samostatné řešení, avšak pro zlepšení výsledků je k této funkci přidáno ještě zjištění průměrného jasu a je použita víceúčelová optimalizace obdobně jako u předchozího řešení 4.3.1. Společně s funkcí, která vypočte průměrný jas v kruhu, je v tomto řešení hledána co nejvhodnější kružnice v obraze podle detekovaných hran, přičemž v oblasti, kterou ohraničuje, má být co nejmenší průměrný jas.

4.3.3 HOG a SVM

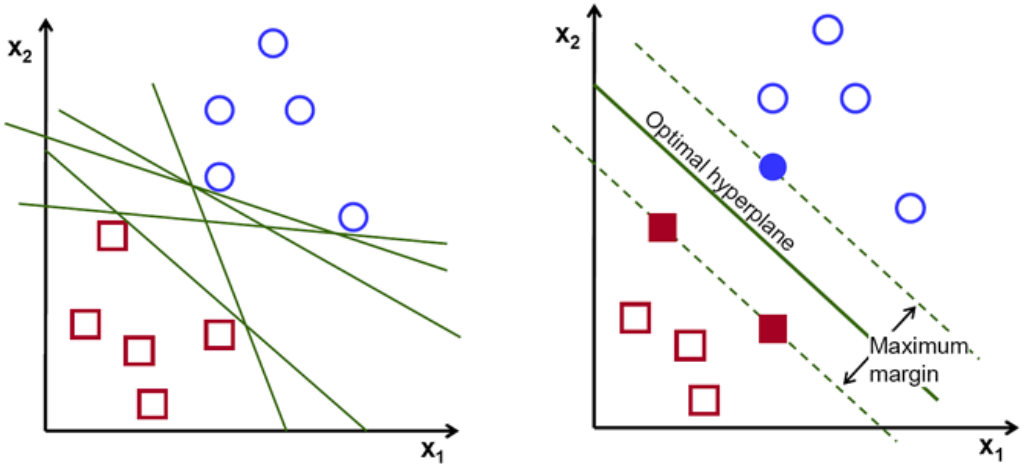
U této metody použité jako účelová funkce je již využito strojového učení, přesněji HOG deskriptoru společně s SVM klasifikátorem, což je jedna z úspěšných metod pro detekci objektů v obraze, proto jsem ji zvolila jako jedno z řešení pro vlastní detekci zornic v obraze. Princip HOG deskriptoru a SVM klasifikátoru je popsán v následujících odstavcích.

- Metoda HOG (Histogram Orientovaných Gradientů) [15] je používána pro detekci objektů v obraze a je založena na výpočtu gradientů podle rozdílu hodnot jasu jednotlivých pixelů. Obraz je rozdělen do několika buněk (například o velikosti 8x8 pixelů), přičemž směr a velikost gradientu jsou spočteny pro každý pixel a následně je vytvořen histogram těchto gradientů pro danou buňku, čímž vznikne pro každou buňku vektor hodnot. Ukázka vypočtených gradientů pro jednotlivé buňky v obraze je na obrázku 13. Získané vektory jsou dále normalizovány přes několik buněk (bloky), aby byl snížen vliv odlišného jasu v různých částech obrazu. Výsledkem algoritmu je vektor příznaků, který vznikne spojením vektorů ze všech bloků. Získaný vektor je následně předán klasifikátoru, jako je například klasifikátor SVM, který byl použit autory článku [15], nebo může být vektor použit jako vstup do neuronové sítě.



Obrázek 13: Vyobrazení vypočtených gradientů pro jednotlivé buňky v obraze [15].

- SVM (Support vector machines) [16] je metoda strojového učení s učitelem a je používána pro klasifikaci objektů dvou odlišných tříd. Cílem při učení je nalézt nadrovinu, která by oddělovala body patřící do jiné třídy a měla by vzdálenost k nejbližším bodům co největší. Příklad ve dvourozměrném prostoru je na obrázku 14. Pokud nejsou body lineárně separabilní, je problém převeden do vyšší dimenze, kde již lze najít vhodnou nadrovinu, která by body oddělila. Tato transformace se nazývá kernel a existuje více druhů kernelů, přičemž každý provádí výpočet jiným způsobem.



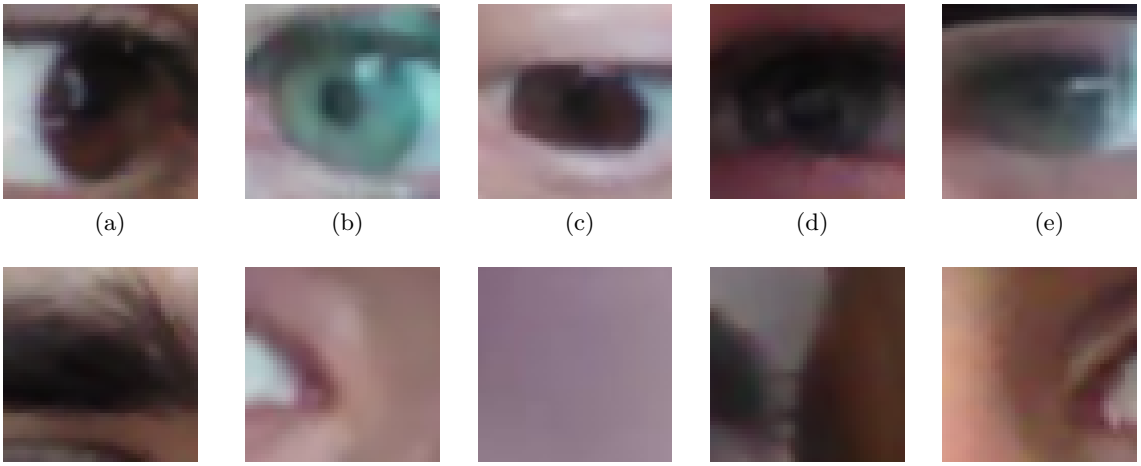
(a) Příklady možných řešení.

(b) Nejvhodnější řešení.

Obrázek 14: Ukázka možných způsobů oddělení bodů jiných tříd pomocí nadroviny (přímky ve dvourozměrném prostoru) 14a a výsledné nejvhodnější řešení 14b [17].

Ve vlastním řešení jsem využila implementaci HOG deskriptoru a SVM klasifikátoru z knihovny OpenCV [20]. Prvním krokem u tohoto řešení je natrénování klasifikátoru SVM pomocí tréno-

vacího datasetu obrázků, který obsahuje 4 649 pozitivních a stejný počet negativních obrázků. Všechny obrazy jsou čtvercové o velikosti 48x48 pixelů, přičemž pozitivní obrazy jsou obrazy zornice s duhovkou a negativní obrazy obsahují části oka a jeho okolí, kde se zornice nevyskytuje. Ukázka pozitivních i negativních obrazů z trénovacího datasetu je na obrázku 15. Při trénování je vždy celý trénovací obraz vstupem do HOG deskriptoru a výstup je použit pro trénování SVM klasifikátoru, jehož úkolem je oddělit od sebe pozitivní a negativní případy.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Obrázek 15: Ukázka pozitivních (15a až 15e) a negativních (15f až 15j) obrázků z trénovacího datasetu.

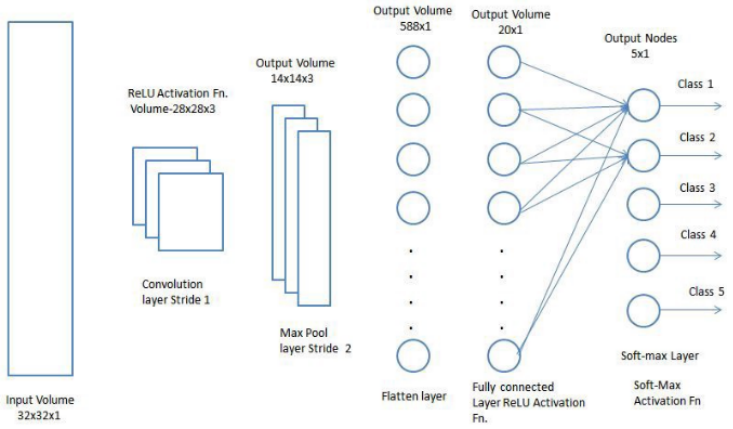
Natrénovaný SVM klasifikátor je využit v samotné účelové funkci při hledání optimálního jedince, jež je určen polohou a velikostí. Při výpočtu hodnoty účelové funkce je podle velikosti a polohy jedince určena čtvercová oblast v původním obraze, která je vstupem pro HOG deskriptor. Výstupní vektor z deskriptoru je použit jako vstup do natrénovaného klasifikátoru a vypočtená hodnota z SVM klasifikátoru je výslednou hodnotou účelové funkce a slouží tak k určení vhodnosti jedince. Výsledek z SVM klasifikátoru představuje v absolutní hodnotě vzdálenost daného vektoru (jedince) od nadroviny oddělující pozitivní a negativní případy, z čehož vyplývá, že čím větší je tato vzdálenost, tím vhodnější je jedinec.

4.3.4 Konvoluční neuronová síť

U posledního řešení pro detekci zornic jsem zvolila využití konvolučních neuronových sítí, jelikož v dnešní době je způsob detekce objektu v obraze pomocí konvolučních neuronových sítí velice často a úspěšně používán. V následujícím odstavci je blíže popsán obecný princip konvolučních neuronových sítí.

- Konvolučních neuronových sítí existuje již spousta druhů a s různými architekturami, přičemž oproti plně propojeným neuronovým sítím se v konvolučních sítích objevují tři nové vrstvy, které mohou být použity vícekrát za sebou a s různým nastavením. Jsou

to vrstvy konvoluční, ReLU (Rectified Linear Unit) a podvzorkovací vrstva (pooling). Úkolem těchto vrstev je získat z obrazu vektor příznaků, který je následně použit jako vstup do plně propojené neuronové sítě. V konvoluční vrstvě jsou provedeny konvoluce vstupního obrazu, v nichž jsou použity různé masky (filtry), jejichž počet a velikost je předem stanovena, avšak samotné hodnoty v maticích filtrů jsou nastavovány v průběhu učení sítě. Po konvoluční vrstvě obvykle následuje nelineární ReLU funkce, která záporné hodnoty nastaví na nulu a ostatní hodnoty ponechá nezměněné. Poslední vrstva slouží ke zmenšení velikosti obrazu, kdy je pro předem danou velikost okna z původního obrazu vypočtena pouze jedna hodnota pro výstupní zmenšený obraz. Výpočet může být proveden různými způsoby, například je vybrána největší hodnota z dané oblasti, nebo je vypočtena suma či průměr hodnot. Použitím těchto vrstev je získán určitý počet obrazů, které jsou následně převedeny do vektoru a využity jako vstup do plně propojené neuronové sítě, která slouží jako klasifikátor. Znázornění celého postupu je na obrázku 16.



Obrázek 16: Znázornění uspořádání jednotlivých vrstev konvoluční neuronové sítě [19].

Ve vlastním řešení jsem využila implementaci neuronové sítě z knihovny dlib [21]. Před použitím konvoluční neuronové sítě jako účelové funkce je potřeba síť natrénovat, přičemž k učení je použit stejný dataset obrázků jako v předchozím řešení 4.3.3 a vstupem do sítě je vždy celý obraz z datasetu. Cílem trénování je naučit síť, aby klasifikovala dvě třídy a to obrazy, kde se zornice vyskytuje, a obrazy bez zornice. Během výpočtu hodnoty účelové funkce je vstupem do již natrénované konvoluční neuronové sítě část prohledávaného obrazu, která je určena parametry jedince, jeho polohou a velikostí. Jako výsledná hodnota účelové funkce je brána vypočtená hodnota ze sítě, která udává, s jakou pravděpodobností patří daná část obrazu do třídy obrazů, kde se vyskytuje zornice. Čím větší je výsledná hodnota (pravděpodobnost), tím vhodnější je daný jedinec.

4.4 Způsoby řešení

Všechny metody použité v implementaci jsou založené na stejném principu s využitím evolučních algoritmů, který je blíže popsán v části 4.1. Způsoby řešení se však liší v použitých evolučních algoritmech a účelových funkcích.

Přehled základních způsobů řešení, které jsem v implementaci použila, je v tabulce 1. V prvním sloupci je vždy uveden zkrácený název metody, kde první část názvu udává použitý evoluční algoritmus, přesněji DE pro diferenciální evoluci a SOMA pro algoritmus SOMA. Druhá část názvu je odvozena od účelové funkce (JP - jas a poloměr, JH - jas a hrany, HOG - HOG a SVM, CNN - konvoluční neuronová síť). Pro každou metodu je v tabulce uveden evoluční algoritmus a účelové funkce, které jsou v daném způsobu řešení využity. V posledním sloupci je informace, zda je u dané metody potřeba strojové učení či ne. V následujících kapitolách jsou popsány principy a odlišnosti řešení při kombinaci evolučního algoritmu s danou účelovou funkcí.

	Evoluční algoritmus	**Účelová funkce**	**Učení**
DE_JP	diferenciální evoluce	jas a poloměr	ne
SOMA_JP	SOMA	jas a poloměr	ne
DE_JH	diferenciální evoluce	jas a hrany	ne
DE_HOG	diferenciální evoluce	HOG a SVM	ano
SOMA_HOG	SOMA	HOG a SVM	ano
DE_CNN	diferenciální evoluce	konvoluční neuronová síť	ano
SOMA_CNN	SOMA	konvoluční neuronová síť	ano

Tabulka 1: Přehled metod použitých ve vlastním řešení. První sloupec tabulky udává název metody, v dalších dvou jsou uvedeny evoluční algoritmy a účelové funkce použité v dané metodě a poslední sloupec udává, zda je u dané metody využito strojového učení nebo ne.

4.4.1 Evoluční algoritmus s účelovými funkcemi jas a poloměr (DE_ JP, SOMA_ JP)

Metody zkráceně nazvané DE_JP a SOMA_JP používají dvě základní účelové funkce, které jsou založené na jasu a poloměru kruhové oblasti v obraze (4.3.1) a jejichž cílem je nalézt co nejtmavší a zároveň co největší oblast. Při použití dvou účelových funkcí je obecně potřeba využít principů víceúčelové optimalizace a v takovém případě je vhodnou volbou evolučního algoritmu diferenciální evoluce, která umožňuje tyto principy snadno použít. Avšak u dvojice účelových funkcí použité v metodách DE_JP a SOMA_JP je možné provést výpočet bez využití víceúčelové optimalizace a lze tyto funkce kombinovat s diferenciální evolucí, kterou využívá metoda DE_JP, i algoritmem SOMA, který je využit v metodě SOMA_JP. V následujících odstavcích je popsán odlišný princip použití dvou účelových funkcí u těchto dvou metod.

- Metoda DE_JP využívá při výpočtu evoluční algoritmus diferenciální evoluce (4.2.1) společně s principem víceúčelové optimalizace. Při běhu algoritmu to znamená, že jsou na začátku výpočtu jedné generace vždy nejprve vytvořeni všichni noví jedinci a společně s těmi původními tvoří jednu velkou populaci. Jelikož má každý jedinec dvě hodnoty, podle kterých se určuje jeho vhodnost, existují v populaci jedinci, kteří jsou lepší, horší, ale i vzájemně neporovnatelní (jedna hodnota jedince je lepší, ale druhá je naopak horší). Z tohoto důvodu je nově vzniklá velká populace rozdělena do množin, které jsou mezi sebou vzájemně porovnatelné (v jedné množině jsou jedinci horší, ve druhé lepší), ale jedinci patřící do jedné množiny jsou vzájemně neporovnatelní (jsou stejně vhodní). Z takto rozdělené populace je nová generace vytvořena z jedinců, kteří patří do nejlepších množin. Na konci výpočtu je potřeba určit jedno nejvhodnější řešení, proto je z poslední vypočtené generace vybrána množina nejlepších vzájemně neporovnatelných jedinců (tzv. Paretova linie), jejichž parametry jsou následně zprůměrovány a nově vypočtené hodnoty jsou výsledné řešení. Příklad takové množiny a výsledného řešení vytvořeného z průměru hodnot je na obrázku 17.

(a) Množina nejlepších řešení (b) Výsledné řešení

Obrázek 17: Příklad množiny nejlepších jedinců 17a a výsledného zprůměrovaného řešení 17b.

- U metody SOMA_JP je jako evoluční algoritmus použita SOMA (4.2.2), přičemž u tohoto evolučního algoritmu nelze snadno využít principů víceúčelové optimalizace jako je tomu u diferenciální evoluce. Avšak u této metody lze použít dvě účelové funkce, tedy dvě

hodnoty pro ohodnocení jedince, bez principů víceúčelové optimalizace. V případě použití dvou účelových funkcí se nabízí jednoduché řešení a to sečíst výsledky z obou účelových funkcí za účelem vytvoření jen jedné hodnoty pro určení vhodnosti jedince a použití algoritmu SOMA stejně jako v případě, kdy by byla použita jen jedna účelová funkce. V takovém řešení nastává jeden problém, pokud jsou hodnoty jedné účelové funkce obecně vyšší než hodnoty druhé funkce (jejich obor hodnot není shodný), ovlivňuje výsledné ohodnocení jedince převážně jen ta funkce, jejíž hodnoty jsou větší, a druhá funkce nemá na výsledek ohodnocení žádný nebo jen minimální vliv. U této metody nabývá první funkce založená na jasu hodnot od 0 do 255 a druhá funkce, která udává velikost oblasti, má obor hodnot od 1 do čtvrtiny velikosti vstupního obrazu (při velikosti obrazu 40x40 pixelů je maximální velikost oblasti 10), což znamená, že mnohem větší vliv na vhodnost jedince má první funkce. Jedním ze způsobů, jak se tohoto problému zbavit, je převést obor hodnot obou funkcí na stejný interval, například od 0 do 1, aby měly obě funkce stejný vliv. Pro takový výpočet je pouze potřeba znát minimální a maximální hodnoty, kterých mohou obě funkce nabývat, a tyto hodnoty jsou u účelových funkcí pro toto řešení známé. Cílem kombinace účelových funkcí použitých v metodě SOMA_JP je nalézt co nejtmavší oblast o co největší velikosti, pokud ale není proveden přepočet hodnot účelových funkcí na stejný interval, je potlačen vliv hledání co největší oblasti a výsledek konverguje k oblastem s minimální velikostí, jelikož tak se nejsnadněji dosáhne nejmenšího průměrného jasu. Rozdíl ve výsledných řešeních je ukázán na obrázku 18, přičemž na obrázku 18a lze vidět, jak výsledné řešení zkonvergovalo k oblasti o minimální velikosti.

(a) Výsledek při rozdílných oborech hodnot.

(b) Výsledek při shodných oborech hodnot.

Obrázek 18: Ukázka výsledných řešení metod s rozdílnými 18a a shodnými 18b obory hodnot účelových funkcí založených na jasu a velikosti oblasti v obraze (4.3.1). Na obrázku 18a je vidět výsledné řešení, které zkonvergovalo k oblasti s minimální velikostí.

4.4.2 Diferenciální evoluce s účelovými funkcemi jas a hrany (DE_JH)

Metoda se zkratkou DE_JH používá obdobně jako předchozí metody (DE_JP, SOMA_JP) dvě účelové funkce. U tohoto řešení je pro zlepšení přesnosti detekce místo funkce odpovídající poloměru kruhové oblasti použita funkce založená na hranách v obraze a druhou funkcí je průměrný

jas oblasti (4.3.2). Účelem kombinace těchto dvou funkcí je nalézt co nejvhodnější kružnici v obraze podle detekovaných hran, přičemž v oblasti, kterou ohraničuje, má být co nejmenší průměrný jas. Jelikož nelze snadno určit obor hodnot funkce založené na detekci hran, jejíž výstupem je počet bodů na okraji oblasti, které jsou detekované jako hranové body, nelze výsledky obou účelových funkcí normalizovat a použít výpočet jako v metodě (SOMA_JP) s algoritmem SOMA. Proto je u tohoto řešení vhodným evolučním algoritmem diferenciální evoluce (4.2.1) společně s využitím víceúčelové optimalizace.

Postup pro tvorbu populací a výběr nejlepšího řešení při použití víceúčelové optimalizace společně s diferenciální evolucí je stejný jako v metodě DE_JP. Při výpočtu v každé generaci jsou nejprve vytvořeni všichni jedinci, společně s původními jsou rozděleni do množin se vzájemně neporovnatelnými jedinci a z nejlepších množin je vytvořena nová populace. Po proběhnutí všech generací je výsledné řešení vypočteno jako průměr parametrů jedinců z nejlepší množiny.

Hrany v obraze jsou detekovány pomocí Cannyho detektoru hran, přičemž detekce je provedena vždy jen jednou pro jeden obraz s okolím oka a při běhu evolučního algoritmu a výpočtu hodnot účelové funkce je počítáno již s předem detekovanými hranami. Cílem tohoto způsobu řešení je najít co nejtmavší oblast, která je zároveň ohraničena co nejvíce body patřícími do detekovaných hran. Ukázka detekovaných hran ve vstupním obraze společně s výsledným řešením je na obrázku 19.

(a) Detekované hrany (b) Výsledné řešení

Obrázek 19: Ukázka detekce hran ve vstupním obraze 19a a výsledného řešení 19b.

4.4.3 Evoluční algoritmus s účelovou funkcí HOG a SVM (DE_HOG, SOMA_HOG)

V metodách DE_HOG a SOMA_HOG je oproti předchozím řešením využito strojové učení, přesněji účelová funkce, která je v těchto dvou metodách použita, odpovídá detekci zornice pomocí HOG deskriptoru a SVM klasifikátoru (4.3.3), což je jedna z úspěšných metod pro detekci objektů. Účelovou funkci založenou na HOG deskriptoru a SVM klasifikátoru lze zkombinovat s diferenciální evolucí, která je použita v metodě DE_HOG, i algoritmem SOMA použitým v metodě SOMA_HOG. Před výpočtem metod je potřeba natrénovat SVM klasifikátor a samotný výpočet je u obou metod velmi podobný, jelikož kombinují jednu účelovou funkci s evolučním algoritmem a takový postup odpovídá obecnému principu evolučních algoritmů (4.2). Cílem je

najít jedince s nejlepším ohodnocením pomocí účelové funkce, ve které je vždy vypočten HOG deskriptor pro danou oblast obrazu a vypočtený vektor je vstupem do předem natrénovaného SVM klasifikátoru, který určí výslednou vhodnost jedince. Na konci výpočtu je vždy vybrán nejlépe ohodnocený jedinec.

4.4.4 Evoluční algoritmus s účelovou funkcí konvoluční neuronová síť (DE_CNN, SOMA_CNN)

V metodách DE_CNN a SOMA_CNN je použita kombinace evolučního algoritmu (4.2) a konvoluční neuronové sítě, která slouží jako účelová funkce (4.3.4). Detekce objektů pomocí konvoluční neuronové sítě je v dnešní době velice často a úspěšně používána, proto jsem tento způsob řešení zvolila ve své implementaci. Metoda DE_CNN používá jako evoluční algoritmus diferenciální evoluci a metoda SOMA_CNN algoritmus SOMA, přičemž postup metod je velice podobný jako u metod DE_HOG a SOMA_HOG popsaných v předchozí části. Konvoluční neuronová síť je nejprve potřeba natrénovat a již natrénovaná síť je využita ve výpočtu účelové funkce. Při ohodnocení jedince je oblast obrazu, která je dána parametry jedince, použita jako vstup do konvoluční neuronové sítě a vypočtená hodnota odpovídající pravděpodobnosti, že se v dané oblasti vyskytuje zornice, udává výsledné ohodnocení jedince. Během výpočtu je hledán jedinec s co nejlepším ohodnocením a jeho parametry určují výsledné řešení.

5 Přesnost a rychlost implementovaných řešení

Implementované způsoby řešení detekují zornice v obraze s různou přesností a rychlostí. Obsahem této kapitoly jsou naměřené rychlosti a vypočtená přesnost detekce jednotlivých metod, včetně různých verzí metod s určitým nastavením dílčích algoritmů, které jsou součástí daných řešení. Dále jsou metody porovnány mezi sebou a jsou srovnány s metodami ExCuSe [5] a ElSe [6] popsanými v částech 2.3, 2.4. V závěru kapitoly jsou navržena zlepšení implementovaných řešení. Výpočty použité v rámci zjišťování přesnosti či rychlosti určité metody jsou popsány v následujících odstavcích.

- Přesnost jednotlivých řešení byla testována na dvou renomovaných datasetech a to GI4E a BioID, jejichž stručný popis je v části 3. Nad každým obrazem z datasetu je proveden výpočet pro detekci zornic a pro všechna výsledná řešení je vypočtena vzdálenost nalezených pozic od referenčních hodnot z datasetu, čímž pro každý obraz vzniknou dvě hodnoty (pro levé a pravé oko), jejichž průměr udává výslednou chybu detekce (v pixelech) pro daný obraz. Jelikož mohou být obrazy různé velikosti a vzájemné porovnání chyb může být zkreslené v závislosti na velikosti vstupního obrazu, je výsledná hodnota chyby detekce normována. Výpočet normalizace je proveden podělením vypočtené hodnoty chyby vzdáleností (v pixelech) referenčních pozic pro levé a pravé oko, čímž je nově vypočtená hodnota nezávislá na velikosti vstupního obrazu. Jednou z hodnot určující přesnost detekce určité metody je zprůměrovaná normovaná chyba přes všechny obrazy z datasetu. Dále je normalizovaná chyba pro každý obraz použita pro výpočet distribuční funkce, která je využita pro porovnání přesnosti detekce jednotlivých metod. Pro další srovnání přesnosti různých způsobů řešení je vypočtena úspěšnost detekce nad daným datasetem, přičemž za správně detekovaný obraz se počítá ten, jehož normovaná odchylka je menší nebo rovna 0,25, a úspěšnost detekce nad daným datasetem je spočtena jako podíl počtu úspěšně detekovaných obrazů a celkového počtu obrazů v datasetu. Výpočet pro normalizaci chyby a hraniční hodnota 0,25 pro úspěšně detekovaný obraz je převzata od autorů BioID datasetu.

- Rychlost výpočtu jednotlivých metod je měřena jako doba od okamžiku, kdy je již vstupem do výpočtu detekce obraz s okolím levého nebo pravého oka, až po získání konečného výsledku detekce zornice v obraze. První část postupu pro nalezení okolí oka v měření času zahrnuta není, jelikož k takovému výpočtu může být použito více způsobů řešení a cílem implementace nebylo tuto část algoritmu optimalizovat. Zároveň je takto možné porovnat rychlost detekce zornic s případy, kdy je vstupním obrazem pouze oblast v okolí oka a oříznutí obrazu není součástí výpočtu. Pro porovnání rychlosti jednotlivých řešení je spočtena doba výpočtu u každého obrazu z datasetu (GI4E a BioID) a hodnota je zprůměrována podle počtu obrazů v daném datasetu. Veškeré měření bylo provedeno na počítači s procesorem Intel Core i5-7200U, RAM 16GB a grafickou kartou NVIDIA GeForce 940MX.

5.1 Diferenciální evoluce, jas a poloměr (DE_JP)

Metoda DE_JP (4.4.1) je jednou z metod využívající základní účelové funkce a parametry, které se nastavují v každé verzi metody, jsou spojené pouze s evolučním algoritmem, konkrétně diferenciální evolucí. U tohoto algoritmu nastavuji čtyři parametry. První určuje počet jedinců tvořící populaci (velikost populace), druhý udává, kolikrát se populace změní v průběhu výpočtu (počet generací). Poslední dva parametry ovlivňují, jak moc jsou argumenty jedinců při tvorbě nové generace mutovány a kříženy. Čím větší jsou hodnoty těchto parametrů, tím více dochází k mutaci a křížení, což znamená, že noví jedinci jsou více odlišní od těch původních. Přehled jednotlivých verzí metody s jejich označením a nastavením parametrů je v tabulce 2.

	Velikost populace	**Počet generací**	**Parametr mutace**	**Parametr křížení**
5/10/4/8	5	10	0,4	0,8
5/20/4/8	5	20	0,4	0,8
10/10/4/8	10	10	0,4	0,8
10/20/4/8	10	20	0,4	0,8
10/30/4/8	10	30	0,4	0,8
20/10/4/8	20	10	0,4	0,8
30/10/4/8	30	10	0,4	0,8
30/30/4/8	30	30	0,4	0,8
10/20/3/7	10	20	0,3	0,7
10/20/5/9	10	20	0,5	0,9

Tabulka 2: Přehled verzí metody DE_JP. V prvním sloupci tabulky je uvedeno označení verze, v dalších čtyřech sloupcích jsou hodnoty udávající nastavení diferenciální evoluce (velikost populace, počet generací, parametr mutace a křížení).

Celkem bylo otestováno deset verzí metody DE_JP. V prvních osmi jsou změněny pouze parametry velikosti populace a počtu generací za účelem porovnání vlivu těchto parametrů na přesnost a rychlost detekce. Z principu algoritmu diferenciální evoluce vyplývá, že čím větší je populace a čím více je počítáno generací, tím delší je doba běhu. Aby nebyla rychlost detekce příliš velká, byl otestován počet generací 10, 20 a 30. Jelikož má tento optimalizační problém relativně malý prostor řešení, dá se předpokládat, že velikost populace 10 až 20 by měla být dostačující, proto jsou testovány verze s 10 a 20 jedinci v populaci. Pro zjištění, jaký vliv na detekci bude mít větší populace, byly otestovány i verze s 30 jedinci, a jelikož má být tento způsob řešení především rychlý, byl počet jedinců nastaven u dvou verzí na 5 jedinců, aby bylo zjištěno, jak se

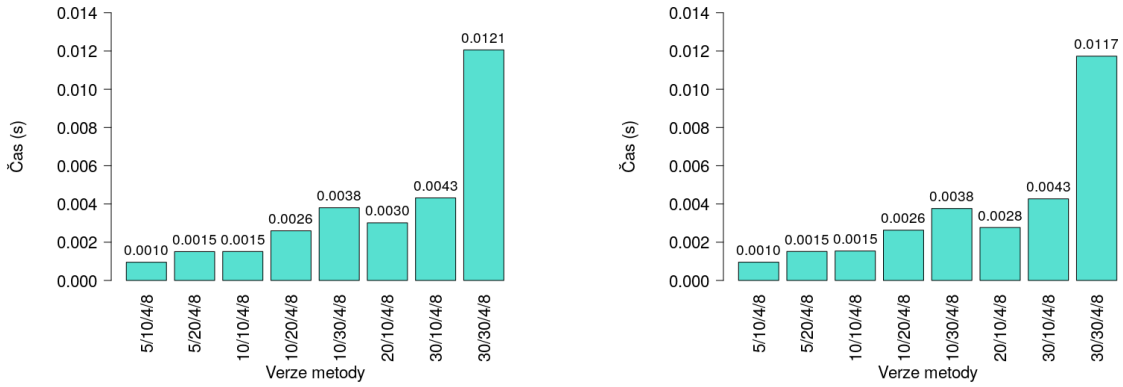
zhorší přesnost detekce na úkor rychlejšího výpočtu. Poslední dvě testované verze (10/20/3/7, 10/20/5/9) mají naopak rozdílné parametry mutace a křížení (v jedné verzi jsou parametry snížené, ve druhé naopak zvýšené) za účelem otestování vlivu těchto parametrů na přesnost detekce. Pro všechny verze metody byla spočtena průměrná chyba, průměrná normovaná chyba, úspěšnost detekce a doba výpočtu nad datasety GI4E a BioID a výsledky těchto měření pro oba datasety jsou v tabulce 3.

	GI4E				**BioID**			
	Chyba (px)	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**	**Chyba (px)**	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**
5/10/4/8	6,9	0,094	95,35	0,0010	6,1	0,111	95,96	0,0010
5/20/4/8	6,3	0,085	94,86	0,0015	5,9	0,106	95,10	0,0015
10/10/4/8	5,7	0,078	96,93	0,0015	5,6	0,100	96,75	0,0015
10/20/4/8	5,1	0,070	96,40	0,0026	5,3	0,096	95,73	0,0026
10/30/4/8	5,0	0,068	95,11	0,0038	5,3	0,095	94,97	0,0038
20/10/4/8	5,3	0,072	96,04	0,0030	5,3	0,095	96,55	0,0028
30/10/4/8	4,9	0,066	96,68	0,0043	5,2	0,095	95,33	0,0043
30/30/4/8	4,6	0,062	94,74	0,0121	5,1	0,092	93,92	0,0117
10/20/3/7	5,5	0,074	93,69	0,0026	5,5	0,098	93,95	0,0029
10/20/5/9	5,5	0,075	96,84	0,0026	5,4	0,097	96,98	0,0027

Tabulka 3: Výsledky měření přesnosti a rychlosti jednotlivých verzí metody DE_JP. V prvním sloupci tabulky je označení verze metody, následující čtyři sloupce uvádějí hodnoty pro dataset GI4E a poslední čtyři sloupce pro dataset BioID. Ve sloupci Chyba je průměrná chyba detekce v pixelech, sloupec Norm. udává průměrnou normovanou chybu v pixelech, Úsp. je úspěšnost detekce v procentech a Čas udává dobu běhu algoritmu v sekundách.

Z výsledků měření je patrné, že průměrná chyba detekce klesá se zvyšujícím se počtem generací a velikostí populace, což odpovídá principu evolučních algoritmů, jelikož více generací a větší populace zajišťují větší počet vytvořených a otestovaných jedinců, čímž se zvětšuje šance na nalezení nejvhodnějšího jedince. Počet testovaných jedinců má naopak negativní vliv na dobu běhu algoritmu, tudíž řešení s menším počtem jedinců v populaci a s méně generacemi mají menší průměrné časy detekce. Například nad datasetem GI4E má verze 10/10/4/8 chybu detekce 5,7 px (0.078 px) a rychlost 0,0015 s, na druhé straně verze s třikrát větší populací 30/10/4/8 má chybu 4,9 px (0,066 px) a rychlost 0,0043 s. Z vypočtené chyby detekce a normované chyby pro verze 10/20/4/8, 10/20/3/7 a 10/20/5/9 vyplývá, že snížení ani navýšení hodnot parametrů mutace

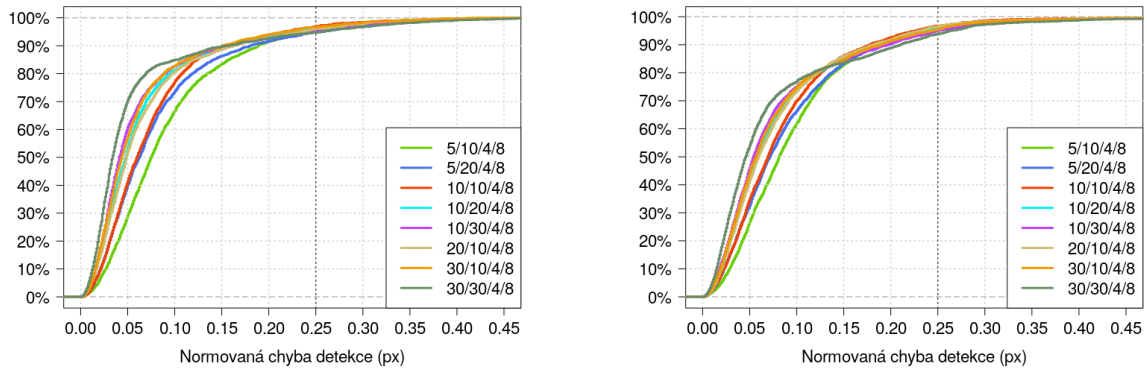
a křížení nemá pozitivní vliv na přesnost detekce, jelikož obě verze s upravenými hodnotami dosahují horší přesnosti. Při porovnání výsledků pro oba datasety je zřejmé, že detekce pro BioID dataset jsou méně přesné, což potvrzuje, že dataset obsahuje obrazy náročnější pro detekci.



(a) GI4E (b) BioID

Obrázek 20: Vyobrazení průměrné rychlosti detekce jednotlivých verzí metody DE_JP nad datasetem GI4E 20a a BioID 20b.

Na obrázku 20 je vyobrazeno porovnání doby běhu jednotlivých verzí metody a z obrázku lze vyčíst, že při navýšení počtu generací nebo velikosti populace se zvýší průměrný čas detekce. Zda je větší hodnota pro velikost populace nebo počet generací nemá na dobu běhu vliv, proto mají verze 10/20/4/8 a 20/10/4/8 velice podobný průměrný čas.

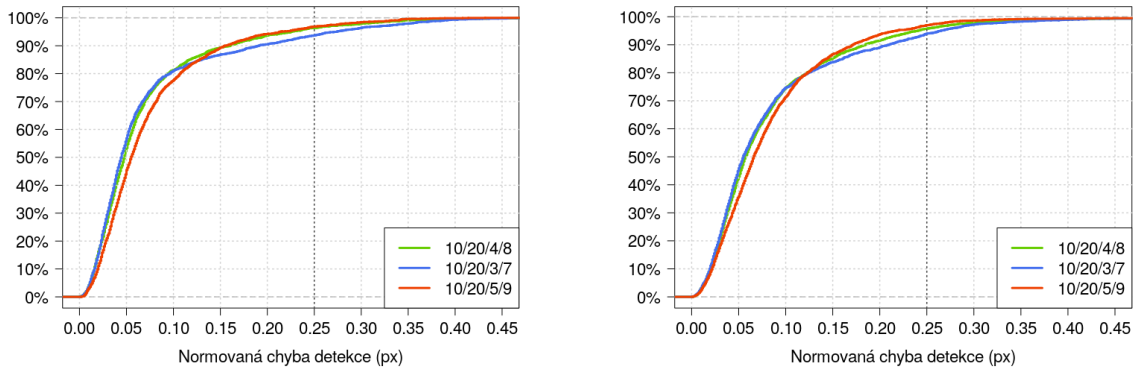


(a) GI4E (b) BioID

Obrázek 21: Srovnání přesnosti detekce jednotlivých verzí metody DE_JP nad datasetem GI4E 21a a BioID 21b.

Pro porovnání přesnosti detekce jednotlivých verzí metod s odlišnými parametry byly spočteny distribuční funkce normalizované chyby detekce. Na obrázku 21 je vyobrazeno porovnání

verzí, ve kterých jsou nastavené odlišné velikosti populace a počty generací. Z vyobrazení je patrné, že metoda s největší populací a nejvíce generacemi (30/30/4/8) dosahuje nejpřesnější detekce. Dále jsou na tom podobně verze 10/30/4/8, 30/10/4/8, 10/20/4/8 a 20/10/4/8, o něco horší detekci mají 10/10/4/8, 5/20/4/8 a nejméně přesná je verze s nejmenší populací a s nejméně generacemi (5/10/4/8). Z obrázku 21 i výsledků v tabulce 3 je také patrné, že i když má verze metody přesnější detekci například v 80 % případů (má nižší průměrnou chybu), dosahuje podobné i horší úspěšnosti detekce (úspěšně detekované snímky mají normovanou chybu pod 0,25 px) než verze, která má detekci méně přesnou.



(a) GI4E

(b) BioID

Obrázek 22: Srovnání přesnosti detekce verzí metody DE_JP s odlišnými parametry mutace a křížení nad datasetem GI4E 22a a BioID 22b.

Na obrázku 22 je vyobrazeno porovnání verzí s odlišným nastavením parametrů mutace a křížení a je zde vidět, že verze s vyššími hodnotami parametrů 10/20/5/9 má sice úspěšnost detekce podobnou jako verze 10/20/4/8, ale přesnost detekce má menší, naopak verze s nižšími hodnotami 10/20/3/7 má podobnou přesnost detekce u 80 % snímků, ale úspěšnost má horší než 10/20/4/8.

5.2 SOMA, jas a poloměr (SOMA_JP)

V metodě SOMA_JP (4.4.1) je použita SOMA jako evoluční algoritmus a účelové funkce založené na průměrném jasu a poloměru. U těchto základních funkcí nejsou žádné parametry, které by se daly nastavit, tudíž u této metody měním jen hodnoty související s evolučním algoritmem, podobně jako u předchozí testované metody (DE_JP). Parametrů, které lze u algoritmu SOMA nastavit, je pět. První dva jsou velikost populace, která určuje, kolik jedinců bude na začátku výpočtu vygenerováno, a počet migrací, což udává, kolikrát bude vypočten posun jedinců k současnému vedoucímu jedinci. Další tři parametry souvisejí s tím, jakým způsobem budou provedeny kroky jednoho jedince při pohybu směrem k vedoucímu jedinci. Jeden parametr udává, jak velké kroky budou při výpočtu provedeny (jak moc hustě bude prohledáván prostor řešení), druhý ovlivňuje, kolik těchto kroků bude provedeno (jak moc dlouhá bude cesta jednoho jedince) a třetím parametrem je zaručena náhoda ve výpočtu a určuje, s jakou pravděpodobností se bude jedinec v určitém směru pohybovat směrem k vedoucímu, v opačném případě zůstane v daném směru stát (daný atribut jedince bude nezměněn).

	Velikost populace	**Počet migrací**	**Velikost kroku**
8/25/21	8	25	0,21
10/20/21	10	20	0,21
10/20/51	10	20	0,51
20/20/21	20	20	0,21
20/20/51	20	20	0,51
20/30/51	20	30	0,51

Tabulka 4: Přehled verzí metody SOMA_JP. V prvním sloupci tabulky je uvedeno označení verze, v dalších třech sloupcích jsou hodnoty udávající nastavení algoritmu SOMA (velikost populace, počet migrací a velikost kroku, která určuje, jak hustě bude prohledáván prostor řešení).

Jednotlivé verze metody, jejichž výsledky testování jsou v této kapitole popsány, jsou v tabulce 4. Podle zdroje [14] je pro parametr ovlivňující délku cesty jedince nastavení na hodnotu 3 dostačující pro všechny úlohy a jelikož tento problém má menší prostor řešení, je u všech verzí této metody nastaven parametr na hodnotu 2. Experimentálně jsem ověřila, že toto snížení hodnoty nemá na přesnost výsledku vliv. Dalším parametrem, který zůstává stejný pro všechny verze, je parametr pro určení, s jakou pravděpodobností se bude jedinec pohybovat v daném směru. Tato hodnota je nastavena na 0,3 (30 %), přičemž jsem zjistila, že zmenšení hodnoty nemá u tohoto řešení na výsledek vliv a navýšení výslednou přesnost zhoršuje pravděpodobně

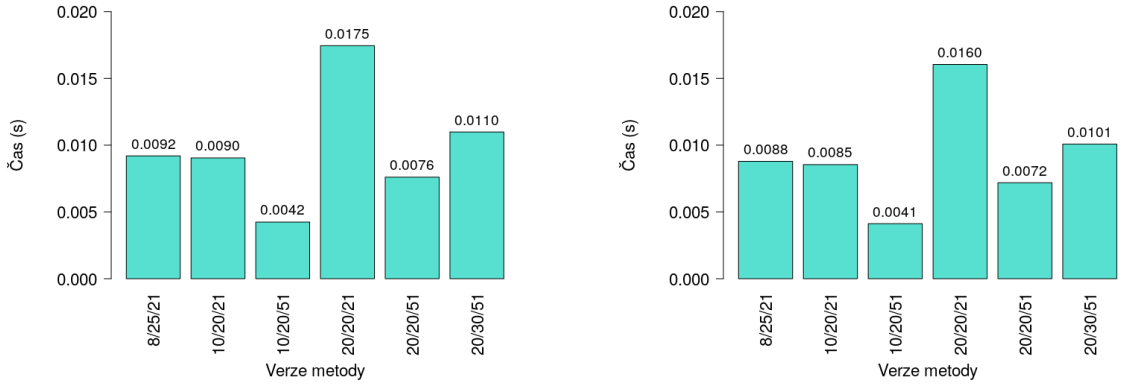
z důvodu, že není zaručena dostatečná různorodost jedinců. V tabulce 4 jsou uvedeny parametry, které jsou rozdílné pro jednotlivé verze metody a jsou to velikost populace, počet migrací a velikost kroku při výpočtu pohybu směrem k vedoucímu jedinci. Všechny tyto parametry ovlivňují, kolik jedinců bude během celého výpočtu evolučního algoritmu vytvořeno a ohodnoceno, což znamená, že změna jednoho z parametrů má vliv na výslednou přesnost a rychlost výpočtu. Pro jednotlivé verze metody jsem nastavila hodnoty těchto parametrů tak, aby bylo možné vliv změny porovnat.

	GI4E				**BioID**			
	Chyba (px)	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**	**Chyba (px)**	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**
8/25/21	6,5	0,088	85,80	0,0092	6,5	0,117	84,32	0,0088
10/20/21	6,5	0,087	86,21	0,0090	6,3	0,113	85,86	0,0085
10/20/51	6,6	0,090	85,92	0,0042	6,7	0,120	84,09	0,0041
20/20/21	6,2	0,084	86,65	0,0175	6,1	0,110	86,06	0,0160
20/20/51	6,5	0,087	86,04	0,0076	6,5	0,116	84,91	0,0072
20/30/51	6,4	0,087	86,25	0,0110	6,5	0,115	84,81	0,0101

Tabulka 5: Výsledky měření přesnosti a rychlosti jednotlivých verzí metody SOMA_JP. V prvním sloupci tabulky je označení verze metody, následující čtyři sloupce uvádějí hodnoty pro dataset GI4E a poslední čtyři sloupce pro dataset BioID. Ve sloupci Chyba je průměrná chyba detekce v pixelech, sloupec Norm. udává průměrnou normovanou chybu v pixelech, Úsp. je úspěšnost detekce v procentech a Čas udává dobu běhu algoritmu v sekundách.

Výsledky měření pro verze metody SOMA_JP jsou v tabulce 5. Z hodnot normované chyby detekce lze opět poznat, že detekce na datasetu BioID je obtížnější než na datasetu GI4E. V rámci datasetu jsou v přesnosti jednotlivých metod jen malé rozdíly. Vliv na přesnost má parametr velikosti kroku, jehož menší hodnota 0,21 zajišťuje lepší přesnost, což lze vidět například u výsledků verze 10/20/21, kde je normalizovaná chyba 0,087 px a úspěšnost detekce 86,21 %, a u verze 10/20/51 s větším krokem ale se stejným nastavením velikosti populace a počtu migrací je normalizovaná chyba 0,090 px a úspěšnost detekce 85,92 %. Stejný vliv má změna parametru velikosti kroku i u verzí 20/20/21 a 20/20/51. Doba běhu jednotlivých verzí metod odpovídá nastavení všech tří parametrů. Pokud jsou hodnoty parametrů vyšší, je čas detekce větší, a naopak.

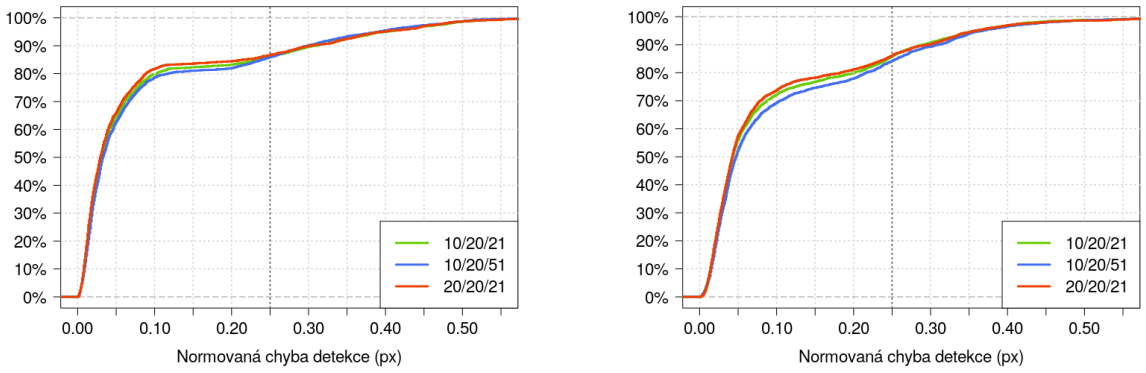
Na obrázku 23 je znázorněno porovnání průměrného času detekce u jednotlivých verzí. Nejnižší hodnotu má 10/20/51, což odpovídá tomu, že u této verze metody je při výpočtu ohodnoceno nejméně jedinců. Naopak je tomu u 20/20/21, kde je čas detekce nejvyšší. Z obrázku lze také vidět, že testovaný dataset nemá vliv na dobu běhu detekce.



(a) GI4E (b) BioID

Obrázek 23: Vyobrazení průměrné rychlosti detekce jednotlivých verzí metody SOMA_JP nad datasetem GI4E 23a a BioID 23b.

Jelikož verze metody SOMA_JP mají přesnost detekce velice podobnou, byly pro porovnání přesnosti detekce pomocí distribuční funkce normalizované chyby zvoleny tři verze (10/20/21, 10/20/51, 20/20/21). Vyobrazení tohoto srovnání je na obrázku 24. Z obrázku lze vyčíst, že největší přesnosti dosahuje 20/20/21, avšak tento rozdíl není příliš velký a verze s nižším počtem jedinců 10/20/21 nemá přesnost detekce výrazně menší. Na obrázku 24 lze také vidět, že detekce pro dataset GI4E je přesnější než pro BioID dataset.



(a) GI4E (b) BioID

Obrázek 24: Srovnání přesnosti detekce jednotlivých verzí metody SOMA_JP nad datasetem GI4E 24a a BioID 24b.

5.3 Diferenciální evoluce, jas a hrany (DE_JH)

Metoda DE_JH (4.4.2) je velice podobná první testované metodě (DE_JP), používá stejný evoluční algoritmus (diferenciální evoluci), je zde využito principů víceúčelové optimalizace a jedna z účelových funkcí také počítá průměrný jas v oblasti. Rozdíl je v druhé z použitých funkcí, u které je potřeba detekce hran v obraze. Parametry, které lze u této metody nastavit, proto nesouvisejí jen s evolučním algoritmem, ale i druhou účelovou funkcí a detekcí hran. Diferenciální evoluce má opět čtyři parametry (velikost populace, počet generací, parametr mutace a křížení) a s detekcí hran souvisí například prahy pro určení, zda se na daném pixelu hrana nachází či ne, nebo velikost filteru pro rozostření obrazu.

Parametry pro nastavení diferenciální evoluce ovlivňující počet testovaných jedinců jsem zvolila stejné jako u verzí metody DE_JP, aby bylo možné metody porovnat, a konstanty pro mutaci a křížení mají hodnoty 0,4 a 0,8, což bylo při testování metody DE_JP ověřeno jako vhodné nastavení. Přehled verzí metody DE_JH s hodnotami parametrů diferenciální evoluce je v tabulce 6.

	Velikost populace	**Počet generací**	**Parametr mutace**	**Parametr křížení**
5/10/4/8	5	10	0,4	0,8
5/20/4/8	5	20	0,4	0,8
10/10/4/8	10	10	0,4	0,8
10/20/4/8	10	20	0,4	0,8
10/30/4/8	10	30	0,4	0,8
20/10/4/8	20	10	0,4	0,8
30/10/4/8	30	10	0,4	0,8
30/30/4/8	30	30	0,4	0,8

Tabulka 6: Přehled verzí metody DE_JH. V prvním sloupci tabulky je uvedeno označení verze, v dalších čtyřech sloupcích jsou hodnoty udávající nastavení diferenciální evoluce (velikost populace, počet generací, parametr mutace a křížení).

Jedna z účelových funkcí použitých v metodě DE_JH počítá počet bodů na obvodu kruhové oblasti, které byly detekovány jako hranové, a zda se bod nachází na obvodu je určeno vzdáleností tohoto bodu od středu kruhu. Tato vzdálenost je dána velikostí jedince, avšak při výpočtu, kdy pozice pixelu jsou dány celočíselně, nelze brát v úvahu jen body, které mají od středu přesně danou vzdálenost. Proto je počítáno s odchylkou udávající, jak moc může být bod vzdálen od ideálního okraje, aby byl ještě považován za bod na obvodu oblasti. Hodnota této odchylky je 0,4 a při testování jsem ověřila, že snížení ani navýšení nemá pozitivní vliv na výsledky detekce.

Detekce hran je provedena vždy jednou pro každý obraz s oblastí okolo oka a lepších výsledků je dosaženo, když je obraz nejprve rozostřen pomocí jednoduchého filtru, který nahrazuje hodnotu jasu pixelu za průměrný jas v okolí pixelu. Velikost masky tohoto filteru je 3x3 pixelů a jeho použitím dojde k odstranění šumu. Vynechání rozostření obrazu vedlo k horším výsledkům, stejně jako zvětšení velikosti filteru, kvůli kterému u takto malých obrazů dojde ke ztrátě příliš mnoha detailů.

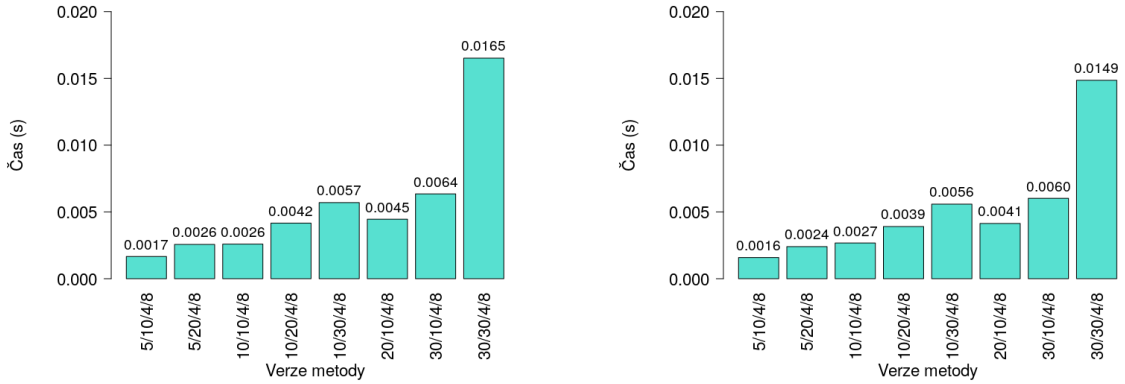
	GI4E				**BioID**			
	Chyba (px)	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**	**Chyba (px)**	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**
5/10/4/8	5,8	0,079	97,98	0,0017	5,7	0,103	96,88	0,0016
5/20/4/8	5,1	0,068	98,67	0,0026	5,2	0,096	97,27	0,0024
10/10/4/8	5,0	0,068	98,62	0,0026	5,2	0,095	98,06	0,0027
10/20/4/8	4,3	0,059	98,62	0,0042	4,8	0,088	97,90	0,0039
10/30/4/8	4,1	0,056	98,62	0,0057	4,6	0,085	97,67	0,0056
20/10/4/8	4,2	0,057	98,83	0,0045	4,8	0,088	97,83	0,0041
30/10/4/8	4,0	0,054	99,03	0,0064	4,7	0,086	97,67	0,0060
30/30/4/8	3,4	0,046	98,79	0,0165	4,4	0,080	97,90	0,0149

Tabulka 7: Výsledky měření přesnosti a rychlosti jednotlivých verzí metody DE_JH. V prvním sloupci tabulky je označení verze metody, následující čtyři sloupce uvádějí hodnoty pro dataset GI4E a poslední čtyři sloupce pro dataset BioID. Ve sloupci Chyba je průměrná chyba detekce v pixelech, sloupec Norm. udává průměrnou normovanou chybu v pixelech, Úsp. je úspěšnost detekce v procentech a Čas udává dobu běhu algoritmu v sekundách.

Pro detekování hran jsem využila Cannyho detektor implementovaný v knihovně OpenCV [20]. U tohoto algoritmu je potřeba nastavit hodnoty dolního a horního prahu, které ovlivňují, jak velké množství hran bude v obraze detekováno. Podle doporučení jsem horní práh nastavila vždy třikrát větší než spodní práh a pro zjištění vhodné hodnoty spodního prahu jsem otestovala více možností. Obecně bylo dosaženo lepších výsledků, když bylo detekováno více hran (práh byl nižší), a podle výsledků experimentů byla vybrána hodnota 40. Jelikož mohou mít vstupní obrazy různou úroveň jasu, vyzkoušela jsem pomocí OTSU metody (rovněž implementované v knihovně OpenCV) určit hodnotu prahu pro každý obraz zvlášť. Toto řešení však nepřineslo žádné zlepšení, bylo dosaženo stejné nebo horší výsledné přesnosti. Jelikož pro výpočet metody DE_JH je lepší, když je detekováno více hran, byla otestována i řešení, kde výsledná hodnota z OTSU metody byla vždy procentuálně snížena. U takového způsobu řešení bylo dosaženo nejlepších výsledků při snížení vypočteného prahu o 20 %. Avšak ani u jednoho z datasetů

nebyla výsledná přesnost lepší než s původním řešením se stejnou hodnotou prahu pro všechny obrazy.

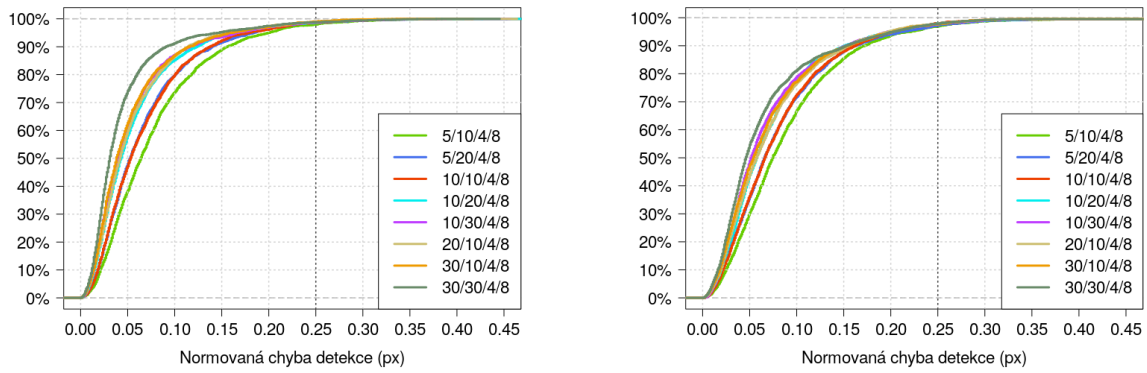
V tabulce 7 jsou výsledky měření pro verze metody DE_JH s různým nastavením parametrů diferenciální evoluce. Všechny verze mají poměrně vysokou úspěšnost detekce, jelikož v žádné není menší než 98 %. Lepší přesnosti dosahují opět metody s větší populací a větším počtem generací, avšak na úkor delšího času výpočtu.



(a) GI4E (b) BioID

Obrázek 25: Vyobrazení průměrné rychlosti detekce jednotlivých verzí metody DE_JH nad datasetem GI4E 25a a BioID 25b.

Srovnání průměrného času běhu je na obrázku 25. Z vyobrazených grafů lze vyčíst, že čas detekce nezávisí na zvoleném datasetu, je však závislý na množství testovaných jedinců během výpočtu.



(a) GI4E (b) BioID

Obrázek 26: Srovnání přesnosti detekce jednotlivých verzí metody DE_JH nad datasetem GI4E 26a a BioID 26b.

Na obrázku 26 je znázorněna přesnost detekce verzí metody DE_JH, přičemž nejlepší přesnosti dosahuje 30/30/4/8 a nejhorší 5/10/4/8, což odpovídá nastaveným parametrům, které ovlivňují počet jedinců. Vliv změny velikosti populace či počtu generací na přesnost detekce u jednotlivých verzí metody DE_JH je stejný jako u metody DE_JP. Tuto podobnost lze pozorovat při srovnání obrázku 26 (DE_JH) a 21 (DE_JP).

5.4 Evoluční algoritmus, HOG a SVM (DE_HOG, SOMA_HOG)

Metody DE_HOG a SOMA_HOG (4.4.3) jsou si velice podobné a jsou prvními testovanými metodami, ve kterých je využito strojové učení. Tyto metody kombinují evoluční algoritmus a detekci zornice pomocí HOG deskriptoru a SVM klasifikátoru, přičemž u jednotlivých verzí metod lze měnit nejen nastavení evolučního algoritmu, ale i parametry související s HOG deskriptorem, jako je velikost buňky či bloku, a SVM klasifikátorem, u kterého mohou být použity různé kernely, které ovlivňují, jakým způsobem je počítána nadrovina pro oddělení tříd. V implementaci obou metod je využit HOG deskriptor a SVM klasifikátor z knihovny OpenCV [20].

Vliv hodnot parametrů pro nastavení diferenciální evoluce (velikost populace, počet generací, parametr mutace a křížení) a algoritmu SOMA (velikost populace, počet migrací, velikost kroku) byl již testován v předchozích měřeních. Pro metodu DE_HOG i SOMA_HOG bylo zvoleno takové nastavení evolučního algoritmu, kde čas detekce není příliš vysoký, ale zároveň je dosaženo dostatečné přesnosti. U diferenciální evoluce je to takové nastavení, kdy hodnota velikosti populace je 10, počtu generací 20, parametru mutace 0,4 a parametru křížení 0,8. U algoritmu SOMA je při takovém nastavení velikost populace 10, počet migrací 20 a velikost kroku 0,21.

	Velikost populace	**Počet generací**	**Parametr mutace**	**Parametr křížení**	**SVM kernel**	**Počet tříd**
10/20/4/8-LIN2	10	20	0,4	0,8	LINEAR	2
10/20/4/8-INT2	10	20	0,4	0,8	INTER	2
10/20/4/8-RBF2	10	20	0,4	0,8	RBF	2
10/20/4/8-LIN1	10	20	0,4	0,8	LINEAR	1

Tabulka 8: Přehled verzí metody DE_HOG. V prvním sloupci tabulky je uvedeno označení verze, v dalších čtyřech sloupcích jsou hodnoty udávající nastavení diferenciální evoluce (velikost populace, počet generací, parametr mutace a křížení) a v posledních dvou sloupcích je uvedeno, jaký kernel je použit pro SVM klasifikátor (SVM kernel) a jaký počet tříd SVM klasifikátor odděluje (Počet tříd).

Klasifikátor SVM jsem použila se třemi různými kernely. Prvním z nich je kernel LINEAR, u kterého není počítán převod problému do vyšší dimenze a je nejrychlejší možností. Druhým

rychlým kernelem, který jsem použila, je INTER a poslední testovaný kernel je RBF, který je doporučovaný jako vhodná volba u většiny úloh. SVM klasifikátor slouží k oddělení dvou odlišných tříd, avšak existuje i verze, která umí klasifikovat jen jednu třídu objektů (obrázky se zornicí) a oddělit ji od ostatních případů (obrázky bez zornice), přičemž při trénování jsou v trénovací množině obrázků jen pozitivní případy, tedy obrázky, na kterých se vyskytuje zornice. Tuto verzi SVM klasifikátoru jsem také otestovala a použila jsem ji společně s rychlým kernelem LINEAR, jelikož výpočet u tohoto klasifikátoru je s ostatními kernely příliš pomalý. Trénování obou verzí klasifikátoru trvalo jednotky maximálně desítky sekund v závislosti na zvoleném kernelu, přičemž ukončující podmínkou pro trénování byl vždy maximální počet iterací 1 000.

	Velikost populace	**Počet migrací**	**Velikost kroku**	**SVM kernel**	**Počet tříd**
10/20/21-LIN2	10	20	0,21	LINEAR	2
10/20/21-INT2	10	20	0,21	INTER	2
10/20/21-RBF2	10	20	0,21	RBF	2
10/20/21-LIN1	10	20	0,21	LINEAR	1

Tabulka 9: Přehled verzí metody SOMA_HOG. V prvním sloupci tabulky je uvedeno označení verze, v dalších třech sloupcích jsou hodnoty udávající nastavení algoritmu SOMA (velikost populace, počet migrací a velikost kroku, která určuje, jak hustě bude prohledáván prostor řešení) a v posledních dvou sloupcích je uvedeno, jaký kernel je použit pro SVM klasifikátor (SVM kernel) a jaký počet tříd SVM klasifikátor odděluje (Počet tříd).

Pro výpočet HOG deskriptoru jsem nastavovala pět parametrů. Prvním je velikost okna, která udává, jak velký bude vstupní obraz, pro který je deskriptor počítán. Obrazy v trénovacím datasetu mají velikost 48x48 pixelů a nejlepších výsledků při použití všech tří kernelů bylo dosaženo při zmenšení této velikosti a nastavení velikosti vstupního obrazu na 32x32 pixelů. Dalšími parametry HOG deskriptoru jsou velikost buňky, která ovlivňuje nad jak velkou oblastí je počítán histogram gradientů, velikost bloku, která udává, přes kolik buněk budou vypočtené histogramy normalizovány, a posun bloku, jehož hodnota ovlivňuje, kolikrát bude proveden výpočet pro jeden blok. Posledním parametrem je počet sloupců v histogramu, který určuje, jak velký bude vektor příznaků pro jednu buňku.

Všechny parametry HOG deskriptoru ovlivňují velikost výsledného vektoru příznaků pro celý vstupní obraz a výchozí nastavení těchto parametrů je velikost buňky 8x8 pixelů, velikost bloku 16x16 pixelů, posun bloku 8 pixelů v horizontálním i vertikálním směru a počet sloupců v histogramu 9. V kombinaci s jednotlivými typy kernelů jsem otestovala různá nastavení těchto parametrů, abych nalezla vhodné nastavení pro každý kernel. Pro získání podrobnějšího popisu obrazu pomocí HOG deskriptoru jsem snižovala velikost buňky, bloku či posunu bloku, čímž je obraz rozdělen do menších oblastí a ve výsledném vektoru příznaků jsou informace o větším

množství detailů. Pro zvětšení vypočteného vektoru příznaků lze zvýšit počet sloupců v histogramu, což také zajistí zpřesnění deskriptoru. Touto úpravou parametrů je však i navýšen celkový čas výpočtu, tudíž pro snížení doby běhu nebo pro získání méně podrobného popisu obrazu jsou naopak velikosti buňky, bloku a posunu bloku zvětšeny a počet sloupců v histogramu snížen. Z výsledků experimentů jsem zjistila, že pro kernel LINEAR (se základní verzí SVM klasifikátoru i s verzí oddělující pouze jednu třídu) a kernel INTER je nejvhodnější nastavení stejné jako výchozí. Větší ani menší vektor příznaků nezlepšil výsledky detekce a při použití většího vektoru příznaků byl navíc zvýšen čas výpočtu. Kernel RBF dosahoval nejlepších výsledků při zvětšení posunu bloku na 16 pixelů v obou směrech a navýšení počtu sloupců v histogramu z 9 na 12, přičemž velikost buňky zůstává 8x8 pixelů a velikost bloku 16x16 pixelů.

	GI4E				**BioID**			
	Chyba (px)	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**	**Chyba (px)**	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**
10/20/4/8-LIN2	4,0	0,054	98,58	0,0116	5,0	0,092	94,84	0,0120
10/20/4/8-INT2	4,1	0,055	98,34	0,0839	4,8	0,088	95,00	0,0830
10/20/4/8-RBF2	3,9	0,053	97,94	0,0591	4,5	0,082	95,99	0,0599
10/20/4/8-LIN1	4,5	0,060	94,01	0,0120	5,4	0,097	91,81	0,0119

Tabulka 10: Výsledky měření přesnosti a rychlosti jednotlivých verzí metody DE_HOG. V prvním sloupci tabulky je označení verze metody, následující čtyři sloupce uvádějí hodnoty pro dataset GI4E a poslední čtyři sloupce pro dataset BioID. Ve sloupci Chyba je průměrná chyba detekce v pixelech, sloupec Norm. udává průměrnou normovanou chybu v pixelech, Úsp. je úspěšnost detekce v procentech a Čas udává dobu běhu algoritmu v sekundách.

Pro každou z metod DE_HOG a SOMA_HOG jsou testovány čtyři verze, přičemž pro DE_HOG jsou u všech verzí stejné parametry diferenciální evoluce a u verzí SOMA_HOG jsou stejné parametry algoritmu SOMA. Nastavení HOG deskriptoru a SVM klasifikátoru jsou pro každou ze čtyř testovaných verzí rozdílná a aby bylo možné mezi sebou porovnat jednotlivé verze obou metod, jsou tato čtyři nastavení HOG deskriptoru a SVM klasifikátoru shodná pro obě metody. Tři z verzí používají běžný SVM klasifikátor oddělující dvě třídy, přičemž v každé z nich je použit jiný kernel (LINEAR, INTER, RBF). Poslední verze využívá SVM klasifikátor pro jednu třídu s kernelem LINEAR. Parametry HOG deskriptoru jsou u všech verzí nastaveny tak, aby odpovídaly nejlepšímu nastavení pro použitý kernel.

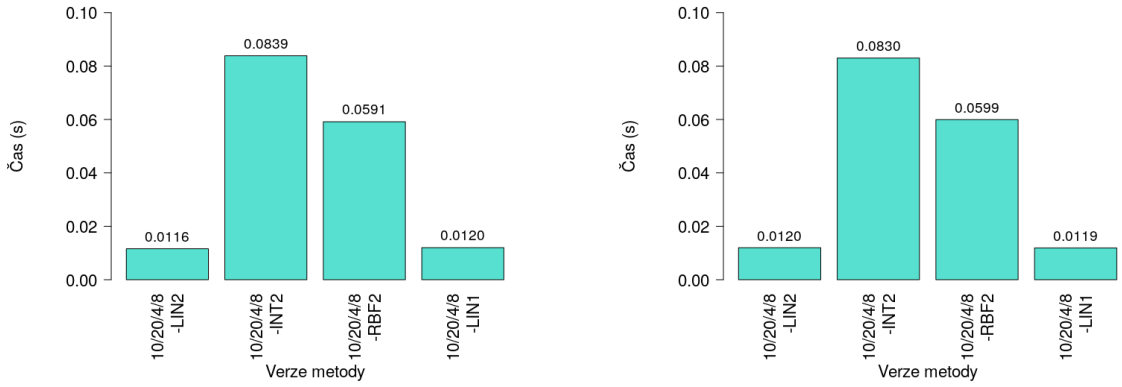
Přehled verzí metody DE_HOG je v tabulce 8 a metody SOMA_HOG v tabulce 9. V označení verze metody je nejprve uvedeno nastavení evolučního algoritmu obdobně jako u předchozích testovaných metod a za pomlčkou je popis odpovídající nastavení SVM klasifikátoru, přesněji zkrácený název použitého kernelu a číslo 1 či 2 udávající, jaká verze klasifikátoru je použita.

Číslu 2 odpovídá běžná verze oddělující od sebe dvě třídy a číslu 1 verze, která odděluje pouze jednu třídu od ostatních. Stejné číselné označení je použito i v posledním sloupci tabulky 8 a 9, který udává, kolik tříd od sebe použitý klasifikátor odděluje.

	GI4E				**BioID**			
	Chyba (px)	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**	**Chyba (px)**	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**
10/20/21-LIN2	6,0	0,080	87,34	0,0728	5,8	0,105	88,23	0,0721
10/20/21-INT2	5,4	0,072	89,28	0,6142	5,4	0,097	90,30	0,6201
10/20/21-RBF2	6,0	0,079	87,18	0,4039	5,4	0,097	89,64	0,4017
10/20/21-LIN1	7,3	0,098	80,14	0,0760	6,4	0,114	85,40	0,0762

Tabulka 11: Výsledky měření přesnosti a rychlosti jednotlivých verzí metody SOMA_HOG. V prvním sloupci tabulky je označení verze metody, následující čtyři sloupce uvádějí hodnoty pro dataset GI4E a poslední čtyři sloupce pro dataset BioID. Ve sloupci Chyba je průměrná chyba detekce v pixelech, sloupec Norm. udává průměrnou normovanou chybu v pixelech, Úsp. je úspěšnost detekce v procentech a Čas udává dobu běhu algoritmu v sekundách.

V tabulce 10 jsou výsledky měření pro verze metody DE_HOG. Z hodnot průměrné normované odchylky a úspěšnosti detekce, která je vždy větší než 90 %, lze vidět, že všechny verze dosahují poměrně dobré přesnosti. Doba běhu se pohybuje od 0,01 s do 0,08 s.

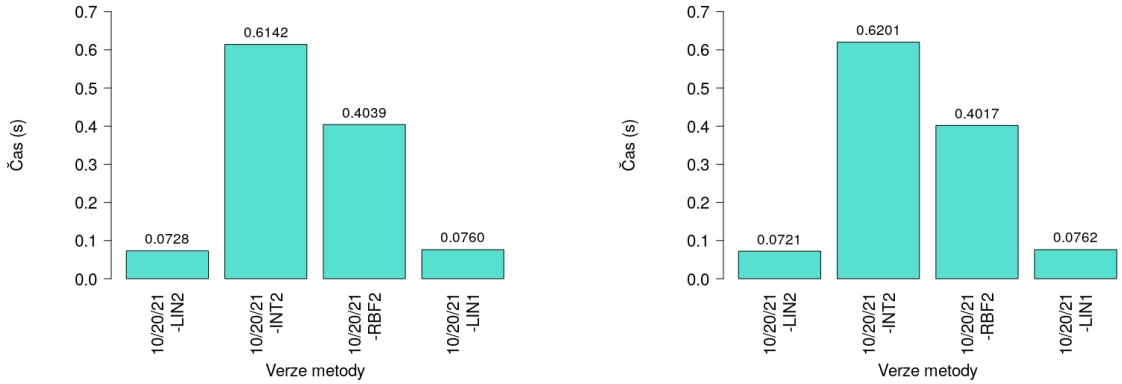


(a) GI4E (b) BioID

Obrázek 27: Vyobrazení průměrné rychlosti detekce jednotlivých verzí metody DE_HOG nad datasetem GI4E 27a a BioID 27b.

Výsledky měření pro verze metody SOMA_HOG jsou v tabulce 11, ve které je vidět, že přesnost detekce není příliš vysoká (maximálně 90 %) a čas výpočtu je od 0,07 s do 0,62 s. Při po-

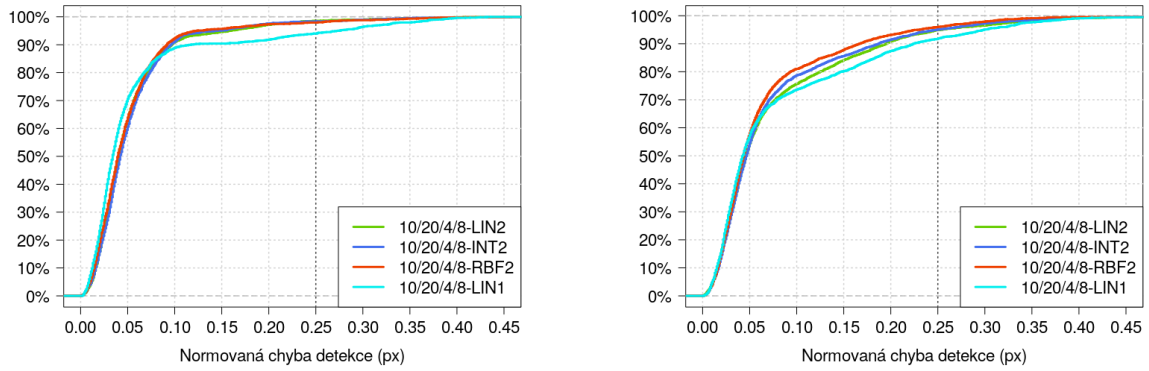
rovnání výsledků pro obě metody je patrné, že verze DE_HOG využívající diferenciální evoluci dosahuje lepší přesnosti s nižším časem výpočtu.



(a) GI4E (b) BioID

Obrázek 28: Vyobrazení průměrné rychlosti detekce jednotlivých verzí metody SOMA_HOG nad datasetem GI4E 28a a BioID 28b.

Na obrázku 27 je vyobrazeno srovnání průměrné doby výpočtu pro verze metody DE_HOG. Z obrázku je patrné, že nejnižší čas výpočtu mají verze 10/20/4/8-LIN2 a 10/20/4/8-LIN1, u kterých je využit kernel LINEAR. Naopak nejpomalejší je verze 10/20/4/8-INT2 s kernelem INTER.

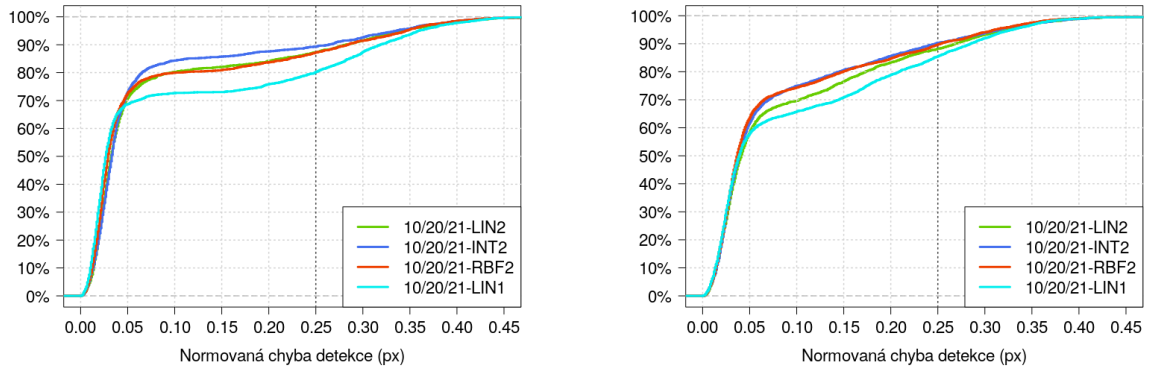


(a) GI4E (b) BioID

Obrázek 29: Srovnání přesnosti detekce jednotlivých verzí metody DE_HOG nad datasetem GI4E 29a a BioID 29b.

Srovnání doby běhu pro metodu SOMA_HOG je na obrázku 28. Nejrychlejší jsou opět verze 10/20/21-LIN2 a 10/20/21-LIN1 využívající kernel LINEAR a nejpomalejší verze 10/20/21-INT2 s kernelem INTER. Při porovnání jednotlivých verzí obou metod je vidět,

že doba běhu při využití evolučního algoritmu SOMA (SOMA_HOG) je vyšší než při použití diferenciální evoluce (DE_HOG). Z obrázků je také patrné, že verze s kernelem RBF mají nižší čas výpočtu než verze s kernelem INTER, přestože kernel INTER má být obecně rychlejší. Důvodem je nastavení HOG deskriptoru, kdy u RBF kernelu je použito nastavení, při kterém je výsledný vektor příznaků menší, díky čemuž je celkový čas výpočtu nižší.



(a) GI4E

(b) BioID

Obrázek 30: Srovnání přesnosti detekce jednotlivých verzí metody SOMA_HOG nad datasetem GI4E 30a a BioID 30b.

Porovnání přesnosti detekce verzí metody DE_HOG je na obrázku 29. Všechny verze, které využívají SVM klasifikátor oddělující dvě třídy (10/20/4/8-LIN2, 10/20/4/8-INT2, 10/20/4/8-RBF2), mají podobnou přesnost detekce. Drobné rozdíly těchto verzí jsou vidět pouze nad datasetem BioID, kde má nejlepší přesnost 10/20/4/8-RBF2 s kernelem RBF. Z obrázku je také patrné, že nejhorší přesnosti nad oběma datasety dosahuje verze 10/20/4/8-LIN1, u které je využit SVM klasifikátor oddělující pouze jednu třídu. Na obrázku 30 je vyobrazeno srovnání přesnosti verzí metody SOMA_HOG. Verze využívající klasifikátor pro oddělení dvou tříd (10/20/21-LIN2, 10/20/21-INT2, 10/20/21-RBF2) dosahují přesnější detekce než verze 10/20/21-LIN1, u které je použit klasifikátor pro oddělení jedné třídy, což je stejný výsledek jako u verzí metody DE_HOG. Při porovnání přesností u metody DE_HOG (obrázek 29) a SOMA_HOG (obrázek 30) je vidět, že kombinace s algoritmem diferenciální evoluce (DE_HOG) dosahuje přesnější detekce nad oběma datasety.

5.5 Evoluční algoritmus a konvoluční neuronová síť (DE_CNN, SOMA_CNN)

Poslední dvě testované metody jsou DE_CNN a SOMA_CNN (4.4.4), u kterých je použita jako účelová funkce detekce zornice pomocí konvoluční neuronové sítě. Evolučním algoritmem u DE_CNN je diferenciální evoluce a u SOMA_CNN algoritmus SOMA. U těchto metod jsou opět nastavovány parametry evolučních algoritmů a především lze použít mnoho různých variant konvoluční neuronové sítě, které se mohou lišit například v architektuře či v nastavení počtu a velikosti filterů v konvoluční vrstvě. V implementaci obou metod jsem použila neuronové sítě z knihovny dlib [21] a při trénování i výpočtu konvoluční neuronové sítě je pro zrychlení využita grafická karta (NVIDIA GeForce 940MX).

Pro parametry evolučních algoritmů jsem zvolila nastavení, které vychází z předchozího testování, a stejně jako u metod DE_HOG a SOMA_HOG jsem použila hodnoty, při kterých je dosažena co největší přesnost za co nejkratší čas. Tomu u diferenciální evoluce odpovídá velikost populace 10, počet generací 20, parametr mutace 0,4, parametr křížení 0,8 a u algoritmu SOMA velikost populace 10, počet migrací 20 a velikost kroku 0,21.

	Počet filterů 1	**Počet filterů 2**	**Počet neuronů**
N8	8	8	8
N16	8	16	16
N32	16	32	32

Tabulka 12: Přehled jednotlivých nastavení konvoluční neuronové sítě použité v metodách DE_CNN a SOMA_CNN. V prvním sloupci je uvedeno označení nastavení, v dalších dvou sloupcích je počet použitých filterů v první (Počet filterů 1) a druhé (Počet filterů 2) konvoluční vrstvě a v posledním sloupci je počet neuronů v první vrstvě plně propojené sítě (Počet neuronů).

Při výběru architektury konvoluční neuronové sítě a nastavení jednotlivých vrstev jsem se inspirovala konvolučními sítěmi použitými v metodě PupilNet v2.0 [7] popsané v části 2.5. Všechny testované sítě mají stejnou architekturu, kde první vrstva je konvoluční, následuje ReLU funkce, podvzorkování podle maximální hodnoty, poté je další konvoluční vrstva, ReLU funkce a poslední částí je plně propojená neuronová síť o dvou vrstvách, přičemž druhá výstupní vrstva má vždy dva neurony, jelikož cílem této neuronové sítě je klasifikovat dvě třídy (obraz se zornicí a obraz bez zornice). Podle sítí v [7] jsem nastavila velikost vstupního obrazu na 24x24 pixelů, velikost filteru v první konvoluční vrstvě na 6x6 pixelů, velikost filteru v druhé konvoluční vrstvě na 5x5 pixelů a posun obou filterů je o 1 pixel v horizontálním i vertikálním směru. Při podvzorkování je počítána maximální hodnota z oblasti o velikosti 4x4 pixelů s posunem 4 pixely. Počet filterů a počet neuronů v první vrstvě plně propojené sítě je u každého nastavení jiný a opět

jsem vycházela ze sítí použitých v [7]. Přehled jednotlivých nastavení počtu filterů a neuronů je v tabulce 12. Označení nastavení vychází z počtu neuronů, v dalších dvou sloupcích tabulky je uveden počet filterů v první a druhé konvoluční vrstvě a v posledním sloupci je počet neuronů v první vrstvě plně propojené sítě.

	Velikost populace	**Počet generací**	**Parametr mutace**	**Parametr křížení**	**Nastavení sítě**
10/20/4/8-N8	10	20	0,4	0,8	N8
10/20/4/8-N16	10	20	0,4	0,8	N16
10/20/4/8-N32	10	20	0,4	0,8	N32

Tabulka 13: Přehled verzí metody DE_CNN. V prvním sloupci tabulky je uvedeno označení verze, v dalších čtyřech sloupcích jsou hodnoty udávající nastavení diferenciální evoluce (velikost populace, počet generací, parametr mutace a křížení) a v posledním sloupci je uvedeno nastavení konvoluční neuronové sítě.

Pro natrénování konvoluční neuronové sítě byla využita metoda zvaná SGD (stochastic gradient descent). Počáteční učící konstanta byla nastavena na 0,01 a vždy, když se přestala snižovat chyba sítě, byla tato konstanta desetkrát zmenšena. Trénování bylo ukončeno ve chvíli, kdy učící konstanta klesla na hodnotu 0,00001. Pro výpočet byla využita grafická karta a doba běhu při trénování byla pro všechna nastavení sítě přibližně 4 minuty.

	Velikost populace	**Počet migrací**	**Velikost kroku**	**Nastavení sítě**
10/20/21-N8	10	20	0,21	N8
10/20/21-N16	10	20	0,21	N16
10/20/21-N32	10	20	0,21	N32

Tabulka 14: Přehled verzí metody SOMA_CNN. V prvním sloupci tabulky je uvedeno označení verze, v dalších třech sloupcích jsou hodnoty udávající nastavení algoritmu SOMA (velikost populace, počet migrací a velikost kroku, která určuje, jak hustě bude prohledáván prostor řešení) a v posledním sloupci je uvedeno nastavení konvoluční neuronové sítě.

Srovnání výsledků měření je provedeno pro tři verze metod DE_CNN a SOMA_CNN. Přehled verzí pro DE_CNN je v tabulce 13 a pro SOMA_CNN v tabulce 14. Všechny verze metody DE_CNN mají stejné nastavení diferenciální evoluce a verze metody SOMA_CNN mají stejné parametry algoritmu SOMA. U obou metod jsou použita tři nastavení konvoluční neuronové sítě, jejichž přehled je v tabulce 12. V označení verze metody je obdobně jako u předchozích tes-

tovaných metod uvedeno nastavení evolučního algoritmu a část za pomlčkou odpovídá nastavení konvoluční sítě.

	GI4E				BioID			
	Chyba (px)	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**	**Chyba (px)**	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**
10/20/4/8-N8	4,8	0,065	96,24	0,0682	4,3	0,078	95,99	0,0671
10/20/4/8-N16	4,5	0,061	97,73	0,0705	4,3	0,079	95,92	0,0695
10/20/4/8-N32	4,6	0,062	97,45	0,1020	4,2	0,077	96,48	0,0973

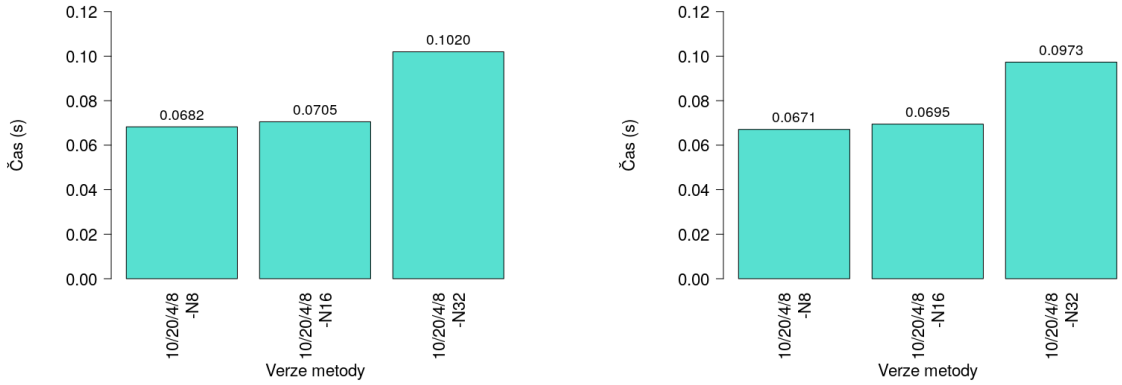
Tabulka 15: Výsledky měření přesnosti a rychlosti jednotlivých verzí metody DE_CNN. V prvním sloupci tabulky je označení verze metody, následující čtyři sloupce uvádějí hodnoty pro dataset GI4E a poslední čtyři sloupce pro dataset BioID. Ve sloupci Chyba je průměrná chyba detekce v pixelech, sloupec Norm. udává průměrnou normovanou chybu v pixelech, Úsp. je úspěšnost detekce v procentech a Čas udává dobu běhu algoritmu v sekundách.

V tabulce 15 jsou výsledky měření pro verze metody DE_CNN. Z hodnot normované odchylky a úspěšnosti detekce je patrné, že přesnost detekce jednotlivých verzí v rámci jednoho datasetu je velice podobná. Čas výpočtu je u každé verze odlišný, což odpovídá tomu, že je použit rozdílný počet filterů. Výsledky měření pro verze metody SOMA_CNN jsou v tabulce 16. Oproti metodě DE_CNN jsou dosažené přesnosti jednotlivých metod menší a čas výpočtu vyšší, přičemž opět platí, že čím více je použito filterů v konvolučních vrstvách, tím delší je doba běhu výpočtu.

	GI4E				BioID			
	Chyba (px)	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**	**Chyba (px)**	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**
10/20/21-N8	5,2	0,069	90,90	0,4675	4,4	0,080	93,39	0,4773
10/20/21-N16	5,1	0,068	93,28	0,4968	4,7	0,086	92,74	0,4865
10/20/21-N32	5,4	0,073	91,46	0,6285	4,6	0,083	92,93	0,6282

Tabulka 16: Výsledky měření přesnosti a rychlosti jednotlivých verzí metody SOMA_CNN. V prvním sloupci tabulky je označení verze metody, následující čtyři sloupce uvádějí hodnoty pro dataset GI4E a poslední čtyři sloupce pro dataset BioID. Ve sloupci Chyba je průměrná chyba detekce v pixelech, sloupec Norm. udává průměrnou normovanou chybu v pixelech, Úsp. je úspěšnost detekce v procentech a Čas udává dobu běhu algoritmu v sekundách.

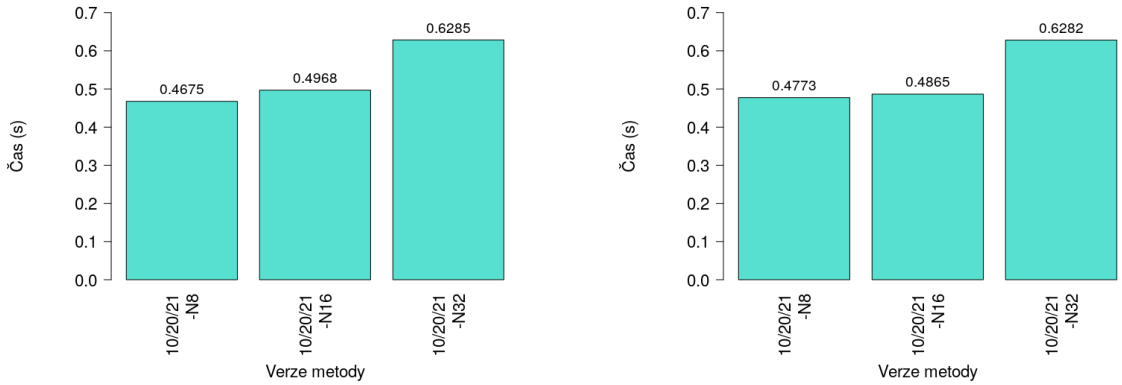
Znázornění průměrné doby běhu výpočtu pro verze metody DE_CNN je na obrázku 31. Z vyobrazení je patrné, že verze 10/20/4/8-N8 a 10/20/4/8-N16 mají čas výpočtu velice podobný a verze 10/20/4/8-N32 je nejpomalejší, což odpovídá tomu, že je v této verzi použito nejvíce filterů a největší počet neuronů v plně propojené síti.



(a) GI4E (b) BioID

Obrázek 31: Vyobrazení průměrné rychlosti detekce jednotlivých verzí metody DE_CNN nad datasetem GI4E 31a a BioID 31b.

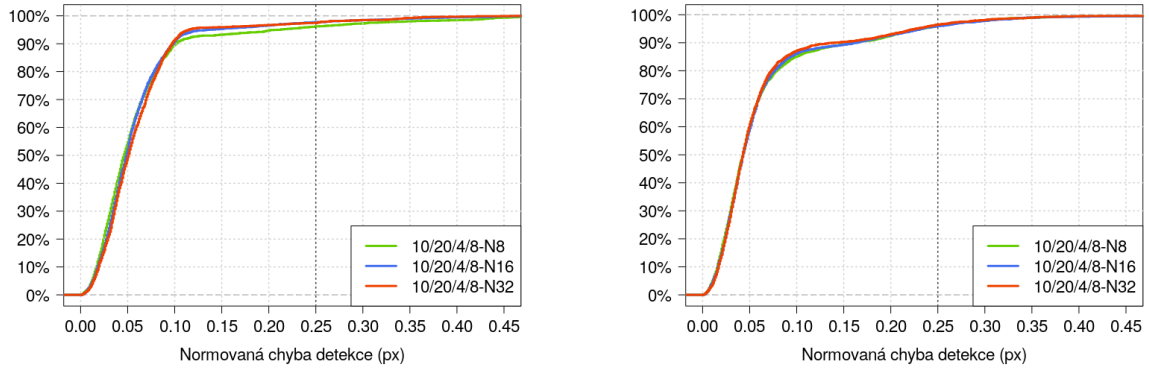
Na obrázku 32 je srovnání času výpočtu pro verze metody SOMA_CNN. Znovu platí, že nejpomalejší verzí je 10/20/21-N32, kde je použito nejvíce filterů a neuronů. Při porovnání zobrazení pro metody DE_CNN (obrázek 31) a SOMA_CNN (obrázek 32) je vidět, že v případě použití diferenciální evoluce je doba běhu nižší.



(a) GI4E (b) BioID

Obrázek 32: Vyobrazení průměrné rychlosti detekce jednotlivých verzí metody SOMA_CNN nad datasetem GI4E 32a a BioID 32b.

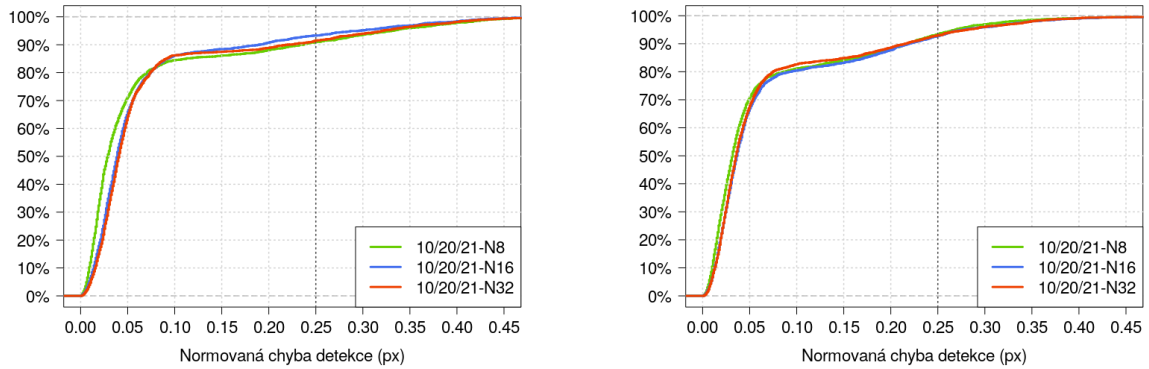
Vyobrazení přesnosti jednotlivých verzí metody DE_CNN je na obrázku 33 a je zde vidět, že všechny verze dosahují téměř shodné přesnosti v rámci jednoho datasetu. Na obrázku 34 je znázorněna přesnost detekce verzí metody SOMA_CNN, přičemž dosažená přesnost je u všech verzí velice podobná nad oběma datasety.



(a) GI4E (b) BioID

Obrázek 33: Srovnání přesnosti detekce jednotlivých verzí metody DE_CNN nad datasetem GI4E 33a a BioID 33b.

Při porovnání přesnosti jednotlivých verzí metod DE_CNN (obrázek 33) a SOMA_CNN (obrázek 34) je patrné, že v případě použití diferenciální evoluce jako evolučního algoritmu je dosaženo větší přesnosti než při použití algoritmu SOMA, jelikož nad oběma datasety mají verze metody DE_CNN větší přesnost. Podobný výsledek byl pozorován již u předchozích testovaných metod DE_HOG a SOMA_HOG.



(a) GI4E (b) BioID

Obrázek 34: Srovnání přesnosti detekce jednotlivých verzí metody SOMA_CNN nad datasetem GI4E 34a a BioID 34b.

5.6 Porovnání implementovaných řešení

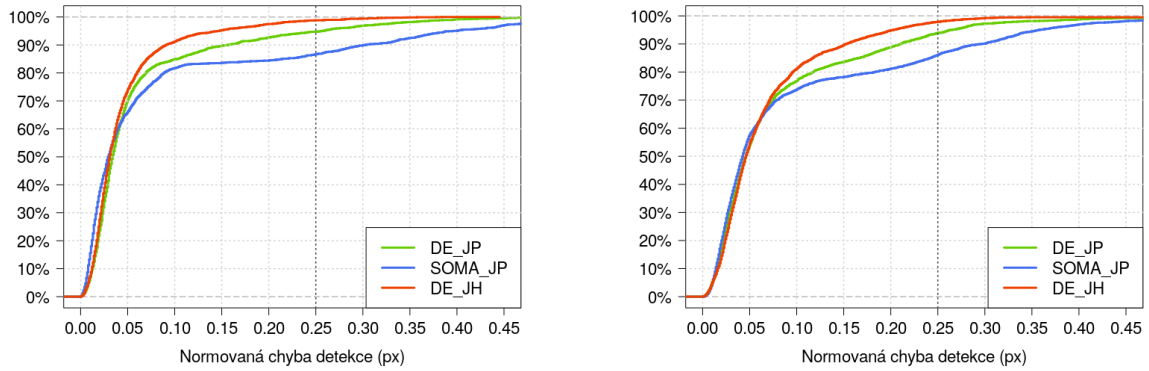
Výsledky měření pro jednotlivé verze metod, které byly představeny v předchozích kapitolách, jsou v této části použity pro porovnání implementovaných metod. V tabulce 17 je shrnutí výsledků verzí metod, které jsou při porovnávání použity. Nejprve jsou mezi sebou srovnány metody, které nevyužívají strojové učení (DE_JP, SOMA_JP, DE_JH), a poté metody používající učení (DE_HOG, SOMA_HOG, DE_CNN, SOMA_CNN). Následně jsou implementovaná řešení porovnána s metodami ExCuSe [5] a ElSe [6], které jsou popsané v částech 2.3 a 2.4.

	GI4E				**BioID**			
	Chyba (px)	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**	**Chyba (px)**	**Norm. (px)**	**Úsp. (%)**	**Čas (s)**
DE_JP 30/30/4/8	4,6	0,062	94,74	0,0121	5,1	0,092	93,92	0,0117
SOMA_JP 20/20/21	6,2	0,084	86,65	0,0175	6,1	0,110	86,06	0,0160
DE_JH 20/10/4/8	4,2	0,057	98,83	0,0045	4,8	0,088	97,83	0,0041
DE_JH 30/30/4/8	3,4	0,046	98,79	0,0165	4,4	0,080	97,90	0,0149
DE_HOG 10/20/4/8-LIN2	4,0	0,054	98,58	0,0116	5,0	0,092	94,84	0,0120
DE_HOG 10/20/4/8-RBF2	3,9	0,053	97,94	0,0591	4,5	0,082	95,99	0,0599
DE_CNN 10/20/4/8-N8	4,8	0,065	96,24	0,0682	4,3	0,078	95,99	0,0671
DE_CNN 10/20/4/8-N16	4,5	0,061	97,73	0,0705	4,3	0,079	95,92	0,0695
ExCuSe	5,7	0,076	93,73	0,0034	7,5	0,141	80,70	0,0024
ElSe	6,2	0,084	86,00	0,0022	6,3	0,113	85,60	0,0031

Tabulka 17: Výsledky měření přesnosti a rychlosti verzí metod, které jsou použity při porovnávání implementovaných řešení. V prvním sloupci tabulky je název metody a označení verze metody, následující čtyři sloupce uvádějí hodnoty pro dataset GI4E a poslední čtyři sloupce pro dataset BioID. Ve sloupci Chyba je průměrná chyba detekce v pixelech, sloupec Norm. udává průměrnou normovanou chybu v pixelech, Úsp. je úspěšnost detekce v procentech a Čas udává dobu běhu algoritmu v sekundách. V posledních dvou řádcích tabulky jsou výsledky pro metody ExCuSe a ElSe.

Na obrázku 35 je vyobrazeno porovnání přesnosti detekce metod, u kterých není využito strojového učení a jejichž účelové funkce jsou založené na průměrném jasu, velikosti oblasti a detekci hran (DE_JP, SOMA_JP, DE_JH). Pro každou metodu byla vybrána taková verze, u které byla detekce nejpřesnější. Pro metodu DE_JP je to verze 30/30/4/8, pro SOMA_JP verze 20/20/21 a pro DE_JH verze 30/30/4/8. Z obrázku 35 lze vyčíst, že metoda DE_JH (98,79 %, 97,90 %), která využívá oproti DE_JP a SOMA_JP účelovou funkci založenou na de-

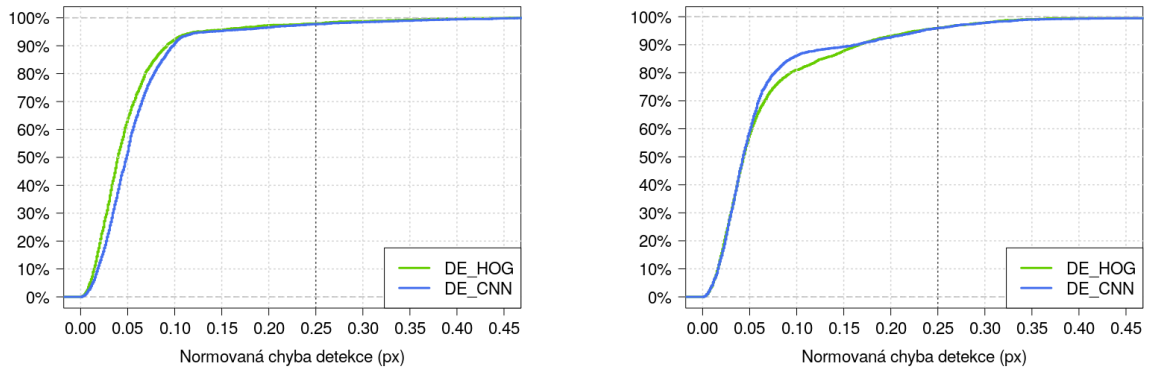
tekci hran, dosahuje největší přesnosti nad datasetem GI4E i BioID. U metod DE_JP (94,74 %, 93,92 %) a SOMA_JP (86,65 %, 86,06 %) jsou použity stejné dvě základní účelové funkce a lepších výsledků je dosaženo u kombinace těchto funkcí s diferenciální evolucí.



(a) GI4E (b) BioID

Obrázek 35: Srovnání přesnosti detekce metod, které nevyužívají strojové učení (DE_JP, SOMA_JP, DE_JH), nad datasetem GI4E 35a a BioID 35b.

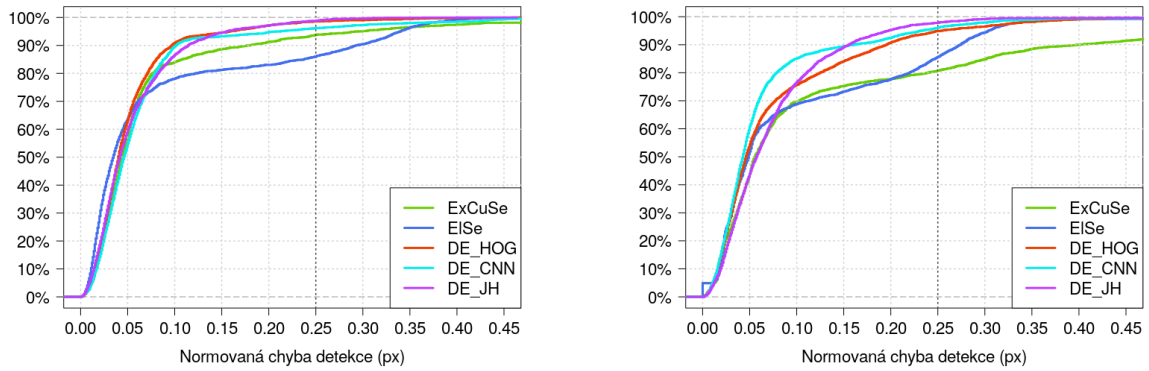
Zhoršená přesnost u metody SOMA_JP může být zapříčiněna i tím, že jsou zde použity dvě účelové funkce bez principů víceúčelové optimalizace a hodnoty funkcí jsou pouze sčítány. Nejrychlejší z těchto tří metod je DE_JP, což je způsobeno tím, že metoda používá dvě základní účelové funkce, jejichž výpočet je rychlý, společně s diferenciální evolucí. Zprůměrovaný čas detekce přes oba datasety je u metody DE_JP 0,012 s, SOMA_JP 0,017 s a DE_JH 0,016 s.



(a) GI4E (b) BioID

Obrázek 36: Srovnání přesnosti detekce metod, které využívají strojové učení (DE_HOG, DE_CNN), nad datasetem GI4E 36a a BioID 36b.

Z výsledků měření pro metody využívající strojové učení (DE_HOG, SOMA_HOG, DE_CNN, SOMA_CNN) již bylo zjištěno, že metody s diferenciální evolucí dosahují lepších výsledků, proto jsou zde porovnány již jen nejpřesnější verze metod DE_HOG a DE_CNN. U metody DE_HOG byla zvolena verze 10/20/4/8-RBF2 používající SVM klasifikátor s kernelem RBF, který je natrénovaný pro oddělení dvou tříd. Porovnávanou verzí u metody DE_CNN je 10/20/4/8-N16, kde je počet filterů u první konvoluční vrstvy nastaven na 8, u druhé na 16 a počet neuronů v první vrstvě plně propojené sítě je 16. Vyobrazení srovnání přesnosti metod DE_HOG (97,94 %, 95,99 %) a DE_CNN (97,73 %, 95,92 %) je na obrázku 36. Dosažená přesnost obou metod je velice podobná. Nad datasetem GI4E je o trochu přesnější metoda DE_HOG, naopak nad datasetem BioID je přesnější metoda DE_CNN. Doba běhu obou metod je téměř shodná, zprůměrovaný čas detekce přes oba datasety je u DE_HOG 0,06 s a u DE_CNN 0,07 s.



(a) GI4E

(b) BioID

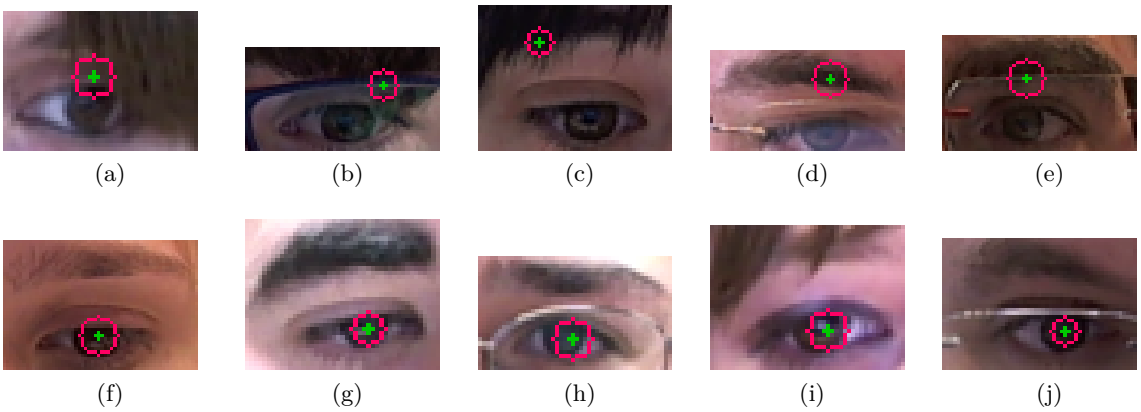
Obrázek 37: Srovnání přesnosti detekce implementovaných řešení s metodami ExCuSe a ElSe nad datasetem GI4E 37a a BioID 37b.

Pro srovnání implementovaných řešení s metodami ExCuSe a ElSe byla vybrána řešení DE_JH, DE_HOG a DE_CNN. Jelikož průměrná doba běhu metod ExCuSe a ElSe je 0,003 s, což je nízký čas, zvolené verze vlastních řešení nedosahují největší přesnosti, ale v rámci daného řešení mají nižší čas běhu. Pro DE_JH je zvolena verze 20/10/4/8 s časem 0,004 s, pro DE_HOG verze 10/20/4/8-LIN2 s časem 0,012 s a pro DE_CNN verze 10/20/4/8-N8 s časem 0,068 s. Srovnání přesnosti daných verzí implementovaných řešení s metodami ExCuSe a ElSe je na obrázku 37. Při porovnání přesností jednotlivých metod nad oběma datasety lze říci, že implementovaná řešení (DE_JH - 98,83 %, 97,83 %; DE_HOG - 98,58 %, 94,84 %; DE_CNN - 96,24 %, 95,99 %) dosahují lepší přesnosti než metody ExCuSe a ElSe. Doba běhu zprůměrovaná přes oba datasety je u metod DE_HOG (0,012 s) a DE_CNN (0,068 s) vyšší než u ExCuSe (0,003 s) a ElSe (0,003 s), pouze metoda DE_JH (0,004 s) má čas výpočtu téměř shodný.

5.7 Problémy a návrhy zlepšení

V průběhu testování byly zjištěny výhody a nevýhody implementovaných řešení a zároveň byly nalezeny problémy, které ztěžují detekci zornice. Tyto problémy jsou zde představeny společně s ukázkou chybných a správných detekcí a dále jsou navržena zlepšení implementovaných metod za účelem zrychlení výpočtu a dosažení větší přesnosti.

Při testování jsem narazila na několik problémů souvisejících se vstupním obrazem, které zapříčiňují nepřesné výsledky. Detekci ztěžují například různé odlesky v očích, brýle, tmavé obočí nebo silné líčení. Pro snížení počtu nepřesných výsledků, které jsou způsobeny zmíněnými důvody, může být například vstupní obraz předem zpracován tak, aby byly odstraněny odlesky v očích. Ukázka chybných detekcí je na obrázku 38 a pro srovnání je na tomto obrázku i ukázka správných detekcí.



Obrázek 38: Ukázka chybných (38a až 38e) a správných (38f až 38j) detekcí zornice.

Z výsledků měření bylo zjištěno, že metody, které jsou založené na základních algoritmech, mají nízkou časovou náročnost, avšak nedosahují příliš velké přesnosti. Naopak řešení, u kterých je využito strojového učení, dosahují větší přesnosti, ale čas potřebný pro výpočet je vyšší. Pro získání větší přesnosti při zachování nízkého času výpočtu může být použita kombinace těchto metod. Například lze použít základní metody při výpočtu evolučního algoritmu pro vytváření nových generací a při výběru nejlepšího jedince z poslední generace může být za účelem zpřesnění použita metoda využívající strojové učení. S touto kombinací by měla být zlepšena přesnost výsledku oproti řešení, které využívá pouze základní metody, a zároveň výpočet nebude příliš časově náročný.

Pro zrychlení doby běhu u navrhovaných řešení může být využita paralelizace při výpočtu evolučních algoritmů. Vytváření jedince není závislé na tvorbě jiného jedince, proto lze výpočet provádět pro každého jedince zvlášť a urychlit tak vytvoření nové generace. Paralelizace může být provedena pomocí spuštění více vláken na procesoru nebo může být využita grafická karta. Pokud by byl výpočet nové generace urychlen, může být řešení použito s větším počtem generací a s více jedinci v populaci, což zlepší přesnost daného řešení.

6 Závěr

Cílem práce bylo seznámit se s metodami pro detekci zornic v obraze a pomocí volně dostupných knihoven naimplementovat detektor zornic.

První kapitola byla věnována metodám pro detekci zornic. Byla zde popsána některá ze současných řešení včetně metod ElSe a ExCuSe, se kterými jsou porovnávány navržené způsoby řešení. V následující kapitole byly představeny některé z dostupných datasetů pro detekci zornic a popsány byly i datasety GI4E a BioID, které jsou použity pro otestování implementovaných metod.

V následujících kapitolách byly popsány navrhované způsoby řešení, jejichž základem je kombinace evolučního algoritmu s různými metodami pro detekci zornic. Evoluční algoritmus je využit pro prohledávání obrazu a k nalezení co nejvhodnější oblasti, které má co nejvíce odpovídat skutečné pozici zornice v obraze. K ohodnocení dané pozice (ke zjištění, jak moc je pravděpodobné, že se v dané oblasti nachází zornice) je použita metoda pro detekci zornice. V implementovaných řešeních jsou využity dva evoluční algoritmy, diferenciální evoluce a SOMA, společně s různými metodami detekce. První dvě z navrhovaných metod jsou založené na základních algoritmech z oblasti analýzy obrazu, přičemž při použití jedné z metod jsou hledány co nejtmavší oblasti v obraze, které mají co největší velikost, a při použití druhé metody je potřeba nejprve detekovat hrany v obraze a cílem je nalézt kruh s co nejmenším jasem, na jehož obvodu bude co nejvíce bodů, které leží na hranách. V dalších z navrhovaných metod je využito strojové učení. První metoda používá pro detekci zornice HOG deskriptor společně s SVM klasifikátorem a ve druhé je k detekci použita konvoluční neuronová síť.

Implementovaná řešení byla testována na dvou renomovaných datasetech (GI4E, BioID) a s různými hodnotami parametrů pro nastavení algoritmů použitých v daném řešení. Z výsledků měření bylo zjištěno, že lepší detekce je dosaženo při použití evolučního algoritmu diferenciální evoluce, s algoritmem SOMA byla řešení méně přesná a pomalejší. Dále z měření vyplývá, že u řešení s metodami, které využívají strojové učení, je dosaženo větší přesnosti než u řešení, kde jsou použity základní metody. Rychlost výpočtu je naopak nižší u řešení s metodami bez strojového učení, kde je další výhodou, že není potřeba mít data pro trénování. Srovnání výsledků detekce s metodami ElSe a ExCuSe bylo provedeno s nejlepšími třemi z navrhovaných řešení. Ve všech z těchto řešení je použita diferenciální evoluce a v prvním je zkombinována se základní metodou využívající detekci hran, ve druhém s metodu založenou na HOG deskriptoru a SVM klasifikátoru a ve třetím s metodou, ve které je použita konvoluční neuronová síť. Při porovnání výsledků dosahovala všechna tři řešení lepší přesnosti než metody ElSe a ExCuSe.

Na tuto práci lze navázat vytvořením řešení, kde bude použita při výpočtu evolučního algoritmu kombinace metody bez učení a metody, ve které je využito učení, za účelem dosažení přesnější detekce s nižší časovou náročností. Detekce pozice zornice může být dále použita pro sledování směru pohledu, čehož se dá využít například v oblasti automobilového průmyslu pro sledování pozornosti řidiče.

Literatura

[1] NGUYEN, Q.X. a S. JO. Electric wheelchair control using head pose free eye-gaze tracker [online]. [cit. 2019-05-31]. DOI: 10.1049/el.2012.1530. Dostupné z: https://www.researchgate.net/publication/260500467_Electric_wheelchair_control_using_head_pose_free_eye-gaze_tracker

[2] HOUGH V, Paul C. Method and means for recognizing complex patterns. 3069654. Uděleno 1962. Dostupné také z: http://www.freepatentsonline.com/3069654.html

[3] Hough Circle Transform [online]. [cit. 2019-06-02]. Dostupné z: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html#hough-circle

[4] PEDRAZA, J.L., M.L. MUÑOZ, R. MÉNDEZ, A. GARCÍA, M.L. CÓRDOBA, A. PÉREZ a F. SÁNCHEZ. Precise Eye-Gaze Detection and Tracking System [online]. [cit. 2019-06-02]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.6318&rep=rep1&type=pdf

[5] PEDRAZA, J.L., Wolfgang FUHL, Thomas KÜBLER, Katrin SIPPEL, Wolfgang ROSENSTIEL a F. SÁNCHEZ. ExCuSe: Robust Pupil Detection in Real-WorldScenarios [online]. [cit. 2019-06-02]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.6318&rep=rep1&type=pdf

[6] PEDRAZA, J.L., Wolfgang FUHL, Thiago C. SANTINI, Wolfgang ROSENSTIEL a F. SÁNCHEZ. ElSe: Ellipse Selection for Robust Pupil Detection in Real-World Environments [online]. [cit. 2019-06-02]. Dostupné z: https://www.embedded.uni-tuebingen.de/uploads/tx_timitarbeiter/ExCuSe_01.pdf

[7] FUHL, Wolfgang, Thiago SANTINI, Gjergji KASNECI, Wolfgang ROSENSTIEL a Enkelejda KASNECI. PupilNet v2.0: Convolutional Neural Networks for CPU-based real time Robust Pupil Detection [online]. [cit. 2020-05-14]. Dostupné z: https://arxiv.org/pdf/1711.00112.pdf

[8] The BioID Face Database [online]. [cit. 2019-06-02]. Dostupné z: https://www.bioid.com/facedb/

[9] The BioID Face Database [online]. [cit. 2019-06-02]. Dostupné z: http://gi4e.unavarra.es/databases/gi4e/

[10] GARBIN, Stephan J., Yiru SHEN, Immo SCHUETZ, Robert CAVIN, Gregory HUGHES a Sachin S.TALATHI. OpenEDS: Open Eye Dataset [online]. [cit. 2020-05-14]. Dostupné z: https://arxiv.org/pdf/1905.03702.pdf

[11] Mpi: Labelled Pupils in the Wild (LPW): Pupil detection in unconstrained environments [online]. [cit. 2020-05-14]. Dostupné z: https://www.mpi-inf.mpg.de/departments/computer-vision-and-machine-learning/research/gaze-based-human-computer-interaction/labelled-pupils-in-the-wild-lpw/

[12] Real-Time Face Pose Estimation. Dlib C++ Library [online]. [cit. 2018-04-22]. Dostupné z: http://blog.dlib.net/2014/08/real-time-face-pose-estimation.html

[13] Canny Edge Detector [online]. [cit. 2019-06-02]. Dostupné z: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

[14] OPLATKOVÁ, Zuzana, Pavel OŠMERA, Miloš ŠEDA, František VČELAŘ a Ivan ZELINKA. Evoluční výpočetní techniky, principy a aplikace. Praha: BEN, 2008. ISBN 80-7300-218-3.

[15] DALAL, Navneet a Bill TRIGGS. Histograms of Oriented Gradients for Human Detection [online]. [cit. 2020-05-14]. Dostupné z: https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

[16] Machine Learning 101: Chapter 2 : SVM (Support Vector Machine) — Theory [online]. [cit. 2020-05-14]. Dostupné z: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

[17] OpenCV: Introduction to Support Vector Machines [online]. [cit. 2020-05-14]. Dostupné z: https://docs.opencv.org/3.4/d1/d73/tutorial_introduction_to_svm.html

[18] The data science blog: An Intuitive Explanation of Convolutional Neural Networks [online]. [cit. 2020-05-14]. Dostupné z: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[19] Towards data science: A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way [online]. [cit. 2020-05-14]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[20] OpenCV [online]. [cit. 2020-05-14]. Dostupné z: https://opencv.org/

[21] Dlib C++ Library [online]. [cit. 2020-05-14]. Dostupné z: http://dlib.net/

Seznam příloh

A Příloha v IS EDISON

Adresářová struktura souborů v příloze v IS EDISON je následující.

```
testovaci_data
    BioID
    GI4E
trenovaci_data
    iris
    neg
ukazky_detekce
zdrojove_kody_detekce
    CMakeLists.txt
    img_test.png
    predictors
    readme.txt
    src
```

Ve složce testovaci_data/GI4E a testovaci_data/BioID jsou ukázky obrázků z datasetů GI4E a BioID. Ve složce trenovaci_data/iris jsou ukázky pozitivních dat pro trénování a složka trenovaci_data/neg obsahuje ukázky negativních obrázků pro trénování. Ve složce zdrojove_kody_detekce je soubor CMakeLists.txt pro sestavení programu pomocí cmake, dále obrázek img_test.png použitý jako vstup programu a soubor readme.txt, kde je popsáno potřebné nastavení pro sestavení a spuštění programu. Složka zdrojove_kody_detekce/predictors obsahuje prediktor potřebný pro detekci zájmových bodů obličeje, která je využita pro oříznutí oblasti očí. Zdrojové kódy aplikace jsou ve složce zdrojove_kody_detekce/src.