VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE POUŽITÍ RETUŠOVACÍCH FILTRŮ VE SNÍMKU S OBLIČEJEM

DETECT THE USE OF RETOUCH FILTERS IN A FACE IMAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ADAM KRAVÁČEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ GOLDMANN**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Kraváček Adam**
Program: Informační technologie
Název: **Detekce použití retušovacích filtrů ve snímku s obličejem**
Detect the Use of Retouch Filters in a Face Image
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s obrazovými filtry, které se běžně používají pro modifikaci fotek v mobilních aplikacích (např. Instagram, Messenger).
2. Nastudujte a sumarizujte dostupné programy pro identifikaci osob podle snímku obličeje. Především se zaměřte na řešení mladší 5 let.
3. Navrhněte algoritmus, který dokáže detekovat ve snímku s obličejem využití minimálně 4 vybraných filtrů.
4. Navržený algoritmus implementujte v libovolném programovacím jazyce jako multiplatformní aplikaci pro operační systémy Windows a Linux. Implementujte jednoduché uživatelské rozhraní.
5. Proveďte experimenty zaměřené na úspěšnost detekce použitých filtrů. Dále pak zjistěte, jak velký vliv má použití těchto filtrů na úspěšnost identifikace osob. K těmto experimentům využijte existující software pro identifikaci osob na základě obličeje.

Literatura:

- KEE, Eric; FARID, Hany. A perceptual metric for photo retouching. *proceedings of the national academy of sciences*, 2011, 108.50: 19907-19912.
- BIANCO, Simone, et al. (ed.). Computational Color Imaging: 6th International Workshop, CCIW 2017, Milan, Italy, March 29-31, 2017, Proceedings. Springer, 2017.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Goldmann Tomáš, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Úprava snímku pomocí filtrů je jeden z nejjednodušších způsobů jak lze v dnešní době vylepšit jeho vlastnosti. Sociální sítě jako Instagram nebo Snapchat, zaměřené primárně na sdílení snímků, nabízejí svým uživatelům při nahrávání snímků filtry, které upravují barvy ve snímku a dělají jej tak atraktivnějším. V případě získávání snímků z těchto platforem by tak mnoho snímků mělo aplikovaný filtr. Tato práce vysvětluje principy těchto filtrů. Dále se zabývá detekcí filtrů ve snímcích s obličejem, experimentuje s několika různými přístupy. Detekce pomocí analýzy histogramu a detekce konvoluční neuronovou sítí se jeví jako nejlepší, a tak jsou implementovány do programu s jednoduchým uživatelským prostředím. Dosahují úspěšnosti 94,44% (histogram) a 99,10% (neuronová síť). Také je zkoumán vliv filtrů na identifikaci osob, který je závislý na aplikovaném filtru. Některé filtry výrazně zhoršují úspěšnost identifikace osob podle snímku obličeje, zatímco jiné mají minimální dopad. Obecně se však dá říci, že změny způsobené filtry nejsou zanedbatelné.

Abstract

These days, altering images via filters is one of the easiest ways of enhancing its properties. Social networks like Instagram or Snapchat, focused primarily on image sharing, offer their users the option to apply filters on their images, which alter their colours to make them look better. If someone was to extract images from these platforms, many of these images would have a filter applied. This thesis explains the principles of these filters and focuses on detection of filters on facial images. Several approaches to detecting filters are being experimented with. Detection by analysis of histograms and detection by convolutional neural network achieve the best results and so are implemented in a program with a simple user interface. They achieved a success rate of 94,44% (histogram) and 99,10% (CNN). This thesis also investigates the impact of filters on facial recognition, where the impact varies depending on the filter used. Some filters have a significant impact on the rate of successful identifications, whereas others have little impact.In general, however, it can be said that the changes introduced by the application of filters are not negligible.

Klíčová slova

zpracování obrazu, histogram, digitální snímky, obrazové filtry, Instagram, umělá inteligence, strojové učení, detekce filtru, snímky obličeje, Diskrétní Fourierova transformace, CelebA, konvoluční neuronová síť

Keywords

image processing, histogram, digital images, image filters, Instagram, artificial intelligence, machine learning, filter detection, facial images, Discrete Fourier transform, CelebA, convolutional neural network

Citace

KRAVÁČEK, Adam. *Detekce použití retušovacích filtrů ve snímku s obličejem.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Goldmann

Detekce použití retušovacích filtrů ve snímku s obličejem

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Goldmanna. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Adam Kraváček
11. května 2021

Poděkování

Tímto bych chtěl poděkovat Ing. Tomáši Goldmannovi za pomoc, ochotu, a zpětnou vazbu při psaní této práce. Dále chci poděkovat rodině a své přítelkyni za podporu.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá problémem detekce použití filtrů na snímcích s obličejem. Cílem je vytvořit postup pro úspěšné rozpoznání filtrů ve snímcích s obličejem. Řešení je poté implementováno jako program s jednoduchým uživatelským rozhraním.

Focení a sdílení fotek se stalo nedílnou součástí našich životů. Dochází k němu na mnoha platformách. Největší platforma na sdílení snímků, Instagram, má více než miliardu aktivních uživatelů[1]. Nabízí použití retušovacích filtrů při každém nahrávání snímků. Jednoduchost aplikace filtru a jednoznačné zatraktivnění fotografie způsobilo obrovský nárůst popularity jejich užívání.

Tyto filtry ale mění vlastnosti snímků. Vlastnosti, které jsou při některých využitích požadovány nezměněné. Například vytváření datasetů pro trénování umělé inteligence, využívání snímků pro rozpoznání osoby, která vykonala trestný čin, nebo jako důkazy pro trestní stíhání.

Proto je tu projekt vytvoření programu pro detekci použití retušovacích filtrů na snímcích s obličejem, jehož výstupem má být aplikace jednoduchá na použití. Pomocí základního uživatelského rozhraní uživatel odhalí, zda byl na snímek aplikován filtr a jaký. K výslednému produktu se dospělo experimentováním s různými postupy. Všechny tyto postupy jsou v textu popsány i s jejich výsledky. Fungující postupy jsou implementovány (v jazyce Python) a volně dostupné online jako open source.

Následující kapitola 2, Rozpoznávání osob podle snímku obličeje, obsahuje základní pojmy týkající se digitálních snímků a jejich zpracování. Popisuje použité metody zpracování obrazu, algoritmy použité v této práci, sumarizaci a popis jednotlivých kroků identifikace podle snímku obličeje, uvádí existující řešení sloužící k identifikaci osob, a zmiňuje některá relevantní práce k řešení detekce retušovacích filtrů. Kapitola 3, Návrh a implementace detektoru, popisuje použité a relevantní datasety, experimenty s různými způsoby detekce filtrů, návrh detektoru retušovacích filtrů a jeho implementaci a testování. Vliv filtrů na identifikaci osob je popsán v kapitole 4.

[1] https://datareportal.com/social-media-users

Kapitola 2

Rozpoznávání osob podle snímku obličeje

Tato kapitola popisuje způsob, jakým jsou digitální snímky reprezentovány (v sekci 2.1), operace nad snímky používané buď filtry nebo při řešení tohoto projektu (v sekci 2.2), v sekci 2.3 popisuje algoritmy klasifikace, se kterými se pracuje v další kapitole, sumarizuje metody identifikace osob podle snímku obličeje s příklady řešení, a zmiňuje jiné práce relevantní k řešení detekce retušovacích filtrů.

Za praktický počátek fotografie se považuje rok 1839. Řada vědců a chemiků se snažila o zachycení obrazu, avšak nejstarší dochovaný snímek zachytil Nicéphore Niépce. Jako první vyvinul způsob permanentního zachycení snímků využitím světla. Byl to ale negativ, a jeho expozice byla příliš dlouhá pro zachycení lidí (8 hodin až několik dní). Za první zachycení člověka se tak považuje snímek zhruba z let 1838/1839. Louis Daguerre spolu s Niépce zdokonalili původní proces a snížili tak čas expozice (okolo 20 minut) [20]. Od tohoto bodu následoval poměrně rychlý postup technologie. Díky rozrůstající se střední třídě v rámci industriální revoluce rostl i zájem o portrétování, a člověk se tak stal jeden z hlavních cílů fotografie [18].

S postupem času se tak na trh dostaly i kamery a systémy pro kontrolu objektů a osob, označovány jako CCTV (Closed-circuit television, uzavřený televizní okruh). Přestože první kamera byla vyrobena už v první polovině 20. století pro kontrolu startů raket V-2 [15], až od osmdesátých let se začaly rozšiřovat do veřejných prostor, jako jsou věznice, letiště, a banky. Původně se využívaly kamery s analogovým přenosem dat, jejichž koaxiální kabely měly omezenou maximální délku (zhruba 100 metrů). Toto limitovalo jejich využití i kvalitu obrazu. Moderní technologie analogového přenosu dosahují lepší kvality (díky kompresi H.264) a jejich kabely přenášejí video až 500 metrů bez zpoždění a ztrát.

Nejmodernější jsou ovšem digitální kamerové systémy (také IP kamery). Přenášejí svá data pomocí protokolu TCP/IP přes sítě Ethernet. Lze je tak zapojit do již existující infrastruktury, a nemají maximální rozlišení omezené přenosovou technologií. Lze k nim také přistupovat vzdáleně, i když to sebou přináší určitá bezpečnostní rizika. Tento vývoj umožnil ještě větší rozšíření kamer po celém světě [7].

Kvůli velkému množství dat se proto automatizuje jejich zpracování použitím metod rozpoznávání osob. Díky tomu je možné např. pokrýt celé město a poté v něm vyhledávat osoby, nebo jednoduše identifikovat osobu ze záznamu.

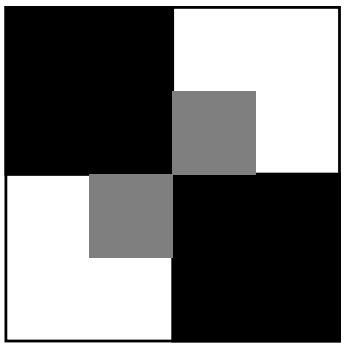
S detekcí a identifikací osob dle obličeje se ale potkáváme v zařízeních které má dnes téměř každý. Moderní telefony disponují s funkcí detekce obličeje pro zaostření kamery,

vyhledává a zaostřuje na obličeje v reálném čase. Aplikace galerie zase na snímcích rozpoznávají dle obličeje osoby, a tak lze procházet snímky nebo v nich vyhledávat dle vyfocených osob, bez manuální klasifikace. Na tyto snímky jsou také aplikovány filtry, které mohou mít vliv na identifikaci pomocí obličeje. Toto vše by ovšem nebylo možné, kdyby snímky nebyly digitální medium.

2.1 Digitální reprezentace snímků

Snímky jsou digitálně reprezentovány jako matice, jejichž jednotlivé hodnoty reprezentují hodnoty pixelů ve snímku (viz obrázek 2.1). Pixely jsou nejmenší částice snímku. Jejich velikost (počet bitů na pixel) záleží na kategorii snímku. Snímky se dají dělit do tří kategorií: binární, odstíny šedi (greyscale), a barevné. Pixely binárních snímků obsahují pouze jeden bit, takže jsou buď bílé nebo černé (1/0). Snímky odstínu šedi jsou sofistikovanější, dokáží znázornit větší množství detailů. Jejich pixely nejsou jen černé nebo bílé, ale mohou mít odstíny šedi. Barevné snímky dokáží reprezentovat ve svými pixely přibližně jakoukoliv barvu [17].

256	256	0	0
256	256	127	0
0	127	256	256
0	0	256	256



Obrázek 2.1: **Ukázka reprezentace snímku maticí pixelů.** Znázorňuje snímek s rozlišením 5×5 v odstínech šedi. Vlevo je vidět matice s hodnotami intenzity pixelů. Vpravo je její zobrazení.

Pro toto řešení jsou nejdůležitější barevné snímky. Právě ty jsou typicky vytvářeny pomocí kamer na telefonech, a jsou potom nejčastěji filtrovány např. při nahrávání na sociální sítě. Typická reprezentace barev ve snímku je model **RGB** Red-Green-Blue (červená-zelená-modrá). Vychází z teorie, že vnímáme barvy pomocí tří typů receptorů v oku. Pixely těchto snímků tak obsahují tři separátní hodnoty (kanály), jednu pro každou z těchto základních barev, které se potom míchají. Snímky s RGBA mají také čtvrtý kanál, nazvaný alfa. Ten udává úroveň průsvitnosti pixelu. Využívá se při úpravách snímků, kdy se mohou snímky překládat přes sebe, nebo při zobrazování snímků na webových stránkách, kde snímkem může prosvítat pozadí stránky [17].

Počet pixelů a poměr stran ve snímku určuje rozlišení, dle rovnice:

$$R = v \cdot s \tag{2.1}$$

kde R je rozlišení, v je výška a s je šířka, v pixelech [17]. Rozlišení se typicky vyjadřuje jako ŠÍŘKA×VÝŠKA, např. snímek s rozlišením 2048×1536 má rozlišení 3,145,728 pixelů, neboli 3,1 megapixelů. Snímky s větším rozlišením můžou obsahovat i více detailů, ale také zabírají více paměti. Při práci s velkým množstvím snímků je tak potřeba zvážit, jaké rozlišení je

potřeba. Nevhodná volba může způsobit pomalejší přenos nebo přesáhnutí kapacity paměti stroje kvůli příliš vysokému rozlišení, či ztrátu podstatných informací příliš nízkou volbou.

Některé snímky ale rozlišení nemají. Snímky lze dělit na rastrové (s rozlišením) a vektorové (bez). Rastrové snímky mají nevýhodu, že při jejich zvětšení mohou být vidět jednotlivé pixely a klesá kvalita. Vektorové snímky se neskládají z pixelů, ale ze základních geometrických útvarů, jako jsou křivky a přímky. Buď jsou ručně vytvořeny specializovaným programem, a nebo použitím programu na převod rastrových snímků na vektorové. Snímky pořízené fotoaparátem jsou rastrové.

2.1.1 Formáty snímků

Obrázek 2.2: **Příklad úrovní komprese u formátu JPEG.** Fotografie s postupně zleva doprava klesající úrovní JPEG komprese (převzato z [19]).

K ukládání digitálních snímků se používá více formátů. Dělí se na ztrátové a bezeztrátové. U bezeztrátových snímků je komprimovaný snímek identický s původním, ovšem komprimace není tak účinná, snímek má větší velikost než se ztrátovou. Ztrátové formáty účinněji redukují velikost snímku. Mohou ho však kompresí upravit tak, že se liší od originálu (rozdílné hodnoty v pixelech). Rozdíly mezi formáty představuje především způsob komprese dat. Nejpoužívanější formáty jsou JPEG, GIF, PNG [17].

- **PNG** (Portable Network Graphics): Open-source bezeztrátový formát. S alfa kanálem (umožňující průhlednost snímku). Využívá vzorců ve snímku ke kompresi.
- **GIF** (Graphical Interchange Format): Vytvoří tabulku s až 256 barvami ze 16 milionů. Pokud má původní obrázek méně jak 256 barev, pak je GIF obrázek identický. Pokud jich má více, tak GIF aproximuje barvy z obrazu barvami z tabulky. Tím je vhodnější pro snímky s méně barvami.
- **JPEG** (Joint Picture Experts Group): Ztrátový (příklad různých úrovní komprese v obrázku 2.2). Optimalizován pro fotografie. Dosahuje vysoké komprese při zachování dobré kvality snímku. Nejčastěji používán v digitálních kamerách.

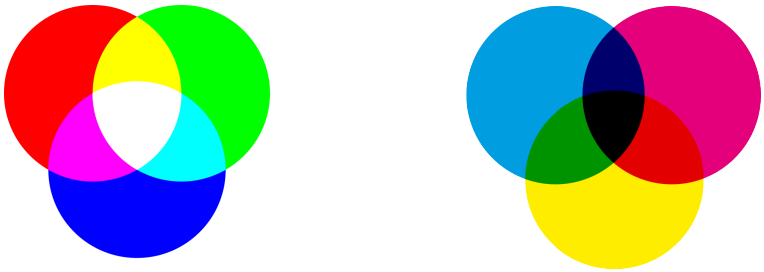
- **TIFF** (Tagged Image File Format): Flexibilní formát, umožňující jak ztrátovou tak i bezeztrátovou kompresi. Informace o kompresi jsou uloženy v samotném snímku.

Některé formáty jsou proprietární. To znamená, že jejich autor (většinou společnost) pomocí licence upravuje možnosti jeho používání. Kvůli těmto omezením bývají méně používané, některé společnosti ale své proprietární formáty zpřístupňují veřejnosti. Například formát BMP od společnosti Microsoft, který má dobrou, veřejně přístupnou dokumentaci [44] a jeho použití není znemožněno patentovou ochranou.

2.1.2 Barvy ve snímku

Tato sekce vychází z knihy *Principles of Digital Image Processing: Fundamental Techniques* [4]. Nejčastější způsob reprezentace barev v obrázku je barevné schéma RGB. RGB (Red-Green-Blue) jsou tři základní barvy (červená, zelená, modrá), z kterých se dají složit všechny ostatní. Při vytvoření snímku jsou zvlášť zachytávány tři intervaly vlnových délek. Pixely mají hodnotu intenzity pro každou z těchto barev. Pokud je reprezentace 8-bitová, každý barevný kanál pak může dosahovat hodnot od 0 do 255. To dohromady poskytuje přes 16 milionů barevných kombinací.

RGB je aditivní barevné schéma. To znamená, že základní barvy se kombinují a vytvářejí tak další, viz obrázek 2.3. Další možností reprezentace barev ve snímku je subtraktivní schéma CMYK (Cyan-Magenta-Yellow-blacK). Je využíváno především při tisku, pro tisknutí s azurovým, purpurovým, žlutým a černým inkoustem. Funguje na principu překrývání barev na bílém pozadí. Inkoust absorbuje světlo, které by se jinak odráželo. Proto se schéma nazývá subtraktivní, inkoust pohlcuje a tak „odečítá“ červenou, zelenou a modrou od bílého světla. Viz obrázek 2.3. Teoreticky by černá barva nebyla potřeba jako inkoust, ale kombinace všech ostatních barev do černé vytváří barevný odlesk, což znemožňuje tisk perfektní černé.

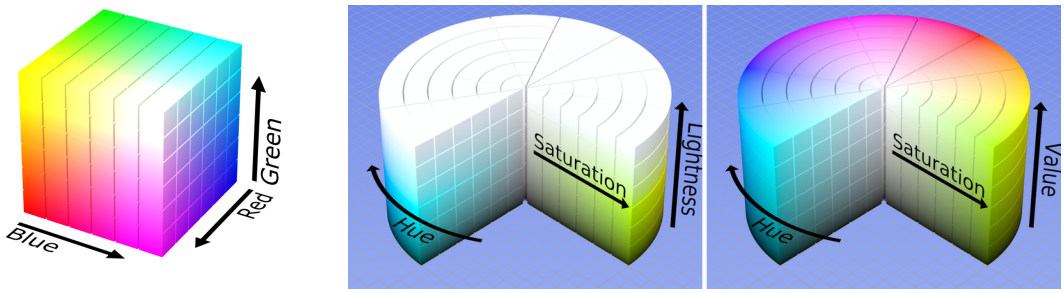


Obrázek 2.3: **Barevné schémata.** Vlevo: RGB, Vpravo: CMYK

Barvy se také popisují v barevném prostoru. RGB i CMYK jsou hardwarově zaměřené modely. Obrázek 2.4 ukazuje prostor definovaný RGB signálem. Je to krychle, protože červená, zelená a modrá mají nezávislé hodnoty. Jejich kombinací se dá vytvořit jakákoliv barva uvnitř krychle.

RGB je vhodné pro použití při programování, díky jednoduchosti manipulace a přímému promítnutí na zobrazovací zařízení. Je však důležité vědět, že vzdálenosti v barevném prostoru RGB neodpovídá přímo našemu vnímání barev (zdvojnásobení hodnoty červeného kanálu nezpůsobí, že nám barva potom připadá dvakrát tak červená). Protože změna souřadnic upravuje barevný tón, saturaci, a jas současně, je výběr barev v prostoru RGB náročný a neintuitivní.

Ruční výběr barev je jednodušší v barevných prostorech, které byly k tomuto účelu vytvořeny. V nich jsou viditelné barevné prvky, jako například saturace, jednotlivě zastoupeny a mohou být nezávisle upravovány. Mezi tyto prostory patří například HSV nebo HSL. Barevný prostor HSV (hue, saturation, value) se skládá ze tří komponent: barevný tón, saturace, hodnota (nebo také jas). Modeluje jak vypadají barvy pod světlem. HSL (hue, saturation, lightness) se skládá z tónu, saturace a světlosti, a modeluje míchání barev, kde světlost představuje množství černé nebo bílé barvy [4].



Obrázek 2.4: **Barevné prostory.** Zleva: RGB (převzato z [22]), HSL a HSV (převzato z [21])

2.1.3 Barevný tón

Barevný tón (odstín) je vlastnost definovaná dominantní vlnovou délkou nebo úhlem od počátečního bodu na barevném kole (viz obrázek 2.4, osa 'Hue' v prostoru HSV). Primární odstíny jsou červená, zelená, a modrá, které se poté promíchávají k vytvoření ostatních odstínů [11].

2.1.4 Kontrast

Kontrast je rozsah efektivně využitých hodnot intenzity snímku. Snímek s vysokým kontrastem využívá celé škály možných hodnot pixelů, zatímco nízký kontrast využívá jen krátkého intervalu v rámci celého rozsahu možných hodnot [4]. Ve snímcích s nízkým kontrastem tak jde hůře rozpoznávat objekty, protože méně vystupují ze svého okolí, celý snímek má podobnou úroveň intenzity. Toto lze jednoduše pozorovat na histogramu (viz sekce 2.2.4).

2.1.5 Jas

Jas je definován jako vlastnost vizuálního vjemu, podle kterého oblast vypadá, že vyzařuje více či méně světla. Je to však těžce vyjádřitelná jednotka. Některé počítačové systémy vyhodnocovaly jas jako průměr všech barevných složek

$$J = \frac{R + G + B}{3} \tag{2.2}$$

kde R představuje intenzitu červené, G intenzitu zelené, a B intenzitu modré, vše většinou na škále 0-256. Toto však neodpovídá tomu, jak lidské oko vnímá barvu. Proto je přesnější výpočet s pomocí vah

$$J = 0,2126 \cdot R + 0,7152 \cdot G + 0,0722 \cdot B \tag{2.3}$$

kde váha každého kanálu odpovídá citlivosti oka na tuto složku [32].

2.1.6 Saturace

Saturace či sytost je barevnost oblasti posuzovaná v poměru k jejímu jasu [10]. Nejvyšší sytosti tak lze dosáhnout použitím jedné barvy s vysokou intenzitou. Při snížení intenzity klesne i sytost. Bílá a černá tak mají nejnižší saturaci, protože všechny barvy mají stejnou intenzitu.

2.2 Zpracování digitálních snímků

Tato sekce popisuje operace prováděné nad snímky v rámci aplikování filtrů (například rotace odstínu) a operace používané při detekci filtrů (například diskrétní Fourierova transformace).

2.2.1 Bodové operace

Bodové operace upravují hodnoty pixelů ve snímku, aniž by měnily jeho velikost, geometrii nebo strukturu. Nová hodnota pixelu závisí jen na předchozí hodnotě pixelu na stejné pozici ve snímku. Je tak nezávislá na ostatních pixelech. Bodovou operaci lze vyjádřit jako

$$a' < -f(a) \tag{2.4}$$

kde a' je nová hodnota pixelu, a je původní hodnota, a funkce *f()* mapuje původní hodnoty na nové. Toto lze vyjádřit i jako

$$I'(u,v) < -f(I(u,v)) \tag{2.5}$$

kde I je hodnota pixelu na pozici *(u, v)*, a I' je nová hodnota na stejné pozici. Pokud je funkce *f()* nezávislá na pozici pixelu, tak se tato operace nazývá globální nebo homogenní. Mezi ně patří modifikace jasu nebo kontrastu, globální prahování, barevné transformace, nebo transformace intenzity.

Pokud se ovšem jedná o mapovací funkci *g()*, která bere v potaz pozici pixelu, jedná se o nehomogenní bodovou operaci dle:

$$a' < -g(a,u,v) \quad nebo \quad I'(u,v) < -g(I(u,v),u,v) \tag{2.6}$$

Typickým příkladem nehomogenní operace je lokální úprava kontrastu nebo jasu pro kompenzaci nerovnoměrného osvětlení při získávání obrazu [4].

Sépie

Převod snímku do odstínu *Sepia* je příklad základní bodové operace. Je dostupný v některých fotoaparátech, a skládají se z něj i další komplexnější filtry. Takto upravené fotografie vypadají jako by byly pořízené v 19. století. Pojem pochází z konce 19. století, kdy se na fotografický pozitiv nanášela sépiová barva získaná z pigmentu sépie obecné [9]. Nad každým pixelem proběhne přepočet hodnot barev dle:

$$R_n = R \cdot 0,393 + G \cdot 0,769 + B \cdot 0,189 \tag{2.7}$$

$$G_n = R \cdot 0,349 + G \cdot 0,686 + B \cdot 0,168 \tag{2.8}$$

$$B_n = R \cdot 0,272 + G \cdot 0,534 + B \cdot 0,131 \tag{2.9}$$

kde R, G a B jsou hodnoty červeného, zeleného a modrého kanálu, a R_n, G_n, B_n jsou jejich nové hodnoty [41].

2.2.2 Filtr

Filtrování je podobně jako bodové operace nástroj k úpravě snímků. Hlavním rozdílem mezi filtrem a bodovou operací je ten, že filtrování typicky využívá více jak jeden pixel pro výpočet nové hodnoty jednoho pixelu ve snímku. Slouží tak k operacím jako je rozmazávání nebo zaostřování [4].

Termín filtr nebo filtrování však nemá jednu přesnou definici, dost často se odvíjí od kontextu jeho použití. Vyskytuje se v mnoha oblastech, ať už filtr vody (nepropouští částice větší než je limit filtru), filtrování signálů (např. dolní nebo horní propust, které nepropouštějí vyšší či nižší frekvence) nebo filtry v informatice, které zpracovávají datový proud (např. program *grep*, který přijímá text a vypisuje jen ten odpovídající jeho parametrům). Tato poslední definice se už více blíží našemu využití, kde je filtrování chápáno jako zpracování snímku, a aplikace různých operací na něj (někdy i jen bodových, např. sépiový nebo černobílý filtr) .

Retušovací filtry využívané na platformách jako je např. Instagram, však v sobě většinou zahrnují několik bodových operací a filtrování. Pomocí kombinace několika metod (změna kontrastu, jasu, rotace odstínu, atd.) se tak dosahuje výsledků moderních filtrů používaných na sociálních sítích.

Obrázek 2.5: **Příklady filtrů ve snímcích s obličejem.** Filtry pochází z aplikace Microsoft Photos. Zleva: Burlesque, Sauna, Neo, Arctic.

2.2.3 Rotace odstínu

Rotace odstínu je jedna z operací, které se provádějí při aplikaci moderních filtrů na snímek. Upravuje barvu ve snímku změnou hodnoty odstínu, který je definován jako úhel od počátečního bodu v kruhu barev. V barevném prostoru HSL nebo HSV to představuje pouze změnu hodnoty kanálu odstínu. Ale protože většina digitálních snímků je uložena v prostoru RGB, nových hodnot po rotaci dosáhneme maticovou operací:

$$\begin{vmatrix} R' \\ G' \\ B' \end{vmatrix} = \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} \times \begin{vmatrix} R \\ G \\ B \end{vmatrix} \tag{2.10}$$

kde proměnných a_{00}, a_{01}, atd. je dosaženo výpočtem z úhlu rotace:

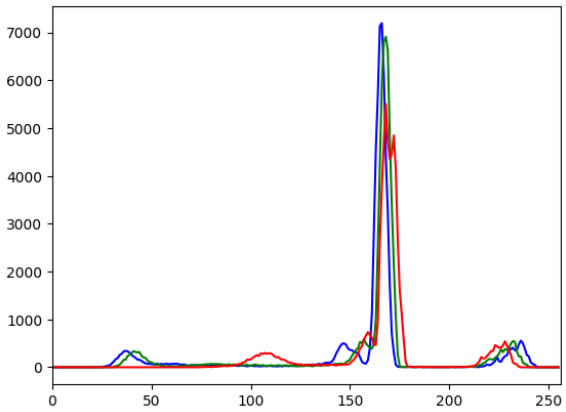
$$\begin{aligned} \begin{vmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{vmatrix} = \begin{vmatrix} +0,213 & +0,715 & +0,072 \\ +0,213 & +0,715 & +0,072 \\ +0,213 & +0,715 & +0,072 \end{vmatrix} &+ \cos R \cdot \begin{vmatrix} +0,787 & -0,715 & -0,072 \\ -0,213 & +0,285 & -0,072 \\ -0,213 & -0,715 & +0,928 \end{vmatrix} \\ &+ \sin R \cdot \begin{vmatrix} -0,213 & -0,715 & -0,928 \\ +0,143 & +0,140 & -0,283 \\ -0,787 & +0,715 & +0,072 \end{vmatrix} \end{aligned} \tag{2.11}$$

kde R je požadovaný úhel rotace odstínu [45]. Dopad rotace odstínu lze vídět na obrázku 2.6, rotace o stejný úhel je např. u filtru *Aden*[1] z knihovny *Pilgram*, kde jednou z operací aplikace filtru je rotace odstínu o 340°.

Obrázek 2.6: **Příklad dopadu rotace odstínu o 340°.** Vlevo je původní snímek, vpravo je po rotaci odstínu. Původní snímek je převzatý z datasetu CelebA [29].

2.2.4 Histogram

Histogram je jednou z nejdůležitějších metod grafického znázornění dat. Klasifikuje data do stejně širokých sloupců, jejichž výška vyjadřuje četnost [35]. Reprezentace snímků histogramem využívá sloupců s šířkou 1, kde hodnota na ose x značí intenzitu barvy a na ose y její četnost ve snímků. Takto histogram popisuje barevné rozložení snímku. Protože snímky obsahují tři barevné kanály (RGB), jeden snímek vytvoří tři histogramy. Pro znázornění trendů v barvách snímku tak jsou tři různé histogramy v jednom grafu na obrázku 2.7.



Obrázek 2.7: **Příklad histogramu RGB snímku.** Graf 3 histogramů, každý od jedné základní barvy. Osa X znázorňuje úroveň intenzity barvy, a osa Y četnost pixelů s danou intenzitou.

[1] Aden – https://github.com/akiomik/pilgram/blob/master/pilgram/aden.py

2.2.5 Šum

Šum ve snímku je náhodná variace barev nebo jasu, většinou vznikající při zachycení snímku. Pro detekci změn ve snímku jde spočítat variance gaussovského šumu v obraze. Používá zero mean operator (nulový průměrný operátor), díky čemuž ji téměř neovlivňuje struktura snímku. Výpočet šumu je dle rovnice 2.13.

$$N = \begin{vmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{vmatrix} \tag{2.12}$$

$$\sigma_n^2 = \frac{1}{36(W-2)(H-2)} \sum_{obrazI} (I(x,y) \circledast N)^2 \tag{2.13}$$

Kde W je šířka a H výška snímku, N maska, I snímek, σ_n standardní odchylka šumu [24].

2.2.6 Diskrétní Fourierova transformace

Tato část 2.2.6 vychází z knihy *Principles of Digital Image Processing: Core Algorithms* [5]. Fourierova transformace je integrální transformace, převádějící signál mezi časově a frekvenčně závislým vyjádřením pomocí harmonických signálů (funkcí sin a cos). Diskrétní transformace je prováděna v diskrétním čase. 2D Fourierova transformace snímku může být vypočítána jako dvě sekvenční jedno-dimenzionální transformace řádků a sloupců (nejdříve se provede 1D DFT nad snímkem po řádcích, a poté nad výsledkem první transformace po sloupcích).

$$G(m,n) = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g(u,v) \cdot e^{-i2\pi(\frac{mu}{M} + \frac{nv}{N})} \tag{2.14}$$

Kde M a N jsou velikost snímku, g vstupní funkce (snímek), G výstupní komplexní funkce, m, n jsou spektrální souřadnice $m = 0, 1, ..., M-1$, $n = 0, 1, ..., N-1$. Výsledná Fourierova transformace je tedy znovu 2D funkce stejné velikosti ($M \times N$) jako původní snímek.

Převedení snímku do frekvenční domény pomocí dekompozice signálů snímku do harmonických funkcí sinus a cosinus použitím Fourierovy transformace nám umožňuje analyzovat jeho geometrické charakteristiky. Změnami ve frekvenční doméně lze ovlivnit určité frekvence, čímž můžeme měnit geometrickou strukturu snímku.

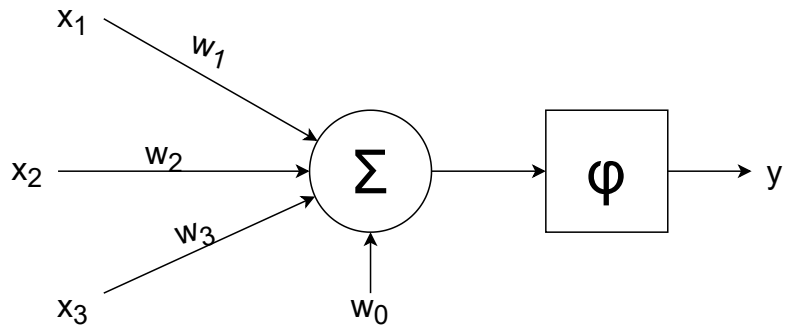
2.3 Klasifikační algoritmy

2.3.1 Viola-Jones

Viola-Jones je algoritmus detekce objektů ve snímku, primárně používán pro detekci tváří. Využívá Haarových příznaků pro detekci obličeje. Funguje díky tomu, že lidské tváře mají společné prvky – například oblast očí je tmavší než horní líce, oblast nosního můstku je světlejší než oči. Tyto prvky jsou poté hledány ve snímku s tváří. Protože je ale nejprve důležité obličej ve snímku najít, prochází se postupně po částech. Porovnávat každou část se všemi možnými příznaky potřebnými pro určitou identifikaci obličeje by ale bylo výpočetně náročné. Proto tento algoritmus využívá kaskádové architektury - stačí porovnat jen s dvěmi příznaky pro vysokou (téměř 100 %) přesnost. Pokud tato část projde, postupuje

na další úroveň, kde je porovnána s více příznaky. Když ne, tak se hledání v této části přeruší, a postupuje se dále ve snímku. Díky tomuto kaskádovému přístupu je hledání obličejů rychlé, ale kvůli způsobu hledání pomocí Haarových příznaků má problémy s tvářemi nakloněnými a pod úhlem [42].

2.3.2 Umělé neuronové sítě



Obrázek 2.8: **Model (umělého) neuronu**

Zdrojem informací v této části 2.3.2 je kniha *Applied Machine Learning* [16], pokud není uvedeno jinak. Umělé neuronové sítě představují výpočetní model používaný v oboru umělé inteligence. Jejich vzorem je funkcionalita biologických neuronových buněk člověka. V současnosti jsou neuronové sítě úspěšně aplikovány na řešení celé řady úkolů z mnoha oblastí, jako jsou obchod, průmysl, doprava, apod. [31].

Nejjednodušším modelem umělé neuronové sítě je tzv. perceptron. Skládá se právě z jednoho neuronu. Typicky se však neurony spojují do vrstev. První vrstva přijímá vstupy do klasifikátoru v různých formách. Poslední vrstva vydává výsledek klasifikace. Vrstvy mezi nimi (takzvané skryté vrstvy) mapují svůj vstup do výstupů, které další vrstva považuje za užitečné. Tímto způsobem se síť naučí vlastnosti, které poslední vrstva shledává užitečné. Proto jsou neuronové sítě zejména užitečné při klasifikaci snímků – sama si najde důležité vlastnosti vstupních objektů. To je důvodem proč jsou nejúspěšnější klasifikátory snímků, jako třeba sítě trénované metodou Meta Pseudo Labels[2], postaveny na principech neuronových sítí.

Umělé neuronové sítě se skládají z (umělých) neuronů, uspořádaných ve vrstvách a propojených vazbami. Každý přijímá množinu vstupů a argumentů, z kterých produkuje číslo které je nelineární funkcí vstupů a parametrů, jak jde vidět na obrázku 2.8, kde:

- $\mathbf{x_i}$ je vstup s indexem i.
- $\mathbf{w_i}$ váha pro vstup s indexem i.
- $\mathbf{w_0}$ je bias (práh).
- φ je aktivační funkce.
- **y** je výstup neuronu.

[2]Meta Pseudo Labels – https://paperswithcode.com/paper/meta-pseudo-labels

A platí:

$$y = \varphi(\sum_{i=1}^{p} w_i x_i + w_0) \tag{2.15}$$

Vstupy **x** mohou být výstupy z jiných neuronů, nebo podněty prostředí. **p** je počet vstupů. Váhy udávají důležitost jednotlivých vstupů. Práh $\mathbf{w_0}$ je speciální případ váhy, manipuluje vstup do aktivační funkce φ [16]. Příkladem jednoduché aktivační funkce je skoková funkce, která je použita v binárním perceptronu, výstup **y** potom bude buď -1 nebo 1 [40]. Standardně používanou aktivační funkcí je avšak usměrněná lineární funkce, krátce ReLU (Rectified Linear Unit), která je nulová pro záporné vstupní hodnoty, ale lineární pro kladné, dle definice 2.16

$$\varphi(u) = max(0, u) \tag{2.16}$$

kde φ je aktivační funkce ReLU, a u je její vstup [16].

Hledání správné hodnoty vah w se nazývá trénování. Toho se dosahuje určením tzv. ztrátové funkce, která odhaduje míru chybovosti klasifikace, a následovným hledáním hodnot minimalizujících výsledek této funkce. Příkladem ztrátové funkce použité v tomto projektu může být křížová entropie

$$\begin{aligned} \frac{1}{N}\sum_i L_{\log}(y_i, s(o(x_i, \theta))) &= \frac{1}{N}\sum_i [-\log p(\text{kategorie příkladu i}|x_i, \theta)] \\ &= \frac{1}{N}\sum_{i \in data} [-y_i^T \log s\ (o(x_i, \theta))] \end{aligned} \tag{2.17}$$

kde N je počet příkladů vstupů x, jejichž kategorii známe. Celkový počet tříd je C. Vektor y_i obsahuje *1 z n* zakódovanou (one hot encode) kategorii příkladu, a je C-dimenzionální. *1 z n* kódování znamená, že pokud je i-tý příklad z kategorie j, potom j-tý prvek vektoru y_i je jedna a všechny ostatní jsou nulové. θ je vektor všech koeficientů (vah), o je vektor prvků (neuronů) sítě a s je funkce softmax, která bere C dimenzionální vektor a vrací C dimenzionální vektor který je nezáporný a jehož prvky mají součet jedna [16].

Sada parametrů θ s nízkým výsledkem ztrátové funkce se typicky hledá pomocí **stochastického gradientního sestupu**. Nepočítá s celým gradientem, ale s odhadem gradientu s náhodnou chybou, díky tomu je výpočetně efektivnější. Je to často vybíraný algoritmus při trénování komplexních klasifikátorů sestavených z více jednotek. Ve vícevrstvých sítích se gradient počítá pomocí zpětné propagace, kdy se určí dopad jednotlivých vah na konečné chybě [16].

Ne vždy je ale přímý sestup gradientu nejlepší způsob, jak minimalizovat funkci. Například při sestupu s pevnou délkou kroku čelíme problému vybrání správné délky, dlouhý krok se k minimu přiblíží rychle, ale bude kolem něj konvergovat a měnit znaménko (překročí minimum). Krátký krok zase povede k pomalému sestupu do minima. Pro rychlý sestup v místech s menší změnou gradientu, a pomalejší sestup v místech s větší změnou, lze využít tzv. optimalizace. Proto tento projekt využívá metodu **Adam**, což je gradientní optimalizace prvního řádu, založená na adaptivních odhadech momentů nižšího řádu [26]. Upravuje tak délku kroku sestupu gradientu (rychlost učení). Je to standardně používaná optimalizační metoda, implementovaná například v knihovnách Keras[3] nebo PyTorch[4].

Neuronové sítě lze dělit na sekvenční a rekurentní. **Sekvenční** sítě jsou organizované ve vrstvách, kde všechny neurony z jedné vrstvy jsou propojeny se všemi neurony následující vrstvy, a při klasifikaci se informace šíří pouze jedním směrem (od vstupní vrstvy

[3] https://keras.io/
[4] https://github.com/pytorch/pytorch/

ke koncové). Používají se např. při řízení dynamických systémů. **Rekurentní** neuronové sítě mohou uchovávat informace ve vnitřní „paměti“, kdy neurony mohou být propojeny s neurony předchozích vrstev. Jsou tak vhodnější pro použití na proudech dat, např. na řeči nebo textu [1].

Sítě s více skrytými vrstvami se nazývají hluboké. Hluboké sítě jsou populární pro použití při řešení problémů s velkým množstvím dat. Jedním z nejčastěji využívaných druhů hlubokých neuronových sítí jsou konvoluční neuronové sítě. Jsou využívány k rozpoznávání snímků pomocí konvoluce. Skládají se z plně propojených vrstev, konvolučních vrstev, pooling a nelineárních vrstev. Jsou úspěšné při použití nad obrazovými daty, jako je klasifikaci snímků, zpracování přirozeného jazyka, a počítačové vidění [1].

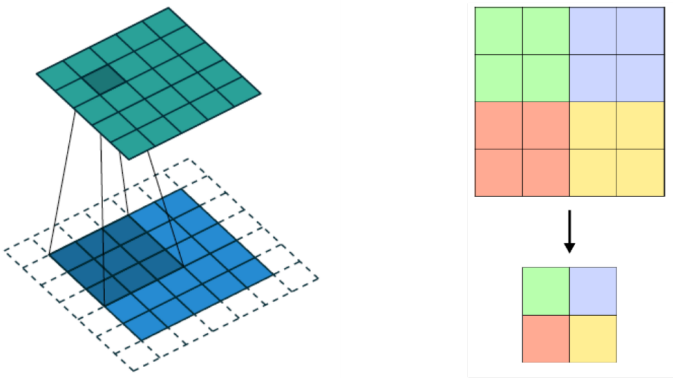
Konvoluce je matematická operace, díky které dokážeme vypočítat podobnost dvou signálů. Je typicky využívána pro hledání vzorů ve snímcích, a je vhodná pro klasifikaci snímků [16]. Aplikuje postupně na snímek **konvoluční matici** (také označovanou jako kernel), pomocí které se poté síť dokáže naučit vzorce ve snímku (jako např. hrany, při více vrstvách konvoluce i složitější vzory, jako např. kruh). Výsledkem konvoluce na snímku je tzv. **příznaková mapa** (feature map). Pro výpočet příznakové mapy N na jedné pozici (pro jeden pixel) se přes místo na původním snímku I přeloží matice W, vynásobí se prvky co leží na sobě, a sečtou se výsledky. Obecně konvoluce signálu je

$$N_n = (I \circledast W)_n = \sum_{m=-\infty}^{\infty} I_m W_{n-m} \tag{2.18}$$

Toto lze upravit pro konvoluci snímků. Jeden bod příznakové mapy N je tak po aplikaci konvoluční matice W na snímek I roven:

$$N_{ij} = (I \circledast W)_{ij} = \sum_{k} \sum_{l} I_{k,l} W_{i-k,j-l} \tag{2.19}$$

kde i, j jsou souřadnice nového snímku, a k, l snímku před konvolucí [12]. Operace konvoluce je znázorněna na obrázku 2.9. Pro optimalizaci algoritmu konvoluce lze překládat matici přes každý např. druhý pixel. Tomuto parametru se říká **krok** (stride). Pro krajní případy, kdy matice přesahuje přes okraj snímku, lze přidat tzv. **padding**, což je přidání sloupců (popř. řádků) ke snímku, typicky s nulovou hodnotou. Konvoluční vrstvy typicky obsahují několik různých konvolučních matic.

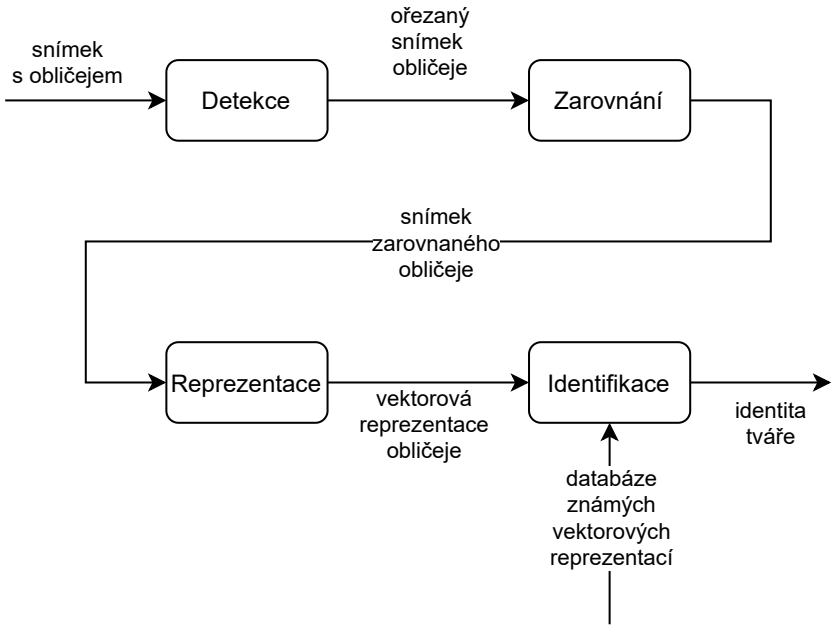


Obrázek 2.9: Vizualizace operací konvoluce (vlevo) s konvoluční maticí velikosti 3×3 a pooling s oknem 2×2 (vpravo) (převzato z [30]).

Při použití kroku o velikosti jeden pixel ale můžeme dostávat redundantní informace, protože aplikované matice se budou ve výsledku překrývat. Tomu lze zabránit buď použitím konvoluční matice menší velikosti a zvětšit krok, nebo použít **pooling** vrstvu. Pooling vrstva (někdy označována jako subsampling) sumarizuje její vstup, typicky buď výpočtem průměru nebo maximální hodnotou (viz obrázek 2.9. Na mapu příznaků na vstupu pooling vrstvy se aplikují nepřekrývající se okna specifikované velikosti (typicky 2×2 nebo 3×3 pixely), ty spočítají buď průměrnou nebo maximální hodnotu v okně, a z jejich výstupů se poté složí snímek menší velikosti (poloviční, při použití 2×2 okna) [16].

2.4 Identifikace osob podle snímku obličeje

Identifikace osob podle snímku obličeje je v dnešní době typicky prováděna pomocí **hlubokých konvolučních neuronových sítí** (DCNN – Deep Convolutional Neural Network) [13]. Hluboké sítě jsou vhodné zejména pro velké množství dat, čehož využila např. společnost Facebook s jejich modelem pro identifikaci *DeepFace*, který dosáhl výsledků podobných člověku. Dalším významným pokrokem v tomto oboru byl například model *FaceNet*, který se učí přímo mapovat snímek do euklidovského prostoru, kde vzdálenost přímo odpovídá míře podobnosti tváře. Tyto řešení a další jsou popsány dále v tomto textu. Typický postup pro identifikaci podle obličeje ze snímku se sestává ze čtyř kroků: **detekce**, **zarovnání**, **reprezentace**, a **klasifikace** [39] (viz obrázek 2.10). Moderní řešení identifikace obličeje dosahují dobrých výsledků v kontrolovaných prostředích, ale jejich přesnost klesá v přirozených podmínkách vzniku snímku, například při nerovnoměrném osvětlení, různých typech obličeje nebo pleti kůže (které se nevyskytovaly v trénovacích datech), rozdílných výrazech tváře, a obzvláště při zakrytí části obličeje (např. slunečními brýlemi) [47].



Obrázek 2.10: Vizualizace kroků procesu identifikace.

2.4.1 Detekce

Detekce obličeje ve snímku a jeho ořezání je první krok pro identifikaci. I když existují už oříznuté a zarovnané datasety (například CelebA [29]), pro praktické použití modelu je to nezbytný krok.

Detekce obličeje typicky probíhá pomocí konvenčních algoritmů, jako například Viola-Jones (popsaný v kapitole 2.3.1), nebo pomocí neuronových sítí. Například **výzkum identifikace tváří v přirozeném prostředí** [47] zmiňuje použití *Multi-task Cascaded Convolutional Network* (MTCNN) pro detekci obličeje. Projekt **OpenFace** pro detekci obličeje před-trénované modely z knihoven dlib[5] nebo OpenCV[6] [2].

Řešení mohou využívat různé způsoby detekce tváří, důležité je, aby tato metoda detekovala obličeje přesně, a případně našla i výchozí body obličeje pro další krok: zarovnání. Dále by také měla být výpočetně nenáročná, protože trénování konvolučních neuronových sítí probíhá na velkých počtech dat.

2.4.2 Zarovnání

Zarovnání obličeje je důležitá část připravení snímku pro jeho reprezentaci. Za cenu zvýšené režie může zvýšit přesnost modelu a snížit variabilitu mezi různými snímky stejné identity tím, že všechny snímky připraví do stejné pozice. Model je tak robustnější, např. natočení snímku nebo vyfocení z různých úhlů na něj nemá takový vliv.

Projekt **OpenFace** používá pro zarovnání obličeje směrem na kameru nejdříve odhad tváře v reálném čase z projektu dlib, díky čemuž zjistí jak je potřeba transformovat tvář pomocí afinní transformace implementovanou knihovnou OpenCV. Každý snímek se snaží transformovat tak, aby oči a spodní ret byly na stejném místě v každém snímku [2].

DeepFace používá sofistikovanou metodu 3D zarovnání, která je vyobrazena na obrázku 2.11. Na začátku jsou detekovány základní body obličeje, pomocí kterých je obličej zarovnán v rámci 2D. Tato metoda ovšem nedokáže vykompenzovat rotace mimo rovinu, které se vyskytují především ve snímcích z nekontrolovaného prostředí. Proto je dalším krokem 3D zarovnání, kdy se 2D zarovnaný obrázek promítne na 3D tvar, který je potom normalizován, čímž je snímek tzv. *frontalizován*. To znamená, že obličej z původního snímku je transformován na pohled, který bychom viděli při předním (frontálním) pohledu do tváře [39].

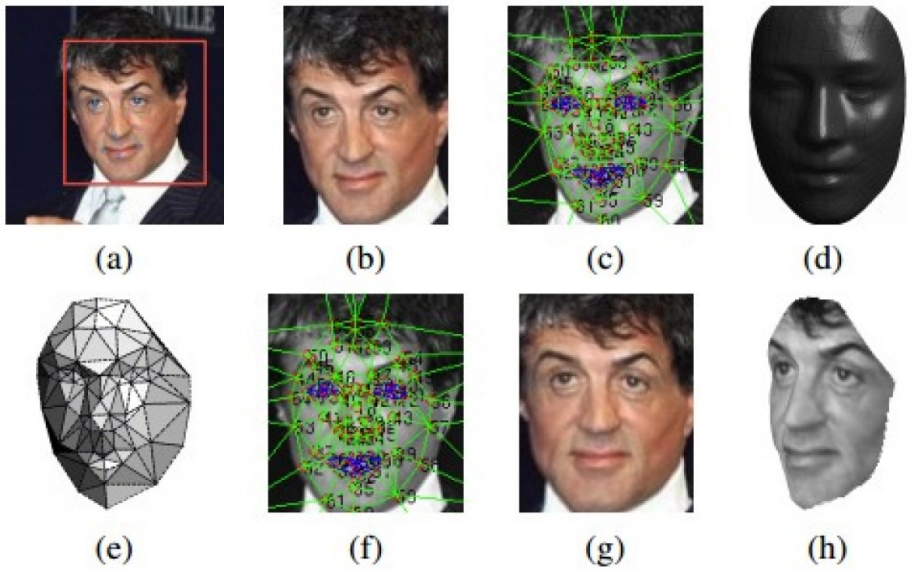
Oproti tomu např. řešení **FaceNet** prozkoumává možnost nezarovnaných snímků, kdy bez zarovnání dosahuje jen o 0,76 % horší přesnosti než verze se zarovnáním na datasetu LFW (*Labeled Faces in the Wild* [23]). Není tak jasné, zda je pro toto řešení vhodné použít zarovnání, které zvyšuje celkovou komplexnost procesu identifikace [34].

2.4.3 Reprezentace

Reprezentace tváře většinou obnáší přenesení příznaků tváře ze snímku do vícedimenzionálního prostoru. V tomto prostoru jsou příznaky stejné identity blízko sebe, zatímco rozdílné identity by od sebe měly být vzdálené.

V **DeepFace** je frontalizovaný snímek předán hluboké konvoluční neuronové sítí, která jej převede na vektor příznaků [39]. Tato síť obsahuje na jejím konci dvě plně propojené vrstvy, které mají zachytit souvislosti mezi zachycenými příznaky, jako např. pozicí a tvarem očí a nosu a pozicí a tvarem úst. Výstup první plně propojené vrstvy je 4096-dimenzionální

[5] dlib – http://dlib.net/
[6] OpenCV – https://opencv.org/



Obrázek 2.11: **Vizualizace procesu zarovnání v projektu DeepFace.** (a) Detekovaný obličej. (b) Ořezaný a 2D zarovnaný obličej. (c) 67 výchozích bodů obličeje a odpovídající Delaunayho triangulace. (d) Referenční 3D objekt transformovaný do roviny 2D zarovnaného obrázku. (e) Viditelnost trojúhelníků z 3D-2D kamery: tmavší trojúhelníky jsou méně viditelné. (f) 67 výchozích bodů vyvozených 3D modelem, které se používají k řízení afinní transformace. (g) Konečný frontalizovaný snímek. (h) Nový pohled generovaný 3D modelem (není součástí DeepFace). Převzato z [39].

vektor, který reprezentuje danou tvář. Výstup druhé vrstvy je předán softmax funkci. Pro zobecnění sítě k identifikaci osob, které se nevyskytovaly v trénovacích datech, se používá výstup první plně propojené vrstvy. Poslední vrstva je tak v síti kvůli trénování. Nevýhodou tohoto přístupu jsou jeho nepřímost a neefektivnost, protože není jisté, zda bude generalizovat správně na nových datech, a velikost výstupního vektoru je velká (DeepFace produkuje 4096-dimenzionální vektor [39]). Oproti tomu **FaceNet** trénuje přímo aby jeho výstup jako byl 128-dimenzionální vektor reprezentující identity. Dosahuje toho použitím ztrátové funkce založené na *Triplet Loss* [34] (toto je více popsáno u řešení FaceNet níže).

Důležitým aspektem správné reprezentace je také trénování modelu a jeho ztrátové funkce. Existuje několik ztrátových funkcí, které se snaží o lepší výsledky při reprezentaci: menší vzdálenost mezi vektory stejné identity, a větší vzdálenost mezi vektory rozdílných identit. Mezi tyto funkce se řadí *CosFace* [43], *SphereFace* [28], *Triplet Loss* [34], a *ArcFace* [13].

2.4.4 Klasifikace

Pro klasifikaci po převodu obličejů na vektory se počítá vzdálenost mezi dvěma výsledky reprezentace. Prvním referenčním, uloženým například v databázi, a vyhodnocovaným obličejem. Počítá se buď euklidovská nebo kosinová vzdálenost.

Nejčastěji používanou metodou pro vyhodnocení, zda dva snímky tváře obsahují stejnou osobu, je **Joint Bayesian model** [47]. Modeluje příznakovou reprezentaci tváře x jako součet normálních rozložení intrapersonální a interpersonální vzdálenosti. Interpersonální vzdálenost μ je vzdálenost mezi dvěma třídami (rozdílné identity), a intrapersonální vzdá-

lenost ϵ platí v rámci jedné třídy (variance snímků jedné identity). Poté platí, že:

$$x = \mu + \epsilon \tag{2.20}$$

kde μ a ϵ jsou modelovány jako Gaussovo rozdělení a jsou odhadnuty z trénovacích dat. Verifikace tváří s Gaussovými rozděleními x_1, x_2 je pak dosaženo poměrem logaritmické pravděpodobnosti $r(x_1, x_2)$

$$r(x_1, x_2) = \log \frac{P(x_1, x_2|H_I)}{P(x_1, x_2|H_E)} \tag{2.21}$$

kde H_I je intra-personální hypotéza (že snímky *náleží* stejné identitě), a H_E je mimo-personální (extra-personal) hypotéza (že snímky *nenáleží* stejné identitě). Poté se z problému verifikace tváře stává problém klasifikace rozdílu $x_1 - x_2$ jako intra-personální nebo mimo-personální variance [6]. Když získáme poměr $r(x_1, x_2)$, stačí jej porovnat s prahovou hodnotou. Pokud je poměr roven nebo větší než prahová hodnota, snímky náleží stejné identitě [47]. Prahovou hodnotu můžeme získat vyhodnocením z trénovacích dat.

2.4.5 Příklady řešení identifikace osob podle snímku obličeje

V této části je popsáno několik významných nebo state-of-the-art řešení identifikace osob podle snímku obličeje.

DeepFace

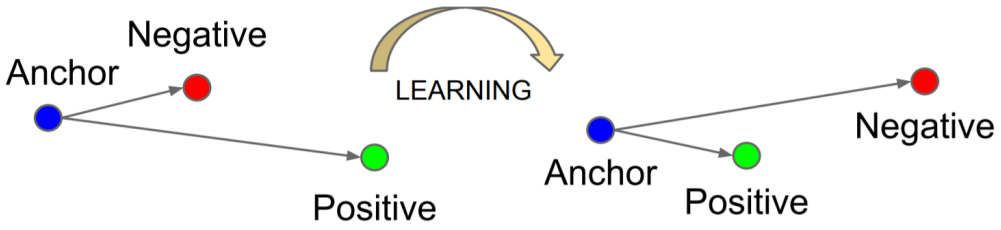
DeepFace reprezentuje nový přístup k identifikaci pomocí hlubokých neuronových sítí. Předchozí řešení využívaly převážně konvenčních metod strojového učení, jako je například metoda podpůrných vektorů (Support Vector Machines – SVM), analýza hlavních komponent (Principal Component Analysis – PCA), nebo lineární diskriminační analýza (LDA). Díky většímu množství dostupných dat a výpočetního výkonu může DeepFace utilizovat hluboké učení k dosáhnutí vysoké přesnosti, konkrétně 97,35 % na datasetu *Labeled Faces in the Wild*.

Vylepšuje metodu zarovnání tváře pomocí analytického 3D modelování obličeje. Zarovnává nejprve v 2D rovině pomocí detekce významných bodů tváře v několika iteracích, dokud není změna mezi obrázky při zarovnání mezi iteracemi zanedbatelná. Poté je provedeno 3D zarovnání, kde je použit 3D model pro transformaci významných bodů tváře do frontálního pohledu. Po zarovnání je barevný snímek obličeje předán hluboké neuronové síti. Ta se skládá z první konvoluční vrstvy s 33 maticemi (filtry) o velikosti $11 \times 11 \times 3$ (okno 11×11 pro všechny barevné kanály RGB), pooling vrstvy vybírající maximální hodnotu z 3×3 okna s krokem 2, další konvoluční vrstvy s 16 maticemi velikosti $9 \times 9 \times 16$. Tyto první tři vrstvy mají za cíl extrahovat nízkoúrovňové prvky, jako hrany a textury. Pooling vrstva je aplikována jen jednou, aby se předešlo ztrátě informací. Poté následují tři lokálně propojené vrstvy, které fungují na stejném principu jako konvoluční, jen má každá souřadnice ve snímku vlastní konvoluční matici (oproti konvoluční vrstvě, kde je použita jedna matice na celém snímku). Je využito toho, že vstupní snímek je zarovnaný, a tak filtry odpovídají jejich místům (například oblast mezi obočím a očima se výrazně liší od oblasti mezi nosem a ústy). Poslední dvě vrstvy jsou plně propojené a slouží k zachycení korelace mezi prvky obličeje. Výstup poslední plně propojené vrstvy je zpracován softmax funkcí na rozložení mezi kategoriemi.

Pomocí kombinace několika modelů trénovaných s různými počátečními parametry dosahuje DeepFace přesnosti 97,35 % na datasetu LFW. Při porovnání s předchozími vydanými řešeními a lidskou přesností v rámci přesnosti a ROC křivky (Receiver Operating Characteristic – popisuje kvalitu klasifikátoru v závislosti na jeho klasifikačním prahu) se tak výsledky DeepFace blíží těm lidským [39].

FaceNet

FaceNet vychází z a vylepšuje DeepFace. Narozdíl od něj se ale učí mapovat přímo do kompaktního euklidovského prostoru. Pro trénování používá trojici snímků (*triplet loss*): referenční snímek (anchor), pozitiv, a negativ (viz obrázek 2.12). Při trénování jsou tak zvoleny kromě vstupního (referenčního) snímku ještě dva další, pozitiv, který má stejnou třídu (identitu) jako referenční snímek, a negativ, který má jinou třídu. Tak můžeme určit výsledný vektor referenčního snímku, jehož vzdálenost od vektoru pozitivu minimalizujeme, ale vzdálenost vektoru negativu maximalizujeme. Pro vyšší rychlost trénování se nepracuje se všemi možnými trojicemi, ale vybírají se takové pozitivy a negativy, které budou mít co největší dopad – liší se co nejvíce od referenčního snímku. Jsou vybírány metodou tzv. *online mining* – přímo při trénování z konkrétní trénovací dávky [34].



Obrázek 2.12: **Vizualizace trénování s Triplet Loss.** Triplet Loss minimalizuje vzdálenost vektorů stejné třídy, ale maximalizuje vzdálenost vektorů rozdílných tříd. Převzato z [34].

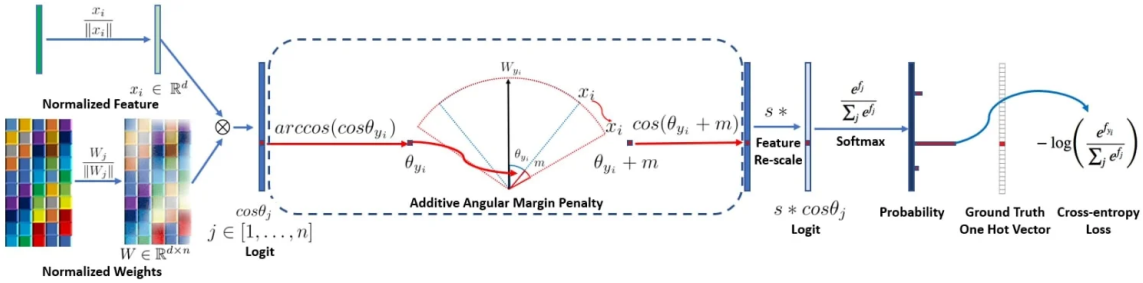
Výsledkem je efektivnější reprezentace tváře – pouze 128 bytů. Navíc model dosahuje vysoké přesnosti i bez sofistikovaného zarovnání tváře, kdy na snímcích stačí pouze těsně oříznout obličej. I přesto poté dosáhne přesnosti 98,87 % na datasetu Labeled Faces in the Wild. Při použití zarovnání dosahuje nové rekordní přesnosti 99,63 % [34].

ArcFace

ArcFace je vylepšení ztrátové funkce stávajících modelů hlubokých konvolučních sítí. Při fázi reprezentace, kdy se snímky převádí na vektor ve více-dimenzionálním prostoru, se využívá speciálních ztrátových funkcí při trénování k vylepšení přesnosti. Tyto funkce se snaží maximalizovat vzdálenost mezi různými třídami (identitami), ale minimalizovat vzdálenost v rámci jedné třídy. Nejrozšířenější ztrátovou funkcí používanou pro klasifikaci je softmax, která ovšem neoptimalizuje explicitně pro tyto vlastnosti, což má dopad na identifikaci při variaci uvnitř tříd (např. rozdílné pózy nebo změna věku) a při použití rozsáhlých testovacích scénářů.

Metoda ArcFace upravuje výsledek vyhodnocení normalizovaných příznaků a vah přičtením penalizace před předáním hodnot funkci softmax (viz obrázek 2.13). Autoři demonstrovali s populárními architekturami konvolučních sítí, ResNet50 a ResNet100, na několika datasetech, že tato metoda rozlišuje lépe mezi třídami než samotná funkce softmax

nebo předchozí používané funkce k tomutu účelu, jako například Triplet Loss nebo SphereFace [13].



Obrázek 2.13: **Vizualizace principu ztrátové funkce ArcFace.** Z normalizovaného příznaku x_i a váhy W se získá logit $\cos\theta_j$ pro každou třídu jako $W_j^T x_i$. Poté se vypočte $\arccos\theta_{y_i}$ a získá úhel mezi příznakem x_i a ground-truth váhou W_{y_i}. Přičte se *angular margin penalty* (úhlový posun příznaku) m k cílovému (ground-truth) úhlu θy_i, vypočte se $\cos(\theta y_i + m)$ a všechny logity vynásobíme příznakovou škálou s. Logity poté pokračují do softmax funkce a podílejí se na ztrátě křížovou entropií k trénování sítě. Převzato z [13].

OpenFace

OpenFace je DCNN model založený na řešení FaceNet, optimalizovaný pro použití na mobilních zařízeních, jehož zdrojový kód je volně dostupný. Jeho cílem je nabídnout volně dostupné state-of-the-art řešení, oproti řešením FaceNet a DeepFace, což jsou soukromé modely které dosáhly své přesnosti trénování na datech nedostupných veřejnosti. Zatímco FaceNet byl trénován na 100-200M snímků a DeepFace byl trénován na 4,4M snímků, OpenFace využívá pouze 500k snímků spojením dvou datasetů pro identifikaci podle snímku obličeje, CASIA-WebFace a FaceScrub[7]. I tak dosahuje dobré přesnosti a výkonosti na datasetu Labeled Faces in the Wild, konkrétně 92,92 % [2].

2.5 Dostupná řešení detekce retušovacích filtrů

V této části je popsáno několik studií, které se zabývají podobnou problematikou jako tato bakalářská práce. Zaměřují se na detekci změn ve snímku obecně. Většinou se vyskytují ve formě popisu postupu detekce, nikoliv jako veřejně dostupného programu nebo knihovny. Na trhu se nevyskytuje program čistě pro detekci použití filtrů ve snímcích s obličejem.

2.5.1 Rozpoznávání filtrovaných snímků s destylizací příznaků

Studie zabývající se vlivem aplikace filtrů na přesnost CNN (konvolučních neuronových sítí). Zjistila, že vizuální efekty filtrů mají významný dopad na nejvýkonnější sítě (ResNet50) sloužící k rozpoznávání objektů. Filtry přenášejí informace o stylu do příznakových map, a tak se studie snaží najít způsob jak tyto informace o stylu odstranit [46].

Představuje tak odlehčený destylizační modul, obsahující plně propojenou neuronovou síť s pěti vrstvami, která přijímá jako vstup kódované příznakové mapy, a jejím výstupem je sada parametrů pro změnu měřítka a posun příznakových map tak, aby vyrušily změny

[7] FaceScrub – http://vintage.winklerbros.net/facescrub.html

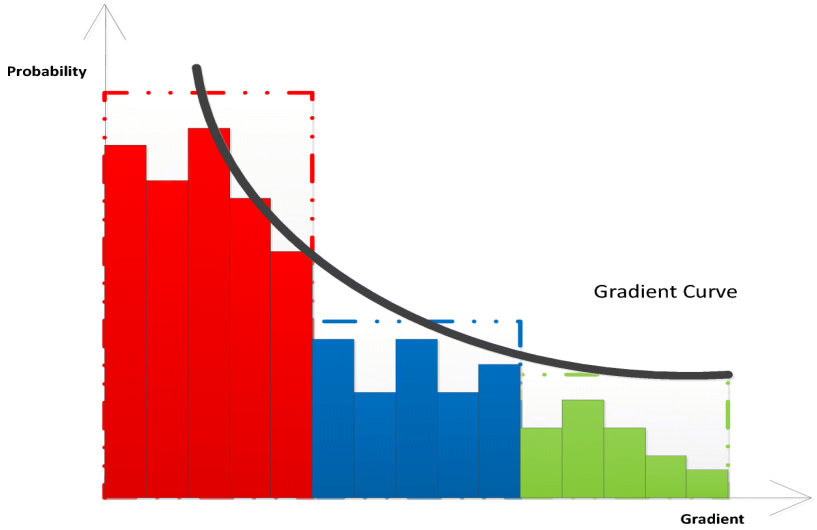
způsobené filtry. Slouží k vylepšení generalizace CNN, do kterých může být zapojena. Pro mne bylo na této studii nejdůležitější její prokázáni vlivu moderních filtrů na výkonnost neuronových sítí, kde po aplikaci filtrů na dataset *ImageNet* přesnost modelu ResNet50 klesla ze 76,13 % na 67,22 % [46].

2.5.2 Detekce retušování

Prozkoumává vliv retušování na rozpoznávání obličeje využitím Supervised Deep Learning. Obsahuje návrh algoritmu založeného na Supervised Restricted Boltzmann Machine (SRBM). Zaměřuje se na klíčové oblasti tváře (oči, ústa, nos), extrahované pomocí Viola-Jones detektoru (viz část 2.3.1). Každá část se poté předává své vlastní Supervised Deep Boltzmann machine (SDBM), která byla na danou část obličeje trénována. Výstupní příznaky tváře z SDBM jsou předány Support Vector Machine (SVM) pro klasifikaci, zda byla část obličeje retušována. Výsledkem je přesnost okolo 87 %. Zaměřuje se na ručně prováděné retušování v programech jako je například Adobe Photoshop [3].

2.5.3 Detekce vyhlazování

Detekce použití vyhlazovacích filtrů na snímku. Využívá detekce hran a následné analýzy histogramu gradientu a analýzy textur. Vyhlazené snímky mají na hranách nižší hustotu a jiné rozložení frekvencí. Gradientní křivka (obrázek 2.14) u neupravených snímků bývá lineární, zatímco upravené snímky mají buď hrany rozdělené do nízkých a vysokých frekvencí (bilaterální filtr), nebo hrany jen s nízkými frekvencemi (ostatní filtry). Detekce vyhlazování dosahuje úspěšnosti nad 90 % [14].



Obrázek 2.14: Grandientní křivka (gradient curve) a její histogram (převzato z [14]).

2.5.4 Hodnocení úrovně retušování

Návrh metriky pro hodnocení změn v retušovaném snímku. Obsahuje osm statistik hodnotících dopad geometrických a fotometrických změn. Tyto změny se poté hodnotí na škále od jedné do pěti podle toho, jak moc se liší od neupraveného snímku. Metrika by mohla být použita např. u upravených fotografií v časopisech nebo online. Tato studie tedy nedetekuje změny, jen je hodnotí. Metrika musí mít dostupné oba snímky (originální a upravený) [25].

Kapitola 3

Návrh a implementace detektoru

Tato kapitola se věnuje relevantním datasetům, experimentům s různými způsoby detekce retušovacích filtrů, konečnému návrhu detekce retušovacích filtrů, jeho implementaci a testování.

3.1 Data

V rámci této práce se používají datasety k trénování i testování přesnosti řešení. Dataset je sbírka dat, v našem případě sbírka snímků obličeje, které mají anotace. Např. v datasetu určeném k identifikaci osob ze snímku podle obličeje musí být ke snímku poskytnuta i informace, kdo se v něm nachází. Datasety jsou využívány k trénování a ohodnocování metod strojového učení. K řešení problému pomocí hlubokých neuronových sítí je klíčové získat správný dataset o dostatečné velikosti, aby se síť dokázala zobecnit řešení vyžadovaného problému (aby síť tzv. generalizovala). Pro tuto práci jsou významné dva následující datasety.

3.1.1 CelebA

Při implementaci a testování této implementace byl využit dataset **CelebA** [29], konkrétně verze *Align&Cropped* (zarovnané a oříznuté snímky). Obsahuje více než 200 000 snímků, každý s jedním obličejem celebrity. Na tomto datasetu jsem testoval algoritmy, posuzoval jejich vhodnost a trénoval modely neuronové sítě. Všechny obsažené snímky mají velikost 178×218, s kterými pracuji bez zmenšování. Zmenšením snímků by šlo dosáhnout rychlejšího vyhodnocování a méně náročného programu (díky menšímu modelu), jenže by se mohly vytratit vlastnosti snímku důležité pro klasifikaci. Dataset obsahuje i 40 anotací vlastností, které ale nejsou využity, k tomuto projektu stačí pouze samotné snímky. Snímky obsahují velké množství rozdílných pozadí, osob, a typů osvětlení, díky čemuž by algoritmy založené na těchto datech měly lépe generalizovat.

3.1.2 Labeled Faces in the Wild

Labeled Faces in the Wild (LFW) je dnes průmyslový standard pro testování modelů určených k identifikaci osob ze snímku obličeje. Obsahuje mnoho fotografií obličejů v nekontrolovaných podmínkách, za různého osvětlení, z různých úhlů, s různými pozadími, a i různými zakrytími obličeje. Snaží se tak přiblížit reálným podmínkám, za kterých je často potřeba identifikovat osobu. Nabízí několik verzí, i s už zarovnanými a oříznutými

snímky [23]. V této práci je několikrát zmíněna u různých modelů identifikace osob podle snímků obličeje, a je i použita při testování vlivu filtrů na identifikaci osob.

3.2 Návrh experimentálních přístupů k detekci filtru

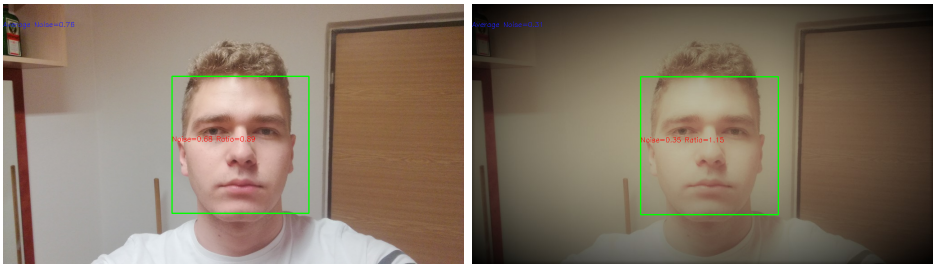
Tato sekce obsahuje popis různých přístupů k detekci filtru, které byly vyzkoušeny v první části vývoje tohoto projektu. Popisuje i jejich úspěšnost, a zda je jejich použití proveditelné.

3.2.1 Detekce šumu v obličeji

První pokus o řešení problému. Snímky by po aplikaci retušovacího filtru měly mít nižší šum v sekci obličeje. Bylo využito metody pro výpočet variance gaussovského šumu v obraze 2.2.5. Šum se počítal v oblasti obličeje, která byla detekována pomocí algoritmu **Viola-Jones** (viz sekce 2.3.1). Tento přístup byl jednoduchý na použití s OpenCV API pro Python, proto byl použit na prototypování, kde jeho nedostatky tolik nevadí. Abych nemusel trávit čas trénováním jen kvůli prototypu, využil jsem předem připraveného modelu pro hledání tváří, dostupného z OpenCV repozitáře.

První snímky, na kterých jsem prototyp testoval, jsem získával vlastním focením a následně ručním aplikováním filtru v aplikaci Fotografie od společnosti Microsoft zahrnutým v operačním systému Windows 10, protože například Instagram nenabízí dostupné API na filtrování obrázků nebo možnost stahovat publikované snímky.

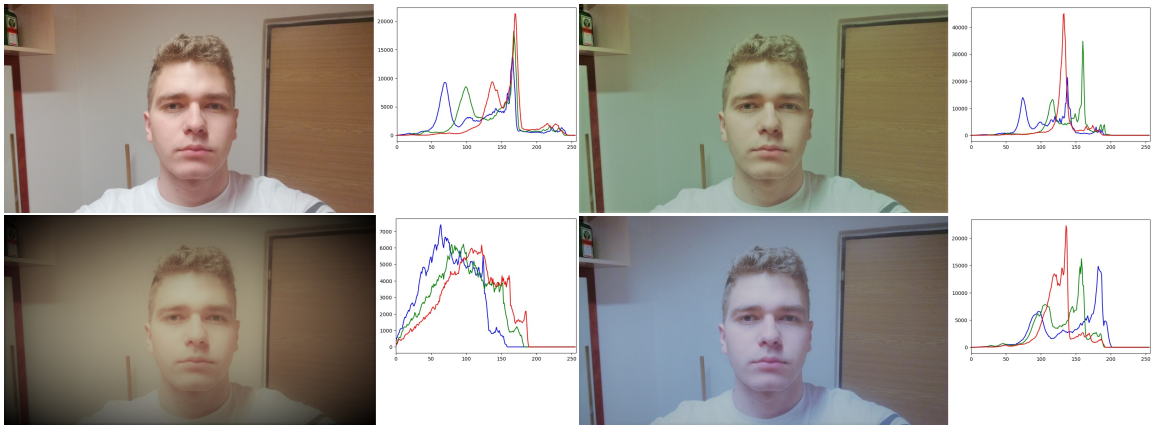
Snímky s použitým retušovacím filtrem vykazovaly většinou nižší hodnoty šumu, než jejich neupravené protějšky. I tak ale často záleželo na druhu retušovacího filtru. Některé filtry dokonce šum zvýšily (viz obrázek 3.1) Problematické však bylo určování hranice úrovně šumu, kdy je obrázek filtrovaný a kdy ještě ne. Najít tuto hranici šumu, která rozdělí všechny snímky správně na filtrované/nefiltrované však nebylo možné, protože se intervaly těchto hodnot výrazně protínají. Metoda funguje dobře na porovnávání dvou stejných obrázků -– ten filtrovaný bude mít většinou nižší hodnotu šumu v obličeji. Další možností, místo porovnávání hodnoty šumu s konstantou, bylo porovnání šumu v obličeji s průměrným šumem ve zbytku snímku. To je problematické, protože různé typy pozadí mají různé hodnoty šumu, a mnoho snímků s obličejem obsahuje i efekt vyhlazení nebo rozostření pozadí.



Obrázek 3.1: **Příklad výsledků pokusu se šumem v obličeji.** Levý obrázek – originál, šum v obličeji 0,68 Pravý obrázek – filtrovaný, šum v obličeji 0,35 .

3.2.2 Analýza barev ve snímku

Dalším řešením byla analýza rozložení barev ve snímku pomocí barevného histogramu (více o histogramech v kapitole 2.2.4). Pomocí OpenCV2 Python API jsem načetl obrázek, a pro



Obrázek 3.2: **Histogramy snímků.** Filtry od levého horního rohu: Bez filtru, Neo, Burlesque, Arctic. Všimněte si intervalů nulových hodnot v grafu u filtrů na místech, kde u původního obrázku nejsou.

každou barvu vytvořil histogram pomocí *opencv2.calcHist()*. Tak byly vytvořeny tři histogramy, s rozsahem hodnot 0-256. Ty byly převedeny do grafu knihovny matplotlib[1].

Filtry upravující hodnoty barev jdou poměrně jednoduše zjistit díky abnormálním hodnotám intenzity barev (viz obrázek 3.2), kde jsou nulové shora omezené intervaly, které by se v normálním snímku neobjevily. Takto lze poměrně spolehlivě detekovat některé filtry. Navíc určité filtry mají krajové oblasti, rohy především, s výrazně nižší intenzitou než ve středové oblasti snímku. Proto je dalším možným postupem, pokud daný filtr neprodukuje nulové intervaly barevného histogramu přes celý snímek, rozdělení snímku na devět sekcí, u kterých se také provede analýza intenzity barev. V krajních sekcích se poté mohou vyskytovat nulové intervaly, a celková intenzita barev je mnohem nižší.

3.2.3 Diskrétní fourierova transformace

Třetím pokusem byla Diskrétní Fourierova transformace 2.2.6. Výstup této funkce představuje snímek ve frekvenční doméně. Skládá se ze dvou částí – komplexní a reálné (magnituda a fáze). K analýze využijeme pouze magnitudy Fourierovy transformace, protože obsahuje většinu informací o struktuře snímku. Toto lze implementovat v jazyce Python pomocí OpenCV2 funkcí *cv2.dft()*. Poté je ještě potřeba pro jednodušší čitelnost přesunout nulovou frekvenci do středu spektra (poslouží funkce *numpy.fft.fftshift()*) z rohů. Díky tomuto posunu jsou potom nízké frekvence ve středu snímku, a vysoké na jeho okrajích. Rozsah výsledných hodnot je ale moc velký pro zobrazení v obrázku. Proto provedeme logaritmickou transformaci, aby byly výsledky v rozsahu 0-255. Ve výsledku potom můžeme vidět rozložení frekvencí snímku.

Na obrázku 3.3 lze vidět, že filtr změnil výstup DFT. Ovšem nepodařilo se mi najít způsob, jak spolehlivě rozlišit samostatný snímek, zda je filtrovaný nebo ne, popřípadě jej převést na vektor pro porovnání. Proto jsem se přestal věnovat tomuto přístupu, a pokračoval s ostatními.

Přímý výpočet DFT by měl dle definice složitost $O(N^2)$. Proto se používá tzv. rychlá Fourierova Transformace (FFT), s $O(N \log N)$. Ta je použita v implementaci nejrozšíře-

[1]Matplotlib – https://matplotlib.org/

Obrázek 3.3: **Spektrální analýza snímku s filtrem** ukazuje, že se filtrovaný snímek (filtr Burlesque, snímek dole) liší od originálu.

nějších matematických programů (jako například Matlab, GNU Octave, atd.). I metoda použitá v tomto řešení z knihovny OpenCV, metoda *dft()*, využívá algoritmu FFT.

3.2.4 Umělá neuronová síť na klasifikaci filtrů z histogramů snímků

Kvůli velkému vlivu filtrů na barvy snímku jsem se nejdříve zaměřil na klasifikaci pomocí histogramů snímků. Domníval jsem se, že dosáhnu vyšší přesnosti, když nebudu brát v potaz obsah snímku, ale jen jeho barevné rozložení.

Vytvořil jsem jednoduchý sekvenční model neuronové sítě 2.3.2 pro vyzkoušení tohoto přístupu. Byl implementován v Pythonu s dvěma plně propojenýma skrytými vrstvami z knihovny Keras[2] na platformě Google Colaboratory[3]. Tyto vrstvy využívaly usměrněnou lineární aktivační funkci (RELU). Poslední, výstupní, vrstva, zodpovědná za rozdělení výsledků do dvou kategorií: filtrované a nefiltrované, využívala aktivační funkce sigmoid.

Síť byla trénována na histogramech snímků z datasetu CelebA [29] obsahující snímky s obličeji. Vybral jsem 10 000 snímků, na které byl aplikován filtr z Python knihovny pilgram[4], a ty, i s jejich nefiltrovanými ekvivalenty. 70 % snímků jsem použil na trénování, 30 % na validaci sítě.

Pro optimalizaci parametrů, jako je počet skrytých vrstev, jejich velikost a velikost vzorku jsem použil vizualizačního nástroje TensorBoard. Díky tomuto nástroji jsem mohl lépe vybrat konfiguraci sítě, která dosahovala lepších výsledků při trénování.

I tak ale byla přesnost správné klasifikace obrázků s filtry pouze 68,4 %. Proto jsem přešel na další přístup k řešení problému, klasifikaci přímo ze snímků místo jejich histogramů.

[2]Keras – https://keras.io/
[3]Google Colaboratory – https://colab.research.google.com/
[4]https://github.com/akiomik/pilgram

3.2.5 Umělá neuronová síť na klasifikaci filtrů ze snímků

Pro detekci s použitím snímků místo histogramů jsem znovu využil datasetu CelebA. Kvůli rozpoznávání 4 různých filtrů a omezené velikosti dostupné paměti pro výpočty jsem vybral jen 2 000 snímků, u kterých jsem na každý aplikoval 4 různé filtry z knihovny pilgram.

Model sítě jsem vytvářel v Pythonu, s pomocí knihovny Keras. Pro zjištění validity tohoto přístupu jsem na začátku vytvořil jednoduchou dopřednou síť. Vstupní vrstva odpovídala velikosti snímků z datasetu (218×178). Další byla *Flatten* vrstva, kvůli transformaci snímků z 2D pole do jedné dimenze. Za ní byly dvě hustě propojené vrstvy s aktivační funkcí *ReLu* (Rectified Linear), o velikostech 64 a 32. Výstupní vrstva využívala aktivační funkci softmax, která vrací pravděpodobnostt, že snímek patří do daných kategorií.

Daná síť dosahovala validační přesnosti 83 %, proto jsem se pro tento přístup rozhodl při tvoření finálního řešení.

3.3 Návrh detektoru retušovacích filtrů

Finální program je spojením úspěšných přístupů k detekci použití filtrů na snímcích s obličejem, které se dají použít jednoduchým multiplatformním uživatelským rozhraním. Neslouží k detekování na velkém množství snímků současně, ale jen na jednom snímku. Uživatel vybere snímek uložený na disku, a použije vybranou metodu detekce. Po vyhodnocení se mu zobrazí výsledek detekce, u vyhodnocení neuronovou sítí i odhadnutá pravděpodobnosti příslušnosti do dané kategorie. Z experimentů jsem vyhodnotil dvě metody jako schopné úspěšné detekce filtrů, analýzu histogramů a neuronovou síť přijímající na vstupu snímky. Možné výsledky detekce jsou buď jeden ze čtyř vybraných filtrů nebo žádný filtr (na snímek nebyl aplikován filtr). Detekovatelné filtry jsou: *Nashville*, *1977*, *Toaster* a *Inkwell* (viz obrázek 3.4), z knihovny Pilgram[5].

Obrázek 3.4: **Příklad detekovaných filtrů.** Příklad snímku, na který jsou aplikovány detekovatelné filtry, a snímek bez filtru, pro porovnání. Původní snímek převzatý z datasetu CelebA [29]. **Zleva: Nashville, 1977, Toaster, Inkwell, snímek bez filtru**.

3.3.1 Detekce analýzou histogramu

Některé filtry mají výrazný a konstatní dopad na rozložení barev ve snímku, díky čemuž se u většiny snímků pohybují hodnoty v některých intervalech pod určitými hranicemi (viz obrázek 3.5). Toto řešení nemá 100 % přesnost, je náchylné na snímky s abnormálním rozložením intenzit barev, ale na druhou stranu může mít lepší výsledky na snímcích co

[5]Pilgram – https://github.com/akiomik/pilgram

se výrazně liší od datasetu na kterém byla trénována neuronová síť, a je rychlejší a méně náročné než detekce neuronovou sítí.

Z analýzy mnoha histogramů lze určit trendy, které se vyskytují v histogramech některých filtrů. Z nich můžeme sestavit podmínky, které musí histogram snímku splňovat, aby šel kategorizovat jako filtrovaný. Díky experimentování s různými způsoby detekce pomocí histogramů jsem došel k závěru, že nejefektivnější je určit horní hranici hodnot na intervalech v histogramu, na které má daný filtr výrazný vliv. Tyto intervaly jdou určit pomocí vykreslení maximálních hodnot mnoha různých histogramů snímků s jedním filtrem. Další možností je spodní hranice (pomocí této metody se ale nevyřadí dostatek snímků s ostatními filtry), nebo omezení průměrem na intervalu, kde přesnost závisí na podobnosti snímku ke snímkům, na kterých se určovala daná hranice.

K určení intervalů hranic pro detekci filtrů a jejich hodnot jsem si vykreslil horní hranici dosáhnutých hodnot pro dané filtry (jako na obrázku 3.5). Pomocí analýzy těchto grafů jsem stanovil několik intervalů maximálních hodnot v histogramu snímku. Cílem bylo dosáhnout takové hranice, aby do ní spadaly všechny snímky vybraného filtru, ale co nejméně snímků s ostatními filtry a nebo bez filtru. Laděním této hranice jsem dosáhl konečné přesnosti na vzorku 40 000 snímků z datasetu CelebA (dataset nebyl použit celý kvůli náročnosti testovacího skriptu na paměť), viz tabulka 3.1:

Aplikovaný filtr:	Nashville	1977	Toaster	Inkwell	Bez filtru
Hranice pro filtr Nashville	99,96 %	29,52 %	1,25 %	0,77 %	2,92 %
Hranice pro filtr 1977	4,29 %	99,97 %	18,02 %	3,18 %	8,6 %
Hranice pro filtr Toaster	0,49 %	4,20 %	99,94 %	0,30 %	7,52 %
Hranice pro filtr Inkwell	0,00 %	0,00 %	0,00 %	100,00 %	0,01 %

Tabulka 3.1: **Propustnost podmínek na histogram pro detekci filtrů.** Procenta určují počet snímků s aplikovaným filtrem (daný legendou nahoře tabulky, osa x), které splňují podmínky na maximální hodnoty v histogramu určitého filtru (legenda vlevo).

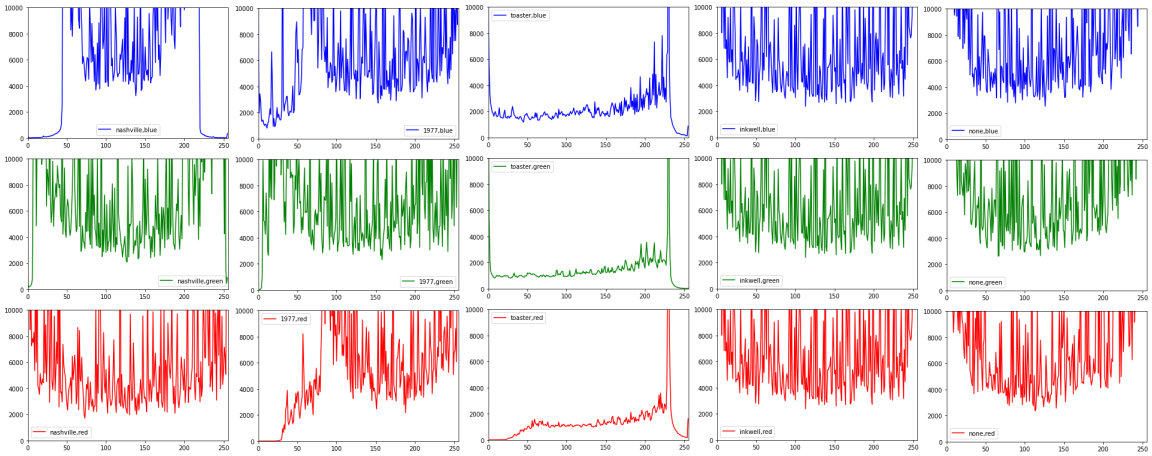
3.3.2 Detekce pomocí neuronové sítě

Experimentováním jsem zjistil, že přímé předání obrázku dosahuje vyšší přesnosti než použití histogramu na vstupu (viz sekce experimentování s neuronovou sítí a histogramy, 3.2.4). Neuronová síť tak přijímá snímky velikosti 178×218 (velikost snímků datasetu CelebA). Výstupem sítě je odhadovaná pravděpodobnost, s kterou patří daný snímek do různých kategorií výstupu. Výsledkem této metody detekce je tak kategorie s nejvyšší pravděpodobností, že do ní snímek patří. Kategorií výstupu je celkem pět, čtyři detekované filtry a jedna kategorie pro snímky bez filtru.

3.3.3 Návrh uživatelského rozhraní

Bylo zvoleno jednoduché uživatelské rozhraní, kde uživatel může vybrat snímek a zobrazí se mu, zda je filtrovaný (a jakým filtrem) nebo ne. Jeho součástí bude i ukázka detekovatelných filtrů, jednak aby uživatel sám mohl vidět, zda výsledek programu dává smysl (může se stát, že výsledek nebude správný), a aby hned věděl které filtry program detekuje. Dále musí uživatel vidět obrázek, který si vybral, a tlačítka na detekci.

Uživatel si bude moct vybrat mezi dvěma metodami detekce, pomocí histogramu nebo neuronové sítě. Metody můžou mít rozdílnou přesnost na různých typech snímků, a pokud



Obrázek 3.5: **Histogramy maximálních hodnot snímků s filtry.** Histogramy s maximálními hodnotami z 40 000 snímků datasetu CelebA. Pro každou úroveň intenzity barvy (osa x) byla vybrána maximální hodnota počtu pixelů s touto intenzitou (osa y). Každý filtr má tři histogramy, jeden pro každou základní barvu (odpovídající barvě v grafu). z histogramů jde vidět, že operace aplikace filtru výrazně omezují hodnoty v některých částech histogramů. **Histogramy maximálních hodnot zleva: Nashville, 1977, Toaster, Inkwell, snímek bez filtru**

se obě metody shodnou na stejném výsledku, tak to může sloužit uživateli k potvrzení jejich výsledku. I načítání modelu, jednorázová událost, bude na uživateli, aby se model nenačítal pokud uživatel nechce. Načítání zabere více času než detekce pomocí modelu, a je tak lepší aby uživatel chvíli počkal po zmáčknutí tlačítka na načtení modelu, a pak mohl pomocí tlačítka na detekci už jen (rychleji) detekovat. Poté se uživateli zobrazí výsledek detekce, a bude moct buď ukončit program, nebo vybrat další obrázek.

3.4 Implementace

Projekt je implementován v jazyce Python verze 3.8.5. Python je rychlý, objektově orientovaný, vysokoúrovňový programovací jazyk vhodný pro rychlý vývoj aplikací[6]. Python jsem zvolil kvůli podpoře pro různé platformy, jednoduchém experimentování v něm a následném převedení experimentů v produkční kód, a kvůli následujícím použitým nástrojům:

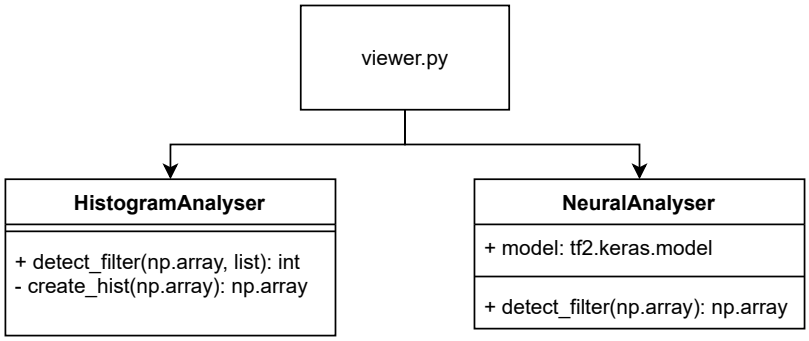
- **Jupyter Notebook**[7]: webová aplikace k vytváření dokumentů s kódem, umožňuje rozdělení kódu do samostatně spustitelných bloků, podporuje interaktivní výstupy jako obrázky nebo grafy, čehož jsem využil hlavně při ladění konstant u detekce analýzou histogramu, nebo trénování různých modelů neuronové sítě, a podporuje mnoho jazyků včetně Pythonu.
- **Google Colaboratory**[8]: Webová služba poskytující prostředí s Jupyter Notebook s webovým rozhraním. Umožňuje programovat pouze z prohlížeče. Zdarma poskytuje prostředí až s 12GB paměti RAM a nVidia Tesla P100 GPU, zdroje však nejsou garantovány a mohou být omezeny nebo odebrány. Při zájmu o výkonnější prostředí

[6]Python – https://www.python.org/doc/essays/blurb/
[7]Jupyter Notebook – https://jupyter.org/
[8]Google Colaboratory – https://research.google.com/colaboratory/faq.html

lze za 9,99$ měsíčně získat přístup k prostředím s rychlejšími grafickými kartami a větší velikostí paměti RAM.

- **Keras**[9]: Otevřený (open-source) Python framework nad knihovnou TensorFlow pro vývoj neuronových sítí, který díky svému jednoduchému a konzistentnímu API umožňuje rychlý vývoj a prototypování modelů. Obsahuje části architektury neuronových sítí, jako jsou vrstvy, aktivační funkce, ztrátové funkce a optimizéry, které jsou připravené pro jednoduché použití programátorem, stačí z nich jen sestavit architekturu sítě.
- **TensorFlow**[10]: Knihovna pro strojové učení, poskytuje Python a C++ rozhraní. Původně pro interní použití v Google, nyní otevřený software.
- **OpenCV-Python**[11]: Knihovna umožňující volání OpenCV metod pro zpracování obrazu z Pythonu. Pro reprezentaci dat z C++ využívá v Pythonu datových struktur knihovny Numpy. Zde využíván pro generování histogramů snímků.

Implementace se skládá ze tří částí (viz obrázek 3.6), které jsou popsané v následujících sekcích. Dvou tříd na detekci v samostatných souborech a hlavního skriptu, ve kterém je definována hlavní smyčka programu a uživatelské rozhraní.



Obrázek 3.6: Ilustrace rozložení implementace programu.

3.4.1 Detekce analýzou histogramu

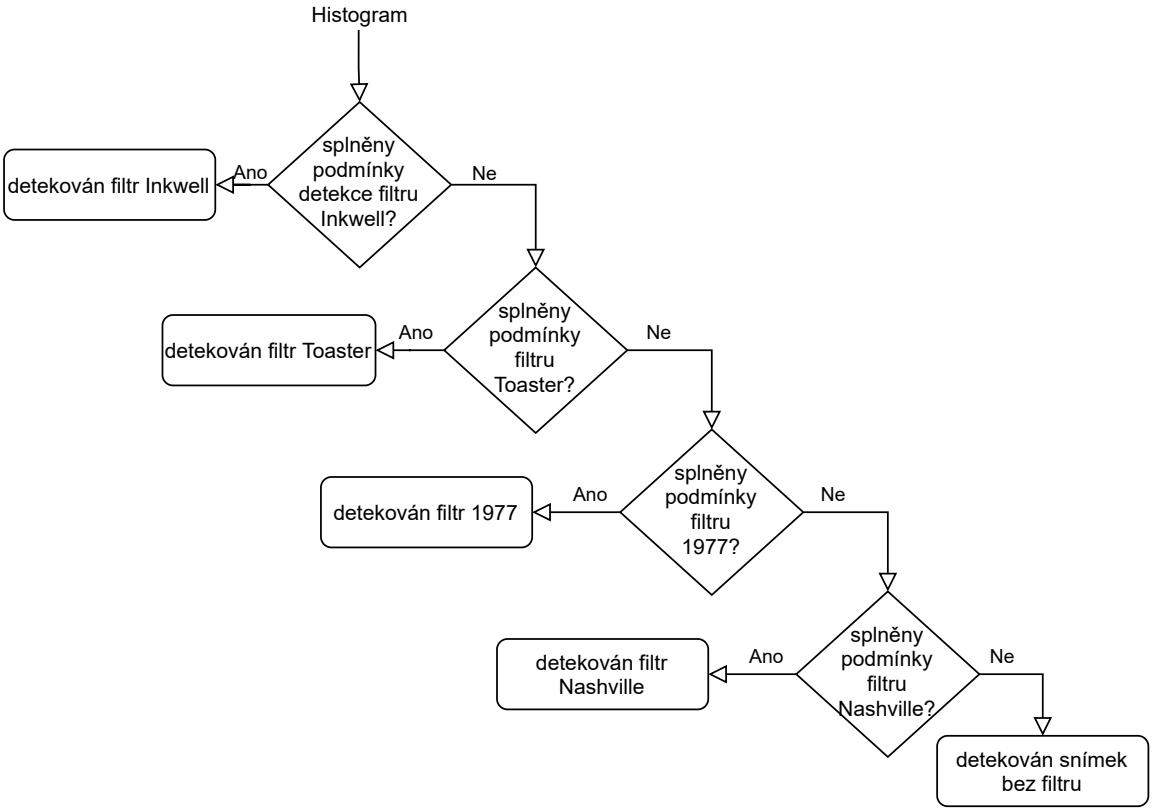
Třída *HistogramAnalyser* pro detekci filtru pomocí histogramu se nachází v souboru *histogram_analyser.py* a obsahuje dvě metody, *___create_hist()* a *detect_filter()*.

Metoda *___create_hist()* je metodou pomocnou. Volá se jen na začátku hlavní metody pro detekci pro převedení snímku na histogram. Nejprve převede snímek z objektu Image knihovny Pillow do formátu ve kterém s ním mohou pracovat metody z knihovny OpenCV (objektu array z knihovny Numpy), a poté pomocí metody *create_hist()* z OpenCV vypočítá histogram zvlášť pro každý barevný kanál. Snímek je na začátku objekt Image z knihovny Pillow protože tato knihovna podporuje mnoho formátů při načítání snímků z disku. Tyto histogramy se poté vloží do jednoho pole a metoda je vrátí. Kvůli tomu, že tuto metodu není potřeba volat mimo její třídu, je tato metoda pouze pomocnou a mohla by být soukromou. Python ale soukromé metody nepodporuje, narozdíl od jazyků jako třeba

[9]Keras – https://keras.io/
[10]TensorFlow – https://github.com/tensorflow/tensorflow/
[11]OpenCV-Python – https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html

Java, a tak jsou na začátku jména metody dvě podtržítka jako označení metody pro vnitřní použití (dle Pythonovské konvence kódu).

Hlavní metoda této třídy je však metoda *detect_filter()*. Slouží k detekci filtru na snímku. Je statická, neboť nevyžaduje instanci třídy ani třídu jako parametr. Přijímá pouze RGB snímek k detekování a seznam filtrů, které lze detekovat, v poli. Seznam filtrů se předává aby při případné změně pořadí filtrů nebo i změně detekovatelných filtrů byla vrácená hodnota odpovídající detekovanému filtru, protože metoda vrací index z tohoto pole odpovídající názvu detekovaného filtru. Zavoláním předchozí zmíněné metody *___create_hist()* jej nejprve převede na histogramy. Poté následuje rozhodovací mechanismus konstrukce if z pěti větví (odpovídající pěti výstupním kategoriím), kde v podmínkách jsou limity maximálních hodnot v intervalech histogramu. Například filtr Nashville zde má šest různých intervalů s omezením maximální hodnoty pro modrý barevný kanál, a tři pro zelený kanál. Protože některé podmínky, kromě filtru, který detekují, mají i přesah a akceptují i větší část snímků s jiným filtrem, viz tabulka 3.1, kde podmínka filtru Nashville akceptuje i 29,52 % snímků s filtrem 1997, je zde použito konstrukce if. Tento problém lze vyřešit tak, že se nejdříve kontroluje, zda je na snímek aplikovaný filtr 1977. Pokud je, metoda vrátí index od kategorie 1977, detekce filtrů se ukončí. Pokud ovšem na snímku není aplikovaný filtr 1977, chyba detekce filtru Nashville, že přijímá i 29,52 % snímků s filtrem 1977, se neprojevuje, protože se tyto snímky nedostanou k podmínce filtru Nashville. Ze stejného důvodu se tak tedy nejdříve vyhodnotí podmínka filtru *Inkwell*, poté *Toaster*, *1997*, *Nashville* a pokud není žádný filtr detekován, tak se snímek vyhodnotí jako bez filtru, jak je vidět ve vizualizaci toku programu při detekci na obrázku 3.7.



Obrázek 3.7: Vizualizace toku programu při detekci filtru ve snímku analýzou histogramu.

Určování limitů histogramu snímků s filtrem probíhalo experimentálně, kde jsem si pomocí knihovny *Matplotlib* vykreslil histogram s maximálními hodnotami z 40 000 snímků s obličejem z datasetu CelebA pro každý filr. Poté jsem nastavoval horní hranice v intervalech histogramu tak, aby podmínky splňovaly snímky z daného filtru, ale co nejméně snímků s ostatními filtry nebo bez filtru. Pro tuto metodu detekce je filtr *Inkwell* zvláštní případ, protože kromě jiných úprav převede snímek do grayscale (stupňů šedé), a tak mají histogramy všech tří barevných kanálů stejné hodnoty. Díky tomu je detekce histogramem na tento filtr přesná, ale na druhou stranu se můžou objevit falešné pozitivy ve formě černobarevných snímků a snímků ve stupních šedi, které tato metoda detekce bude vždy považovat za upravené filtrem *Inkwell*.

3.4.2 Detekce neuronovou sítí

Tato metoda detekce je implementována třídou *NeuralAnalyser*, která se nachází v souboru *neural_analyser.py*. Třída obsahuje konstruktor pro načtení modelu, a metodu instance (instance method) *detect_filter()*. Objekt musí být před použitím inicializován, aby se načetl model z disku do paměti. Tato operace je relativně paměťově a časově náročná (model má 0,5 GB), a stačí ji provést jen jednou, proto je načítání modelu na požádání, při inicializaci objektu, která probíhá na žádost od uživatele v uživatelském rozhraní. Model ve formátu H5 se tak z disku načítá v konstruktoru, pomocí metody *load_model()* knihovny Keras ze souboru *model.h5* v podsložce *resources*.

Třída obsahuje metodu pro detekci filtrů na snímku, *detect_filter()*, která musí být zavolána na inicializovaném objektu, aby byl načtený model. Model akceptuje snímky velikosti 178×218, proto metoda akceptuje jen tuto velikost. Proto je v hlavní smyčce programu, před zavoláním metody na detekci, každý obrázek převeden na velikost 178×218 metodou *resize()* z knihovny Pillow. Metoda detekce snímek přidá do prázdného pole (neuronová síť bere jako vstup pouze pole), a nad modelem sítě se zavolá metoda *predict()* se snímkem jako parametrem. Ta vrací diskrétní rozdělení pravděpodobnosti přes cílové kategorie (tzn. přes vybrané filtry a kategorii bez filtru). Toto rozložení vrací i samotná metoda detekce jako její výsledek. Pro vybrání kategorie s nejvyšší pravděpodobností je použita po zavolání metoda *argmax()* z knihovny Numpy, která vrátí index položky s nejvyšší hodnotou.

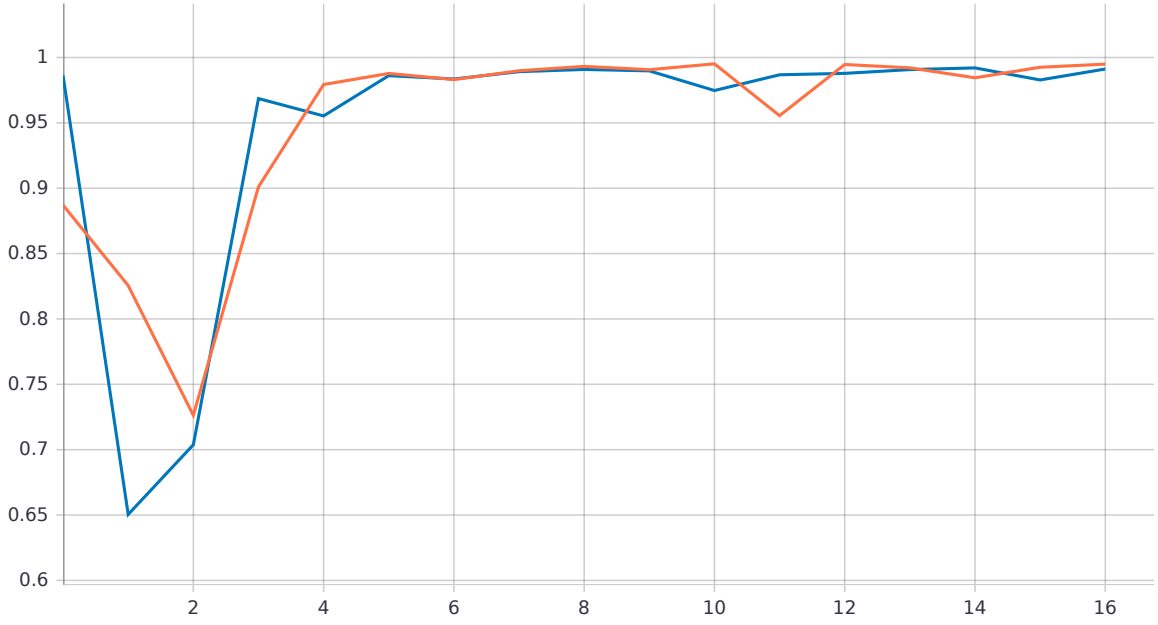
Finálního modelu bylo dosaženo inkrementálními změnami a experimentováním na platformě Google Colaborator (krátce Google Colab). Model byl trénován na datasetu CelebA (více v sekci 3.1). Z něj bylo vybráno 12 000 snímků (kvůli omezení velikosti paměti při trénování). Od každého snímku byla vytvořeny verze s detekovatelnými filtry, a jedna původní verze bez filtru. z těchto snímků pak byly vytvořeny n-tice (tuple) spolu s anotací označující použitý filtr (nebo jeho nepoužití). Celkově se tak model trénoval na 59 000 snímcích (12 000 snímků pro každou výstupní kategorii, mínus 1 000 snímků pro testování modelu). Pole těchto n-tic bylo poté pseudo-náhodně promícháno metodou *shuffle()* z Python modulu random[12], aby se předešlo přetrénování (overfittingu) sítě. Poté byla data rozdělena zvlášť na snímky a anotace. Anotace, obsahující číslo značící index příslušné kategorie snímku, byly převedeny na binární matici tříd, kvůli použití kategorické křížové entropie jako ztrátové funkce. K tomu byla použita pomocná metoda *to_categorical()* frameworku Keras. Ta převede vektor čísel na matici s počtem sloupců odpovídající počtu výsledných kategorií, a počtem řádků odpovídající délce původního vektoru čísel. Na každém řádku se tak vyskytují nuly a jedna jednička na pozici označující příslušnou kategorii. Řádky tak obsahují kód *1 z n* (anglicky one-hot encoding). Pro trénování se poté použije 70 % dat, zbylých

[12] random.shuffle() - https://docs.python.org/3/library/random.html#random.shuffle

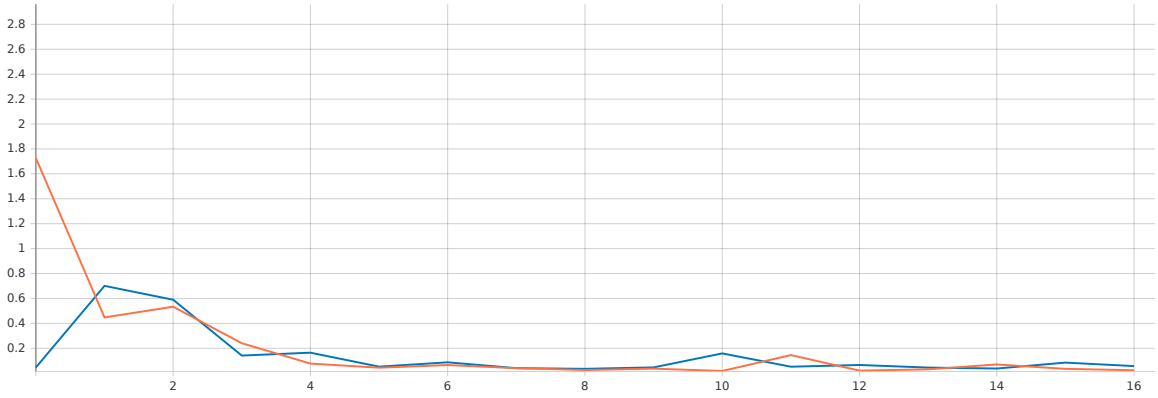
30 % je použito na validaci (nastavením parametru *validation_split* metody *fit()* na 0,3) sítě při a po trénování.

Konečná architektura sítě byla vyvinuta z modelu popsaného v sekci o experimentování 3.2.5. Z různých parametrů neuronové sítě, jako velikost dávky (batch size), počet plně propojených vrstev, velikost plně propojených vrstev a aktivační funkce, byla vytvořena pole. Poté se trénovaly modely s kombinacemi těchto parametrů, a podle výsledků se určovaly další parametry, pro maximalizaci validační přesnosti a minimalizaci validační chyby. Největšího zlepšení dosáhlo přidání konvolučních vrstev, kdy se úspěšnost detekce zvýšila o 10 %.

Dospělo se tak k modelu konvoluční sítě s třemi páry konvolučních a pooling vrstev, následovanými pěti skrytými vrstvami. Konvoluční sítě slouží k extrahování příznaků ze snímků, a každá je následovaná max-pooling vrstvou, která slouží ke zmenšení feature map a zjednodušení obrázku. Ty jsou následovány *Flatten()* vrstvou, která převádí výstup poslední pooling vrstvy do jedno-dimenzionálního pole, které je poté předáváno na vstup následující plně propojené vrstvě neuronové sítě. Těchto vrstev je zde 5, a slouží k zachycení důležitých prvků ve snímku. Velikosti (počty neuronů) těchto vrstev jsou 1024, 512, 341, 256, a 204, a používají aktivační funkce ReLU (popsána v sekci neuronové sítě, 2.3.2). Na výstupu je poslední plně propojená vrstva, o velikosti odpovídající množství výstupních kategorií, s aktivační funkcí softmax. Model byl trénován 17 epoch, po kterých dosáhl maximální přesnosti, s velikostí dávky 64 filtrovaných snímků. Vývoj přesnosti a výsledku ztrátové funkce v jednotlivých epochách trénování neuronové sítě lze vidět na obrázcích 3.8 (přesnost) a 3.9 (ztrátová funkce).



Obrázek 3.8: **Graf přesnosti při trénování.** Oranžové hodnoty jsou trénovací, modré jsou validační. Na ose X jsou vyznačeny epochy. Na ose Y je přesnost sítě podle epochy trénování.



Obrázek 3.9: **Graf výstupů ztrátové funkce při trénování.** Oranžové hodnoty jsou trénovací, modré jsou validační. Na ose X jsou vyznačeny epochy. Na ose Y výstupní hodnota ztrátové funkce.

3.4.3 Uživatelské rozhraní

Uživatelské rozhraní je implementováno pomocí wrapperu na framework *Tkinter*[13], *PySimpleGUI*[14]. Nejdříve se specifikuje rozložení (layout) okna pomocí 2D pole objektů. Toto pole obsahuje několik polí, každé obsahuje jeden řádek, ve kterém jsou komponenty *PySimpleGUI*, jako třeba *Text*, *Button* nebo *Image*, zobrazeny vedle sebe. Při definici rozhraní okna jsou komponenty inicializovány počátečními hodnotami, jako třeba jaký text obsahuje textové pole, nebo text zobrazený na tlačítku. V tomto programu je uživatelské rozhraní rozděleno do sloupců, které mají každý vlastní rozložení, oddělených vertikálním oddělovačem. Levý sloupec obsahuje komponenty pro nahrání snímku z místního disku do programu pro detekci, zobrazení celého názvu vybraného snímku, zobrazení vybraného snímku, a tlačítka pro metody detekce filtrů a načtení modelu neuronové sítě. Levá část tak obsahuje funkční část programu, kterou uživatel aktivně ovládá, zatímco pravá část obsahuje statický příklad snímků s vybranými filtry, které program detekuje. Ukázku uživatelského rozhraní lze vidět v obrázku 3.10.

Reakce rozhraní na akce uživatele jsou implementovány v hlavní smyčce programu. Program po inicializaci běží v nekonečné smyčce a čeká na akce uživatele. V každém běhu smyčky se čte z objektu okna název poslední události a její hodnotu. Poté je podle události provedena akce, jako vybrání obrázku, nebo detekce filtru.

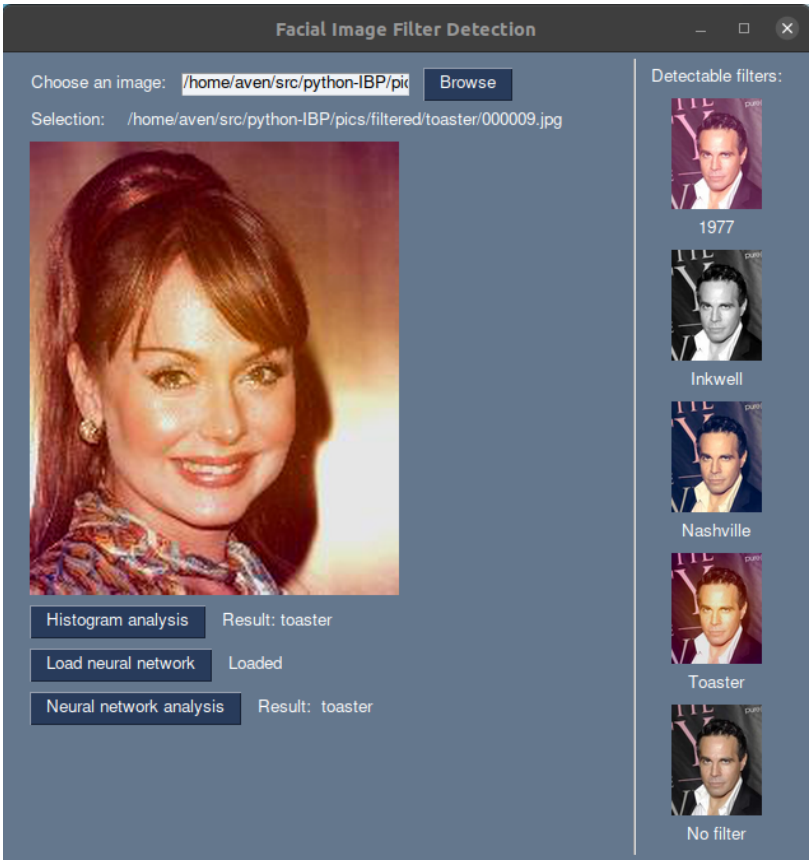
3.5 Úspěšnost detekce

Tato kapitola popisuje proces a výsledky experimentů s detekcí filtrů. Zaměřují se na úspěšnost detekce.

3.5.1 Experimenty s detekcí analýzou histogramu

K experimentu s detekcí retušovacích filtrů metodou analýzy histogramu byl použit skript *histogram_analyser_test.py*. Bere snímky ze složky specifikované proměnnou *PATH*, kde názvy snímků jsou šesticiferná čísla začínající od jedničky, ve formátu JPG. Načítání snímků

[13]Tkinker – https://docs.python.org/3/library/tkinter.html#module-tkinter
[14]PySimpleGUI – https://github.com/PySimpleGUI/PySimpleGUI



Obrázek 3.10: **Příklad uživatelského rozhraní** při detekci filtru na snímku oběma metodami po vyhodnocení výsledků, na operačním systému Ubuntu 20.04 s prostředím pracovní plochy GNOME 3.36.

je takto navržené protože jsem experimentoval na datasetu CelebA, který má tuto strukturu. Skript poté projde všechny detekovatelné filtry (specifikované v obrázku 3.4). Napřed načte obrázek z disku, aplikuje na něj daný filtr, a provede detekci. U každého filtru počítá počet korektních detekcí, který je poté převeden na procenta z celkového počtu snímků a uložen do pole s úspěšnostmi všech filtrů, které je na konci vytisknuto na standardní výstup. V rámci tohoto experimentu jsem používal finální verzi detekce analýzou histogramu, kde jsem dosáhl následujících výsledků (v tabulce 3.2). Celková přesnost této metody detekce retušovacích filtrů aplikovaných na dataset CelebA je tedy 94,44 %.

Filtr:	Nashville	1977	Toaster	Inkwell	Bez filtru
Přesnost:	96,25 %	89,24 %	100 %	100 %	86,70 %

Tabulka 3.2: **Přesnost určování filtru pomocí analýzy histogramu.** Procenta určují počet snímků z části datasetu CelebA o 100 000 snímcích, na kterých byl detekován správný filtr metodou analýzy histogramu pro danou kategorii výsledku (osa X).

3.5.2 Experimenty s detekcí neuronovou sítí

Experiment zaměřený na přesnost detekce neuronovou sítí se nachází ve skriptu *neural_analyser_test.py*, který podobně jako skript pro experiment detekce analýzou histogramu testuje každou kategorii výstupu zvlášť. Stejně jako předchozí skript, nejdříve načte snímek, aplikuje na něj filtr, a poté zavolá metodu detekce neuronovou sítí a porovná, zda se výsledek rovná aplikovanému filtru. Pokud ano, započítá jej do výsledné statistiky experimentu. Výsledek tohoto experimentu lze vidět v následující tabulce 3.3:

Filtr:	Nashville	1977	Toaster	Inkwell	Bez filtru
Přesnost:	99,80 %	99,50 %	99,70 %	99,90 %	99,00 %

Tabulka 3.3: **Přesnost určování filtru pomocí neuronové sítě.** Procenta určují počet snímků z části datasetu CelebA o 10 000 snímcích, na kterých byl detekován správný filtr neuronovou sítí pro danou kategorii výsledku (osa X).

Celková přesnost této metody detekce, zjištěná tímto postupem, nad datasetem CelebA, je tedy 99,55 %. Toto odpovídá i informacím o přesnosti získaných při návrhu a vývoji tohoto modelu, kde validační přesnost v posledním kroku byla 99,12 %, a přesnost získaná při ohodnocení (metoda *evaluate()* knihovny Keras) modelu nad 1 000 snímky (neobsaženými v trénovacím vzorku datasetu) byla 99,10 %.

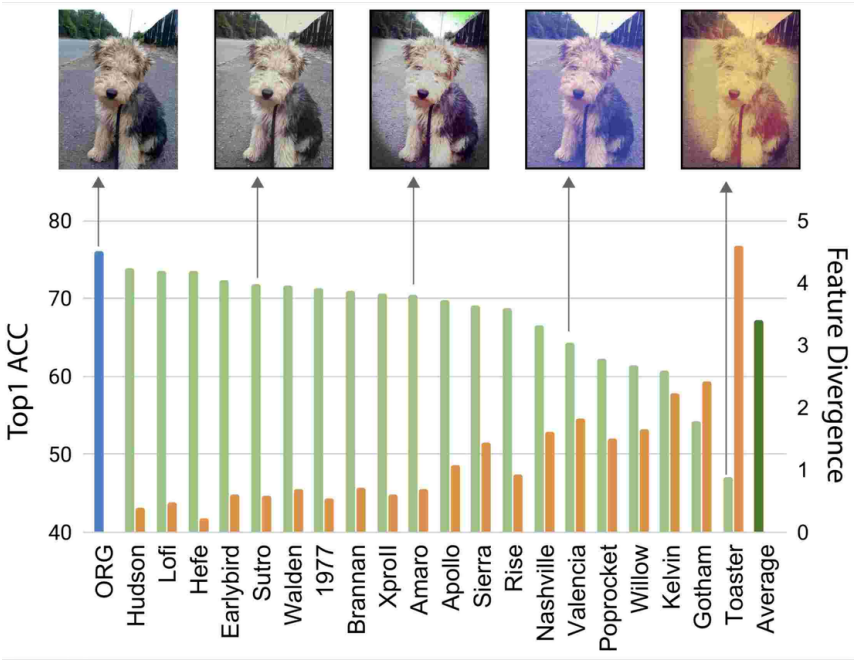
Kapitola 4

Vliv obrazových filtrů na identifikaci osob

Tato kapitola se věnuje zkoumání vlivu filtrů na identifikaci osob, popisuje relevatní práce věnující se této problematice, provedený experiment vlivu filtrů na identifikaci osob, a jeho výsledky.

I když to není často řešené téma, vlivu filtrů na klasifikaci nebo identifikaci se v posledních několika letech věnovalo několik studií. V rámci studie *Recognizing Instagram Filtered Images with Feature De-stylization* (Rozpoznávání snímků s instagramovými filtry pomocí de-stylizace příznaků) její autoři vytvořili modifikovanou verzi datasetu *ImageNet*, používaného pro trénování sítí k rozpoznávání objektů ze snímku, kde na každý snímek aplikovali 20 různých instagramových filtrů. Tento dataset nazvali *ImageNet-Instagram*. Poté na tomto novém datasetu zkoušeli několik nejpřesnějších konvolučních neuronových sítí k rozpoznávání obrazu. Výsledky poukazují na to, že změny způsobené filtry vedou k velkému rozdílu příznaků oproti originálním snímkům, což vede k poklesu přesnosti. Průměrná *top-1* přesnost klesla ze 76,13 % na 67,12 % při použití datasetu *ImageNet-Instagram* místo původního datasetu *ImageNet*. Přesnost top-1 je úspěšnost při klasifikaci snímku do kategorie s nejvyšší odhadnutou pravděpodobností. Dále se používá řesnost top-5, která značí úspěšnost při klasifikaci snímku pokud je správná kategorie mezi pěti s nejvyšší odhadnutou pravděpodobností. Některé filtry, jako například Gotham nebo Toaster, aplikují výrazné změna na obrázek, a způsobují významný pokles přesnosti (21,13 % pro Gotham a 29,03 % pro Toaster) [46]. To lze vidět na obrázku 4.1.

Ve studii nazvané *Provenance analysis for instagram photos* (Analýza původu instagramových fotografií) je zkoumán vliv filtrů na referenční šum fotoaparátu (SNP – *sensor pattern noise*). SNP je šum vyskytující se v každé fotografii, podle kterého je možné identifikovat zařízení, které pořídilo snímek, protože rozdílné senzory fotoaparátů způsobují rozdílný šum. Bylo zjištěno, že aplikace některých filtrů má na šum ve fotografii výrazný dopad, zatímco jiné filtry mají dopad minimální a ze šumu jdou získat informace i původu snímku i po jejich aplikaci. Filtry tak dělí na dvě skupiny, podle jejich dopadu. První skupina obsahuje filtry s výrazným dopadem na šum ve snímku, a obsahuje filtry jako např. Toaster nebo nebo Hudson. Zde je tak vidět, že různé filtry mají lišící se dopad na rozdílné vlastnosti snímku. Filtr Toaster výrazně ovlivnil přesnost v předchozí studii, ale naopak filtr Hudson měl dopad nejmenší. Druhou skupinou jsou filtry s minimálním dopadem na šum ve snímku, obsahuje filtry jako např. Nashville, 1977, a Walden [33].



Obrázek 4.1: Ukázka vlivu filtrů na přesnost DCNN *ResNet50* předtrénované na datasetu ImageNet (v zelené) a odchylka příznaků (v oranžové). Na ose X jsou různé typy filtrů. Na levé ose Y je přesnost, a na pravé ose Y odchylka příznaků. Převzato z [46].

Studie *Filter-Invariant Image Classification on Social Media Photos* (Klasifikaci obrazu na snímcích ze sociálních medií neovlivněná filtry) představuje způsob rozpoznávání obrazu, který ignoruje změny ve snímku způsobené filtry. Její součástí je i experiment nad třemi moderními konvolučními neuronovými sítěmi k rozpoznávání obrazu, kde aplikace filtru Valencia na validační dataset *ILSVRC2012 Val* má výrazný dopad na jeho přesnost [8]. To lze vidět na tabulce 4.1.

ILSVRC2012 Val	Bez filtru	S filtrem
AlexNet [27]:	56,87 % / 80,30 %	30,14 % / 53,70%
VGG-Net [36]:	68,27 % / 92,50 %	50,95 % / 74,58 %
GoogLeNet [37]:	68,70 % / 88,90 %	47,85 % / 73,02 %

Tabulka 4.1: **Dopad filtrů na přesnost rozpoznávání obrazu.** Přesnost *top-1/top-5* různých moderních řešení (osa X) rozpoznávání snímku na datasetu *ILSVRC2012 Val* s filtrem Valencia. Převzato z [8].

4.1 Experiment

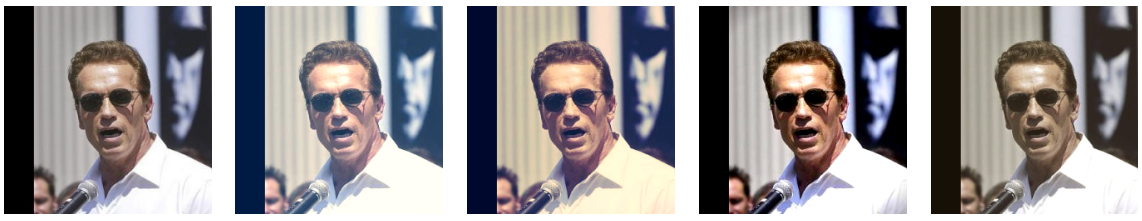
K testování vlivu filtrů na identifikaci osob jsem použil Python knihovnu *face_recognition*[1]. Tato knihovna slouží k rozpoznávání osob podle snímků obličejů, k čemuž využívá C++ knihovny pro strojové učení *dlib*[2]. Dlib je otevřená a volně dostupná (*open-source*) sada

[1] face_recognition – https://github.com/ageitgey/face_recognition
[2] dlib – http://dlib.net/

moderních nástrojů k vývoji aplikací strojového učení. Používá se v průmyslu i na akademické půdě, v široké škále domén, včetně robotiky, vestavěných zařízení, mobilních telefonů a velkých vysoko-výkonových výpočetních prostředí.

Metody identifikace z knihovny *face_recognition* tak dosahují přesnosti 99,38 % při identifikaci osob na datasetu LFW díky použití dlib třídy *face_recognition_model_v1*[3], která převádí snímky obličeje na 128-dimenzionální vektory. Mezi vektory stejné osoby by měla být malá vzdálenost (alespoň menší než je hranice pro identifikaci (*treshold*)). Snímky rozdílných osob by zase měly mezi sebou mít větší vzdálenost (větší než je hranice pro identifikaci).

Jazyk Python jsem vybral abych mohl využít výpočtů pomocí grafických karet na službě Google Colaboratory, což výrazně zrychlilo testování. To probíhalo nad snímky z datasetu LFW. Z datasetu jsem vybral několik osob, které měly v datasetu alespoň 15 rozdílných snímků, protože při vyšším počtu identit, proti kterým se poté vyhodnocované snímky porovnávají, klesala při experimentech přesnost modelu. Takto jsem mohl testovat na větším počtu snímků s menším počtem unikátních identit. Vektorovou reprezentaci obličeje z prvního snímku každé z vybraných osob jsem uložil do samostatného pole, které poté sloužilo jako databáze známých identit při rozpoznávání tváří na snímcích. Na ostatní snímky osoby jsem buď aplikoval vybraný retušovací filtr (opět jsem použil retušovací filtry z knihovny Pilgram), nebo jej nechal bez úprav, a také je převedl do 128-dimenzionální vektorové reprezentace. Filtry vybrané pro tento experiment aplikované na snímku z použitého datasetu LFW jsou vidět na obrázku 4.2. Poté jsem všechny tyto reprezentace tváří ze snímků co se nevyskytují v databázi známých identit postupně proti ní porovnával pomocí funkce k rozpoznávání osob *compare_faces()*, a počítal *euklidovské vzdálenosti* snímku k jednotlivým identitám metodou *face_distance()*.



(a) Bez filtru (b) Walden (c) Nashville (d) Lofi (e) Slumber

(f) Willow (g) Toaster (h) Inkwell (i) Hudson (j) 1977

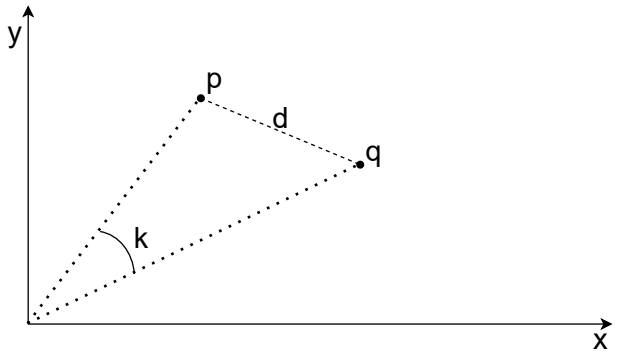
Obrázek 4.2: **Ukázka filtrů použitých v tomto experimentu.** Seřazeno podle přesnosti identifikace dosáhnuté s daným filtrem (vlevo nahoře nejvyšší přesnost, nejmenší přesnost s filtrem vpravo dole), která je v tabulce 4.2. Filtry byly aplikované na snímek převzatý z datasetu LFW [23].

[3]Třída pro identifikaci – http://dlib.net/python/index.html#dlib.face_recognition_model_v1

Euklidovská vzdálenost je délka úsečky mezi dvěma body v euklidovském prostoru. V případě tohoto experimentu je to prostor 128-dimenzionální. Při výpočtu této vzdálenost se může vyjít z Pythagorovy věty. V rovnici 4.1 je tak popsán výpočet euklidovské vzdálenosti d mezi dvěma body p a q ve 128-dimenzionálním prostoru:

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_{128} - q_{128})^2} \tag{4.1}$$

kde p_n a q_n jsou souřadnice bodů p a q v n-té dimenzi [38]. V některých řešení identifikace osob podle snímku obličeje se používá i kosinová vzdálenost (nebo kosinová podobnost), jako například ztrátová funkce CosFace [43]. Kosinová vzdálenost je míra podobnosti mezi dvěma nenulovými vektory, a je rovna kosinu úhlu mezi nimi. Rozdíl mezi těmito dvěmi způsoby výpočtu vzdálenosti lze vidět na obrázku 4.3. Pro výpočet vzdálenosti v tomto řešení je ale použita euklidovská vzdálenost, které je počítána pomocí metody *face_distance()*, která bere na vstup pole referenčních reprezentací tváří (z databáze známých identit) a testovanou reprezentaci tváře, a vrací pole (euklidovských) vzdáleností mezi reprezentací testované tváře a každou z referenčních reprezentací z databáze známých identit.



Obrázek 4.3: Vizualizace rozdílu mezi kosinovou podobností k a euklidovskou vzdáleností d dvou bodů p a q.

Rozhodnutí, zda tvář ve vyhodnocovaném snímku patří jedné z osob v databázi identit, oproti které je snímek porovnáván, je určeno hranicí vzdálenosti. Pokud vzdálenost dvou vektorových reprezentací tváří nepřekročí tuto hranici (je menší než), dají se považovat za reprezentace stejné osoby. Pokud jsou ovšem vzdálenosti větší než stanovená hranice, jedná se o dvě rozdílné identity. Při tomto experimentu byla použita implicitně nastavená hranice euklidovské vzdálenosti $0,6$, která dosahovala dobrých výsledků na snímcích datasetu LFW bez aplikovaného filtru.

Při tomto experimentu bylo použito snímků 24 rozdílných identit z datasetu LFW, v unikátních 1025 snímcích. Z těchto snímků bylo celkem 24 (pro každou identitu jeden) použito pro vytvoření databáze identit. Na zbylých 1001 snímků byl aplikován daný filtr, a byly porovnáváno s uloženými identitami. Výsledné hodnoty experimentu, průměrné hodnoty vzdálenosti vektorových reprezentací rozdílných a stejných identit (osob), přesnost klasifikace, a i úspěšnost detekce obličeje ve snímku, lze vidět v tabulce 4.2.

Aplikovaný filtr	Přesnost identifikace osob	Přesnost detekce obličeje	Prům. vzdál. stejných osob	Prům. vzdál. rozdílných osob
Žádný	97,60 %	100,00 %	0,43	0.83
Walden	97,30 %	100,00 %	0,44	0,82
Nashville	62,08 %	74,83 %	0,60.	0,81
Lofi	48,13 %	96,30 %	0,66	0,82
Slumber	48,13 %	69,03 %	0,65	0,81
Willow	22,18 %	75,22 %	0,74	0,81
Toaster	20,05 %	78,22 %	0,75	0,80
Inkwell	17,88 %	90,05 %	0,78	0,80
Hudson	17,13 %	89,81 %	0,79	0,81
1977	17,13 %	89,81 %	0,79	0,81

Tabulka 4.2: **Vliv filtrů na identifikaci osob pomocí snímku obličeje.** Seřazeno sestupně podle přesnosti identifikace osob. Lze vidět dopad aplikace daných filtrů (osa X) na různé metriky a přesnosti (osa Y). *Přesnost identifikace osob* je při porovnání snímku s databází známých identit. *Přesnost detekce obličeje* je detekce pomocí nástrojů knihovny dlib, jakožto součást převedení obličeje na jeho vektorovou reprezentaci. *Průměrná vzdálenost stejných osob* (stejné identity) a *Průměrná vzdálenost rozdílných osob* (rozdílných identit) jsou vyjádřeny jako euklidovská vzdálenost, která je používána pro následovnou klasifikaci (s hranicí identifikace 0,6).

Z výsledků experimentu tak lze vidět, že aplikace některých filtrů má výrazný dopad na identifikaci osob, i při použití moderního řešení. Filtry obecně zvětšují vzdálenost mezi tvářemi stejné identity, a tak i zhoršují přesnost identifikace. Tento výsledek odpovídá výsledkům předchozích zmíněných studií, že filtry mají dopad na standardní řešení problémů rozpoznávání snímků, u kterého nebylo s podobnými úpravami počítáno.

Řešením těchto problémů může být buď upravit model tak, aby nebyl ovlivněn změnami od filtrů (jako je ukázáno v jedné ze zmíněných studií [8]), odstranit ze snímků informace o stylu, které do něj filtry vnáší (taktéž zmíněno v předchozí studii [46]), nebo dopředu detekovat snímky s aplikovaným filtrem (jak je řešeno v této práci).

Jde vidět, že některé filtry mění takové vlastnosti snímku, že klesá přesnost identifikace osob ze snímku obličeje (např. filtr Hudson) nebo i klesá přesnost detekce obličeje ve snímku (např. filtr Slumber), zatímco jiné filtry mají dopad minimální (např. filtr Walden). Toto odpovídá i ostatním studiím, kdy filtr Toaster negativně ovlivňoval přesnost ve studii šumu [33] i rozpoznávání obrazu [46], zatímco filtr Walden měl v obou studiích malý vliv.

Hodnoty z tohoto experimentu byly získány skriptem *FilterImpactOnRecignition.ipynb* v prostředí Google Colaboratory.

Kapitola 5

Závěr

Cílem této práce bylo najít spolehlivý způsob detekce použití retušovacích filtrů na snímcích s obličejem. Tento záměr byl splněn, podařilo se najít dvě metody detekce, obě s vysokou přesností.

Práce také obsahuje vysvětlení principu filtrů v digitálních snímcích, neuronových sítí, a sumarizaci programů pro identifikaci osob podle snímku obličeje spolu s několika příklady těchto programů. Navrhl jsem a vyzkoušel několik přístupů k detekci filtrů, z nichž dva se ukázaly jako použitelné v praxi. První byla metoda detekce pomocí analýzy histogramu, kde filtr omezoval maximální hodnoty v daných intervalech histogramu. U této metody se podařilo dosáhnout přesnosti 94,44 %. Druhou metodou byla detekce pomoci konvoluční neuronové sítě s přesností 99,55 %, která byla trénována na datasetu CelebA. Tyto metody detekce retušovacího filtru na snímku s obličejem jsem implementoval v jazyce Python, spolu s jednoduchým uživatelským rozhraním. Program je možné spustit na platformách Windows i Linux. Byly provedeny testy úspěšnosti metod detekce filtrů, kde metody dosáhly dobrých výsledků (94,44 % a 99,55 %).

Na závěr byl zkoumán vliv filtrů na identifikaci osob podle snímků obličeje. Nejdříve jsem popsal několik podobných prací zkoumajících vliv filtrů na různé operace detekce snímků. Poté jsem provedl vlastní test vlivu filtrů na moderním řešení identifikace podle snímku obličeje. Bez filtru tato metoda identifikace dosahovala přesnosti 97,60 %. Po aplikaci filtrů přesnost klesla, u některých filtrů méně (filtr Walden, přesnost 97,30 %), u některých zase více (filtr Hudson, přesnost 17,13 %). Lze tak říci, že filtry mají jednoznačný vliv na úspěšnost identifikace osob podle snímku obličeje, a je tak vhodné to buď vzít v potaz při vyvíjení nového řešení, nebo při předávání snímků k identifikaci.

V rámci této práce jsem se dozvěděl o tom, jak fungují filtry se kterými se setkávám téměř denně. Naučil jsem se pracovat s neuronovými sítěmi, naučil se jak fungují konvoluční neuronové sítě, a proč jsou tak úspěšné. Získal jsem přehled o způsobech identifikace osob podle snímku obličeje a poznal jsem moderní řešení tohoto problému.

Výsledný program detekuje čtyři různé filtry podobné filtrům používaným na platformě Instagram, a dokáže rozpoznat i absenci filtru. V budoucnu by bylo možné jej rozšířit pro detekci více filtrů, nebo zkusit, zda by šlo natrénovat neuronovou síť na detekci filtrů obecně, bez zaměření se na několik vybraných filtrů.

Literatura

[1] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A. et al. State-of-the-art in artificial neural network applications: A survey. *Heliyon*. Elsevier Ltd. 2018, sv. 4, č. 11. ISSN 2405-8440.

[2] Amos, B., Ludwiczuk, B. a Satyanarayanan, M. *OpenFace: A general-purpose face recognition library with mobile applications*. CMU-CS-16-118, CMU School of Computer Science, 2016.

[3] Bharati, A., Singh, R., Vatsa, M. a Bowyer, K. W. Detecting Facial Retouching Using Supervised Deep Learning. *IEEE Transactions on Information Forensics and Security*. IEEE. 2016, sv. 11, č. 9, s. 1903–1913. ISSN 1556-6013.

[4] Burger, W. *Principles of digital image processing : fundamental techniques*. London: Springer, 2009. Undergraduate topics in computer science, 7592. ISBN 978-1-84800-190-9.

[5] Burger, W. a Burge, M. J. *Principles of Digital Image Processing: Core Algorithms*. London: Springer London, 2009. Undergraduate Topics in Computer Science. ISBN 978-1-84800-194-7.

[6] Chen, D., Cao, X., Wang, L., Wen, F. a Sun, J. Bayesian face revisited: A joint formulation. In:. 2012, sv. 7574, č. 3, s. 566–579. ISBN 9783642337116.

[7] Chen, N. *Analog vs Digital vs IP Cameras* [online]. CCTV Singapore, listopad 2020 [cit. 2020-12-12]. Dostupné z: https://www.cctvsg.net/analog-vs-digital-vs-ip-cameras/.

[8] Chen, Y.-H., Chao, T.-H., Bai, S.-Y., Lin, Y.-L., Chen, W.-C. et al. Filter-Invariant Image Classification on Social Media Photos. In:. říjen 2015, s. 855–858. DOI: 10.1145/2733373.2806348.

[9] Clair, K. *The Secret Lives of Colour*. John Murray (Publishers), 2016. ISBN 9781473630819. Dostupné z: https://books.google.cz/books?id=v3ZujwEACAAJ.

[10] Commission Internationale de L'Eclairage. *CIE S 017/E:2011 ILV: International lighting vocabulary*. 1. vyd. Commission Internationale de L'Eclairage, 2011.

[11] Cotnoir, L. *Hue, Value, Saturation* [online]. Leigh Cotnoir [cit. 2020-12-18]. Dostupné z: http://learn.leighcotnoir.com/artspeak/elements-color/hue-value-saturation/.

[12] Damelin, S. B. a Miller, W. *The mathematics of signal processing*. Cambridge University Press, 2012.

[13] Deng, J., Guo, J., Xue, N. a Zafeiriou, S. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. *ArXiv.org*. Ithaca: Cornell University Library, arXiv.org. 2019. ISSN 2331-8422. Dostupné z: http://search.proquest.com/docview/2071279519/.

[14] Ding, F., Shi, Y., Zhu, G. a Shi, Y.-Q. Smoothing identification for digital image forensics. *Multimedia Tools and Applications*. New York: Springer US. 2019, sv. 78, č. 7, s. 8225–8245. ISSN 1380-7501.

[15] Dornberger, W. *V-2*. New York: Bantam Books, 1979. ISBN 978-0553126600.

[16] Forsyth, D. *Applied Machine Learning*. Cham: Springer International Publishing, 2019. ISBN 9783030181130.

[17] Furht, B., Akar, E. a Andrews, W. A. *Digital Image Processing: Practical Approach*. Cham: Springer International Publishing, 2018. SpringerBriefs in Computer Science. ISBN 9783319966335.

[18] Gillespie, S. *The early American daguerreotype : cross-currents in art and technology*. Cambridge, Massachusetts Washington, DC: MIT Press The Lemelson Center, Smithsonian Institution, 2016. ISBN 9780262034104.

[19] Gäbler, M. a AzaToth. *Gradual JPEG artifacts example, with decreasing quality from right to left*. 2011 [cit. 2020-29-01]. Creative Commons Attribution 3.0 Unported, https://creativecommons.org/licenses/by/3.0/. Dostupné z: https://en.wikipedia.org/wiki/File:Felis_silvestris_silvestris_small_gradual_decrease_of_quality.png.

[20] Hirsch, R. a Green, C. *Seizing the Light: A History of Photography*. McGraw-Hill, 2000. ISBN 9780697143617. Dostupné z: https://books.google.cz/books?id=vftTAAAAMAAJ.

[21] Horvath, M. *HSL HSV cylinder color solid comparison*. Jan 2008 [cit. 2021-11-03]. Creative Commons Attribution 3.0 Unported, https://creativecommons.org/licenses/by/3.0/. Dostupné z: https://commons.wikimedia.org/wiki/File:HSL_HSV_cylinder_color_solid_comparison.png.

[22] Horvath, M. *RGB color solid cube*. Jan 2008 [cit. 2021-11-03]. Creative Commons Attribution 3.0 Unported, https://creativecommons.org/licenses/by/3.0/. Dostupné z: https://commons.wikimedia.org/wiki/File:RGB_color_solid_cube.png.

[23] Huang, G. B., Mattar, M., Berg, T. a Learned Miller, E. *Labeled faces in the wild: A database for studying face recognition in unconstrained environments*. 2007.

[24] Immerkær, J. Fast Noise Variance Estimation. *Computer Vision and Image Understanding*. Elsevier Inc. 1996, sv. 64, č. 2, s. 300–302. ISSN 1077-3142.

[25] Kee, E. a Farid, H. Perceptual metric for photo retouching. In:. National Academy of Sciences, 2011, sv. 108, č. 50, s. 19907–19912. ISSN 0027-8424.

[26] Kingma, D. a Ba, J. Adam: A Method for Stochastic Optimization. *ArXiv.org*. Ithaca: Cornell University Library, arXiv.org. 2017. ISSN 2331-8422. Dostupné z: http://search.proquest.com/docview/2075396516/.

[27] Krizhevsky, A., Sutskever, I. a Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. a Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, sv. 25. Dostupné z: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[28] Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B. et al. SphereFace: Deep Hypersphere Embedding for Face Recognition. 2017.

[29] Liu, Z., Luo, P., Wang, X. a Tang, X. Deep Learning Face Attributes in the Wild. In: *Proceedings of International Conference on Computer Vision (ICCV)*. December 2015.

[30] Maier, A., Syben, C., Lasser, T. a Riess, C. A Gentle Introduction to Deep Learning in Medical Image Processing. *ArXiv.org*. Ithaca: Cornell University Library, arXiv.org. 2018. ISSN 2331-8422. Dostupné z: http://search.proquest.com/docview/2160088588/.

[31] Marček, D. *Supervizované a nesupervizované učení z dat: statistický a soft přístup*. Ostrava: Vysoká škola báňská - Technická univerzita, Ekonomická fakulta, 2016. 59–60 s. ISBN 978-80-248-3884-7.

[32] Poynton, C. *Color FAQ - Frequently Asked Questions Color*. Listopad 2006 [cit. 2020-12-02]. Dostupné z: http://poynton.ca/ColorFAQ.html.

[33] Quan, Y., Lin, X. a Li, C.-T. Provenance analysis for instagram photos. In:. Springer Verlag, 2019, sv. 996, s. 372–383. ISBN 9789811366604.

[34] Schroff, F., Kalenichenko, D. a Philbin, J. FaceNet: A unified embedding for face recognition and clustering. In: *IEEE Conference on Computer Vision and Pattern Recognition. Proceedings*. 2015, 07-12-, s. 815–823. ISBN 9781467369640. Dostupné z: http://search.proquest.com/docview/1770328968/.

[35] Scott, D. W. Histogram. *Wiley Interdisciplinary Reviews: Computational Statistics*. Hoboken, USA: John Wiley & Sons, Inc. 2010, sv. 2, č. 1, s. 44–48. ISSN 1939-5108.

[36] Simonyan, K. a Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2014.

[37] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. et al. Going Deeper with Convolutions. 2014.

[38] Tabak, J. *Geometry: The Language of Space and Form (History of Mathematics)*. Hardcover. Facts on File, květen 2004. 222 s. ISBN 978-0816049530.

[39] Taigman, Y., Yang, M., Ranzato, M. a Wolf, L. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In: *IEEE Conference on Computer Vision and Pattern Recognition. Proceedings*. 2014, s. 1701–1708. ISBN 9781479951178. Dostupné z: http://search.proquest.com/docview/1677924193/.

[40] Titterington, M. Neural networks. *Wiley Interdisciplinary Reviews: Computational Statistics*. Hoboken, USA: John Wiley & Sons, Inc. 2010, sv. 2, č. 1, s. 1–2. ISSN 1939-5108.

[41] Toub, S. .NET Matters: Sepia Tone, StringLogicalComparer, and More. *MSDN Magazine*. United Business Media LLC. 2005, sv. 20, č. 1. ISSN 1528-4859.

[42] Viola, P. a Jones, M. Robust Real-Time Face Detection. Boston: Kluwer Academic Publishers. 2004, sv. 57, č. 2, s. 137–154. ISSN 0920-5691.

[43] Wang, H., Wang, Y., Zhou, Z., Ji, X., Gong, D. et al. CosFace: Large Margin Cosine Loss for Deep Face Recognition. 2018.

[44] White, S. *Bitmap Storage* [online]. 2018 [cit. 2020-29-01]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/gdi/bitmap-storage.

[45] World Wide Web Consortium. *Filter Effects* [online]. Aug 2011 [cit. 2021-14-03]. Dostupné z: https://www.w3.org/TR/SVG11/filters.html#feColorMatrixValuesAttribute.

[46] Wu, Z., Wu, Z., Singh, B. a Davis, L. Recognizing Instagram Filtered Images with Feature De-stylization. *ArXiv.org*. Ithaca: Cornell University Library, arXiv.org. 2019. ISSN 2331-8422. Dostupné z: http://search.proquest.com/docview/2331699871/.

[47] Yan, J., Zhang, L., Wu, Y., Guo, P., Zhang, F. et al. Research on face recognition method based on deep learning in natural environment. In: *Proceedings - 2017 IEEE 8th International Conference on Awareness Science and Technology, iCAST 2017*. Institute of Electrical and Electronics Engineers Inc., 2017, 2018-, s. 501–506. ISBN 9781538629659.

Příloha A

Obsah přiloženého paměťového média

Přiložené paměťové médium má následující adresářovou strukturu:

- `xkrava02-Detekce-filtru.pdf` ..... dokument technické zprávy
- `src-latex/` ..... zdrojové kódy technické zprávy
- `src-python/` ..... zdrojové kódy projektu
  - `notebooks/` ..... složka s Jupyter notebooky
  - `resources/` ..... zdroje pro běh programu
    - `model.h5` ..... model neuronové sítě
  - `README.md` ..... popis spuštění programu, požadavků a zdrojových souborů