VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

APLIKACE PRO AUTOMATICKÉ VYHODNOCENÍ VĚROHODNOSTI GENEROVANÉHO SNÍMKU OBLIČEJE

APPLICATION FOR AUTOMATIC EVALUATION OF THE FIDELITY OF THE GENERATED FACIAL IMAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JIŘÍ ŠOTOLA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ GOLDMANN**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

156900

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Šotola Jiří**
Program:	Informační technologie
Název:	**Aplikace pro automatické vyhodnocení věrohodnosti generovaného snímku obličeje**
Kategorie:	Zpracování obrazu
Akademický rok:	2023/24

Zadání:

1. Seznamte se s problematikou generování syntetických snímků obličeje a zjistěte, jaké jsou dostupné generované datasety se snímky obličeje.
2. Sumarizujte informace o alespoň 3 algoritmech pro rozpoznávání osob podle obličeje.
3. Navrhněte řešení, které bude určovat věrohodnost generovaných snímků obličeje.
4. Navržené řešení implementujte v programovacím jazyce Python.
5. Aplikaci otestujte a vyhodnoťte, čím je věrohodnost generovaných snímků obličeje nejvíce ovlivňována.

Literatura:

- CHOI, Yunjey, et al. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018. p. 8789-8797.
- DENG, Jiankang, et al. Arcface: Additive angular margin loss for deep face recognition. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019. p. 4690-4699.

Při obhajobě semestrální části projektu je požadováno:
Body 1 a 2.

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:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	6.11.2023

Abstrakt

Tato práce se zaměřuje na návrh a implementaci aplikace pro ověření věrohodnosti syntetický generovaných snímků, která se kvůli rozlehlosti tohoto tématu směruje k ověření podobnosti obličejových znaků originálního snímku a snímku z něho vygenerovaného. Pro tuto aplikaci je vyvinut model *Gen_Verifier* založený na siamských nerunových sítí, u kterých používá ztrátovou funkci *contrastive loss*. Tento model byl trénován a testován na datové sadě *LFW*, kde se dostal až k přesnosti 91 %. Pro testování generovaných snímků je použit model *StarGAN*, který generoval snímky obličeje se změnou barvy vlasů, pohlaví a stáří. Výsledné testování na generovaných snímcích ukázal, že model *StarGAN* vytváří obličeje, které se s originálem na 87,53 % shodují.

Abstract

This work focuses on the design and implementation of an application for verifying the fidelity of a synthetically generated images, which, due to the vastness of this topic, is aimed at verifying the similarity of the facial features of the original image and the image generated from it. For this application, a *Gen_Verifier* model is developed based on Siamese networks, which uses the *contrastive loss*. This model was trained and tested on the *LFW* dataset, where it reached an accuracy of 91 %. The *StarGAN* model is used to test the generated images, which generated facial images with changes in hair color, gender and age. The resulting testing on the generated images showed that the *StarGAN* model produces faces that are similar in 87.53 % cases.

Klíčová slova

neuronové sítě, neuron, perceptron, siamská síť, rozpoznávání obličeje, GANs, StarGAN, RelGAN, ArcFace, MagFace

Keywords

neural networks, neuron, perceptron, siamese network, facial recognition, GANs, StarGAN, RelGAN, ArcFace, MagFace

Citace

ŠOTOLA, Jiří. *Aplikace pro automatické vyhodnocení věrohodnosti generovaného snímku obličeje*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Goldmann

Aplikace pro automatické vyhodnocení věrohodnosti generovaného snímku obličeje

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.

.....
Jiří Šotola
9. května 2024

Poděkování

Touto cestou bych chtěl poděkovat svému vedoucímu Ing. Tomáši Goldmannovi za poskytnutí cenných rad a za pomoc při psaní mé bakalářské práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Vytváření nových snímků s obličejem je komplikovaný proces, který by měl splňovat potřebná kritéria pro dosažení kvalitnějšího výstupu. Tyto kritéria jsou jednoduché problémy, které jsou z lidské perspektivy snadno vidět jako, že hlava je do určití míry oblá a nebo, že nos je mezi očima, ale pro počítač to je komplexní problém, který sám jednoduše nevyřeší. Z tohoto důvodu jsou navrhovány nové systém pro řešení těchto problémů.

Systémy pak kvůli tomu dělíme do moha skupin jako klasifikační nebo regresní s cílem snadnějšího vyřešení skupiny problémů, které můžou být například tvorba náhodného obličeje pro prezentaci nového vzhledu sociálních sítí, kde obličej nemá reálnou osobu a tudíž s ní mohou vytvořit zcela imaginární osobnost. Dalším příkladem je úprava obličeje hledané osoby se záměrem snadnější identifikace. Pro tento úkol je vytvořena kolekce snímků s různými styly vlasů, výrazů a i stářím, kde je minimálně jeden snímek přiblížený skutečné podobě, která umožní snadnější hledávání a dostihnutí. Tyto snímky mohou pomoci nejen policii, ale i u jiným modelům, které se na nich poté mohou učit.

V tomto případě je nutné ověřit podobnost obličejů s originální snímkem a vygenerovaným snímkem, protože je důležité nezměnit jeho obličejové znaky pro správnou identifikaci a protože modely na to často nejsou přímo kontrolované. Jejich kontrola poté spočívá v podobě dotazníků na skupinu lidí, které se snaží nejlépe ohodnotit celý s snímek a ne jeho přesnou podobu s originálem, která je někdy kritická.

Cílem této práce je vyvinutí aplikace v programovacím jazyce *Python* pro vyhodnocení věrohodnosti syntetických snímků i s reálnými. Aplikace je schopna ověřit podobnost identit na snímcích ve formě binárního výstupu, ve kterém je přenášena pravdivostní hodnota. Další její vlastností je skupinové zpracování, kde se pomocí struktury jmen a adresářů zpracuje široké množství dvojic s analýzou na výstupu. Výstup se skládá z výpisu do terminálu nebo jako extrakce do souboru.

Pro tvorbu takové aplikace jsou nutné informace a znalost, které jsou sepsány v následujících kapitolách. Prvním, čím se práce zabývá je generování syntetický snímků v kapitole 2 pro podrobnější pochopení a realizaci funkce modelů, které jsou v podobě generativních soupeřících sítí se základem v neuronových sítí. V této kapitole jsou také uvedeny důležité datové sady snímků v určitých formátech pro vývoj modelu. S další kapitolou 3 je uvedeno i prozkoumání různých modelů pro rozpoznávaní snímků s obličeji podle osob. Následuje kapitola Návrh a implementace 4, kde je popis vyvinutého modelu a jeho zpracování do konzolové aplikace společně s přípravou datových sad. Výsledný program vyústí v podobě kapitoly Testování a experimenty 5, kde je model ověřen porovnáním s jiným modelem a experimentováno s generátorem.

Kapitola 2

Generování syntetických snímků obličeje

Generování syntetických snímků obličeje je proces, při kterém probíhá tvorba nových snímku obličeje s pomocí znalostí obličejových znaků, na kterých byli trénovány pomocí reálných snímků pro dosažení nejvěrohodnějšího a realističtějšího výsledku [30].

Způsobů tohoto generování je ohromné množství od generování náhodných obličejů, generování snímků se změněným výrazem až po převádění obličeje ze 2D do 3D a následného otočení s vytvořením nového snímku [10, 12, 34].

Pro užší zaměření je zabýváno s generátory, jaké zpracovávají změnu vlastností obličeje, ve kterých nejvíce vynikají generativní soupeřící sítě s největší věrohodností a přesností. Tento typ sítě je představen v sekci 2.2. Základem pro generativní soupeřící sítě jsou dvě neuronové sítě. Tyto sítě fungují na principu hry s nulovým součtem, kde zisk jednoho je ztrátou druhého. Takový to způsob pomáhá ke vzniku kvalitnějších snímků [10]. S neuronovými sítěmi je pak seznáno v sekci 2.1. Dále se představí různé implementované modely v sekci 2.3 a na konec jsou datové sady pro generaci snímků v sekci 2.4.

2.1 Neuronové sítě

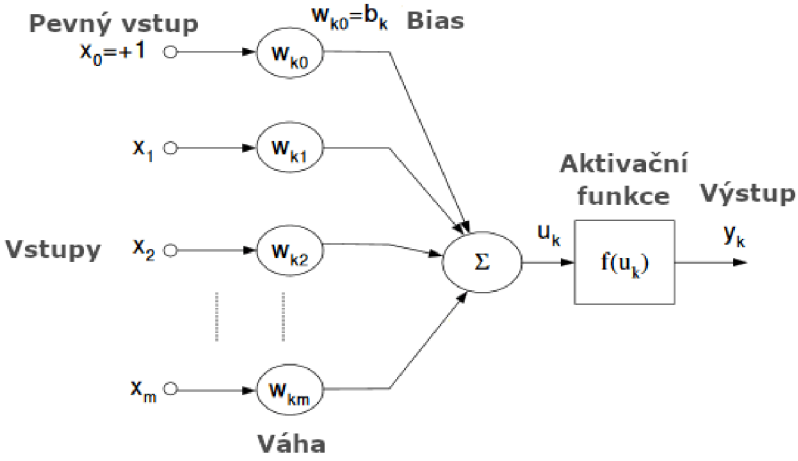
Neuronové sítě jsou založené na sofistikovaném pracování lidského mozku, kde stovky miliard neutronů jsou paralelně spojeny mezi sebou za účelem zpracování určité informace. Tato funkcionalita se výzkumníkům podařila do určité míry napodobit a přenést do počítačového světa. Neuronové sítě nám velmi pomáhají s řešením složitějších problémů jako u překladu jazyků nebo rozeznávání různých vzorů ve snímku, ale stále nejsou schopny s napodobením lidských emocí a vědomí [35].

Tato sekce se zabývá neuronovými sítěmi ze zdrojů [35, 26].

Perceptron

Perceptron je základní jednotkou v neuronových sítích, která je definovaná jako matematický model biologického neuronu. Model perceptronu se skládá z množiny vstupů, těla a právě jednoho výstupu, který je vyobrazen na obrázku 2.1.

Každý vstup se skládá z příchozí hodnoty a jeho váhy. Tyto dvě čísla se navzájem vynásobí a vstoupí do těla perceptronu. V těle perceptronu se všechny vynásobené hodnoty sečtou a přičte se hodnota zvaná *bias* neboli práh. Dalším krokem se vloží sumarizovaná



Obrázek 2.1: Model perceptronu [16].

hodnota do aktivační funkce a výsledek se předá na výstup perceptronu. Celkový mechanismus je poté možné popsat vzorcem 2.1.

$$y_k = f(b_k + \sum_{i=1}^{n} x_i w_{ki}) \tag{2.1}$$

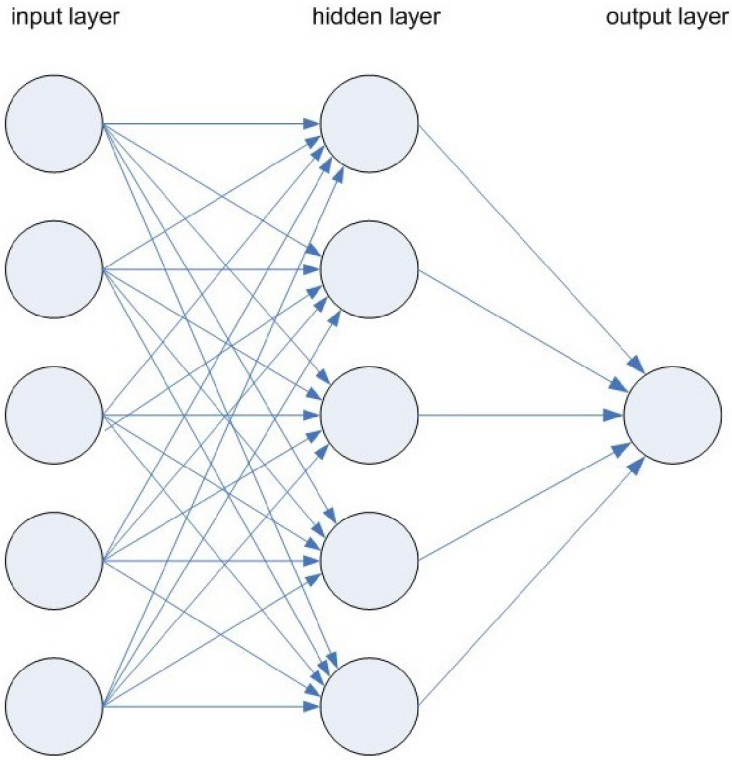
Tento vzorec znázorňuje x_n jako vstupní hodnoty, w_{ki} jako váhy jednotlivých vstupů a y_k jako výstupní hodnotu. b_k představuje práh a k označení perceptronu. n reprezentuje počet vstupů a f je aktivační funkce [35, 16].

Struktura

V neuronových sítích se perceptrony skládají do určitých vrstev pro dosažení optimálnějších výsledků a k řešení složitějších problémů. Každá vrstva se skládá z množiny perceptronů, které jsou pomocí vazeb napojeny na všechny perceptrony předchozí vrstvy a podle typu sítě i mezi sebou. Příklad je vyobrazen na obrázku 2.2 s tři vrstvou neuronovou sítí [19]. Tyto vrstvy poté dělíme na vstupní vrstvu, skrytou vrstvu a výstupní vrstvu. Vstupní vrstva slouží k přijetí dat ze vstupu a distribuci do skryté vrstvy. Skrytá vrstva má za účel zpracovat tyto data pomocí prahů a vah, které se skrze učení je mohou modifikovat, aby měli, co nejmenší chybovost. Skrytá vrstva může mít i více vnitřních vrstev pro zlepšení kvality dat na její výstupu do výstupních vrstvy [35]. Výstupní vrstva následně vyobrazí výsledky, pomocí hodnoty aktivace na koncových perceptronech. Z výsledků se následně zjišťují kategorie, predikce a tak dál.

Aktivační funkce

Aktivační funkce je část perceptronu, která rozhoduje o vlastnostech výstupních hodnot. Její funkcionalitou je zjišťování, jak hodnoty na výstupu mají vypadat v podobě mapování do určitého intervalu. Do aktivační funkce se při velkém výběru vybírají funkce podle požadavků, jaké má perceptron splňovat a jaké budou mít pozitivní vliv na architekturu sítě [29].



Obrázek 2.2: Třívrstvá neuronová síť [19].

Jeden z nejednodušších algoritmů je binární kroková funkce, která odesílá na výstup hodnotu 0, dokud nepřijde zlomový bod a nezačne odesílat 1, nebo lineární funkce, která data nezpracovává a okamžitě je odesílá. Aktivačních funkcí je nezměrné množství, ale nejzákladnější jsou *sigmoid* [9], *ReLu* [6], *tanh* [29] a *softmax* [8]. K těmto funkcím existují ještě jiné varianty jako *leaky ReLu* [29], které kompenzuji nevýhody původních funkcí nebo rozšiřují funkčnost perceptronu. V případu *leaky ReLu* je umožněno při trénování jít i do záporných čísel, aby se perceptron nestal ve specifických případech nefunkčním.

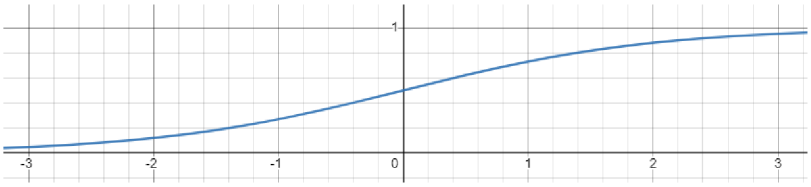
Sigmoid

Sigmoid je typ aktivační funkce, která mapuje všechny vstupní hodnoty v podobě reálných čísel do intervalu od 0 do 1. Tato funkce je znázorněna na obrázku 2.3. Kvůli jeho mapování, tak se tato funkce používá u pravděpodobností, kde dokáže najít, v jaké části přechodu od záporného čísla ke kladnému se nachází. Funkce je daná předpisem 2.2.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Vlastnosti této funkce jsou, že se skládá z tří části. První část, která je do vstupní hodnoty -3, je možné jí ohodnotit jako 0, protože nabývá hodnot extrémně blízkých k 0. V intervalu od -3 do 3 to je skoro lineárně rostoucí křivka a od 3 výše je možná ohodnotit jako 1, ze stejného důvodu jako první interval. Z hlediska informatiky, tak tato funkce není moc efektivní, protože funkce je náročná výpočet a je nevhodná pro některé hodnoty, protože je velmi citlivá pří přechodu hodnoty 0,5, která odděluje záporné a kladné vstupní

hodnoty. Mezi neutrální vlastnosti patří postupné tlumení počátečních a koncových hodnot. V pozitivním pohledu funkce dobře odfiltruje odlehlé hodnoty, které jsou v extrémně vysoké nebo extrémně nízké a namapuje je na hodnoty blízké 1 a 0. Pro tyto vlastnosti se funkce se používá u primárně u výstupní vrstvy [9].

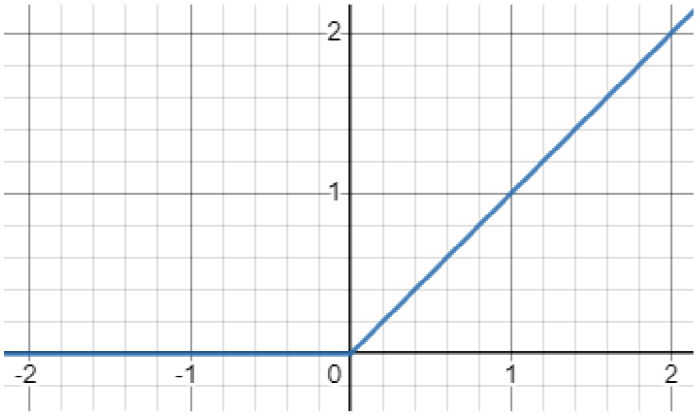


Obrázek 2.3: Aktivační funkce *sigmoid*.

ReLU

Tato funkce je jedna z nejpoužívanějších z důvodu její jednoduchosti a snadné předvidatelnosti. Její průběh je v grafu 2.4, kde můžeme zjistit, že se funkce pohybuje po v reálných číslech, ale výstupní je v rozsahu reálných nezáporných čísel. Tato funkce zpracovaná všechny záporná číslo mapováním do nuly, čímž mnohem snadněji předpovídá výstup, a samotný výpočet, který není náročný a učí se poměrně rychle. Nevýhoda této funkce je možnost zablokování perceptronu. Tento problém vznikne, když při učení váhy jsou nastaveny tak, že jejich součet s jakýmkoli vstupem je záporný, nebo nulový a když to zpracuje *ReLU*, tak výstup bude vždy nulový. Tímto poté nelze ovlivňovat hodnocení a ani změna vah není možná, protože při učení změna vah se spočítá pomocí vlivu na hodnocení a ten je vždy nulový. Primární využití této funkce je ve skrytých vrstvách. Funkce je dána tímto předpisem 2.3.

$$f(x) = \begin{cases} x & \text{pro } x \geq 0 \\ 0 & \text{jinak} \end{cases} \tag{2.3}$$



Obrázek 2.4: Aktivační funkce *ReLU*.

Softmax

Softmax je speciální aktivační funkce, která se používá u klasifikačních neuronových sítí v poslední plně propojené vrstvě. Používá se, protože uvádí pravděpodobnost výsledku pře-

počítanou ze všech perceptronů v dané vrstvě a tím zaručuje, že součet všech perceptronů ve vrstvě je rovnal jedné. Respektive zjistí výsledek v perceptronech. Ten poté sečte a konečný výsledek perceptronu se vytvoří hodnotou v perceptronu děleno hodnotou celkovým součtem [8].

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.4}$$

Je reprezentovaná funkcí u rovnice 2.4, kde K je počet perceptronů ve vrstvě a z_i je označení perceptronu.

Trénování neuronové sítě

Neuronové sítě jsou tvořené z velkého množství perceptronů, které jsou složené z nastavitelných vah a prahů. Pro zvýšení kvality a přesnosti je nutné nastavit tyto váhy a prahy na optimální čísla. Jeden ze způsobů je nastavit je ručně. Tento způsob je velmi neefektivní a u větších sítí i nemožný pro lidi, protože komplexnost některých řešený je neskutečně velká. Druhý způsob je učením, kde se používají automatické nastavování vah a prahů s pomocí datových sad, které slouží jako příklady s výsledky pomocí, jakých se optimalizují natavení. Celkově jsou dva typy učení. První je učení s učitelem a druhý je učení bez učitele [35].

První typ funguje na principu známého vstupu a výstupu, kde je předem připravená datová sada s informacemi o vlastnostech vzorků. Neuronová síť se pak učí tak, že vezme vstupní vzorek, zpracuje ho, výsledek porovná s předpokládaným výsledkem a upraví váhy neuronové sítě. Toto je v základu *forward-propagati* a *back-propagation*, které jsou popsané důkladně níže. Pro správnou funkcionalitu je potřeba funkce s názvem ztrátová, které vypočítává přesnost a odchýlení sítě.

Druhý typ je tedy učení bez učitele. V tomto případě nejsou poskytnuty žádné vlastnosti navíc jako s učitelem, ale jsou poskytnuty parametry, jako například kolik má mít výstupů. Hlavním principem je vložení velké množství dat, které se neuronová síť snaží roztřídit do shluků podle, kterých bude hodnocena. Tento princip je velmi praktický, ale je složitější na výrobu, potřebuje víc dat na učení a potřebuje speciální funkci pro zajištění funkcionality.

Back-propagation

Tento učící algoritmus se používá u vícevrstevných neuronových sítí [35].

Tato metoda se skládá ze dvou fází. První fáze je *forward-propagation*, kde neuronové sítě počítají normálně výsledek pomocí vah a prahů. Výsledek je poté číselný vektor, který je porovnán ve ztrátové funkci za vypočtením výsledku chyby. V druhé fázi zvaná *back-propagation* se propaguje vypočtená chyba do vah a prahů směrem od výstupu ke vstupu v poměrech, které jsou před počítané podle vlivu změny na výsledku. Tento proces se opakuje, dokud není pravděpodobnost chyby dostatečně nízká.

Ztrátová funkce

Ztrátová funkce neboli *loss function* je funkce, která vypočítá odchylku výsledku neuronových sítí a ideálního výsledku pro každý výstup neuronové sítě. Ztrátová funkce je klíčovou pro hodnocení kvality neuronových sítí, výpočet chybovosti a zpětného trénování.

Cílem trénování je minimalizovat hodnotu ztrátové funkce. To se obvykle dělá pomocí optimalizačních algoritmů, jako je gradientní sestup (*gradient descent*), které upravují váhy a prahy tak, aby minimalizovaly ztrátovou funkci [35].

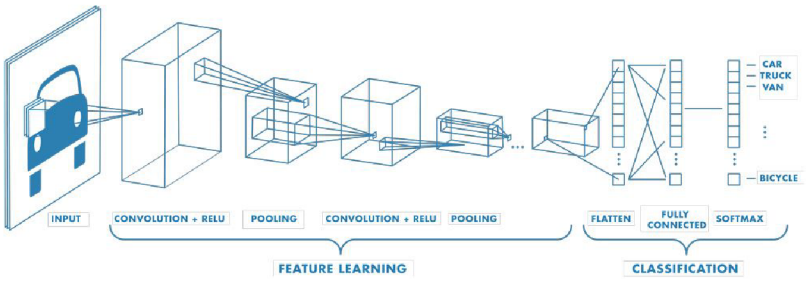
Ztrátové funkci dělíme na dva primární typy. První je regresní, kde se porovnává s výstupní hodnotu regresní neuronové sítě, kde regresní znamenaná předpovídání hodnoty na základě vstupů. Druhý je klasifikační, kde se porovnává s výstupem klasifikační neuronové sítě, která produkuje soubor pravděpodobnostních vektorů na určení správné třídy [1].

Jedním z příkladů je ztrátová funkce *Binary Cross-Entropy Loss* [39]. Tato funkce používá binární klasifikaci pro předpověď výstupu podle vloženého vzorku do dvou skupin. K tomuto je v poslední vrstvě aktivační funkce *Sigmoid* [8], která pomocí ztrátové funkce určuje pravděpodobnost, do které skupiny patří. Je popsána rovnicí 2.5, kde p je pravděpodobnost, že budoucí vstup bude patřit do jedné skupiny, a y je označení pro vstup.

$$L = -(y \log(p) + (1 - y) \log(1 - p)) \tag{2.5}$$

Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou jedny z typů neuronových sítí. Tyto sítě se specializuji na zpracovaní obrazu pomocí konvoluce, ale jinak se chovají jako základní neuronové sítě [7]. Tyto sítě jsou složené ze vstupní vrstvy, alespoň jedné konvoluční vrstvy, několik *pooling* vrstev, několik plně propojených vrstev a výstupní vrstvy. Standartě jsou tvořeny ve strukturách, kde se konvoluční vrstvy a *pooling* vrstev střídají za sebou a poté jsou dány propojené vrstvy. Tato struktura vidět na obrázku 2.5 [28].



Obrázek 2.5: Struktura konvoluční neuronové sítě [23].

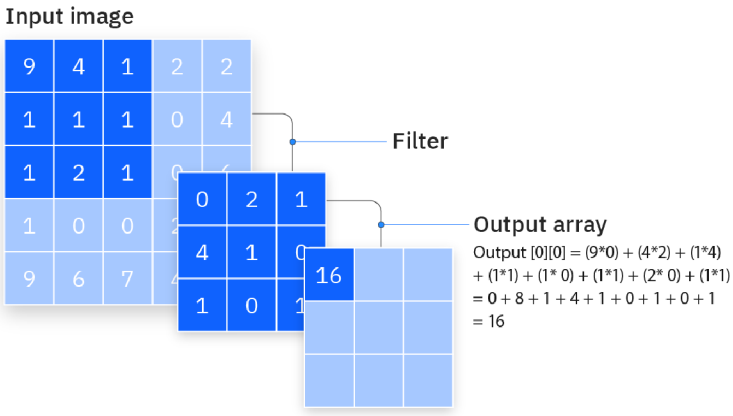
Konvoluční vrstva

Konvoluční vrstva, jak už od názvu lze odvodit, tak pracuje na principu matematické konvoluce, kde pomocí specializovaných filtrů dokáže zvýraznit různé vzory (hrany, čáry, atd) [28].

Vstupem této vrstvy jsou vstupní data a filtry ve 2D polích a výstup je 2D pole zvané příznaková mapa (*feature map*), ve kterém se pomocí intenzity odstínu dozvíme výsledek konvoluce jako na příkladovém obrázku 2.6.

Konvoluce je znázorněna výrazem 2.6.

$$G[i, j] = X * H = \sum_k \sum_l X[k, l] \cdot H[i - k, j - l] \tag{2.6}$$



Obrázek 2.6: Příklad konvoluce obrázku s filtrem [20].

Jednotlivé proměnné výrazu jsou X, H, i, j. X je vstupní pole 2D hodnot. H je vstupní pole filtru. G je výstupní pole filtru. i a j jsou pozice právě vypočítávaného bodu.

Pooling vrstva

Pooling vrstva [28] se nachází za vrstvou konvoluční. Slouží k snížení prostorové dimenze vstupu tím, že okolní hodnoty agreguje do jedné hodnoty. Způsob agregaceje možný ovlivnit povolovacím polem. Její vstup pole hodnot a tedy v tomto případě příznaková mapa z konvoluční sítě a výstup je zmenšená příznaková mapa.

Tato vrstva má dva typy: *max pooling* a *average pooling*. První typ *max pooling* vrací maximální hodnotu z předané oblasti a druhý typ *average pooling* vrací průměr hodnot z předané oblasti, kde oblast je pole vstupních hodnot.

Plně propojená vrstva

Tato vrstva se dává na konec konvolučních neuronových sítí. Nachází se různém množství od dvou a více. Slouží jako normální neuronová vrstva na učení ze vstupu, kde v každé vrstvě jsou všechny výstupy z předchozí vrstvy spojeny do každého perceptronu ve stávající vrstvě ovlivňující váhy při každém učení.

ResNet-50

ResNet-50 je model konvoluční neuronové sítě, která je složena z 48 konvolučních vrstev a 2 pooloing vrstev. Tyto vrstvy jsou vytvořeny speciální metodou tak, aby při učení velkého množství vrstev nevyprchala změna vypočítaná z gradientu, která nastavuje váhy perceptronů a tím zvýšit jejich účinnost. Tato metoda prochází mezi vrstvami na propagaci změny vah tak, že konvoluční vrstvy rozřadí do skupinek (reziduální bloky), které sdílejí stejnou změnu a u kterých se zmenší změna pouze o jednotku jedné vrty místo množství vrstev ve skupině. Výsledná změna se poté propaguje další skupince a tento cyklus se opakuje do konce modelu [17].

Siamská neuronová síť

Siamská neuronová síť je jedna z typů neuronové sítě [3], která se specializuje na porovnávání výstupů ze stejných neutronových sítí.

Obecně se skládá ze dvou a nebo více stejných podsítí, které produkují výstupní vektory. Tyto vektory vyústí do vrstvy, která započítá jejich vzdálenost pomocí, které je možné zjisti jejich odlišnosti. Toto byl jeden ze způsobů u dvou podsítí, jak zjistit odlišnost, ale je prvně nutné analyzovat data na zjištění rozmezí. U třech podsítí je příklad v porovnávaní chtěného snímku s jedním odlišným a jedním podobným. Pokud vstup má menší vzdálenost s podobným snímkem, tak je vstup podobný, pokud to je naopak, tak je odlišný.

Použité podsítě u všech těchto typů musí splňovat určité podmínky. Hlavní z nich je, že musí mít stejnou architekturu a sdílet konfiguraci prahů a vah tak, aby pro stejné snímky vytvořily vždy stejné vektory nehledě na podsíti. Jedním z příkladů jsou sítě pro verifikaci podpisů či podobnosti obličejů.

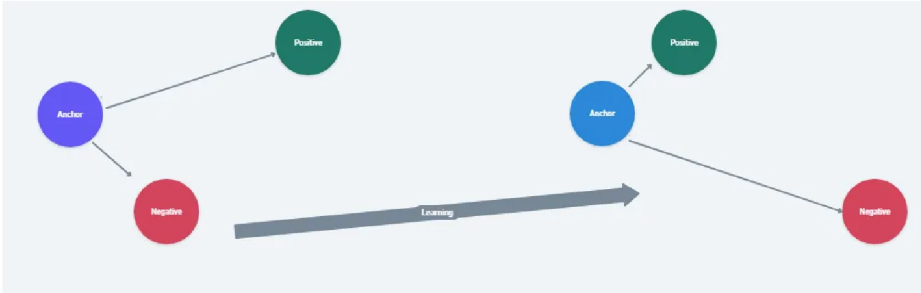
Používané ztrátové funkce v Siamských sítích jsou primárně dvě *Triplet loss* a *Contrastive loss*, avšak používají se i jiné, ale často nejsou tak přesné.

Triplet loss

Triplet loss funguje na principu přirovnaní tří vstupů, kde jeden je zakotvený jako výchozí bod, druhý je pozitivní (je podobný) a třetí je negativní (je odlišný). Cílem modelu je zmenšit vzdálenosti od pozitivního vstupu a od negativního se co nevíce oddálit. Tento úkol je znázorněn na obrázku 2.7. Toto fungování pomáhá snížit vzdálenosti mezi stejnými třídami snímků a oddálit rozdílné, ale tím se výpočet pro její funkci stane náročnější.

$$L\left(A,P,N\right)=\max\left(\|\,\mathrm{f}\left(A\right)-\mathrm{f}\left(P\right)\|_2-\|\,\mathrm{f}\left(A\right)-\mathrm{f}\left(N\right)\|_2+\alpha,0\right) \tag{2.7}$$

Tato ztrátová funkce je v rovnicí 2.7, kde $f(x)$ je funkce, která zpracovává vstupy a vytvoří z nich pole vektorů. α je nastavitelný práh. Poslední trojice A, P a N jsou vstupy znázorňující výchozí bod, pozitivní vstup a negativní vstup.

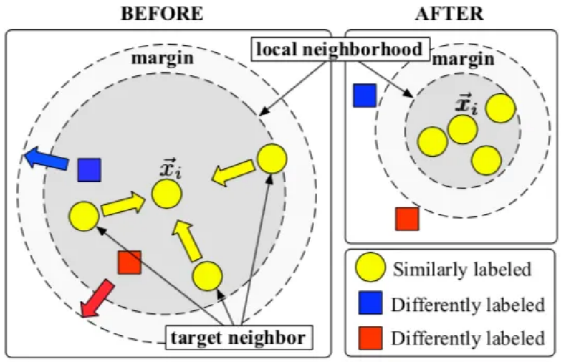


Obrázek 2.7: Obrázek ztrátové funkce *Triplet loss* [3].

Contrastive loss

Contrastive loss je ztrátová funkce, která porovnává pouze dva vstupy. První je zakotvený a má funkcionalitu také jako výchozí bod. Druhý snímek variabilní bod na porovnání. Cílem této funkce zmenšit vzdálenost s podobnými snímky a zvětšit ji u odlišných. Tento úkol je znázorněn na obrázku 2.8. *Contrastive loss* je poté zapsaná i vzorcem 2.8, kde y_{pred} je vypočítaný výsledek z modelu, y_{true} je pravdivý výsledek a m velikost rozmezí mezi nimi.

$$L = y_{true} \cdot y_{pred}^2 + (1 - y_{true}) \cdot (max(m - y_{pred}, 0))^2 \tag{2.8}$$



Obrázek 2.8: Obrázek ztrátové funkce *Contrastive loss* [36].

2.2 Generativní soupeřící sítě

Generativní soupeřící sítě anglicky *Generative Adversarial Networks* se zkratkou GANs jsou model neuronových sítí, které fungují na speciálním typu učení pro zvýšení jejich kvality. Jak již jméno napovídá, tak je složena z sítí, které soupeří mezi sebou s cílem být lepší než druhá s tím, že jedna získá na začátku vzor a jiná se ho snaží napodobit. Tyto sítě se postupně začnou vylepšovat mezi sebou než se dostanou k limitům jejich architektury. Generativní sítě se obecně skládají ze dvou hlavních složek – generátor a diskriminátor.

Tato sekce je ze zdrojů [21] a [5].

Generátor

Generátor je část modelu GANS, která se stará o generaci falešných snímků za účelem se zlepšit a získat kvalitnější výstup. Toto docílí při projití dostatečném množství pokusů s ohodnocením ověřovacího prvku – diskriminátor.

Prvním výstupem je vždy náhodný šum, který se pomocí vah upravuje tak, že zašle snímek soupeřící síti neboli diskriminátoru a ten jí vrátí číselnou hodnotu neboli skóre, jak moc špatně dopadl jeho výstup. Generátor poté vezme odeslaný snímek, vypočítá hodnotu, jakou se váhy musí posunout pro každý pixel a propaguje ho zpět do sítě ať je lepší nebo horší než milý snímek. V toto hraje roli diskriminátor, protože čím lepší bude výsledek, tím vetší bude poté posunutí vah a naopak platí to samé. Čím horší bude výsledek, tím menší bude poté posunutí vah. Výsledkem je poté pomalé posunování vah směrem k typu snímků, které mají větší skóre.

Tato síť má vnitřní vstup, který nastaven náhodně při kaženém spuštěni a slouží pro získání variability výstupu a možného zlepšení, kde každá iterace tento vstup jemně omezí cestou na výstup pomocí vah.

Generátor nepracuje vždy pouze s náhodným šumem, ale na vstupu může mít i snímek, se kterým musí umět pracovat. Z tohoto důvodu se do generátorů dávají i jiné sítě, které se starají o analýzu a o úpravu snímků. První síť zpracovává analyzuje obrázek a hledá objekty, které má změnit. Druhá tyto objekty nahradí nebo jinak změní a zahladí je do snímku.

Tento typ generátorů poté nemusí mít jen jeden vstup se snímkem, ale muže mít i vstup s vektory, ve kterých je uloženo, do jakého typu se má snímek změnit. Toto umožní zpracovat více typů přeměn do jednoho modelu.

Generátory požívají technologie neuronových sítí, kde jednou z možných ztrátových funkcí je *Binary Cross-Entropy loss*. V případě analyzační sítě jsou použity konvoluční neutronové sítě.

Diskriminátor

Diskriminátor je část modelu GANS, která se stará o kontrolu celého modelu, kde některé typy mají i více diskriminátorů pro dosažení lepších výsledků.

Diskriminátor musí umět kontrolovat výstup, zda splňuje dané požadavky, a proto při první inicializaci je tato část naučena, jak požadované snímky vypadají, a ke kterým má dávat lepší hodnocení.

V příkladu na generování snímků, tak generátor vždy s počátku vygeneruje náhodný šum, které když přijdou k diskriminátoru, tak je musí ohodnotit podle snímků, ze kterých se učil. Toto hodnocení je často od nuly do jedné a čímž říká, že se s tímto snímkem zmýlil s takovou pravděpodobností. Naopak pokud generátor je už na vysoké úrovni, tak se diskriminátor začne mýlit, ale diskriminátor ví, že vše, co přijde od generátoru není podle originálních požadavků. Tímto může zjisti svoji chybu změnit své hodnocení. Také se tímto může učit i ze snímků u generátorů, které požadují nějaký snímek jako vstup. Tyto snímky se poté dají použit pro zlepšeni diskriminátoru.

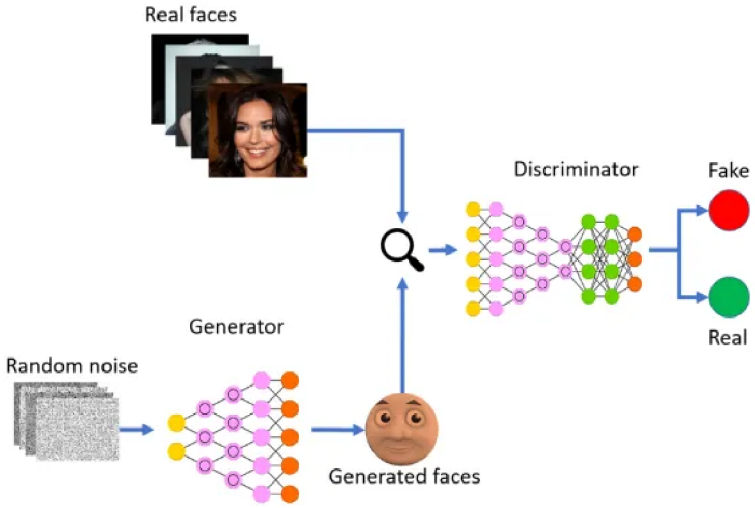
Jedním z kontrol, které musí diskriminátor provádět je kontrola realističnosti snímků, nebo podoby k originálu. Pokud generátor nemá žádný vstup, tak diskriminátor musí umět prakticky jen dva typy informací a to je, zda snímek splňuje typ, co má mít na výstupu, a zda je realistický, protože pokud generátor vytváří z větší části snímek náhodně, tak je velká možnost, že přijde s něčím novým. Pokud se poukáže na realitu, tak se musí počítat například s tím, že některé plochy jsou pouze značené jedním typem barev jako bělmo u očí a nikoliv jiná varianta. Jednoduše řečeno je nutné diskriminátor naučit základní poznatky, co musí umět vyhodnotit.

U generátorů se vstupy je také nutné taky kontrolovat podobnost výstupu se vstupem, neboť je důležité zachovat základní strukturu, která se nemá měnit, aby model neudělal z žirafy slona.

Pro diskriminátory se požívají často také technologii konvolučních neutronovích sítí, kde jednou z možných ztrátových funkcí je *Binary Cross-Entropy loss*, ale není tomu vždy tak, protože záleží, co kontrolují.

Struktura a učení

Generátor a diskriminátor jsou spojeny za sebou, jak je znázorněno v obrázku 2.9. Dohromady pracují tak, že generátor vygeneruje snímek a ten zašle do diskriminátoru, který ho zpracuje, zda splňuje požadavky. Pokud je rozhodnuto, že nesplňuje požadavky, tak se aktualizuje generátor pro zajištění lepšího generování. Pokud je rozhodnuto, že splňuje požadavky, tak je aktualizován diskriminátor, protože ho ohodnotil špatně. Tímto způsobeném se poté síť sama učí a tento způsob je zván učení bez dozoru (*unsupervised learning*) u generativních sítí.



Obrázek 2.9: Model generativní soupeřící sítě [2].

2.3 Algoritmy pro generování syntetických snímků obličeje

Modelů na generování syntetických snímků obličeje, již je vytvořeno velké množství a každý byl stvořen za specifickým cílem, tudíž má své výhody a nevýhody. V této sekci jsou představeny modely i s jejich funkcemi jako *StarGAN*, *RelGAN*, atd.

Další důležitou informací je využití pojmů doména a atributy, které jsou použité ve specifickém významu. Atribut snímku je myšlen jako jeho nějaká vlastnost. Například u snímků s obličejem to je barva vlasů atd. Hodnota atributu je význam této vlastnosti s příkladem hodnoty jako černé vlasy a doména je soubor snímků se stejnými hodnotami atributů.

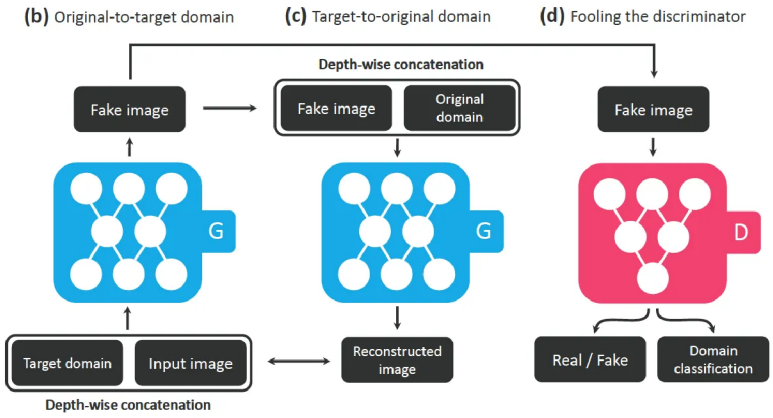
StarGAN

StarGAN je model pro generování syntetických snímků založený na generativní soupeřící síti, která obsahuje jeden generátor a jeden diskriminátor v podobě dvou konvolučních neuronových sítí. Jeho výsadou je transformace vstupních snímků na výstupní v cílové doméně, kde typ cílové domény není daný pouze jednou doménou od jeho trénování, ale je dán širším množstvím domén. Další z jeho předností je, že se dokáže učit z jedné datové sady se snímky o různých atributech místo použití více modelů, které by uměli změnu na zpět.

Tato sekce je ze zdroje [12].

Funkce tohoto modelu je možná shrnou to tří procesů. První je generování snímku se zadanou cílovou doménou, která vygeneruje snímek na porovnání. Druhý proces je porovnávání v diskriminátoru, který zjišťuje, zda je snímek reálný a v jaké doméně se nachází. Třetí proces je zpětná generování, kde vygenerovaný snímek je generován do originální domény pro zjištění zkreslenosti od originálního snímku. Celý tento proces je znázorněn na obrázku 2.10.

Tento model se učí reálností generovaného snímku, správným převedení do cílové domény a jeho zpětnou generací na penalizaci odlišností od originálu. Tyto úkoly jsou zpracovávány ztrátovou funkcí, která je zde rozdělena do několika pod funkcí.



Obrázek 2.10: Model StarGANu [12].

Adversarial Loss je funkce slouží pro rozeznaní reálnosti snímku. Je znázorněn rovnicí 2.9, kde generátor generuje $G(x, c)$ snímek x do cílové domény c a $D_{src}(x)$ je rozdělení pravděpodobnosti přes zdroje v diskriminátoru. Generátor se snaží hodnotu L_{adv} zmenšit a diskriminátor ji zvětšit.

$$L_{adv} = logD_{src}(x) + log(1 - D_{src}(G(x, c))) \tag{2.9}$$

Domain Classification Loss. Tato funkce slouží pro optimalizaci tvorby snímku do různých domén. Je složena ze dvou funkcí 2.10 a 2.11, kde 2.10 je pro diskriminátor a 2.11 je pro generátor.

$$L^r_{cls} = -logD_{cls}(c'|x) \tag{2.10}$$

$$L^f_{cls} = -logD_{cls}(c|G(x, c)) \tag{2.11}$$

$D_{cls}(c'|x)$ představuje rozdělení pravděpodobnosti přes domény, kde c' je originální doména. Diskriminátor pomocí něj optimalizuje rozeznání cílové domény. Při učení minimalizování jeho funkce se učí od reálného snímku originální doménu. Generátor ji snaží minimalizovat pro lepší optimalizaci výsledků cílové domény.

Reconstruction Loss. Tato optimalizace funguje za účelem dosažení věrnosti svého původního snímku. Užívá tuto rovnici a je aplikovány v generátoru.

$$L_{rec} = ||x - G(G(x, c), c')||_1 \tag{2.12}$$

Full Objective. Jako konečný výraz je možný reprezentovat těmito dvěma rovnicemi 2.13, 2.14.

$$L_D = -L_{adv} + \lambda_{cls} L^r_{cls} \tag{2.13}$$

$$L_G = L_{adv} + \lambda_{cls} L^f_{cls} + \lambda_{rec} L_{rec} \tag{2.14}$$

λ_{cls} a λ_{rec} jsou hyper-parametry, které řídí důležitost jednotlivcích ztrátových funkcí a rekonstrukce ve srovnání se soupeřící ztrátou.

Z důvodu nejednoznačnosti cílové domény z učení na vícero datových sadách byl vytvořen vektor masky na zaznačení klasifikace domény a na vytřídění nespecifikovaných atributů. Tato klasifikace je znázorněna na rovnici 2.15, kde C_n jsou jednotlivé klasifikace hodnot atributů datových sad m.

$$c = [c_1, ..., c_n, m] \tag{2.15}$$

Na porovnání se staršími modely byli použity *DIAT*, *CycleGAN* a *IcGAN*. Na těchto sítích byla použité datové sady *CelebA* [22] s *RaFD*, kde 90 % bylo použito na trénování a zbylých 10 % bylo použito na experimenty pro porovnání věrohodnosti výsledku. Ve výsledném porovnání StarGAN měl mnohem lepší výsledky než ostatní sítě, ale zobrazili se některé nedokonati v transformaci brýlí a šperků anebo při změně barvy vlasů se změnil trochu i výraz.

RelGAN

Model pro generování syntetických snímků *RelGAN* je založen na generativní soupeřící síti, která obsahuje jeden generátor a tři diskriminátory. Tato síť je tvořena čtyřmi konvolučními neuronovými sítěmi s tím, že diskriminátory sdílí šest konvolučních vrstev. Vstup je tvořen snímkem s vektorem, který určuje, do jaké domény se má dostat relativně k vstupnímu snímku, a na výstup je snímek v cílové doméně, která také podporuje více změn najednou. Tento model si hodně zakládá na tomto vektoru, kde se snímek mění podle hodnoty atributu tak, že kde je hodnota nulová, tak se nic nemění, ale pokud je hodnota 1 nebo -1, tak je snímek změněn pouze o tyto hodnoty. Respektive model snaží tento snímek upravit jen v tom atributu a neměnit žádný jiný, avšak mění ho relativně (například přidá více úsměvu, ale nemusí se potom úplně smát). Tato sekce je ze zdroje [37].

Funkcionalita je rozdělena do čtyř částí. První je generování snímku tvořená nezbytnými změnami. Druhá je tvořena prvním diskriminátorem D_{Real}, který se snaží kontrolovat reálnost snímku. Třetí část je druhý diskriminátor D_{Match}, který porovnává generovaný snímek se snímkem z cílové doméně podle originálního snímku a relativního vektoru. Poslední část tvořena třetím diskriminátorem D_{Interp}, u jakého se snaží diskriminátor zjistit míru interpolace generovaného snímku s originálním, zda mají stejnou strukturu.

Tento model byl porovnáván se *StarGAN* a *AttGAN*, kde byl znázorněn výraznější pokrok u propagace změny atributu na obličeji bez vedlejších změn a zachování obličejových rysů z originálního snímku.

Adam²Net

Jedním z jiných přístupů přistoupilo *Adam²Net*, který jen generuje snímky z pomocí hlubokých konvolučních neuronových sítí s grafem adaptivní inference (*Adaptive Inference Graph*). Tato síť funguje na bázi spínání určitých neuronových vrstev podle vstupu. Cílová doména zde povoluje více změn nejednou. Tento model je cílený na generaci kvalitních

snímků mezi více doménami za limitovaných výpočetních zdrojích. Tato sekce je ze zdroje [25].

Funkcionalita *Adam²Net* je tvořena enkodérem obsahu, enkodérem stylu a zpětným dekodérem. Enkodéry slouží k vyextrahování obsahu a stylu snímku. Extrahování obsahu a stylu poté procházejí sekvencí bloků, které za běhu pomocí adaptivního grafu modifikují obsah tak, aby obsahoval požadovaný výstup v dané doméně. Po projití bloků se obsah dekóduje na výstupní snímek. Tento způsob umožňuje použít lineární množství bloků podle generačních domén na místo kvadratického množství generátorů.

Tento model byl zkoušen na omezené velikosti domény na datové sadě *CelebA* [22], kde si vedl poměrně dobře, ale jeho výkon nebyl tak dobrý jako model *StarGAN*, který používal výkonnější techniku.

2.4 Datové sady

V této části jsou představeny některé datové sady, které jsou použitelné pro generaci snímků na rozpoznávání, a nebo pro vyhodnocování identit ze snímků. Mají dobré využití pro hodnocení pro modely, protože mají dobře popsané vlastnosti snímků jako například atributy nebo identity na snímcích. Tyto datové sady se jmenují *CelebA* a *LFW - People*.

CelebA

CelebA je zkratka pro datovou sadu se jménem *Celeb Faces Attributes Dataset* [22]. Tento dateset obsahuje přes 200 tísíc obličejových snímků od 10 177 lidí. Každý tento snímek je také obsahuje až 40 atributů, které jsou jako příklad barva vlasů, pohlaví a nebo i nějaké výrazy obličeje jako smích. Tato datová sada se může využít pro trénování nebo verifikaci správně zařazených atributů.

LFW - People

LFW - People je datová sada pro rozpoznávání obličejů a celým jménem je *The Labeled Faces in the Wild face recognition dataset* [18]. Tato datová sada obsahuje přes 13 000 snímků s popsanými jmény od osob, které na nich byli zachyceni. Tyto snímky jsou seřazeny do složek, kde 1680 osob obsahuje dva a více snímků. Převážná část to jsou celebrity, které byli zachyceny na různých webech.

Kapitola 3

Rozpoznání osob podle obličeje

Rozpoznávání obličeje na snímku je problém, který se skládá z identifikace a ověření osob podle charakteristických rysů tváře. Tento problém je snadno řešitelný pro lidi, ale až do nedávna byl náročným problém pro počítačové vidění, protože můžou nastat nečekané úskalí jako, když se obličej začne měnit časem nebo zraněními, jeho pozice se změní nebo na něj budou působit vnější vlivy jako je sluneční svit.

K řešení tohoto počítačového problému nám pomáhají metody hloubkového učení, které jsou schopny využít rozsáhlé datové sady s pestrým množstvím tváří pro možný vývoj a trénování modelů. Tento přístup poté umožňuje moderním modelům fungovat na stejné úrovni a později i překonat schopnosti rozpoznávání tváří u lidí.

V této kapitole je popsán tento problém identifikace a ověřování osob na fotografiích 3.1, proces na jeho detekci, zarovnání a extrakce rysů je v sekci 3.2 a nakonec je seznámeno s procesem na rozpoznání obličejů v sekci 3.3.

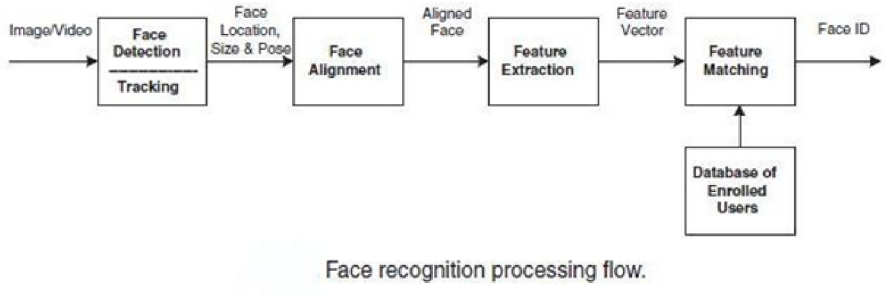
Tato kapitola je tvořena větší částí ze zdroje [4].

3.1 Identifikace a ověřování osob na fotografii

Často je potřeba rozeznat osoby podle obličeje k autentizaci anebo k již zmíněné identifikaci nebo ověření. Autentizace slouží jako omezení přístupu ke zdrojům s případem pro omezení přístupu jen důvěryhodným osobám a zamezení jiným do areálu. Ověření je už porovnaní dvou obličejů u případu automatické potvrzení osoby podle jejich obličeje s fotkou na občanském průkazu a identifikace je s významem nalezení obličeje, které se používá v sociálních sítích, kde se můžou přidělovat jména osobám na fotce. Tyto případy se shluknou do jednoho problému a ten se obecně nazpívá automatické rozpoznávání tváře. Týká se poté jak statických fotografií, tak i videí a přímých přenosů.

Nad tímto problémem se začalo uvažovat od 50. a 60. let 20. století, kde výzkum automatického rozpoznávání obličeje se zahájil v 70. letech. Tento výzkum vyústil ve studie o rozpoznávání obličejů, které vyvinuly své první modely na začátku 90. let [32]. V posledních dvou desetiletí se posunul jejich vývoj kvůli pokročilejším technikám a výhodnějšímu technickému vybavení. Dnes totiž tento pokrok umožňuje lidem spolehlivě rozeznávat obličeje na snímcích a jak se jmenují, i když mají starší obličeje nebo nosí sluneční brýle, atd.

Rozpoznávání obličeje je popisován jako proces, který je rozdělen do čtyři kroků: detekce obličeje, zarovnání obličeje, extrakce rysů a rozpoznání obličeje. Je reprezentován na obrázku 3.1.



Obrázek 3.1: Proces rozpoznávání obličeje [4].

1. **Detekce obličeje** – Lokalizování obličejů na snímku a jejich extrahování.
2. **Zarovnání obličeje** – Zarovná snímek tak, aby byl konzistentní s databází a nebo jiným snímkem.
3. **Extrakce rysů** – Extrahuje znaky ze snímku obličeje, které jsou užitečné pro rozpoznání.
4. **Rozpoznávání obličejů** – Provádí porovnání obličeje s biometrickými rysy jiných obličejů z databáze.

Tento proces muže být implementován programy anebo moduly pro každá krok. V určitých případech se mohou některé kroky zkombinovat do jednoho.

Dále je rozvinutá detekce obličeje a rozpoznávání obličejů, protože zarovnání pracuje na jednoduchém oříznutí nebo transformaci snímku, která je u modelů obsažená v detekci společně s částí extrakce, a zbytek extrakce rysů je rozeznání objektu obličejového znaku, které je vysvětleno v rozpoznávání obličeje.

3.2 Detekce obličeje a jejich algoritmy

Detekce obličeje je netriviálním krokem v rozpoznávání obličeje. V tomto kroku nachází problém rozpoznávaní objektů, u kterého je potřeba nejen najít umístění každého obličeje, ale i jeho rozsah z oblasti (*Region of Interest*). Největší stížení je v tom, že lidská tvář ve svém vzhledu je dynamický objekt, který má vysokou míru variability, což činí detekci obličeje v počítačovém vidění o to obtížnější.

Detekce musí být robustní vzhledem k podmínkám, kdy je záznam pořízen. Nemělo by nastat, že se osoba trochu pootočí hlavou, nosí vousy anebo snímek má vyšší jas, a obličej není rozeznán, protože pak není připuštěn ani k samotnému rozeznávání obličeje a výsledek skončí neúspěchem.

Algoritmy zabývající se detekcí obličeje

Pro detekci se používají dva typy přístupů. První využívá ručně vyrobené filtry, které lokalizují tváře na snímku na základě hlubokých znalostí domény. Tento princip může být velmi rychlí a účinný, když se filtry shodují, ale mohou drasticky selhat při nevytvoření všech variant. Druhý přístup je pomocí vnímaní snímku jako celku, kde se učí, jak automaticky

lokalizovat a extrahovat obličeje. K tomuto principu zapadají neuronové sítě. Konkrétní případy jsou *Cascaded Convolutional Networks* a *RetinaFace*.

Multi-task Cascaded Convolutional Networks

Pro detekci obličejů existuje model *Multi-task Cascaded Convolutional Networks*, který se skládá ze tří kaskádovitě spojených konvolučních neuronových sítí. Detekce obličejů je jedna část, kterou tento model zvládá, protože dokáže najít umístění obličeje a také obličejové znaky. Tento model je ze zdroje [40].

Vylepšení tohoto modelu je v jeho výkonosti a přesnosti, protože detekce je rozložená do tří částí, které na sebe navazují konvoluční sítě. První část má za účel najít možné kandidáty na obličej v celém snímku. Použitá konvoluční síť mělká a rychlá. Tato síť používá jako ztrátovou funkci *Cross-entropy loss*. Druhá část je více komplexní a z kandidátů vybere obličeje, které projdou hodnocením silnější konvoluční sítě, a poté jsou umístěné v rámečku. Poslední část už jen detekuje obličejové znaky a vylepší výsledek vycentrováním. Použitá konvoluční síť u tomto případě je už větší a pomalejší. Poslední dvě konvoluční sítě používají ztrátovou funkci *Euclidean loss*, která je v rovnici 3.1, kde $\hat{y}_i$ je regresní cíl získaný ze sítě a y_i je pevně daný koordinát s obsahem buď vlastností rámečku obličeje nebo charakteristické znaky obličeje.

$$L_i = ||\hat{y}_i - y_i||_2^2 \tag{3.1}$$

Tento model byl porovnáván mezi sedm dalších modelů *RCPR*, *TSPM*, *Luxand face SDK*, *ESR*, *CDM*, *SDM* a *TCDCN* na datové sadě *WIDER FACE easy* [22], kde dopadl nejlépe s 81 % přesností.

RetinaFace

RetinaFace je pokročilejší model, který je vnitřně složen ze tří paralelních procesů, jaké na sebe dohlížejí a společně se ovlivňují. Tyto procesy poté mají na starost najít místo s obličejem a vypočítat jeho skóre společně s nalezením jeho oblasti. Další má na starost rozeznat obličejové znaky a poslední porovnaná 3D obličej, který je mapovaný na 2D snímek, s nalezení obličejem. Její základní struktura je tvořena z hlubokých konvolučních neuronových sítí, kde se vyskytovaly i reziduální bloky [14].

Tento model používá speciální více úlohovou ztrátovou funkci, která vypadá takto 3.2.

$$L = L_{cls}(p_i, p_i^*) + \lambda_1 p_i^* L_{box}(t_i, t_i^*) + \lambda_2 p_i^* L_{pts}(l_i, l_i^*) + \lambda_3 p_i^* L_{pixel} \tag{3.2}$$

Výpočet je rozdělen do čtyř celků. První se stará o rozpoznání neboli detekci obličeje. Je daná výrazem $L_{cls}(p_i, p_i^*)$, kde p_i je pravděpodobnost, že v místě i je obličej, a p_i^* je hodnota, která představuje nulu, nebo jedničku podle toho, zda to je, nebo není obličej. L_{cls} poté představuje ztrátovou funkci *softmax loss for binary classes*.

Druhá část je na predikci vlastností oblasti, kde t_i jsou předpovídané koordynáty odsazení x s y a dál je šířka a výška. t_i^* jsou poté vlastní pevné koordynáty. Jako ztrátová funkce L_{box} je použita *smooth-*L_1 [15].

Třetí část $L_{pts}(l_i, l_i^*)$ slouží pro rozeznávání obličejových rysů v pěti znacích, kde l_i jsou předpovídané znaky a l_i^* jsou skutečné znaky. Ztrátová funkce tomto případě je podobná jako u predikce rámečků.

Jako balanční proměnné jsou nastaveny $\lambda_1 - \lambda_3$ na hodnoty 0,25, 0,1 a 0,01, což znamená zvýšení významu prvních celků kvůli přesnosti a důležitosti, protože pokud nalézaní obličejových rysů něco najde a vyšší část, která hledá obličeje jako celek, neshledá, že to je obličej, tak má vyšší důležitost vyšší část.

Poslední je L_{pixel} se jménem *Dense Regression Loss* pro porovnání vybraného obličeje s obličejem, který byl vytvořen ve 3D a zachycen na 2D plátno s podobnými podmínkami jako reálný porovnávaný obličej. Tento proces je popsán rovnicí 3.3

$$L_{pixel} = \frac{1}{W * H} \sum_{i}^{W} \sum_{j}^{H} ||\mathcal{R}(\mathcal{D}_{P_{ST}}, P_{cam}, P_{ill})_{i,j} - I^*_{i,j}||_1 \tag{3.3}$$

Tato rovnice obsahuje W a H, které jsou šířka a výška k příslušnému místu obličeje $I^*_{i,j}$. Písmeno $\mathcal{R}$ znační *render* respektive vyobrazení 3D obrázku na 2D plochu. Dále $\mathcal{D}_{P_{ST}}$ nastavuje předvybraný 3D pohled na před-připravený obličej, P_{cam} jsou parametry pro pozici kamery a P_{ill} přestavuje parametry pro osvícení snímku. Model byl trénován na upravených snímcích z *CelebA* [22] a *WIDER FACE* [38].

Tento model byl testován na mnoha datových sadách a jedna nich byla *WIDER FACE easy*[22], kde dopadl s 96 % přesností. Pokud se zapojil do celku rozpoznávání obličeje, tak s modelu *ArcFace* dosáhli lepšího výsledku než s *Multi-task Cascaded Convolutional Networks*. Hodnotili se datové sady *LFW*, *CFP-FP* a *AgeDB-30*, kde z předchozího hodnocení 99,83 %, 98,37 % a 98,15 % s modelem *Multi-task Cascaded Convolutional Networks* udělala 99,86 %, 99,49 % a 98,60 %. Jak je vidět, tak *RetinaFace* zlepšila detekci i u snímků s horší kvalitou s předchozích tří datových sad.

3.3 Rozpoznávání obličeje a jejich algoritmy

Tento krok je velmi různorodý a lze ho přizpůsobit konkrétním potřebám některých problému. Obecně se můžeme popsat jako řízenou úlohu prediktivního modelování trénovanou na vzorcích se vstupy a výstupy, kde vstupy jsou vždy snímky, které je obsahují jeden obličej, a výstup je záleží na úloze, ale obecně to je označení nějaké třídy (číslo v kategorické třídě, atd.) anebo pravděpodobnosti či redigované číslo. V dnešní době to je výstup z neuronové sítě v podobě *feature vector*.

Úlohy, které by mělo zvládat jsou nalezení shodného obličeje, nalezení podobných obličejů a schopnost rekonstrukce, kde by mělo být schopné ze svých znalostí vygenerovat obličej z databáze svých zpracovaných obličejů [4], které byli převedené například na *feature vector*.

Rozeznáváni by mělo také mělo fungovat ve dvou režimech. První je režim ověření, kde je vstupní snímek porovnáván s jiným a má zajistit, že se obě identity na snímkách shodují. Druhý je identifikace, kde se vstupní snímek hledá svoji identitu v databázi obličejů.

Algoritmy zabývající se rozpoznáním obličeje

Modelů pro rozeznání obličeje je pestré množství a vznikali již od 90. let, kde byly sestrojeny jedeny z prvních modelů na holistickém přístupu [4]. Poté se kolem roku 2000 začali vyvíjet přístup k učení místních funkcí (metoda mělkého učení), které zlepšili výkon z přibližně 60 % na nějakých než 90 %. Později se vyvinuli systémů hlubokého učení pro rozpoznávání obličejů kolem let 2014 a 2015, které posunuly přesnost datových sad v průběhu let na 99 % [4].

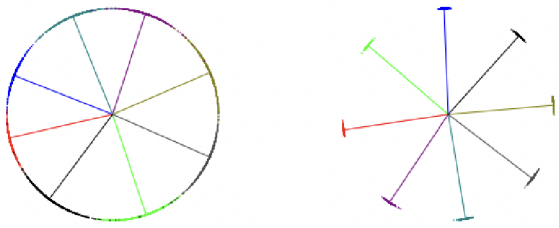
V této době se používají systémy hlubokého učení, které nejčastěji pracují na neuronových sítí. Tyto systémy budou představeny na malém vzorku modelů: *ArcFace*, *MagFace* a *QMagFace*.

ArcFace

ArcFace je model na rozeznání obličejů vytvořený na hlubokých konvolučních neutrálních sítí. Je založen nad myšlenou úhlového okraje (*angular margin*), který snadněji rozděluje třídy obličejových rysů. Další výrazný pokrok je v trénování, kde se docílilo použití mechanismu, který snadněji zpracovává snímky s horší kvalitou nebo nechtěným šumem a tím docílil ještě větší robustnosti. *ArcFace* se nejen používá k identifikaci či ověření, ale má funkčnost generování obličejů na aproximaci cílové identity pomocí gradientu sítě a *batch normalization* vrstev, které pomáhají s uloženými hodnotami pro normalizaci.

První myšlenka úhlového okraje, která zde byla propagována, byla vnesena do ztrátové funkce se jménem *Additive Angular Margin loss*. Cílem této funkce je rozřadit obličejové rysy do různých tříd a potom tyto naučené třídy posunovat dál od sebe v úhlovém prostoru. Pro snadnější porozumění si můžeme přestavit, že se jednotlivé vektory, které hodnotí stejnou třídu, se mapují na hypersféru (povrch koule) kolem určitého úhlu. Pro každou třídu je jeden úhel. A toto vektorové ohodnocení má určitý rozplyl, takže se můžou jednotlivé třídy okrajově překrývat. Do této doby byli vytvořeny modely na ohraničení prostoru, ale *ArcFace* přišel se způsobem, jak soustředit vektorové ohodnocení co nejblíže k úhlu třídy od rozhodovací hranice v lineární složitosti vůči jiné třídě. Výsledek je třídy s přesnými hranicemi a prostor mezi nimi, kam nespadají žádné vzorky. Tento vzor je vyobrazen na obrázku 3.2 a mechanismus je poté převeden do matematické rovnice 3.4, kde parametr s, který slouží pro škálovaní, dále θ_j popisuje úhel mezi třídou j a m je velikost mezery mezi třídami.

$$L_i = -\log\left(\frac{e^{s\cos(\theta_{y_i}+m)}}{e^{s\cos(\theta_{y_i}+m)} + \sum_{j=1, j\neq y_i}^{N} e^{s\cos(\theta_j)}}\right) \tag{3.4}$$



(a) Norm-Softmax (b) ArcFace

Obrázek 3.2: Vyobrazení, jak *ArcFace* posouvá vektory prvků různých tříd dál od rozhodovací hranice [13].

Další funkcionalitu, která zavedená v *ArcFace*, je technika učení pod-center, které usnadňují snadnější učení a třídění snímků od velkého šumu z reálného světa. Obecně se chová *ArcFace* jako *Triplet loss*, která je snaží přiblížit z pozitivním vzorků a oddálit od negativních, ale u *ArcFace* se snaží přiblížit k jednou z pod-centrů, které reprezentují jednotlivé

třídy vzorků se společnými znaky. U negativních je to na stejném principu, ale vzdálenost neporovnávají pouze s pod-centrem reprezentující jednu třídu, ale se všemi ostatními třídami. Tímto se přidá vliv ostatních tříd na výpočet vzdálenosti a třídy se stanou nezávislé mezi jakými třídami se porovnávají.

Další výhodou je, že pod-centra se řadí do dvou skupin: dominantní a nedominantní. Při trénování se model učí na různých vzorcích, které zapadnou k některým pod-centrům příslušící třídě. Zde se začnou projevovat typy pod-center. Pokud je vzorek nějak zašuměný, tak se přiřadí k nedominantnímu pod-centru pomocí vzdálenosti od nejbližšího centra, a poté má menší vliv u hodnocení. Pokud je kvalitní, tak je přiřadí do dominantního pod-centra, který má největší vliv na hodnocení. Dominantní pod-centrum je jen jedno v pod-třídě, kterou v tom případě reprezentuje, a shromažďuje naprostou většinu čistých snímků, kterých je větší množství a tudíž má potom i větší vliv. Toto rozdělení poté pomáhá i se zátěží, kde všechny vzorky nejdou přes jednu třídu, ale přes pod-centra, kde vzorky nejvíce budou přecházet přes ty dominantní a zátěž nebude centralizovaná.

Tento model byl vyzkoušen na několika různých datových sadách jako *LFW* [18] nebo *CFP-FP* [31], kde prošel s nejlepším hodnocením 99,5 %. Tímto lze celkově shrnout, že *ArcFace* skvěle používá svoje vyhodnocení s vylepšenou odolností vůči variacím. Na datových sadách bylo použito velké množství snímků a tím lze říct, že je dobře funguje v úlohách identifikace obličeje ve velkém měřítku, kde je počet identit obrovský.

Zdroj této sekce je [13].

MagFace

MagFace je jeden z rozšiřujících modelů modelu *ArcFace*, který se zaměřuje na vnesení kvality vzorku do učení modelu. Hlavním cílem je učení na principu, že když vezmeme tři stejné snímky s rozdílnou kvalitou, tak se model více naučí z toho kvalitního než z toho horšího. Respektive se zhoršující kvalitou se bude zvětšovat vzdálenost od středu třídního rozdělení tak, aby neporušila výhody předchozího modelu. Druhým cílem je minimalizovat cenu změny na již předchozím modelu na zajištění větší přesnosti.

Řešení nabízí tento model v podobě zvýšení nebo snížení délky počátečního vektoru reprezentujícího vlastnost okraje. Tato změna se poté projeví do ztrátové funkce *MagFace* 3.5, kde originál rovnice *ArcFace* je na 3.4.

$$L_i = -\log \frac{e^{s\cos(\theta_{y_i}+m(a_i))}}{e^{s\cos(\theta_{y_i}+m(a_i))} + \sum_{j=1,j\neq y_i} e^{s\cos(\theta_j)}} + \lambda_g g(a_i) \tag{3.5}$$

Rovnice popisuje parametr s, který slouží pro škálovaní, dále θ_j popisuje úhel mezi třídou j a f_i (vlastnost snímku). $m(a_i)$ je aditivní úhlový okraj kalkulovaný na kvantitě snímku a $g(a_i)$ je regulatizér, který oceňuje kvalitnější snímky. λ_g je hyper-parametr užívaní pro balancování klasifikační a regularizační ztráty.

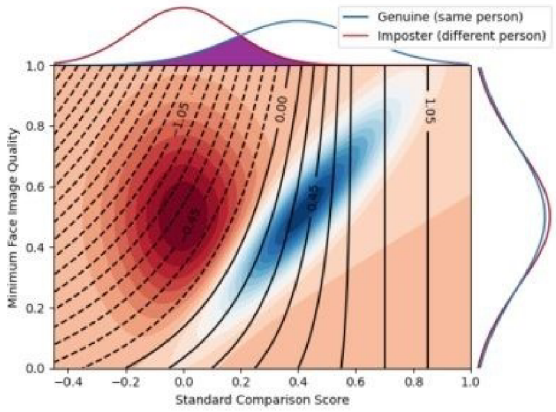
Toto vylepšení se promítlo do experimentů s datovými sadami LFW a CFP-FP, kde *MagFace* získal ohodnocení 99,83 % a 98,46 % průměrně lepší skóre jak *ArcFace* o 0,04 %. Utěžších datových sad jako IJB-B a IJB-C měl okolo 90 % a 95 % s jednou výjimkou a průměrně bylo skóre lepší kolem 1 %. Těmito výsledky můžeme vyvodit, že přidání kvality má pozitivní vliv a dopad na celý model.

Vytvořeno ze zdroje [24].

QMagFace

QMagFace pokračuje v rozvoji *MagFace*, kde přidává speciální funkci *Quality-aware comparison function*, která slouží pro zvýšení přesnosti rozhodování v zejména v případech, kdy se třídy se vzorky obličejů v hodnocení hodně překrývají, aniž by obsahovali stejné obličeje. Tento model se pak je poté pro obtížné situace jako porovnávání různě starých lidí, otočených nebo snímků z různou kvalitou.

Tato metoda přichází s *Quality-Aware Comparison Scoring* a to by se dalo popsat jako upravený způsob hodnocení rozdílů mezi dvěma osobami. Tento způsob je vyobrazen na obrázku 3.3, kde na osách je znázorněna dřívější způsob vyhodnocení a čárami v obrázku je znázorněna nová metoda s pomocí *Quality-Aware Comparison Scoring*. Různé třídy jsou reprezentována červenou a modrou barvou. Na tomto obrázku je také dobře vidět, jak se tyto třídy překrývají a jak je tato nová metoda pomáhá oddělit.



Obrázek 3.3: Vizualizace *Quality-Aware Comparison Scoring* [33].

Tato funkcionalita je dosažena funkcí 3.6, kde $s = cos(e_1, e_2)$ je skóre vyjádřené podobností dvou šablon, které jsou vytvořené *MagFace* ztrátovou funkcí se dvěma snímky a $q1, q2$ jsou znázornění kvalit těchto snímků. α a β jsou proměnné pro trénování.

$$s'(s, q_1, q_2) = min\{0, \beta * s - \alpha\} * minq_1, q_2 + s \tag{3.6}$$

S tímto modelem bylo experimentováno na datových sadách AgeDB, CFP-FP, a tak dál, kde se umístil první s 98,50 % a 98,74 % o 0,2 % před ostatními. Z těchto výsledků je dobře znázorněno vylepšená kvalita hodnocení modelu, ale naskytli se i limitující faktory. Tato metoda není moc účinná u nízko-kvalitních snímků a při práci s videm je potřeba agregace pro získání snímků s větší kvalitou.

Tato sekce je ze zdroje [33].

Kapitola 4

Návrh a implementace

Tato práce se zaměřuje na vyhodnocení podobnosti generovaných snímků v mém případě konkrétněji na tvorbu modelu na porovnaní podobnosti identit originálního snímku obličeje a vygenerovaného snímku, který byl vytvořen z originálního snímku úpravou jeho atributu. Cílem je robustní program, který na výstupu dává číslo o podobnost identit. Pro tento úkon je vytvořena siamská neuronová síť, která porovnává identity dvou vstupních snímků a na výstup podává číslo od nuly do jedné.

Obsah kapitoly je rozřazen do tří částí. První část 4.1 obsahuje návrh sítě, kde je popsána architektura sítě, požité prvky a předpokládaný výstup. Další část 4.2 je tvořena přípravou dat potřebných pro jeho učení, ověření jeho funkčnosti a následné testovaní. Poslední část 4.3 se skládá ze samotné implementace v jazyce *Python*, přehledem kódu a trénování modelu.

4.1 Návrh

Tato sekce se specializuje na návrh modelu a funkčnost aplikace. Jak jsem již zmínil, tak jako základ modelu jsem vybral siamskou síť, která je s její strukturou níže. Po tomto následuje návrh funkčnosti aplikace, která popisuje, jak se pracuje s aplikací v konzoly s popisem jeho argumentů, které lze použít.

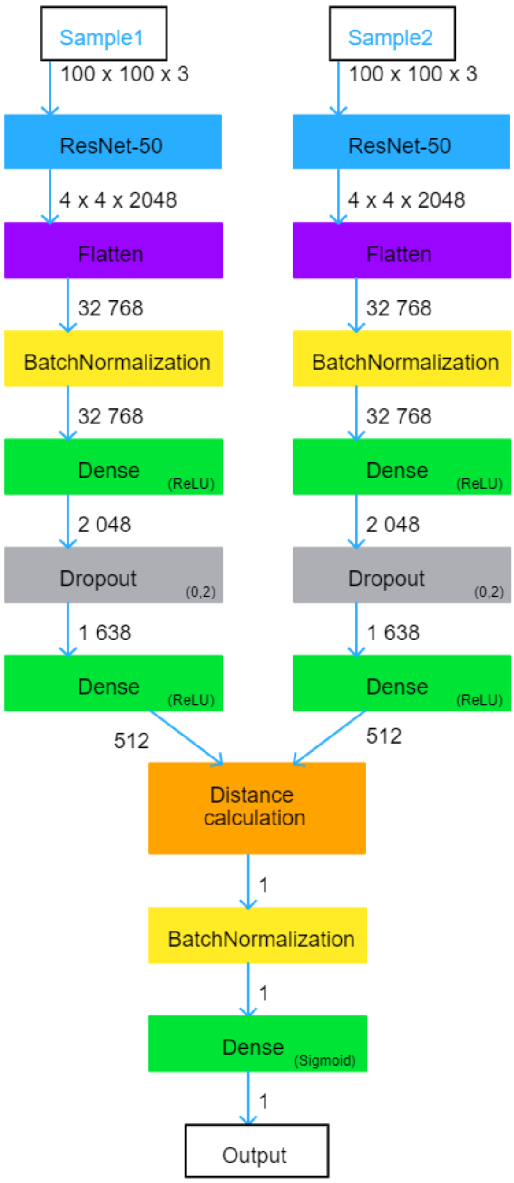
Siamská neuronová síť

Důvod, pro který jsem vybral siamskou sít je v jeho porovnání vstupů. Klasifikační neuronové sítě potřebují mít pevný počet tříd na to, aby je mohlo správně rozdělit na rozdíl od siamských neuronových sítí, kde se vypočítají pouze vektory s následným měřením vzdálenosti. Tímto výpočtem jsem schopen analyzovat na veliké množství různých snímků. Další jeho výhodou je jeho robustnost. Tato síť je výpočetně složitější na trénování, které proběhne pouze jednou, ale poté poskytuje rozpoznaní snímků o malé kvantitě s velkou přesností [3]. Poslední je kvůli použití dvou stejných podsítí, tak je možnost si uložit, výstupní vektor místo celého snímku pro rozpoznáváni. V tomto duchu jsem vybral i ztrátovou funkci *Contrastive loss*. Tato funkce používá pouze dva snímky, a proto konečná architektura je snadnější na výpočet a lehčí na ovládání.

Struktura modelu

Strukturu jsem rozložil do dvou částí. První část je o tom, jak vypadají podsítě na extrakci vektorů a druhá je o výpočtu odlišnosti. Celková struktura je zachycena na obrázku

4.1, kde jsou jednotlivé vrstvy zobrazeny barevně a šipky znázorňující přenesení vektorů a rozdělení částí modelu.



Obrázek 4.1: Obrázek modelu.

Podsíť na extrakci vektorů je ve formě konvoluční neuronové sítě. Ze začátku při příjmu snímku se pro extrakci prvně připraví data než vstoupí do sítě. Tato předpříprava se provede převedení obrazu z RGB do BGR a posune čísla znázorňující barvy tak, aby byli při průměru rovné nule. Tento krok pomůže k rozeznávání v konvolučích vrstvách.

Při vstupu do modelu první vrstva je složena z modelu *ResNet-50* pro přípravu snímku skrze konvoluční vrstvy. Tento model je blíže popsán v sekci 2.1.

Další vrstva je *Flatten*, která všechny zpracované snímky z konvolučních vrstev v podobě trojrozměrného pole převede do jednorozměrného pole o délce násobků tří délek předchozího pole. Tímto umožníme zpracování dalšími vrstvami.

Hodnoty přivedené z předchozí vrstvy jsou už seřazené v jednom poli, ale stále jsou nijak neupravena od konvoluce a mohou skrze ní procházet extrémní hodnoty. Na vyřešení tohoto problému je potřeby vrstva *BatchNormalization*. Tato vrstva normalizuje hodnoty a transformuje je blízko k nule se směrodatnou odchylkou přibližně okolo jedné. Tato vrstva, ale má jednu nevýhodu v tom, že se nechová stejně při učení a poté při predikci, protože používá jiný výpočet. Tento výpočet je ovlivněn průměrováním hodnot, kde při učení se používán průměr ze skupinky snímků (*batch*), na kterých je učena zároveň, ale při predikci používá pohyblivý průměr z vstupních hodnot a z naučených při učení. Tento princip je umožňuje ustálit hodnoty, pokud přijde vysoký nebo nízký extrém z předchozí vrstvy.

Následují vrstvy jsou dvě plně propojené a jedna odpadávací. Plně propojené vrstvy mají jako aktivační funkci ReLU. Tyto vrstvy se starají se o výpočet vektorů pomocí naučení z trénovací sady nastavováním vah a prahů. Odpadávací vrstva slouží k zabránění *overfitting* tím, že vypustí některé perceptrony z hodnocení v průběhu učení.

Tímto se dostáváme do druhé části, kde se počítá odlišnost pomocí vzdálenosti. Hlavní vrstva této části je výpočet Euklidovské vzdálenosti mezi vektory z podsítí. Vzdálenost je počítána výrazem 4.1, kde x a y jsou pole vektorů podsítí, mezi kterými se počítá vzdálenost rozhodující o podobnosti.

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{4.1}$$

Další vrstvou je *BatchNormalization*, která se stará o transformaci vzdáleností u podobných snímků, aby se přibližovali k nule a u odlišných snímků, aby se přibližovali k jedničce.

Pro snadnější nalezení prahu, která rozhoduje jako dělič mezi podobnými a odlišnými snímky, je přidá ještě jedna vrstva. Vrstva je jedením perceptron s aktivační funkcí *Sigmoid*, který hledá rozhodující práh a odděluje hodnoty více od sebe. Tato vrstva také nastavuje výstup na interval od nuly do jedné, znázorňující pravděpodobnost, jak výsledek dopadl.

Předpokládaná výstup by měl být po těchto operacích okolo nuly, pokud se snímky podobají, a pokud ne, tak se měli pohybovat okolo jedné, ale i přes všechny tyto operace je možné se budou některé vzorky pohybovat okolo poloviny vzdálenosti mezi nulou a jedničkou a bude obtížné je oddělit, kvůli variabilitě vstupních snímku. Tímto je možné zjistit, zda jsou odlišné nebo ne, a na výstupu programu je znázornit. Ve výsledku výstup modelu je pravdivostní jednotka, které bude ovlivněna vstupními snímky.

Funkcionalita programu

Tento program se jmenuje `Gen_Verifier` a je to aplikace do příkazového řádku, který má čtyři argumenty, kde jeden je možné rozdělit do tří možností. Jeho zápis vypadá takto:

```
Gen_Verifier file_path_A file_path_B [Options] [file_path_result]

Optinons:
    [-v|--version]
    [-h|--help]
    [-dr]
```

První i druhý argument je na zadání cesty k snímku nebo složky se snímky na porovnávání, třetí je o možnostech zadání výsledku s verzí a nápovědou. Poslední je na možné uložení výsledku do souboru.

Aplikace se chová ve třech módech podle toho, co uživatel zadá za první dva argumenty `file_path_A` a `file_path_B`. První je porovnávání snímku se snímkem, kde je porovná a vypíše výsledek. Druhý mód je porovnávání snímku se složkou. V tomto případě program prolistuje složku a porovná každý snímek se snímkem v argumentu. Z těchto dat udělá průměr a vypíše ho na výstup. U posledního módu je porovnávaná složka se s složkou. V tomto případě program vezme za sebou abecedně jdoucí snímky v a porovná je mezi sebou, výsledek zprůměruje a vypíše.

Argument `-v` anebo `--version` spustí vypsání verze na standardní výstup. Pokud jsou i jiné argumenty, jak jsou ignorovány a tento se zpracuje s předností.

Argumenty `-h` anebo `--help` dělají podobnou věc jako předchozí argument, ale tento při obsahu v argumentech zobrazuje nápovědu a v prioritách je až po argumentu s verzí.

Pátý argument `-dr` modifikuje podrobnost výpisu. Pokud není obsažený, tak bude vypisovat strohé informace jak porovnaní dopadlo. Porovnání snímku se snímkem jednu hodnotu a u ostatních tedy v módě dva a tři průměr. Pokud je obsažený, tak u módů dva a tři vypíše podrobnější výsledky a tím, že vypíše v ještě hodnoty snímku proti snímku před průměrováním.

Poslední argument je `file_path_result`. Tento argument slouží ke zjištění, kam se má výstup vypsat. Pokud není obsažen v argumentech, tak se všechny informace zapisují na standardní výstup. Pokud je zapsaný v argumentech, tak se výsledek zapíše do souboru na nakonec a oddělí se novým řádkem. Soubor musí mít koncovku `.txt`. Pokud soubor není, tak ho vytvoří a pokud cesta ukazuje na složku tak se program ukončí se speciálním návratovým kódem a vypíše to hlášku do na standardní chybový výstup.

4.2 Příprava dat

Pro trénovaní, porovnávání a testování jsou nutné mít datové sady odpovídající požadavků pro modely nebo funkce, kterou budou zastávat. Konkrétně je potřeba datová sada pro nyní navrhovaný model pro zajištění jeho funkcionality. Dále je potřeba zajistit datový model s generovanými snímky a jejich originály. Pro tento účel je potřeba získat model pro generovaní syntetických snímků a datovanou sadu, kterou se musí naučit. Tato datová sada musí být dostatečně velká, aby se zní model dokázal naučit a aby zbyly nám snímky na další zpracování v porovnávání. Dohromady to tvoří dvě datové sady, kde jedna musí být částečně vytvořena přes model *StarGAN*.

První datová sada je cílená na právě vyvíjený model pro učení a verifikaci. K tomuto je potřeba mít snímky se záběrem na obličej a u každého snímku znát jeho identitu. Tyto parametry splňuje datová sada *LFW - People*, která je popsána v sekci 2.4. Příklady z této datové sady jsou zobrazeny na obrázcích 4.2. Pro učení modelu se vytřídí snímky osobností, které jsou jen po jednom, aby datová sada vyla rovnoměrnější. Následně se vytvoří náhodné dvojice dvou stejných a dvou rozdílných identit na snímku ve stejném zastoupení. Jelikož jsou vyřazeny složky osob pouze s jedním snímkem, tak obě skupiny budou mít stejné zastoupení a jedna nebo druhá skupina nebude rozměnitelnější a budou obě rovnocenné. Vytvářet náhodné dvojice můžeme, protože vybrané složky obsahují přes 9 100 těchto snímků a možnost, že se vybere stejnou dvojici v jedné várce na učení je velice malá.

Druhá datová sada snímků je potřeby u generování syntetických snímků. Od této datové sady je potřeba, aby byla dostatečně velká a obsahovala atributy snímků, protože se s ní bude učit model generátor *Stargan* ze zdroje [11] a druhá část je použita jeho výsledné porovnání. Pro tento účel je použitá sada *CelebA*, která je popsána v sekci 2.4. Tato sada je rozdělena na tři části. První dvě třetiny jsou použité pro trénovaní generátoru [11]

Obrázek 4.2: Příklady z datová sady obrázků *LFW - People* [18].

a poslední třetina je na porovnání do právě vyvíjeného modelu tak, že vybere snímek, ten se zpracuje generátorem [11] a vznikne nový syntetický snímek. Tyto dva snímky se poté označí, že jsou pár, a vytvoří se z nich nová datová sada snímků zpracovaná pro porovnání v počtu 2 000 kusů. Příklady originálních a generovaných snímků jsou na obrázku 4.3 a 4.4, kde první má změnu pohlaví, druhý obrázek má změnu barvy vlasů a třetí stáří.

Obrázek 4.3: Příklad originálních snímků z snímků *CelebA*.

4.3 Implementace

Tato sekce je zpracována na popis navrhovaného modelu, který je zpracovaný v jazyce *Python*. Dále je práce s potřebnými knihovnami. Jednotlivé třídy či metody jsou podrobně popsány s ukázkou kódu s následným trénováním.

Obrázek 4.4: Příklad generovaných syntetických snímků.

Python

Python je vysokoúrovňový skriptovací programovací jazyk, který je dynamický typovaný s *Garbage collection* [27]. Tento jazyk také podporuje více programovacích paradigmat jako je objektově orientovávané programovaní na funkcionální programování. Jelikož je tento jazyk je skriptovací, tak je i interpretovaný a to mu pomáhá mít velký rozsah mezi velké množství platforem, ale za cenu snížení jeho výkonu. Avšak některé funkce se výrazně urychlili pomocí optimalizací, které ale někdy přidají čas strávený při instalacích. Dynamické typování přináší poměrnou volnost při implementacích, ale chovají se jako dvousečný meč, protože statická analýza nemůže odhalit tolik chyb a zjistí se až po spuštění, a také protože při vyšších abstrakcích nemusí programátor vědět jaké jsou typy parametrů nebo návratových hodnot při nedokonalé dokumentaci od jiných programátorů, i když v posledních verzích přidali možnost slabého typování u argumentů, návratových hodnot a proměnných. Tyto vlastnosti však pomáhají k rychlému a snadnému vývoji obzvláště na prototypech nebo malých projektů, a proto se stal jedním z nejoblíbenějších programovacích jazyků.

Důvod vybrání *Pythonu* nebylo tolik složité. *Python* obsahuje jednu z nejrozšířenějších základen pro neuronové sítě, ale hned za jím je R a Java. Je to z důvodu, že *Python* je více abstrahovaný, snadnější a lehčí. Obsahuje knihovny jako TensorFlow a Keras, které neskutečně usnadní tvorbu a trénování modelů. Za to silněji typované jazyky jako C++ se k velké složitosti moc nepoužívají, ale mají i své místo ve více kritických částech jako jsou jádra umělé inteligence a tak dál.

Potřebné knihovny

Velkou část kódu jsou knihovní volání, které abstrahují nějakou činnost. Tato činnost je vetšinou složitý problém nebo se často opakuje, tím pádem je dobré tento problém vyřešit, optimalizovat ho a zpřístupnit ho ostatním, aby se nemuselo řešit stejný problém stokrát a udělat v něm o to více chyb. Bohužel, ale vždy se časem nejde nějaká chyba, která je nutná opravit. Proto se spraví a přidá do aktualizace, kde se promítne tam, kde je použita. Ve výsledku se tato činnost urychlí a, když se přepočte na čas, který byla vyvíjena oproti, kdyby to měl každý dělat znovu, tak se to mnohem násobně vydělá. Tento princip programování je tím pádem hodně efektivní, neboť zmenšuje riziko chyb, opakovaní kódu a šetření hodně času.

První z jich je knihovna `NumPy`[1]. Tato knihovna přidává možnost pracovat s více rozměrnými poli, vektory a i maticemi. Dále nabízí široké množství matematických funkcí. Je výborné zpracovaná na práci se statistikou a správou větších dat. Případ tohoto programu to je prakticky všude v modelu.

[1] https://numpy.org/

Další knihovnou je `OpenCV` [2], která nabízí ohromné množství funkcí pro zpracovaní snímků a videí. Jako příklad jeho metod jsou funkce na zvětšení a zmenšení snímků, na vkládání obrazců do snímku a nebo vložení okrajů.

`Pandas`[3] je knihovna napsaná pro manipulaci a analýzu dat. Zejména nabízí datové struktury a operace pro práci s číselnými tabulkami a časovými řadami. V případě tohoto projektu s ní byla tvořena většina grafů.

Knihovna `FaceAIKit`[4] je zaměřená na detekci a rozpoznávání obličejů. Tato knihovna používá model *Retina* pro detekci a modely s využitím *ArcFace* a *MagFace* pro rozpoznávání. Její tvorba byla uskutečněna vedoucím práce Ing. Tomášem Goldmannem.

`Scikit-learn`[5] je knihovna pro strojové učení. Zabývá se klasifikačními, regresními a shlukovými algoritmy včetně dělení datových sad na trénovací a validaci, jako je použito v tomto programu.

`Keras`[6] je knihovna, která poskytuje rozhraní *Pythonu* pro tvorbu umělích neuronových sítí. `Keras` funguje jako rozhraní pro knihovnu `TensorFlow`, která se jimi zabírá podrobněji.

Knihovna `TensorFlow`[7] je určena pro strojové učení a umělou inteligenci. Může být použita v celé řadě úkolů, ale zaměřuje se zejména na trénink a použití hlubokých neuronových sítí.

Dále jsou použité další knihovny jako je například Různá rozhraní operačního systému (`os`) nebo Systémově specifické parametry a funkce (`sys`), ale ty jsou už integrované v *Pythonu* samotném a není možné je kompletně popsat kvůli jejich rozmanitosti, tak jsou zde pouze zmíněny.

Struktura programu

Tato sekce slouží k představení struktury a rozřazení funkcionality programu, především představení tříd a metod, kde popsáno, jak jde datový tok a kde se můžou naskytnouti problémy. Také je pod každou metodou vypsán kód, jak vypadá výchozí volání a společně s argumenty.

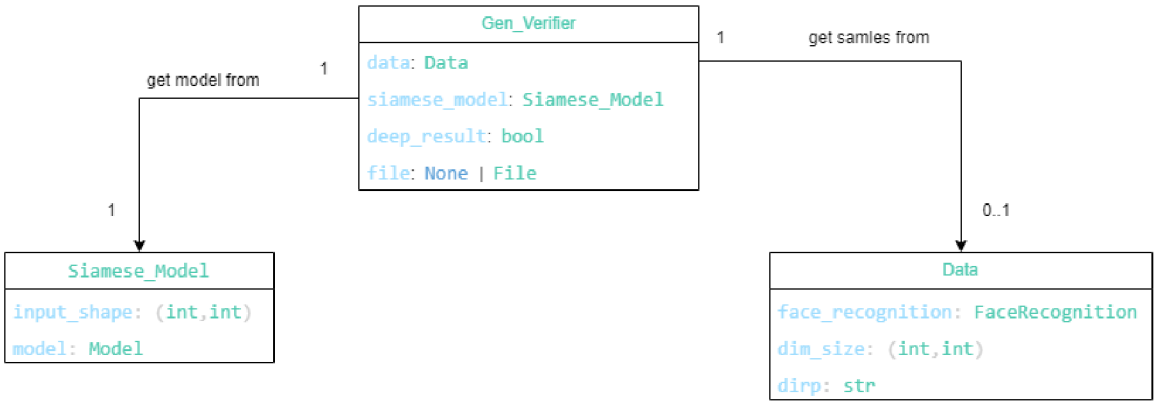
Program je roztěžen do tří tříd `Gen_Verifier`, `Data` a `Siamese_Model`, přičemž každá z těchto tříd má svoji úlohu. Tyto třídy mají svoji hierarchii, která umožňuje abstrahovat různé funkce v mnohem jednoduší formě než jsou v původním funkci. Hlavní třída v tomto programu je `Gen_Verifier` a zbylé dvě jedna abstrahuje model a druhá přístup k datové sadě a vstupních snímků. Jsou zobrazené na diagramu 4.5.

Třída Data

Třída `Data` se zaměřuje na zpracování snímků a načtení vzorků z na datové sady. Objekt této třídy obsahuje tři proměnné a pět metod.

První proměnná je `face_recognition`. Tato proměnná obsahuje model pro detekci snímku z knihovny `FaceAIKit`, která je použita dále v metodách. Je uložena takto v proměnné, protože je často používána a její inicializace trvá okolo 20 sekund na počítači, který není moc hardwarově vybavený, a když by se měla inicializovat v každém spuštění metody na detekci obličeje u 1 000 snímků, tak by to trvalo bez optimalizací okolo 6 hodin a to je

[2] https://opencv.org/
[3] https://pandas.pydata.org/
[4] https://pypi.org/project/face-ai-kit/
[5] https://scikit-learn.org/stable/
[6] https://keras.io/
[7] https://www.tensorflow.org/



Obrázek 4.5: Diagram tříd.

započítaná pouze inicializace nikoli vykonání kódu na detekci, nalezení a vystřihnutí obličeje ze snímku. Tato proměnná je inicializována v konstruktoru a je nastavena na model *ArcFace*, protože později je ukázáno, že budou porovnávány snímky stejného rozlišení a není nutnost je třídit.

Druhá proměnná je `dim_size`, do které je uloženo konečné rozměry snímku obličeje. Je inicializována v konstruktoru a výchozí nastavení je na typ *tuple* a na hodnotu `(100, 100)` rozměrů obličeje, ale je možné ho případně změnit.

Poslední proměnná je na uložení cesty k datové sadě pro metody na získání vzorků se jmenném `dirp`. Její výchozí hodnota je `'img/LFW - People/lfw_funneled'`.

První metoda je konstruktor, který nastavuje tři předchozí proměnné. Jeho dva argumenty jsou `dim_size` a `dirp`, pokud nepočítáme `self` jako odkaz na objekt. `dim_size` slouží pro možnost nastavení druhé objektové proměnné, které jsou rozměry snímku obličeje. `dirp` nastaví cestu do složky, kde je uložená datová sada *LFW* s vychází hodnotou `'img/LFW - People/lfw_funneled'`. Je zapsána tímto kódem.

```
__init__(self,dirp = 'img/LFW - People/lfw_funneled')
```

Další metodou je `read_image`. Tato metoda slouží k přečtení snímku s jeho formátováním. Obsahuje argumenty `filename` a `face_num`, kde `filename` slouží jako cesta ke snímku a `face_num` slouží pro výběr správného obličeje ze snímku, pokud jich je na snímku více. Funkce této metody je na zpracování snímku z udané cesty a připravení na model. Pro tuto funkci se snímek přečte z cesty a projde detekcí obličeje. Pokud snímkem obsahuje více jak jeden obličej, tak si vybere požadovaný obličej. Následuje kontrola, zda má nějaké obličeje a jaké kvality jsou. Pokud nesplní nějako vlastnost, tak se metoda vrátí s hodnotou `None`. Dále se transformuje na požadovanou velikost, předzpracuje snímek pro model a v návratové hodnotě vrátí snímek. Pokud nějaká metoda selže, z důvodu špatného vstupu, tak se opět vrátí s `None`.

```
read_image(self,filename,face_num = 0)
```

Následuje metoda pro získání vzorků z datové sady. tato metoda se jmenuje `get_data`. Obsahuje argument `total_sample_size`, který nastavují množství vzorků. Částečná funkcionalita je popsána v sekci Příprava dat 2.4, kde je z datové sady náhodně vybraná dvojice o určitém množstvím a se stejným zastoupením snímků se stejnou osobou a snímků s rozdílnou osobou v poli za sebou. Tyto vzorky uložené v několika dimenzionálním poli jsou poté

vráceny jako návratová hodnota společně s polem o výsledcích jejich porovnávání. Pokud se nepodaří získat takovou sadu vzorků, tak se vrátí `None`.

```
get_data(self, total_sample_size)
```

`Split_data` je metoda, která vytvoří náhodně zamíchanou vzorkovou sadu o množství `total_sample_size` vzorcích a rozdělí je v poměru `split` pro testování. Výchozí nastavení je, že jedna čtvrtina slouží pro testování. Navrací čtyři hodnoty `x_train`, `x_test`, `y_train` a `y_test`, kde `x_train` s `x_test` jsou vzorky pro model a `y_train` s `y_test` jsou výsledky, které mají být dosaženy.

```
split_data(self, total_sample_size, slit =.2)
```

Poslední metoda je `get_data_from_dir`, která načte všechny snímky ve složce `dirpath` a následně je vloží do návratové hodnoty.

```
get_data_from_dir(self, dirpath)
```

Třída Siamese model

Druhá třída je `Siamese_Model`, která obsluhuje a abstrahuje model siamské sítě. Tento model tudíž slouží k tvorbě a nahrávání vah modelu. Dál slouží k trénování, vyhodnocování a předpovídání hodnot. Objekt této třídy má dvě proměnné a osm metod.

První z atributů této třídy je `input_shape`. Do tohoto parametru je uložena vlastnost, jak má vypadat vstup do modelu a jaké rozměry má mít. Výchozy nastavení je na `(100,100)`.

Druhý atribut je `model` typu `Model`, který je základní jednotkou celé třídy. Třída `Model` je z knihovny `Keras`, kde která je naložená na neuronových sítích, a tudíž třída `Siamese_Model` je pouze zapouzdření nad třídou `Model`.

První metodou je opět konstruktor, který nastavuje `input_shape` přes argument se stejným jménem, jaký je nastavený na výchozí hodnotu, a vytvoří i s kompilací model siamské sítě. Pokud atribut `input_shape` je nastaven na výchozí hodnotu, tak je možnost nastavení argumentu `load` a třída automaticky načte váhy a prahy do modelu. Pokud se ale rozměry vstupu v argumentu změní, tak tato možnost už nelze využít.

```
__init__(self, input_shape = (100,100),load = True)
```

Druhá metoda je `predict_once`, která ze dvou snímků zjistí číslo, které předpoví, zda si jsou osoby na nich podobné. Snímky se dosazují do argumentů `sample1` a `sample2`. Tato funkce vrací číslo od nuly do jedné, kde jednička znamená stejnou osobu a nula představuje rozdílnou osobu.

```
predict_once(self,sample1, sample2)
```

Metoda `evaluate` slouží k vyhodnocení přesnosti sítě a vypočítání ztrátové funkce. Její argumenty jsou `x_test` a `y_test`. Kde `x_test` pěti rozměrné pole se vzorky a `y_test` je jedno rozměrné pole s výsledky. Návratová hodnota je ze dvou čísel první je hodnota ztrátové funkce a druhá je hodnota přesnosti, ale je nutné podotknout, že přesnost je podle výsledků je převracená. Podrobnější vysvětlení je v následující sekci Učení modelu 4.3.

```
evaluate(self,x_test,y_test)
```

Metoda `fit` je na trénování modelu na vzorcích `x_train`, `x_test`, `y_train` a `y_test`, kde `x_train` s `y_train` jsou vzorky na trénování modelu a `x_test` s `y_test` jsou na ověření přesnosti. Poslední argument `epochs` znamená kolikrát se má učit na těchto datech.

`fit(self,x_train, x_test, y_train, y_test, epochs = 10)`

Metody `load_weights` a `save_weights` slouží k práci s ukládáním nahráváním vah modelu na disk.

```
load_weights(self,path)
save_weights(self,path)
```

Funkce `euclidean_distance` je statická metoda pro výpočet vzdálenost několika vektorů a argument `vects` je pole dvou vektorů, které přijdou na porovnání. Vzorec je možné najít v sekci Návrh modelu 4.1 s rovnicí 4.1.

```
euclidean_distance(vects)
```

Funkce `contrastive_loss` je statická metoda, která slouží jako ztrátová funkce podle stejného stejného jména pro správnou funkci siamských sítí.

```
contrastive_loss(y_true, y_pred)
```

Poslední tři metody `subnetwork`, `set_up_subnetworks` a `get_siamese_model` se soustředí kolem implementace modelu, která je teoreticky popsáná v sekci Návrh modelu 4.1, ale části jako rozřazení do metod tam nejsou rozebrány. Na tvorbu podsítí je použita metoda `subnetwork`, která vytvoří konvoluční neuronové podsítě, jaké se nají použít v metodě `set_up_subnetworks` na spojení podsítí dohromady a zajištění jejich správné funkce sdílením vah. Poslední metoda `get_siamese_model` přidá nakonec druhou část modelu jako euklidovskou vzdálenost, vytvoří z celku model se vstupy a kompiluje ji se ztrátovou funkcí *contrastive loss* a optimalizátorem *RMSprop*. Jako velikost vstupu je použit atribut `input_shape`, který obsahuje výchozí hodnoty. Z tohoto je možné vypočítat, že z konvoluce přijde 32 768 perceptronů a tak je na to připraveno 2048 a 512 perceptronů v plně propojených vrstvách.

```
subnetwork(self)
set_up_subnetworks(self)
get_siamese_model(self)
```

Třída Gen Verifier

`Gen Verifier` je hlavní programu, která se stará o propojení tříd `Data` a `Siamese_Model` skrze jejich výměnu proměnných. Další její funkčností je zpracovávat veškerý vstup v argumentech programu, vypočítat výsledek a ten poté vypsat na stanovené místo. Tato třída obsahuje čtyři proměnné a třináct metod.

První dvě proměnné jsou `model` a `siamese_model`, které jsou vytvořeny v konstruktoru, a jsou to instance stejnojmenných tříd, jaké vykonávají funkce popsané v předešlých pod sekcích 4.3 a 4.3.

Druhé dvě proměnné jsou `deep_result` a `file`. Tyto proměnné jsou nastavovány podle argumentů programu. Proměnná `deep_result` je pravdivostním typem a zapisuje, jak má být podrobný výstup, kde výstup je popsán v podsekci Funkcionalita 4.1. Zato proměnná `file` je na zaznačení pro výstupu programu. Pokud je nastavená na `None`, tak výsledky se vypisují na standardní výstup, pokud ne, tak je zapsána na souboru, který je pod touto proměnnou uložen.

Konstruktor v této třídě má jeden argument, který nastavuje adresu složka pro datovou sadu od třídy `Data`, a dále nastavuje atributy objektu na výchozí hodnoty. Atributy `data`

s `siamese_model` jsou nastavené jako stejnojmenných třídy a atributy `deep_result` s `file` jsou nastaveny na `False` a `None`.

`__init__(self)`

Metoda `train_model` se stará o kompletní učení modelu. Má dva argumenty a to jsou `total_sample_size`, který nastavuje počet vzorků na kterých se model má učit, a `epochs`, jaký nastavuje počet kol v kolika se má učit. Výchozí nastavení je na deset kol.

```
train_model(self,total_sample_size, epochs = 10)
```

Tato metoda se jménem `evaluate_model`, poskytuje možnost zjistit hodnotu ztrátové funkce a přesnost modelu, kde přesnost modelu je převrácená. Dále poskytuje možnost argumentem `total_sample_size` kontrolovat množinu vzorků, na kterých to je testováno. Výchozí nastavení je na tisíci vzorcích.

```
evaluate_model(self,total_sample_size = 1000)
```

Čtvrtá metoda je `predict_once_model`, která ze dvou snímků získá číslo, které předpoví, zda si jsou osoby na nich podobné či nikoliv. Snímky se vkládají do argumentů `sample1` a `sample2`. Tato funkce vrací číslo od nuly do jedné, kde jednička znamená stejnou osobu a nula představuje rozdílnou osobu. Je to zapouzdření funkce `predict_once` ve třídy `Data`.

```
predict_once_model(self,sample1, sample2)
```

Metody `load_weights` a `save_weights` zapouzdří metody z `Siamese_Model` pro snadný a přehledný přístup.

```
load_weights(self,path)
save_weights(self,path)
```

Metoda se jménem `write_to` slouží pro výpis výsledků programu podle nastaveného umístění v atributu `file`. Pokud je `None`, tak ho napíše na standardní výstup.

```
write_to(self,string)
```

Funkce `process_argv` zpracuje argumenty programu a nastaví cílové umístění výstupu společně s možností podrobnějšího výstupu. Argumenty jsou validní, pokud se vrátí návratová hodnota nula jinak proběhla nějaká chyba při zpracování.

```
process_argv(self,argv)
```

Metody `process_file_to_file`, `process_file_to_dir` a `process_dir_to_dir` slouží pro zpracování snímků ze vstupu argumentů programu výstup podle módu v jakém jsou zaslány. Tyto módy jsou sepsány v podsekci Funkčnost. Obecně to zpracuje a vypíše výsledky podle předzpracovaných dat.

```
process_file_to_file(self,argv)
process_file_to_dir(self,argv)
process_dir_to_dir(self,argv)
```

Poslední metoda je `main`. Tato metoda slouží k spuštění modelu pro rozpoznávání osob z příkazové řádky a tudíž s prací celého programu. Jsou do ní zkombinovaná většina metod, které byly použity zde popsány. Nejsou v ní zkombinovány metody pro učení a validaci, protože to není hlavní účel této metody a programu.

Tok programu

Program je konstruován na spuštění dvou příkazů, aby proběhlo celé vyhodnocení podobnosti, zpracování argumentů a výpis výsledků, které je kódem v následujícím odstavci.

```
from Classes.Gen_Verifier import Gen_Verifier

verifier = Gen_Verifier()
verifier.main()
```

Prvně probíhá inicializace, která je popsána v předešlé podsekci Třída `Gen Verifier` u konstruktoru, a pak proběhá metoda `main`, která předá kontrolu nad programem metodě `process_argv`, která zpracuje argumenty. Tato funkce předá kontrolu opět metodě `main`, která podle typu prvních dvou argumentů spustí jeden z tří módů s příslušnou metodou `process_file_to_file`, `process_file_to_dir`, nebo `process_dir_to_dir`, které mají vestavěný rozeznávací model a výpis hlášek. Poté se vše ukončí s návratovým kódem 0 při úspěchu a jiné číslo při neúspěchu.

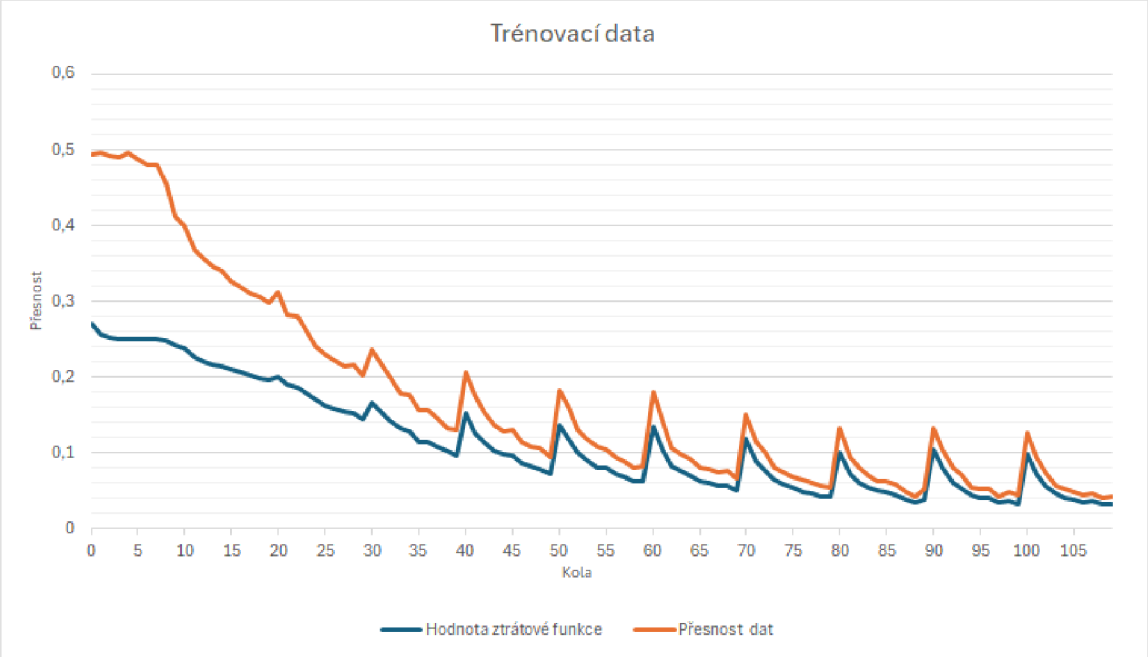
Učení modelu

Neuronové sítě jsou velmi pokročilý nástroj pro analýzu složitých dat jako jsou snímky. Tyto sítě se nastavují pomocí prahů a vah, které jsou systematicky tvořeny učením. Těchto vah a prahů je i v nejednodušších neuronových sítí několik stovek, což způsobuje jedinou možnost zlepšení přesnosti modelu pomocí učení. Základ učení je trénovat model do největší přesnosti, kde ho nesmíme trénovat moc a ani málo, neboť pokud je model natrénovaný málo, tak má nižší přesnost než by mohl mít celkově a z druhé strany pokud je přetrénován, tak model začne ztrácet svoji přesnost také.

Model popsaný v této práci byl učen na sekvencích 8 000 vzorků, které jsou rozděleny na deset kol. Počet sekvencí, kolik je na tento model použito, je jedenáct, ale na konečné váhy jsou použity desáté.

Na obrázcích 4.6 a 4.7 jsou znázorněny grafy, které reprezentují učení modelu s vyznačenou hodnotou ztrátové funkce a přesnosti. Na první pohled je vidět, že modelu klesá hodnota ztrátové funkce, ale přitom klesá i hodnota i přesnosti. Toto je způsobeno kombinací tohoto modelu a ztrátové funkce, která se snaží co nevíce stlačit výsledky k nule, a tím se vytvořil invertovaný model. Jelikož jsou vstupem modelu dva snímky a výstup je od nuly do jedné, tak při náhodném vývěru dostaneme 50 % šanci, že se trefíme, jako je vidět na těchto grafech. Tím pádem, pokud začne klesat nebo i stoupat přesnost, tak model se stává vždy přesnější akorát jeho hodnoty v klesajícím případě jsou převrácené. Model by se stal nefunkčním pouze, pokud by přesnost pohybovala pořád okolo 50 % a neklesala by a ani by nestoupala.

Dál je vidět na trénovacích datech z grafu 4.6, jak klesají hodnoty, ale jsou v nich výstupky. Tyto výstupky jsou způsobeny kombinací způsobu trénování a struktury modelu, konkrétněji při učení nastavení parametrů kol (*epochs*) s velikostí trénovací skupiny (*batch_size*) a u struktury to je vrstva *BatchNormalization*. Tato vrstva jak bylo zmíněno dříve, tak slouží k normalizaci a používá tam průměr z trénovací skupiny. Problém nastane, když se zmenší velikost trénovací skupiny. V tom případě vždy, když se na začátku trénování (spouštění `fit`) nahrají data potřebná pro normalizaci, ale jelikož je jich nedostatek, tak hodnota ztrátové funkce se zvětší. Poté se s dalším kolem se tyto data zlepší kvůli příchodu nových dat, které se zprůměrují a hodnota ztrátové funkce se zase zmenší. Toto se opakuje s každým spuštěním trénovaní. Jediná výhoda byla, že počet kol bylo 10, protože dokázali



Obrázek 4.6: Graf vývoje trénovacích dat.

vykompenzovat tuto ztrátu a ještě se posunout dál. Důvod, proč to nebylo vidět na obrázku 4.7 s testovacími daty je takový, že ta vrstva má jiný výpočet při testování.

Toto ustálení je lépe, ale vidět na datech z grafu 4.7, kde přesnost zobrazena na testovacích datech. Při každé sekvenci model odložil 20 % vzorků bokem a provádí na nich testování hodnoty ztrátové funkce a jeho přesnosti. Tímto stylem je možno zjisti, zda v modelu nenastává *overfitting*, neboli zaměřování modelu k jednomu typu vzorků a nikoli jako k celku. Toto chování naštěstí ještě nenastalo v obecném modelu jako je tento. Naštěstí je vidět, že tento trend, zde nevzniká, a to protože model je učen po sekvencích.

Model byl hodnocen v datové sadě *LFW*, kde získal přesnost až 91 %. Pro toto hodnocení se použil model ze 100 kroku.

Chybové hlášky

Při zadání špatného vstupu do programu program na vrátí kódy chyby a na standardní chybový výstup vypíše chybovou hlášku s problémem, který nastal.

Návratové kódy

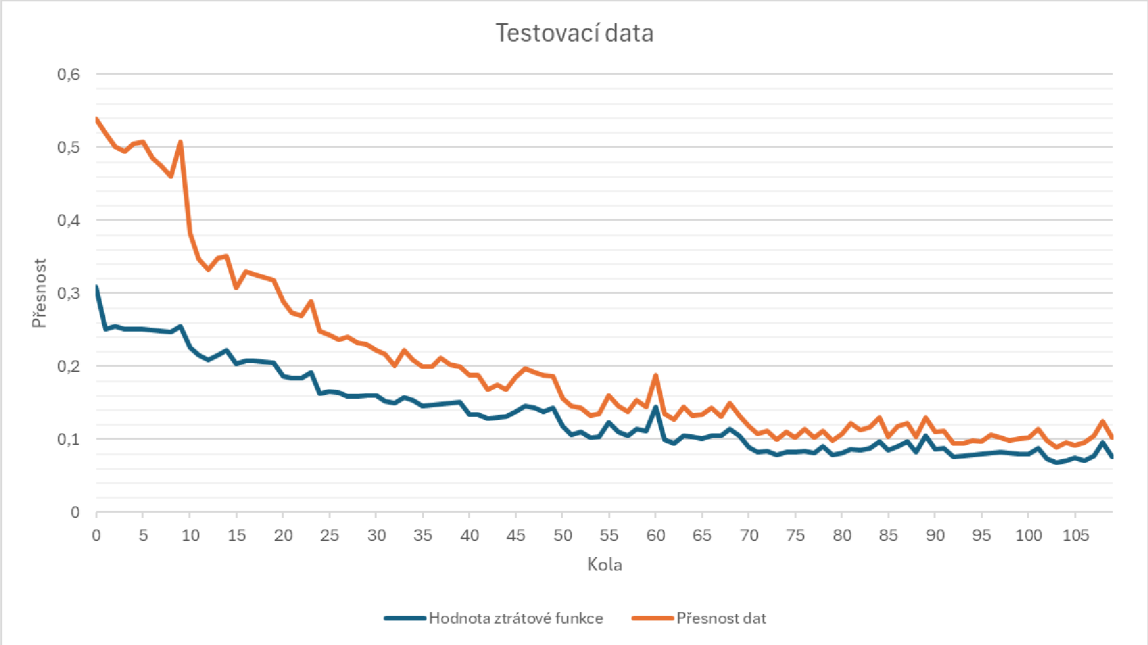
0: Správně ukončený program.

1: Chyba v počtu argumentů. Počet argumentů je jiný než je možný.

2: Špatné použití argumentu `-dr`. Tento argument je použit dvakrát.

3: Chybné zadání v argumentu pro výpis do souboru. Soubor nekončí koncovkou *.txt* nebo soubor nelze otevřít.

4: Chybné zadání argumentu `file_path_A`, který se nechce správně otevřít a formátovat.



Obrázek 4.7: Graf vývoje testovacích dat.

5: Chybné zadání argumentu `file_path_B`, který se nechce správně otevřít a formátovat.

4.4 Použití implementovaného programu

Použití tohoto programu není úplné nejjednodušší, protože je poměrně velký na velikosti. Sice kód je velice malý a nepřesáhl ani 100 kB, ale uložené váhy do modelu mají přes 350 MB, a když se všechny knihovny zabalí pomocí nástroje `pyinstaller` do spustitelného souboru, tak mají kolo 3 GB. Proto jsem se rozhodl program šířit bez zakódovaní do binárního a čerpat z toho i jiné vlastnosti.

Implementovaný program je možný použít dvěma způsoby. První a primární způsob, za kterým byl vyvinut, je použití z příkazové řádky jako konzolová aplikace. Toto lze docílit druhým příkazem níže, spuštěný v kořenové složce projektu na otevření nápovědy. U tohoto je nutnost mít instalovaný *Python 3.11.5* a všechny knihovny sepsané v `requirements.txt` (kód níže – první příkaz).

```
python -m pip install -r requirements.txt

python -m Gen_Verifier --help
```

Druhým způsobem je možné program integrovat jako knihovnu do jiného projektu a používat přímo model na vypočítávání, kde stačí složku `Gen_Verifier` zkopírovat společně s váhami z kořenové složky a vžit je do jiného projektu.

Aplikace je uložena dveřmi způsoby. V papírové verzi je nahrána celá aplikace na přenosném zařízení, ale v elektronické je nahrána na VUT Google Disk a v příloze je jen soubor *README.md* s odkazem na nahranou aplikaci. Toto řešení je aplikováno z důvodu nedostatečného místa v elektronické příloze.

Seznam souboru je v struktuře adresáře, kde první složka slouží k uložení tříd a struktur. Druhá složka obsahuje váhy do modelu. Další obsažený soubor je *README.md*, který obsahuje uživatelskou příručku. Jako poslední tam je soubor `requirements.txt` s knihovnami, které jsou nutné instalovat před použitím.

Kapitola 5

Testování a experimenty

Testování je nejdůležitější část vývoje software i hardware, neboť nám umožňují porovnání předpokládaného výstupu a výstupu z určitých celků programu za daných podmínek. Z tohoto je poté možné zjisti, zda model funguje správně, nebo někde nejsou chyby. Obecně testování ověřuje nějakou schopnost programu, kterou by měl mít. Oproti tomu experimenty hledají nové řešení a prozkoumávání nedostatky v testování.

V této kapitole je model testován za účelem ověřování jeho správné funkčnosti vůči jeho předpokládaného výstupu u reálných snímků, udělaná analýza hodnot na výstupu, následné porovnání s již zavedeným řešením a zhodnocení přesnosti a kvality. Toto vše je téma první sekce 5.1. Následující sekce 5.2 je o prozkoumání funkčnosti na vygenerovaných snímcích. Poslední sekce 5.3 je zaměřená na vývoj, kterým tento model prošel, než získal výsledky takové jaké má, a jaké experimenty se museli provést, aby model byl tak přesný.

5.1 Porovnávání modelu

Pro zjištění kvality je důležité porovnat stávající model s ostatními modely, neboť je složité zjistit, jak si vede model mezi ostatními, když není testován na stejných vzorcích.

Pro tento úkol používám modul z knihovny `FaceAIKit`, který byl učen se ztrátovou funkcí *ArcFace*. Na vstupních snímcích jsou provedeny stejné úpravy jako na vstupu do stávajícího modelu a to jsou nalezení obličeje, oříznutí ho ze snímku a transformování rozlišení snímku obličeje.

Pro porovnání jsou vyrobeny čtyři grafy, u kterých jsou zobrazeny vlastnosti výstupu. Tyto grafy jsou získané ze dvou sad vzorků, které jsou vytvořeny podle snímků osob se stejnými obličeji a osob s rozdílnými obličeji. Počet vzorků je 2 000 na každou sadu.

Výsledky modelu Gen Verifier

Výsledky modelu `Gen_Verifier` jsou zpracovány v grafech typu jádrové odhady hustoty se jmény Graf s hodnotami výsledků u snímků se stejnými obličeji a Graf s hodnotami výsledků u snímků. Tyto grafy jsou na obrázcích 5.1 a 5.2, kde je vidět, jak se chová výstup, který se má pohybovat kolem 0, nebo 1.

Model je invertovaný, jak je zmíněno v sekci Implementace 4.3, tudíž má hodnotu jedné na výstupu u snímku se stejnými obličeji a hodnotu nuly u snímku s rozdílnými obličeji.

Další jeho chtěnou vlastností je, že má přesně oddělené hodnoty mezi typy dvojic snímků ve vzorcích až na chybové vzorky, které se přesně promítli na opačnou část spektra.

Obrázek 5.1: Graf s hodnotami výsledků u snímků se stejnými obličeji modelu Gen Verifier.

Hodnoty se primárně pohybují okolo nuly a jedné. Od nuly níže a od jedné výše se nic nevyskytuje, ale malého množství dat se pohybuje mezi nulou a jedou konkrétněji mezi hodnotami 0,2 až do hodnoty 0,8. Tudíž většina hodnot se vychýlila do dvou konkrétních bodů.

Pro nalezení rozhodují hodnoty je použit graf, který sloučil dva předešlé a průsečíkem hodnot v intervalu od nuly do jedné je vyjádřena mezní hodnota. Tento graf zobrazen na obrázku 5.3 a zobrazuje pásmo podobných hodnot výsledků kolem hodnoty 0,4.

Výsledky z modelu FaceAIKit

Výsledky těchto dat jsou v grafu Graf hodnot výsledků modelu FaceAIKit na obrázku 5.4, který obsahuje obě křivky, protože už není nutné zobrazovat průběh obou grafů.

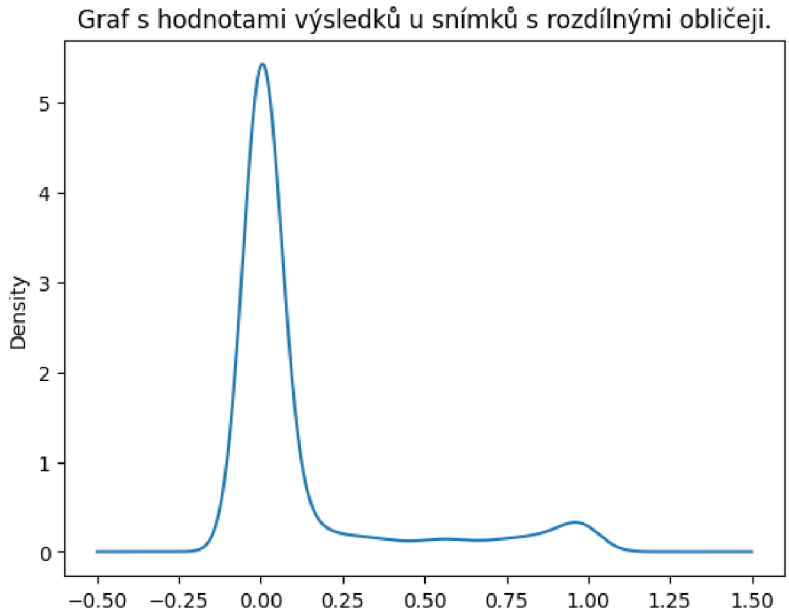
Z tohoto obrázku je vidět, že se hodnoty pohybují u obou typů na kladné ose grafu, kde první typ, který je se stejnými obličeji, je není jednoznačný tolik jako ten druhý, který obsahuje rozdílné obrázky. Je zobrazen jako dvojitý kopeček spojený dohromady, který může být způsoben nedokonalostí datové sady nebo hodně malou chybovostí modelu.

Jelikož se zde snímek nepohybuje na stupnici nuly od jedné, jak jsou data roztroušeny pouze do dvou Gaussových křivek, které se protínají v jenom bodě mezi vrcholky. Tento průsečík je v bodě s hodnotou 1,14, kterou zle používat jako mezní bod.

Celkové výsledky

Pro celkové hodnocení je vypočítáno pomocí úspěšnosti roztěženi modelu mezi dvě kategorie pomocí mezního bodu, který byl vyobrazen v předchozích podsekcích u každého modelu.

Výsledek je možné spočítat z předchozích hodnot, kde se použijí hodnoty ze vzorků se stejnými obličeji a spočítá se počet vzorků, které spadají pod hranici. To samé se udělá u vzorků s rozdíleními obličeji, ale spočítají se všechny vzorky, které spadají nad hranici. Toto proces je opakován pro druhý model.



Obrázek 5.2: Graf s hodnotami výsledků u snímků s rozdílnými obličeji modelu Gen Verifier.

Výsledky těchto modelů jsou následující. Přesnost modelu `Gen_Verifier` je 90,7 % a přesnost modelu z modulu `FaceAIKit` je 97,13 %. Z toho leze usoudit, že výsledky tohoto modulu nejsou lepší jako z knihovny `FaceAIKit`, ale jsou pořád uspokojivé. Naproti tomu model, který byl vytvořen v této práci, docílil většího prostoru mezi vzorky se stejnými obličeji a rozdílnými obličeji a jeho hodnoty spadali do intervalu od nuly do jedné, bez nutnosti hledání hranice.

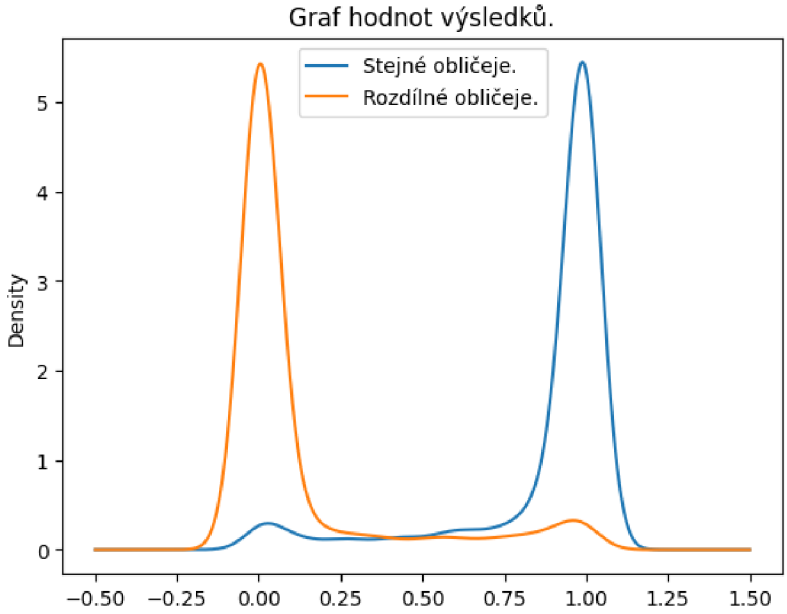
5.2 Testování generovaných snímků

Hlavním cílem tohoto projektu je vytvořit program na vyhodnocení věrohodnosti obličejů u synteticky generovaných snímků, a proto jsou do teď nabyté zkušenosti z předchozích kapitol použité k porovnávání identit na snímcích. Zaměření je na analýze snímků, vytvoření postupu a shrnutí výsledků na provedeném porovnání v modelu *StarGAN*.

Analýza datové sady

Pro vyhodnocení věrohodnosti byla použita datová sada, kterou jsem vytvořil podle sekce Příprava dat 2.4 druhá část. Tato sada obsahuje dvojici originálního snímku a vygenerovaného po 2 000 vzorcích a rozlišením 128 × 128. V tomto případě jsou takové datové sady snímků vytvořeny tři a každá má změnu jiného atributu pro zlepšení vyobrazení pohledu na celý model *StarGAN*.

První je tvořena změnou atributu barvy vlasů. Každý vstupní snímek je analyzován a je mu změněn a barva vlasů na hnědou bez měnění účesu. Tato činnost by neměla měnit žádné vlastnosti obličeje a tím pádem by neměla měnit i osobu na ni zachycenou, ale je možnost, že se obličej nějak změní kvůli nezachycenému atributu z datové sady anebo šumu z pozadí.



Obrázek 5.3: Graf hodnot výsledků modelu Gen Verifier.

Druhá datová sada je na bázi změny pohlaví. To znamenaná, že mužskému obličeji jsou přiděleny ženské rysy a ženskému obličeji jsou přiděleny mužské rysy. Tyto akce už zasahuje do identifikace a je očekávané, že se úspěšnost zmenší.

Poslední datová sada je vytvořena stárnutím obličeje. Její výsledek je, že obličej zestárne o několik desetiletí, ale nejsou zde žádné záruky o podobnosti. U této datové sady se očekává ještě menší procento, protože většina snímků se nevytvořila velmi kvalitně do reálného snímku.

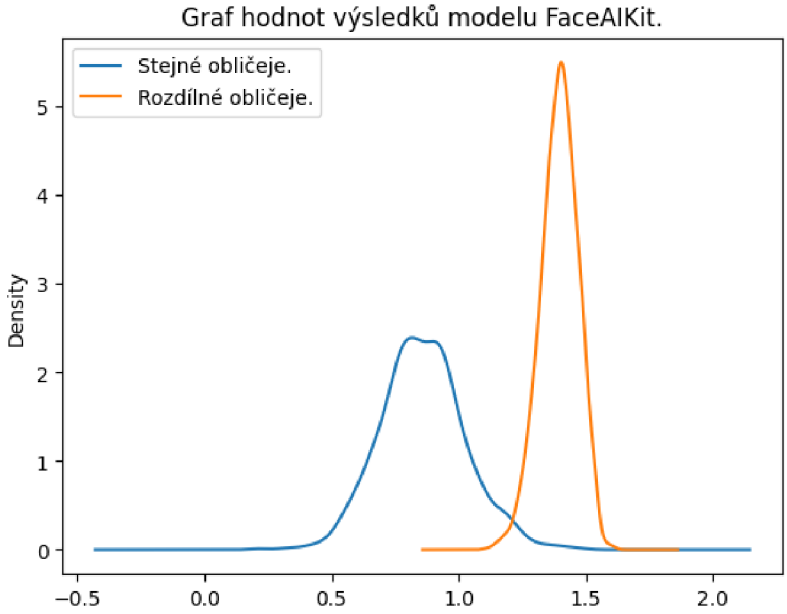
Vyhodnocení věrohodnosti vzorků

Vyhodnocení věrohodnosti datových sad z generátorů syntetických snímků proběhne ve dvou fázích. První fáze je o zjištění samotné podobnosti na datech získaných skrze model při porovnávání snímků, které jsou vloženy do grafu pro podrobnější analýzu. Z druhé části je poté spočítaný výsledek na procenta z prahu z minulé sekce Porovnání modelů 5.1.

Graf je zobrazen na obrázku 5.5, kde je vidět, jak jsou podobné různé datové sady. Tento graf je vytvořen odhadem hustoty za pomocí Gaussových jader, takže pod křivkou je vyjádřeno množství vzorků v jeho obsahu. Jelikož je založený na Gaussových křivkách, tak nejvyšší lokální bod zobrazuje, jaká hodnota se nevíce objevuje a šířkou křivky zjistíme jeho přibližný rozptyl. Podle tohoto grafu je vidět také, že hodnoty jsou pod nulou a nad jedničkou, ale to je jen klam odhadování Gaussových křivek, protože tato křivka je osově souměrná přes osu y a pokud do toho tyto hodnoty vložím, tak program pouze odhadne, jak má vypadat. Nepředpoví přesnou hodnotu, protože data nejsou tak vyhlazená.

První vyznačená je pro referenci a zobrazuje vzorky s různými osobami na snímcích.

Druhá je vyznačená podobnost vlasů, kde je možné si všimnout, že mají vysokou podobnost a málo výsledků na opačném spektru. Toto znamená, že generátory se drželi původního snímku a zaměnili pouze vlasy.



Obrázek 5.4: Graf hodnot výsledků modelu FaceAIKit.

Třetí je pohlaví, kde se vidět, že má vysoké hodnoty a také že generátory neměnili rozměry a znaky obličejů, ale měnili pouze jejich vzhled, který podle mého úsudku, není tak převratný, protože každé pohlaví má určité podobnosti v obličeji a pokud se znaky podobají více originálu než skupině, do které má pařit. Tak to bude vypadat podivně. Ale toto je jen připomínka, že tato toto vyhodnocení nemůže mít pevně dané hranice, protože nejsou přesně možné určit.

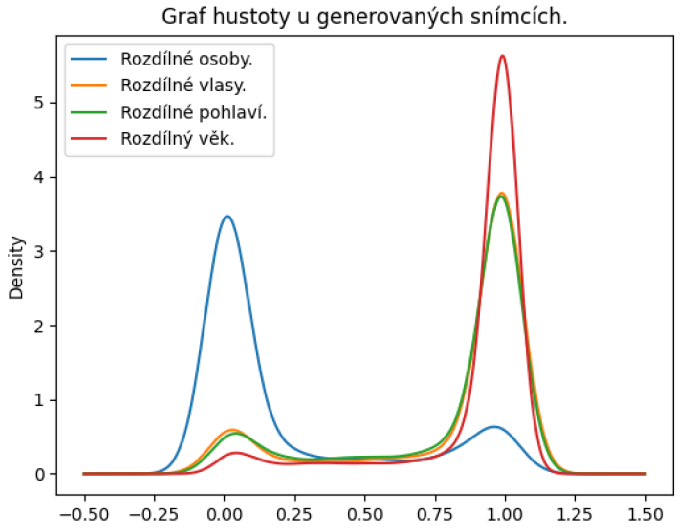
Poslední je věk, který má velice vysokou shodu vůči jiným porovnáním, ale může se tak podobat, protože starší obličeje, na kterých to bylo trénováno, měli odhadem maximálně 60 let a do této doby se obličej tolik nezmění. Přibude více vrásek a obličeji se může zmenšit tuková vrstva pod kůží, což nezmění tolik v obličejových rysech, ale jen je to složitější na rozpoznání.

Při vyhodnocení výsledků, tak u 23 vzorků nebylo možné detekovat obličej na snímku a bylo automaticky vyřazeno z datové sady a se zbylými vzorky dopadlo vyhodnocení tak, že model `Gen_Verifier` přiřadil vzorkům s rozdílnými vlasy přesnost 86,83 %, vzorkům s rozdílným pohlavím hodnotu 84,17 % a vzorkům s rozdílným věkem 91,6 %. Průměrné hodnocení modelu *StarGAN* je 87,53 %.

Z tohoto lze usoudit, že model *StarGAN* má poměrně vysokou přenosnost v obličejových rysů, protože když se vezme na vědomí, že model `Gen_Verifier` má 91 % přenosnost, tak reálná hodnota může být i o 9 % vyšší a celkově průměrná hodnota dosahovat k 96 %. Toto by znamenalo, že model změnit jistě obličejové rysy u 4 % obličejů a 22 % je možnost, že se snímek nepodobá s originálem. V tomto výsledku je zahrnuto i chybovost z předchozího testování, kde 9 % může být falešně pozitivních nebo negativních.

5.3 Vývoj a experimenty

Než byl vytvořen model `Gen_Verifier`, tak bylo uděláno velké množství prototypů, které v kontrolovaných podmínkách fungovali poměrně dobře, ale pokud se používali se snímky



Obrázek 5.5: Graf hustoty u generovaných snímcích.

z reálného světa, tak se jejich přesnost velice zhoršila. Toto vedlo k provedení úprav a experimentů na těchto řešení do modelu, který je v této práci.

První model

První model, který byl sestrojen na rozpoznávání, byla Siamská neuronová síť, která je použita na černobílé datové snímků sadě s množstvím 400 kusů. Tyto snímky byli vytvořeny od čtyřiceti různých osob po deseti snímcích. Struktura snímku byla velice jednoduchá taktéž. Obličej byl vždy ve středu snímku, nebylo moc vidět těla a pozadí bylo jenom bílé. Další výhody jako, že obličeje nebyli jinak otočené nebo všechny osoby byli jednoho typu. Tím je myšlené, že to byl dospělí muž bez vousů, na krátko ostříhaný, s žádným kloboukem nebo brýlemi a měli všichni stejnou barvu pleti. Jako ulehčení také všechny snímky měli stejnou velikost. Tyto ideální snímky bylo použity pro natrénování a analýzu.

Model vypadal velice jednoduše. Předpracovaní bylo na principu normalizace a stačilo vydělení snímku číslem 255.

Jako podsíť používal konvenční neuronovou síť, která obsahovala sekvence 3×3 konvoluční vrstvy s aktivační funkcí *ReLU*, následovala 2×2 *max pooling* vrstva a poslední byla *dropout* vrstva s parametrem 0,1. Tato sekvence se opakovala třikrát. Poté přišlo vyhlazení snímků do jedné řady pomocí vrstvy *Flatten*. Poslední tři vrstvy byli plně propojené, kde první z nich obsahovala 128 neuronů, nasledovala *dropout* vrstva s parametrem 0,1 a poslední byla opět plně propojená s 50 neurony. Obě plně propojené vrstvy měli aktivační funkci *ReLU*.

Tyto síť se spojili a směrovali do poslední vrstvy *Lambda* s výpočtem Euklidovské vzdálenosti. Jako ztrátovou funkci jsem použil *Contrastive loss* s *RMSprop* optimalizátorem.

Tento model dosahoval 94 % úspěšnosti, ale jeho výstup byl pořád poměrně rozházený. Proto do modelu byla přidána ještě jedna plně propojená vrstva o jenom neuronu s aktivační funkcí *Sigmoid*. Poté začala mít přesnost okolo 97 %, ale měl jednu nepříjemnou vlastnost jako jeho předchůdce a to, že se stal po učení invertovaný.

Model na barevné obrázky

Původní návrh na model pro barevné obrázky bylo nalézt obličeje na snímcích, převést tyto snímky do černobílé, transformovat je do potřebné velikosti a vložit je do stávajícího modelu. S tímto experimentem jsem hluboce selhal, protože model dosáhl maximálně 62 % úspěšnosti na ostřihané datové sadě *LFW* o 400 snímcích.

Tímto neúspěchem jsem začal modifikovat model. První věcí, co bylo uděláno bylo odstranění převodu snímku na černobílý a vložena speciální konvoluční vrstva na začátek podsítí modelu, která konvertovala hloubku snímku na tři snímky, které procházeli další konvolucí. Tímto bylo také zvýšeno množství perceptronů u plně propojených vrstev na 512 a 64. Tímto krokem se model zlepšil o pár procent.

V dalších pár modelech byl navýšen počet konvolučních vrstev s různými typy a perceptronů plně propojených vrstev. S tímto bylo i zvetšení rozlišení přijímaných snímků na 50×50. Těmito kroky se model dostal až na hodnoty okolo 75 % účinnosti, ale s každou přidanou vrstvou byla míra učení pomalejší a méně účinná.

Proto byly do modelu přidány místo konvolučních vrstev reziduální bloky s konvolucí. Reziduální bloky byli přidány v rámci modelu *ResNet50*, kde byli odstraněny všechny plně propojené vrstvy a většina nastavení byla přepsána. Další přidaná věc byl před příprava dat normalizací a změnou kanálů. Tím to krokem jsem získal více konvolučních vrstev a zvýšenou kvalitu, kde přesnost skončila někde okolo 86 %.

Jelikož z reziduálních bloků při vycházeli občas vysoká čísla, tak byli přidané normalizace po reziduálních blocích a po výpočtu Euklidovské vzdálenosti a společně s navýšením perceptronů u plně propojených vrstev. Tímto se model dostal na 89 % přesnosti.

Po dalších experimentech v počtu perceptronů v plně propojených vrstvách se počty dostali až na 2048 a na 512, kde se začala ustalovat přesnost u 91 % a přidání perceptronů moc už nepomáhalo.

Kapitola 6

Závěr

Úkolem této práce tvorba aplikace, která se stará o vyhodnocení věrohodnosti syntetický generovaných snímků obličeje. Toto vyhodnocení věrohodnosti se nezabývá realističnosti nebo vzhledem snímků, ale probíhá ve formě rozpoznávání obličejů, kde jsou obličeje extrahovány z originálních a vygenerovaných snímků a následně porovnány v podobnosti obličejových rysů.

Pro uskutečnění aplikace jsou v práci představené témata, která se zabývají touto problematikou. První téma, které je přestavené, je generování syntetických snímků obličeje, kde jsou popsány neuronové sítě, základní bloky pro generování snímků, a generativní soupeřící sítě, které se starají o samotné generování syntetických snímků ve kvalitě, jaká není přesně vyjádřená. K tomuto tématu jsou také připojeny dvě datové sady, které jsou použité při učení modelů a následném generování snímků.

Druhé téma je o rozpoznání osob podle obličeje. Zde je uvedeno, jak tento proces funguje v digitálním světe a jaké jsou vytvořené modely pro jeho řešení, které jsou později používány jako reference kvality, kam se model má dostat.

Po znalostech získaných z předchozích témat je navrhnut model *Gen_Verifier*, který je jádrem aplikace pro porovnávaní. Model je tvořen siamskou neuronovou sítí, která je poté implementována a trénována s použitím několika tříd pro snadnější ovládání a správnou funkčnost aplikace.

Poslední částí je testování a experimentování na datových sadách *LFW* a *CelebA*, které slouží na zjištění kvality modelu a poté kvality generátoru *StarGAN*. Testy jsou zpracovány ve dvou fázích. První zpracovává reálné obrázky pro zaručení kvality modelu a druhý zpracovává testy na vygenerovaných snímcích.

Výsledky po trénování modelu na reálných snímcích a následné testování ukázali, že model dosahuje 91 % přesnosti na datové sadě *LFW*, kde většina výstupních hodnot je u čísel 0 a 1 a málo hodnot se pohybovali mezi mini. Toto spektrum hodnot bylo chtěné, protože to mnohem ulehčí rozhodování. U druhého testování, tentokrát na generovaných snímcích z datové sady *CelebA*, kde hodnocení modelu dosáhlo průměrně 87,53 %. Z toho je množené říct, že okolo 4 %, které model *StarGAN* vygeneroval se nepodobá originálu s tím, že se nesmí přehlédnout k chybovosti z prvního testování.

Z tohoto je možné usoudit dva výsledky. První je k modelu *Gen_Verifier*, který měl chybovost 9 %. Tato chybovost není příliš velká a na vyhodnocení věrohodnosti by teď ještě prošla, ale na budoucí použití je potřeba použít přesnější nastroj, neboť ostatní modely se vyvíjejí s vysokou rychlostí a takové výsledky byly před 8 lety.

Druhý výsledek zasahuje do testovaného generátoru *StarGAN*, u kterého je určeno, že 4 % vygenerovaných obličejů se nepodobá originálu. Pokud by se tento model zaměřoval

přímo na generování, co nejpodobnějších snímků, tak by byl poměrně dobré. Ale na to, že tento model nemá jednotku specializovanou na toto ověřování, tak má výborné výsledky.

Literatura

[1] Alake, R. *Loss Functions in Machine Learning Explained* [online]. Data Camp, 2023 [cit. 2024-04-15]. Dostupné z: https://www.datacamp.com/tutorial/loss-function-in-machine-learning.

[2] Algoscale. *How To Use GAN To Generate Images* [online]. 2022 [cit. 2023-12-12]. Dostupné z: https://algoscale.com/blog/how-to-use-gan-to-generate-images/.

[3] Benhur, S. *A friendly introduction to Siamese Networks* [online]. Towards Data Science, 2020 [cit. 2024-03-20]. Dostupné z: https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942.

[4] Brownlee, J. *A Gentle Introduction to Deep Learning for Face Recognition* [online]. Deep Learning for Computer Vision, 2019 [cit. 2023-12-27]. Dostupné z: https://machinelearningmastery.com/introduction-to-deep-learning-for-face-recognition/.

[5] Brownlee, J. *A Gentle Introduction to Generative Adversarial Networks (GANs)* [online]. 2019 [cit. 2023-12-30]. Dostupné z: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/.

[6] Brownlee, J. *A Gentle Introduction to the Rectified Linear Unit (ReLU)* [online]. 2020 [cit. 2024-03-30]. Dostupné z: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[7] Brownlee, J. *How Do Convolutional Layers Work in Deep Learning Neural Networks?* [online]. Deep Learning for Computer Vision, 2020 [cit. 2023-12-27]. Dostupné z: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

[8] Brownlee, J. *Softmax Activation Function with Python* [online]. 2020 [cit. 2024-03-30]. Dostupné z: https://machinelearningmastery.com/softmax-activation-function-with-python/.

[9] Brownlee, J. *A Gentle Introduction To Sigmoid function* [online]. 2021 [cit. 2024-03-30]. Dostupné z: https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/.

[10] Chauhan, N. S. *Generate Realistic Human Face using GAN* [online]. KDnuggests, 10. března 2020 [cit. 2023-12-12]. Dostupné z: https://www.kdnuggets.com/2020/03/generate-realistic-human-face-using-gan.html?__cf_chl_tk=DkjPXVyYj1604EMWgzLxoLGq7ND_Hw1VGJXtNj.KVDU-1703156934-0-gaNycGzND9A.

[11] Choi, Y. *StarGAN - Official PyTorch Implementation.* 2020. Dostupné z: https://github.com/yunjey/stargan.

[12] Choi, Y., Choi, M., Kim, M., Ha, J.-W., Kim, S. et al. *StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation.* 2018.

[13] Deng, J., Guo, J., Yang, J., Xue, N., Kotsia, I. et al. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 1. vyd. Institute of Electrical and Electronics Engineers (IEEE). říjen 2022, sv. 44, č. 10, s. 5962–5979. DOI: 10.1109/tpami.2021.3087709. ISSN 1939-3539. Dostupné z: http://dx.doi.org/10.1109/TPAMI.2021.3087709.

[14] Deng, J., Guo, J., Zhou, Y., Yu, J., Kotsia, I. et al. *RetinaFace: Single-stage Dense Face Localisation in the Wild.* 2019.

[15] Girshick, R. *Fast R-CNN.* 2015.

[16] Guiné, R. *The Use of Artificial Neural Networks (ANN) in Food Process Engineering.* Březen 2019. DOI: 10.18178/ijfe.5.1.15-21.

[17] He, K., Zhang, X., Ren, S. a Sun, J. *Deep Residual Learning for Image Recognition.* 2015.

[18] Huang, G. B., Ramesh, M., Berg, T. a Learned Miller, E. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments.* 07-49. University of Massachusetts, Amherst, October 2007.

[19] IBM. *The Neural Networks Model* [online]. IBM Corporation, 2021 [cit. 2023-12-12]. Dostupné z: https://www.ibm.com/docs/en/spss-modeler/18.0.0?topic=networks-neural-model.

[20] IBM. *Convolutional neural networks* [online]. IBM Corporation, 2022 [cit. 2023-12-12]. Dostupné z: https://www.ibm.com/topics/convolutional-neural-networks.

[21] Keen, M. *What are GANs (Generative Adversarial Networks)?* [online]. IBM Technologyt, 2021. Dostupné z: http://neuralnetworksanddeeplearning.com/.

[22] Liu, Z., Luo, P., Wang, X. a Tang, X. *Deep Learning Face Attributes in the Wild.* December 2015.

[23] *How CNNs Work* [online]. Math works, 2021 [cit. 2023-12-12]. Dostupné z: https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html.

[24] Meng, Q., Zhao, S., Huang, Z. a Zhou, F. *MagFace: A Universal Representation for Face Recognition and Quality Assessment.* 2021.

[25] Nguyen, T.-P., Lathuilière, S. a Ricci, E. *Multi-Domain Image-to-Image Translation with Adaptive Inference Graph.* 2021.

[26] Nielsen, M. A. *Neural Networks and Deep Learning* [online]. Determination Press, 2018. Dostupné z: http://neuralnetworksanddeeplearning.com/.

[27] Pythonu, V. *What is Python? Executive Summary* [online]. 2020 [cit. 2024-04-15]. Dostupné z: https://www.python.org/doc/essays/blurb/.

[28] Saha, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* [online]. Towards Data Science, 2018 [cit. 2024-04-15]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[29] Sharma, S. *Activation Functions in Neural Networks* [online]. Towards Data Science, 2017 [cit. 2023-12-15]. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[30] Shinde, S., Pradhan, T., Ghorpade, A. a Tale, M. *Face Generation from Textual Features using Conditionally Trained Inputs to Generative Adversarial Networks*. 2023.

[31] Taherkhani, F., Talreja, V., Dawson, J., Valenti, M. C. a Nasrabadi, N. M. *Profile to Frontal Face Recognition in the Wild Using Coupled Conditional GAN*. 2021.

[32] Taskiran, M., Kahraman, N. a Erdem, C. Face recognition: Past, present and future (a review). *Digital Signal Processing*. 1. vyd. Červenec 2020, sv. 106, č. 1, s. 102809. DOI: 10.1016/j.dsp.2020.102809.

[33] Terhörst, P., Ihlefeld, M., Huber, M., Damer, N., Kirchbuchner, F. et al. *QMagFace: Simple and Accurate Quality-Aware Face Recognition*. 2022.

[34] Tran, L., Yin, X. a Liu, X. Representation Learning by Rotating Your Faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1. vyd. Institute of Electrical and Electronics Engineers (IEEE). prosinec 2019, sv. 41, č. 12, s. 3007–3021. ISSN 1939-3539.

[35] Wang, S.-C. *Artificial Neural Network*. 1. vyd. Boston, MA: Springer US, 2003. 81–100 s. ISBN 978-1-4613-5046-0, 978-1-4615-0377-4.

[36] Wang, Z. *Contrasting contrastive loss functions* [online]. Towards Data Science, 2020 [cit. 2024-03-20]. Dostupné z: https://towardsdatascience.com/contrasting-contrastive-loss-functions-3c13ca5f055e.

[37] Wu, P.-W., Lin, Y.-J., Chang, C.-H., Chang, E. Y. a Liao, S.-W. *RelGAN: Multi-Domain Image-to-Image Translation via Relative Attributes*. 2019.

[38] Yang, S., Luo, P., Loy, C. C. a Tang, X. *WIDER FACE: A Face Detection Benchmark*. 2015.

[39] Yathish, V. *Loss Functions and Their Use In Neural Networks* [online]. Towards Data Science, 2020 [cit. 2023-12-15]. Dostupné z: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9.

[40] Zhang, K., Zhang, Z., Li, Z. a Qiao, Y. Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. *IEEE Signal Processing Letters*. 1. vyd. Institute of Electrical and Electronics Engineers (IEEE). říjen 2016, sv. 23, č. 10, s. 1499–1503. DOI: 10.1109/lsp.2016.2603342. ISSN 1558-2361. Dostupné z: http://dx.doi.org/10.1109/LSP.2016.2603342.