VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

ROZPOZNÁVÁNÍ OSOB PODLE OBLIČEJE V NEFRONTÁLNÍCH POZICÍCH

FACE RECOGNITION OF PERSONS IN NON-FRONTAL POSITIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JOZEF HORVÁT**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ GOLDMANN**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

143842

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Horvát Jozef, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Strojové učení
Název:	**Rozpoznávání osob podle obličeje v nefrontálních pozicích**
Kategorie:	Umělá inteligence
Akademický rok:	2022/23

Zadání:

1. Seznamte se s problematikou rozpoznávání osob podle obličeje. Především se zaměřte na řešení pro rozpoznávání osob v nefrontálních pozicích.
2. Vyhledejte relevantní datasety pro trénování a testování algoritmů strojového učení pro rozpoznávání osob podle obličeje ve frontálních i nefrontálních pozicích.
3. Navrhněte algoritmus pro rozpoznávání osob podle obličeje v nefrontálních pozicích.
4. Proveďte implementaci algoritmu z předchozího bodu s využitím knihovny Pytorch nebo Keras.
5. Na dodaném datasetu proveďte srovnání vašeho řešení s algoritmem ArcFace. Zjistěte, jak hodně dokáže vaše řešení při rozpoznávání podle obličeje kompenzovat vliv natočení. Navrhněte další rozšíření algoritmu.

Literatura:

- KAWULOK, Michal; CELEBI, Emre; SMOLKA, Bogdan (ed.). *Advances in face detection and facial image analysis*. Springer, 2016.
- CHOWDHURY, Srija; SIL, Jaya. FACE RECOGNITION from NON-FRONTAL IMAGES Using DEEP NEURAL NETWORK. In: *2017 Ninth International Conference on Advances in Pattern Recognition (ICAPR)*. IEEE, 2017. p. 1-6.
- KAWULOK, Michal; CELEBI, Emre; SMOLKA, Bogdan (ed.). *Advances in face detection and facial image analysis*. Springer, 2016.

Při obhajobě semestrální části projektu je požadováno:
Body 1, 2 a 3.

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:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	3.11.2022

Abstrakt

Táto práca sa zaoberá vytvorením modelu schopného rozpoznávania osôb podľa tvári na obrázkoch v nefrontálnych pozíciách. Riešenie stavia na algoritme ArcFace a využíva anotácie uhlov naklonenia a natočenia tváre.

Abstract

This thesis deals with the creation of a model capable of recognizing people from faces in images in non-frontal poses. The solution builds on the ArcFace algorithm and uses annotations of tilt and rotation angles of the face.

Kľúčové slová

rozpoznávanie tvárí, nefrontálne rozpoznávanie, konvolučná neurónová sieť, strojové učenie, ArcFace, TensorFlow2, Keras

Keywords

face recognition, non-frontal recognition, convolutional neural network, machine learning, ArcFace, TensorFlow2, Keras

Citácia

HORVÁT, Jozef. *Rozpoznávání osob podle obličeje v nefrontálních pozicích.* Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Goldmann

Rozpoznávání osob podle obličeje v nefrontálních pozicích

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Tomáša Goldmanna. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Jozef Horvát
15. mája 2023

Poďakovanie

Rád by som poďakoval pánovi Ing. Tomášovi Goldmannovi, za množstvo cenných rád a pripomienok, ktoré mi poskytol pri spracovaní tejto diplomovej práce. Výpočtové zdroje boli poskytnuté v rámci projektu e-INFRA CZ (ID:90140), ktorý podporilo Ministerstvo školstva, mládeže a telovýchovy Českej republiky.

Obsah

Kapitola 1

Úvod

Problematika rozpoznávania osôb podľa tváre je veľmi pestrá a široká. Má využitie napríklad v dohľadových systémoch, prístupových termináloch, používa sa pre forenzné účely a v mnohých ďalších situáciách. V reálnom prostredí sa pracuje s fotkami a obrázkami, ktoré nie sú zachytené v kontrolovaných podmienkach. Obrázky môžu mať veľmi tmavé pozadie, svetlo nemusí byť najlepšie, alebo osoby sú zachytené v rôznych pozíciách. V tejto práci sa venujem práve tomuto poslednému prípadu. Cieľom diplomovej práce je vytvoriť program, ktorý bude umožňovať rozpoznávanie osôb podľa tvárí v nefrontálnych pozíciách.

Existuje plno *state-of-the-art* riešení pre rozpoznávanie tvárí. Mnohé dokážu dobre pracovať s obrázkami z reálneho sveta. Modely sú trénované na rôznych dátach, kde sú osoby v rôznych pozíciách (frontálnych alebo nefrontálnych), s rôznymi pozadiami a osvetleniami. Modely sú trénované na obrovskom množstve dát (milióny obrázkov s tisíckami identít). Medzi takéto algoritmy patria napríklad *ArcFace*, *FaceNet* alebo *DeepFace*, a mnoho ďalších.

Tieto algoritmy majú obrovské množstvo implementácií v rôznych knižniciach. Je ich možné ľahko nainštalovať a využiť pomocou zopár riadkov kódu. Niektorí autori dokonca ponúkajú natrénované modely, takže použitie je o to ľahšie. Medzi takéto knižnice patria napríklad *DeepFace* alebo *InsightFace*.

Spomenuté riešenia majú väčšinou iný prístup k nefrontálnemu rozpoznávaniu. Problém riešia v prvotných fázach pri fáze zarovnania tváre. Model detekuje uhol natočenia tváre, a snaží sa tvár natočiť do frontálnej podoby a vycentrovať. Druhý prístup je taký, že obrázok sa nepredspracuje, ale nechá sa priamo na neurónovú sieť, aby sa naučila rozpoznávať tváre v týchto pozíciách.

Moje riešenie bude pracovať s uhlovými anotáciami: vertikálne a horizontálne natočenie tváre. Tieto anotácie budú spolu s obrázkami a ich identitami na vstupe do siete. S týmito anotáciami bude pracovať upravená chybová funkcia. Riešenie bude postavené na algoritme ArcFace, ktoré zároveň bude slúžiť ako referenčný bod toho, či vlastné modifikácie majú nejaký dopad na úspešnosť upraveného modelu.

V kapitole 2 popisujem štandardný proces rozpoznávania osôb na základe tvárí, uvádzam algoritmy, implementované riešenia načrtnutých algoritmov a nakoniec datasety. V kapitole 3 prichádzam s návrhom riešenia. Popisujem frameworky pre vytváranie modelov strojového učenia. Uvádzam implementačné nástroje a datasety, ktoré použijem v práci. Následne uvádzam implementáciu riešenia. V kapitole 4 popisujem vykonané experimenty na trénovaných modeloch, výsledky a porovnanie s nemodifikovanou ArcFace implementáciou.

Kapitola 2

Rozpoznávanie osôb podľa tvárí

V tejto kapitole popisujem ako prebieha rozpoznávanie osôb podľa tvári. Na začiatku uvádzam základné postupy a techniky. Ďalej opisujem niektoré známe algoritmy. Podrobnejšie sa zaoberám algoritmom ArcFace, ktorý následne používam pri vývoji môjho riešenia. Na konci kapitoly uvádzam datasety, ktoré sa používajú pre trénovanie a testovanie sietí.

2.1 Proces rozpoznávania

Rozpoznávanie tvárí je technika identifikácie alebo verifikácie osoby pomocou jej tváre na fotografii alebo videu. Táto úloha počítačového videnia zachytáva, analyzuje a porovnáva vzory detailov tváre danej osoby. Informácie o procese rozpoznávania tvárí som vo všeobecnosti čerpal z následujúceho zdroja [30].

Môžeme rozlišovať medzi dvoma druhmi rozpoznávania [31] a to:

- verifikácia,
- identifikácia.

Verifikácia osoby na základe tváre je mapovanie *one-to-one*: danej tváre k známej identite (môžeme sa spýtať otázku: "Je to on?"). Identifikácia osoby je mapovanie *one-to-many*: danej tváre k databáze známych tvárí (môžeme sa spýtať otázku: "Kto je to?").

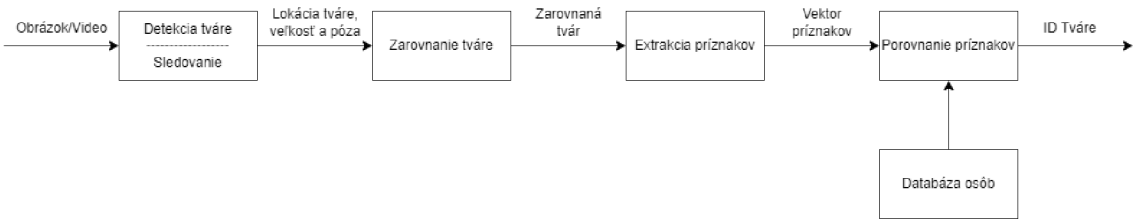
Proces rozpoznávania tvárí je možné rozdeliť do týchto hlavných častí:

- detekcia tváre,
- zarovnanie tváre,
- extrakcia príznakov,
- určenie podobnosti alebo klasifikácia.

Proces rozpoznávania tvárí je zobrazený na obrázku číslo 2.1.

2.1.1 Detekcia tváre

Pomocou detekcie tváre sa snažíme zistiť, či sa na obrázku nachádza tvár, identifikovať jej miesto, extrahovať a vytvoriť komprimovaný súbor pre budúcu extrakciu príznakov. Existuje niekoľko metód ako sa dá tvár detekovať na obrázku.

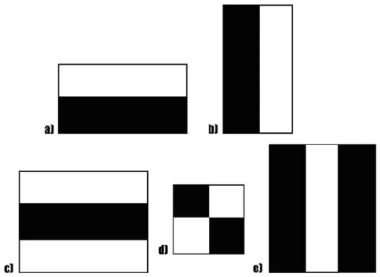


Obr. 2.1: Proces rozpoznávania osôb z obrázkov [30].

Viola-Jones

Viola-Jones (alebo tiež Haar cascade) bol od roku 2001 po dlhú dobu *state-of-the-art* algoritmus pre detekovanie tvárí. Uviedli ho P. Viola a M. J. Jones vo svojej práci [55]. Na algoritme bolo vykonaných veľa vylepšení. Výhodou je, že má veľmi jednoduchú architektúru a beží rýchlo. Nevýhodou je, že dáva zlé výsledky a nefunguje na nefrontálnych obrázkoch. Informácie som čerpal zo zdroja [4].

Haarové príznaky služia na detekciu hrán a miesta, kde dochádza k náhlej zmene v intenzite pixelov. Príznaky sú ilustrované na obrázku 2.2. Tmavé miesta sú pixely s hodnotou 1 a svetlé s hodnotou 0. Úlohou je získať priemernú hodnotu pixelov ležiacich v bielej a tmavej oblasti. Ak je rozdiel priemerov blízky 1, tak bola detekovaná hrana.



Obr. 2.2: Haarové príznaky [4].

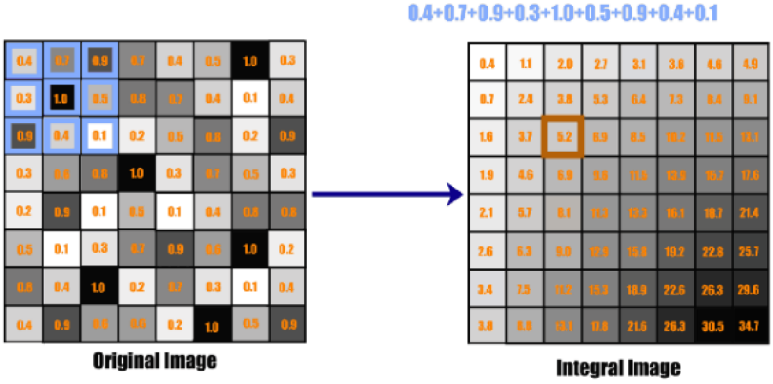
Tento výpočet sa urobí postupne pre celý obrázok (pre každý príznak), čo je ale výpočtovo veľmi náročné. Riešením je *integrálny obraz*, ktorý sa vypočíta z pôvodného obrázku. Obsahuje pixely, ktoré sú súčtom všetkých pixelov pôvodného obrázku ležiacich od neho naľavo a hore. Výpočet môžeme vyjadriť pomocou rovnice:

$$II(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.1}$$

kde $i(x', y')$ je hodnota pixelu na pozícií (x', y') [5]. Princíp je znázornený na obrázku 2.3. Výhoda integrálneho obrazu je že nám stačí pracovať v jednom kroku len so 4 pixelmi.

Na začiatku vývoja algoritmu sa pracovalo so 180 000 príznakmi. Nie všetky však boli relevantné pre rozoznávanie tvárí. Bol preto použitý algoritmus *AdaBoost*, ktorý zredukoval obrovskú množinu príznakov na 6 000 príznakov.

AdaBoost (Adaptive Boosting) je technika pre vytváranie silnejších modelov strojového učenia pomocou kombinácie slabších modelov. Využívajú sa na to rozhodovacie stromy, ktoré majú len koreň a dva listy (decision stump). Metóda vytvára strom pre každý atribút



Obr. 2.3: Integrálny obraz [4].

prípadu, pozerá ako klasifikuje tieto prípady a prideľuje váhy prípadom (väčšiu váhu zle klasifikovaným prípadov) a aj stromom (väčšiu váhu úspešnejším stromom) [29].

Dlib (HOG)

Dlib (HOG) je model založený na HOG [11] (histogram orientovaných gradientov) príznakoch a SVM. HOG sa používa pre extrakciu príznakov z obrázkov. Je to jeden z rýchlejších modelov bežiacich na CPU. Dokáže pracovať s frontálnymi a čiastočne aj s nefrontálnymi obrázkami. Má problém s malými obrázkami, a niekedy neberie, ignoruje niektoré časti brady a čela.

HOG deskriptor sa zameriava na štruktúru a tvar objektu. Obrázok je rozdelený do niekoľkých segmentov a pre jednotlivé segmenty sa generujú histogramy pomocou využitia veľkosti a orientácie gradientu.

Na výpočet gradientu sa používa 1D maska $[-1, 0, 1]$, ktorú môžeme označiť ako G_x (a transponovanú ako G_y). Pomocou nej vypočítame veľkosť a uhol gradientu v smere x a y následovne:

$$\mu = \sqrt{G_x^2 + G_x^2} \tag{2.2}$$

$$\theta = |tan^{-1}(G_y/G_x)| \tag{2.3}$$

Obrázok je následne rozdelený do pravidelnej mriežky (používajú sa bunky o veľkosti 8x8 pixelov). Pre každý blok je vypočítaný histogram s deviatimi časťami, kde každá časť sleduje uhol v rozmedzí 20 stupňov. Kvôli senzitivite gradientov na osvetlenie je nutné urobiť normalizáciu. Následne môžeme získať príznaky z celého obrázku. Informácie som čerpal zo zdroja [54].

Dlib (CNN)

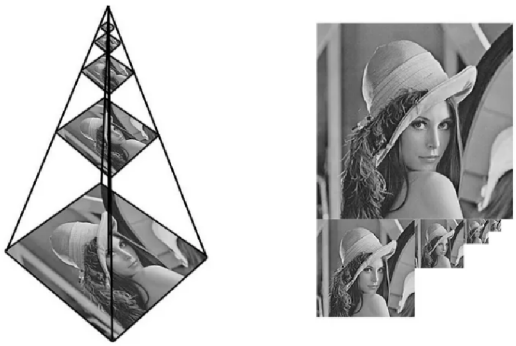
Dlib (CNN) je metóda používajúca MMOD (Maximum-Margin Object Detector) s konvolučnou neurónovou sieťou. Model funguje veľmi rýchlo na GPU a umožňuje prácu s tvárami na obrázkoch vo frontálnych aj nefrontálnych pozíciách. Model si vie poradiť aj so zakrytými časťami tváre. Nevýhodou tejto metódy je, že je trénovaná na tvárach o veľkosti 80x80 pixelov, takže neumožňuje detekciu menších tvárí. Metóda je veľmi pomalá na CPU.

MTCNN

Multi-task Cascaded Convolutional Networks [19] je model, ktorý umožňuje detekciu a zarovnanie tvárí. Metóda dáva najpresnejšie výsledky z vyššie spomenutých metód. Model vie pracovať s rôznymi natočeniami a veľkosťami tváre a jej zakrytiami. Nevýhodou je, že je pomalšia v porovnaní s HOG a Haar cascade metódou.

Proces pozostáva z troch etáp konvolučných neurónových sietí, ktoré sú schopné rozoznať tváre a pozície landmarkov (oči, nos a ústa).

Na začiatku (ešte pred samotným algoritmom) sa zoberie obrázok a zmení sa jeho veľkosť na rôzne mierky, tak aby sa vytvorila pyramída z obrázkov. Tá je vstupom následujúcej kaskádovej siete. Táto príprava dát je ilustrovaná na obrázku 2.4.



Obr. 2.4: Integrálny obraz [19].

Prvý štádiom je sieť P-Net (proposal network). Je to plná konvolučná sieť. Sieť získava kandidátne okná a regresné vektory ich ohraničení. Po ich získaní sa spoja prekrývajúce sa oblasti. Toto je nakoniec výstupom tejto časti algoritmu.

Druhým štádiom je sieť R-Net (refine network). R-Net je konvolučná sieť. Vstupom do siete je výstup predchádzajúcej siete s popísanými úpravami. Znižuje počet kandidátov, vykonáva kalibráciu s regresiou ohraničenia a využíva ne-maximálne potlačenie (NMS) na zlúčenie prekrývajúcich sa kandidátov. Výstupom sú dva vektory, a to 4-prvkový a 10-prvkový vektor. Prvý značí ohraničujúci rámček tváre a druhý lokalitu landmarkov tváre.

Tretím štádiom je sieť O-Net (output network). Je podobná sieti R-Net, ale popisuje tvár ešte podrobnejšie. Výstupom je poloha piatich landmarkoch tváre.

2.1.2 Zarovnanie tváre

Zarovnanie tvárí sa používa v počiatočných fázach rozpoznávania tvárí. Google uvádza, že zarovnanie tvárí zvyšuje presnosť ich modelu pre rozpoznávanie tvárí FaceNet z 98,87% na 99,63% [47]. Pre zarovnanie tvárí môžeme použiť metódu MTCNN, alebo využiť knižnicu OpenCV, či Dlib, ktoré poskytujú funkcionalitu 2D zarovnania tváre. Následne uvádzam niektoré ďalšie algoritmy a modely, ktoré je možné využiť pre tento účel.

Kaskádové neurónové siete

Kaskádové neurónové siete pozostávajú z niekoľkých úrovní sietí s rôznymi parametrami. Prvá úroveň slúži na prvotný odhad tvárových landmarkov z veľkých vstupných regiónov. Vyžaduje to hlbokú neurónovú sieť. Ďalšie úrovne spracovávajú menšie regióny v okolí odhadov vyšších vrstiev. Tieto úrovne sa snažia znížiť rušenie. Parametre každej úrovne sa

učia samostatne, takže sa nezachytáva korelácia medzi sémanticky súvisiacimi charakteristikami obrázka. Pre riešenie tohto problému sa môžu použiť rekurentné neurónové siete [57].

SDUNet

SDUNet (Stacked Dense U-Nets) je algoritmus používajúci husté U-siete. Pre zvýšenie účinnosti model predstavuje novú topológiu a zvyšuje kapacitou siete bez navýšenia výpočtovej zložitosti modelu (dosahuje to pomocou pridania nového agregačného bloku). Pre 3D-68 landmarky model dosahuje úspešnosť 99,80% na datasete LFW [21].

DeepFace

DeepFace je model pre rozpoznávanie osôb podľa tvárí, v ktorom je predstavená metóda 3D zarovnania tváre založenej na 67 fiduciálnych bodoch. Podrobnejší popis je v tejto časti práce 2.3.1.

2.1.3 Historické metódy extrakcie príznakov

Extrakcia príznakov je základný a veľmi dôležitý krok pri rozpoznávaní tvárí. Jeho úlohou je extrahovať črty tváre, ktoré následne voláme príznaky. Sú to biologické komponenty, ktoré sa líšia od človeka k človeku. Existuje viacero metód ako extrahovať rôzne kombinácie týchto príznakov. Sú známe aj ako uzlové body. Žiadni dvaja ľudia nemajú všetky uzlové body rovnaké, okrem identických dvojčiat.

Úlohou extrakcie príznakov je znížiť dimenzionalitu dát tým, že vytvára už z existujúcich príznakov nové príznaky. Pôvodné príznaky sa zahodia. Tieto nové príznaky sumarizujú informácie z pôvodnej sady príznakov.

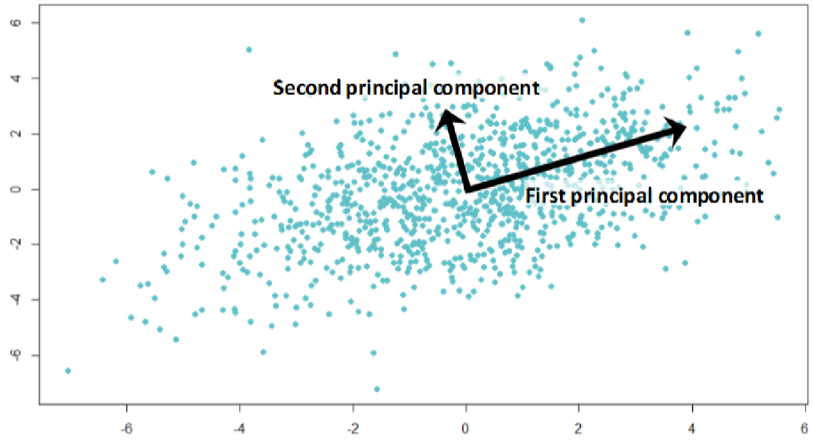
Na rozdiel od extrakcie príznakov poznáme výber príznakov. V tomto prístupe nevytvárame žiadne nové príznaky. Snažíme sa vyberať príznaky, ktoré sú najviac dôležité (napríklad najlepšie ukazujú rozdiel medzi jednotlivými triedami) a zahadzujeme menej dôležité. Existuje niekoľko prístupov k extrakcii príznakov, uviedol by som najprv historické a následne používané v súčastnosti.

Principal Component Analysis

Principal Component Analysis (PCA) [6] slúži na transformáciu vysoko-dimenzionálnych dát na nízko-dimenzionálne, zatiaľ čo sa snaží zachovať čo najviac informácií ako je možné. Algoritmus PCA patrí medzi algoritmy učenia bez učiteľa.

PCA je definovaná ako ortogonálna lineárna transformácia, ktorá transformuje dáta do nového súradnicového systému, takže najväčší rozptyl nejakej skalárnej projekcie dát bude ležať na prvej súradnici (prvý hlavný komponent), druhý najväčší na druhej súradnici a tak ďalej. Príklad hlavných komponent je zobrazený na obrázku 2.5.

Na začiatok je vhodné dáta štandardizovať. Dosiahneme to tým, že od každého dáta odčítame strednú hodnotu a rozdiel podelíme štandardnou odchýlkou. Takéto dáta budú mať porovnateľnú škálu, čím sa vyhneme problémom. Následne si vypočítame kovariančnú maticu, ktorá nám ukáže korelácie medzi premennými. Hlavné komponenty určíme pomocou vlastných vektorov a vlastných čísel. Pre každú dimenziu dát existuje jeden vlastný vektor a vlastné číslo. Vlastný vektor kovariančnej matice udáva smer osi, kde je najväčší rozptyl, čo vlastne nazývame hlavný komponent. Vlastné číslo udáva veľkosť rozptylu. Ak zoradíme



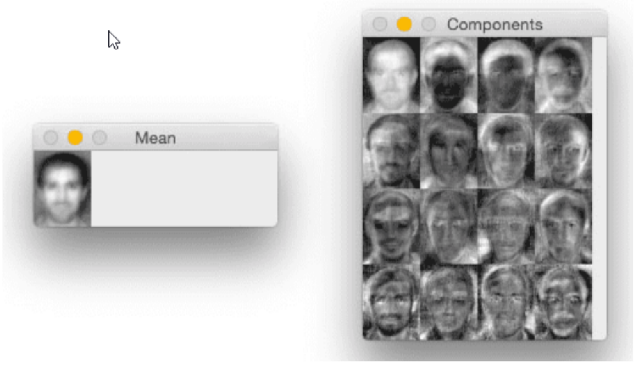
Obr. 2.5: Hlavné komponenty [2].

vlastné vektory podľa veľkosti ich vlastných čísel, dostaneme zoradené hlavné komponenty [26].

PCA je možné použiť na identifikáciu osôb pomocou algoritmu **EigenFaces** [43]. Jeho autormi sú Sirovich a Kirby, ktorý algoritmus predstavili v roku 1987 vo svojej práci [50]. Algoritmus v krátkosti popíšem.

Na začiatok je si potrebné pripraviť dáta. Z obrázkov si potrebujeme vytvoriť jednorozmerný vektor. Dosiahneme to tak, že obrázok o rozmeroch $K \times K$ *sploštíme* do K^2 -dimenzionálneho vektoru. Spravíme to pre každý obrázok. Jednorozmerné vektory sa uložia do matice o rozmeroch $Z \times K^2$, kde Z je počet obrázkov v datasete.

Následne sa aplikuje na maticu algoritmu PCA. Dostaneme tak EigenFace vektory. Tie značia dimenzie, v ktorých sa tváre líšia najviac. Zachytávajú tak rozdiely v štruktúrach tváre. Vizualizáciu je možné vidieť na obrázku 2.6. Identifikácia sa vykonáva pomocou týchto vektorov.



Obr. 2.6: Vizualizácia EigenFace vektorov. Svetlejšie časti značia väčší rozptyl a tmavšie menší alebo žiadny [43].

Novú tvár môžeme reprezentovať ako skalárny súčin medzi splošteným vstupným obrázkom tváre a N EigenFace vektorov. Každá tvár je tak reprezentovaná ako lineárna kom-

binácia hlavných komponentov. Pre identifikáciu sa použije euklidovská vzdialenosť, medzi takýmito vektormi.

Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) je algoritmus učenia s učiteľom pre redukciu dimenzionality dát. Metóda LDA sa snaží maximalizovať vzdialenosť stredných hodnôt a minimalizovať rozptyly jednotlivých tried. Algoritmus tak znižuje prekrytie jednotlivých tried a tým zlepšuje výsledky klasifikácie. Ak chceme použiť algoritmus LDA predpokladáme, že vstupné dáta majú normálne rozdelenie, inak nemusíme dostať dobré výsledky.

LDA je možné tiež využiť pre rozpoznávanie tvári a to pomocou algoritmu **FisherFaces** [46]. Vylepšuje algoritmus EigenFaces. Algoritmus extrahuje hlavné komponenty, ktoré odlišujú jednú osobu od druhej. Algoritmus najprv aplikuje PCA a následne LDA, ktoré je tiež známe ako Fisher Linear Discriminang (FLD). Možeme to vyjadriť matematicky ako:

$$W = W_{fld}^T W_{pca}^T \tag{2.4}$$

,

kde W_{pca} je matica vlastných vektorov, ktoré sú výsledkom algoritmu PCA a W_{fld} je matica vlastných vektorov, ktoré sú výsledkom FLD algoritmu. Výhodou tejto metódy je, že je imúnna voči šumom a vlastnosť jedinca nedominuje rysom iných osôb.

Independent Component Analysis

Independent Component Analysis (ICA) [41] je metóda slúžiaca na separovanie nezávislých zdrojov zo zmiešaného signálu. Na rozdiel od techniky PCA, ktorá sa sústreďuje na maximalizovanie rozptylu dát, metóda ICA sa zameriava na nezávislosť jednotlivých komponentov. Algoritmus ICA má niekoľko obmedzení. Nezávislé komponenty generované technikou ICA musia byť navzájom štatisticky nezávislé, nesmú byť z normálneho rozdelenia a musí ich byť toľko, koľko je pozorovaných zmesí.

Locally Linear Embedding

Locally Linear Embedding (LLE) [34] je metóda nelineárnej redukcie dimenzií dát. Ide o metódu učenia bez učiteľa. Algoritmus LLE sa snaží znížiť n-dimenzionálny priestor, zatiaľ čo sa snaží zachovať geometrické vlastnosti originálnej nelineárnej štruktúry.

Technika LLE najprv hľadá k-najbližších susedov k bodom. Následne aproximuje každý dátový vektor ako váženú lineárnu kombináciu svojich k-najbližších susedov. Nakoniec počíta váhy, ktoré najlepšie rekonštruujú vektor z jeho susedov a potom vytvorí nízkodimenzionálne vektory najlepšie rekonštruované týmito váhami.

Informácie o historických metódach extrakcie príznakov som mimo iného čerpal zo zdroja [25].

2.1.4 Moderné prístupy extrakcie príznakov

Medzi moderné prístupy k extrakcii príznakov patrí využitie príznakových vektorov (embeddingov). Je to spôsob ako reprezentovať diskrétne (kategorické) hodnoty pomocou vektorov so spojitými hodnotami. Majú tú výhodu, že dokážu redukovať dimenzionalitu kategorických premenných a rozumne ich reprezentovať v transformovanom priestore.

Embeddingy sú reprezentované funkciou $f(x) \in \mathbb{R}^d$. Funkcia mapuje obrázok x do d-dimenzionálneho euklidovského priestoru.

Hlavným dôvodom pre použitie príznakových vektorov je hľadanie najbližších susedov v priestore embeddingov. Hodí sa to napríklad pri zhlukovaní kategórií , alebo vytváraní odporúčaní.

Embeddingy prekonávajú limitáciu techniky *one-hot encoding* kategorických premenných. Technika *one-hot encoding* má tú nevýhodu, že pre obrovské množstvo tried potrebujeme obrovský vektor, ktorý by ich kódoval. Ďalšou nevýhodou je, že neposkytuje informáciu o blízkosti jednotlivých kategórií.

2.1.5 Porovnanie príznakov

Úlohou porovnania príznakov je povedať, či sa jedná o tú istú osobu, ak máme na porovnanie dve obrázky (overenie identity osoby) alebo kto sa na obrázku nachádza (určenie identity osoby). Prístupy porovnania príznakov môžeme preto rozdeliť na dve skupiny a to porovnanie pomocou metrík na základe vzdialenosti alebo klasifikáciu.

Metriky na základe vzdialeností

Pre určenie podobnosti dvoch príznakových vektoroch môžeme použiť metódu euklidovskej vzdialenosti alebo kosínusovej podobnosti, ktoré teraz popíšem.

Euklidovská vzdialenosť je prístup, ktorý hľadá najmenšiu vzdialenosť medzi dvoma rôznymi príznakovými vektormi. Táto metóda je vhodná pre datasety, ktoré majú menší počet tried a dáta majú menšiu dimenzionalitu.

Kosínusová podobnosť je veľmi podobná metóde pracujúcej s euklidovskou vzdialenosťou. Rozdiel je v tom, že nezisťujeme vzdialenosť medzi dvoma vektormi, ale počítame medzi nimi kosínus uhla. Čím je výsledok bližší k jednotke, tak tým je väčšia pravdepodobnosť zhody.

Používanie euklidovskej vzdialenosti alebo kosínusovej podobnosti sa používa hlavne pri detekcii, či daná tvár patrí danému človeku, alebo či sa jedná o tú istú osobu na rôznych obrázkoch.

Klasifikácia

Úlohou klasifikácie je povedať, kto je na obrázku. Trvá to len niekoľko sekúnd. Existuje množstvo rôznych druhov klasifikátorov. Spomeniem len niekoľko spojených predovšetkým s rozpoznávaním tvárí.

Support vector machine (SVM) je algoritmus, ktorý sa snaží nájsť hyper-rovinu v n-dimenzionálnom priestore, ktorý jasne klasifikuje jednotlivé dáta. Existuje mnoho možností ako nájsť správne hyper-roviny. Pri metóde SVM sa snažíme nájsť rovinu s najväčším okrajom, t. j. s najväčšou vzdialenosťou medzi dátami dvoch tried. Klasifikátor tak má väčšiu istotu pri rozhodovaní [15].

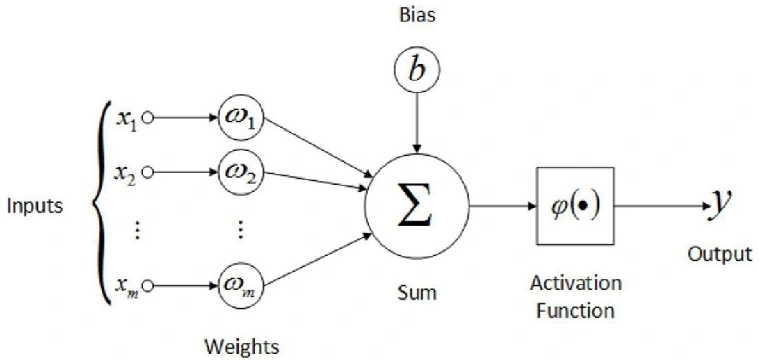
K-najbližších susedov je neparametrický klasifikátor. Algoritmus závisí na čísle k, ktoré si je možné zvoliť ľubovoľne. Dáta patria do triedy, ktorá ma majoritné zastúpenie medzi k-najbližšími susedmi. Okrem tvrdého priradenia je možné urobiť aj mäkké priradenie, kedy vytvoríme škálu, s akou určitosťou vieme tvrdiť, že dáta patria do danej triedy. Algoritmus k-najbližších susedov je možné využiť spolu s technikou PCA, ktorý vyrieši problémy spojené s dimenzionalitou.

Ak nemáme dostatok dát na vytvorenie klasifikátora, môžeme využiť príznakové vektory. Vieme ich vygenerovať pomocou dostupných nástrojov (Dlib) a následne pomocou euklidovskej vzdialenosti alebo kosínusovej podobnosti napríklad určiť z obrázkov, či sa jedná o tú istú osobu [45].

2.2 Neurónové siete

Neurónová sieť je sieť prepojených vrstiev skladajúcich sa z malých častí nazývaných umelé neuróny, ktorá vykonáva matematické operácie vďaka ktorým nachádza vzory v dátach. Neurónová sieť je používaná pre klasifikáciu, predikcie a tak ďalej. Umelý neurón je inšpirovaný skutočným biologickým neurónom, jeho štruktúrou a aj funkciou. Zdrojom informácií o neurónových sietiach boli [27] a [59].

Umelý neurón má ku každému vstupu priradenú váhu. V bázovej funkcii vstupy s váhami vynásobí a sčíta. K tomuto sa pridáva ešte bias. Výsledok bázovej funkcie je vstupom pre aktivačnú funkciu, ktorá pridáva do neurónu nelinearitu. Schéma neurónu je ilustrovaná na obrázku 2.8.



Obr. 2.7: Umelý neurón [7].

Matematicky môžeme umelý neurón vyjadriť následovne:

$$x \longmapsto \sum_{i=0}^{n} w_i x_i + b = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n + b = \mathbf{w} \cdot \mathbf{x} + b \tag{2.5}$$

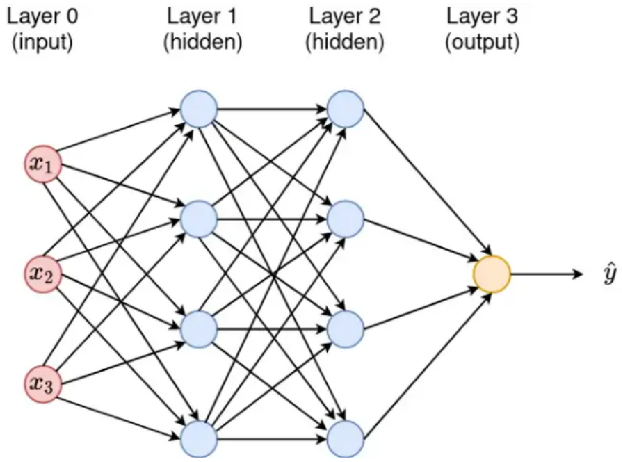
$$\hat{y} = g(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.6}$$

kde $\mathbf{w}$ je vektor váh, $\mathbf{x}$ je vektor vstupu, b je bias a g je aktivačná funkcia.

Neurónová sieť je tvorená z niekoľkých neurónov zoradených vo vrstvách. Poznáme vstupnú vrstvu, výstupnú vrstvu a niekoľko vrstiev medzi nimi, ktoré sa nazývajú skryté vrstvy. Vstupom neurónovej siete sú n-dimenzionálne dáta. Vstupom pre každú skrytú vrstvu je výstup predchádzajúcej vrstvy. Posledná vrstva nám dáva výsledok, t. j. odhadovanú triedu pre daný vstup. Obrázok 2.8 ilustruje neurónovú sieť o veľkosti 3-4-4-1.

Pre učenie siete je potrebné mať chybovú funkciu. Chybová funkcia určuje odhad toho ako zle sa darí sieti pri odhadovaní výsledkov pre dané vstupy. Úlohou učenia je minimalizovať túto funkciu.

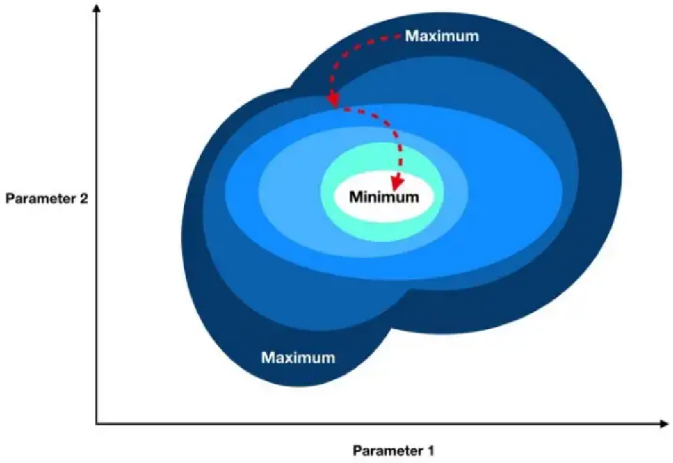
Pre problém minimalizovania sa používa **gradientný zostup**. Gradient funkcie je vektor, ktorého jednotlivé elementy sú parciálnymi deriváciami vzhľadom na každý parameter. Gradient nám povie ako sa zmení chybová funkcia, keď aplikujeme malú zmenu na určitý



Obr. 2.8: Neurónová sieť so štyrmi vrstvami o veľkosti 3-4-4-1. Sieť má jednu vstupnú, dve skryté a jednu výstupnú vrstvu [27].

parameter. Vieme sa takto jednoducho dostať k minimu funkcie. Myšlienka gradientného zostupu je ilustrovaná na obrázku 2.9.

Pomocou algoritmu **backpropagation** vieme vypočítať aký má podiel každý neurón siete na chybe a opraviť váhy s biasmi, tak aby sa im lepšie darilo pri odhadoch. Propagovanie chyby začína pri výstupnej vrstve a postupuje naspäť sieťou až k vstupnej vrstve. Veľkosť chyby špecifického neurónu je priamo úmerná vplyvu daného neurónu na výstup.



Obr. 2.9: Gradientný zostup [59].

2.2.1 Aktivačné funkcie

Aktivačná funkcia určuje, či má byť neurón aktivovaný alebo nie. Existuje niekoľko druhov aktivačných funkcií [49]. Najznámejšie aktivačné funkcie stručne popíšem.

Binárna funkcia

Binárna funkcia je jedna z najjednoduchších aktivačných funkcií. Jej výsledkom je 0 (vstup nepatrí do triedy), alebo 1 (vstup patrí do triedy). Umožňuje tvrdé klasifikovanie do tried.

$$f(x) = \begin{cases} 0 & \text{pre } x < 0 \\ 1 & \text{pre } x \geq 0 \end{cases} \tag{2.7}$$

Sigmoida

Sigmoida naberá hodnoty od 0 až po 1. Vďaka tejto vlastnosti môžeme odhadovať pravdepodobnosť zaradenia do triedy. Používa sa pri binárnej klasifikácií.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

Softmax

Softmax [14] je generalizácia sigmoidy. Používa sa pre viac-triednu klasifikáciu.

$$f(x) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{2.9}$$

TanH

TanH (Hyperbolický tangens) je funkcia podobná logistickej sigmoide, ale jej obor hodnôt je od -1 po 1. Výstup funkcie je symetrický okolo nuly, čo umožňuje rýchlejšiu konvergenciu [1].

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.10}$$

ReLU

ReLU (Rectified Linear Unit) je jedna z najviac používaných aktivačných funkcií. Vo veľkom sa používa v konvolučných neurónových sieťach. Výhodou tejto funkcie je (napr. oproti sigmoide alebo TanH), že rieši problém miznúceho gradientu (miznutie chyby pri spätnej propagácií veľmi hlbokou neurónovou sieťou).

$$f(x) = \begin{cases} 0 & \text{pre } x < 0 \\ x & \text{pre } x \geq 0 \end{cases} \tag{2.11}$$

2.2.2 Chybové funkcie

Úlohou chybovej funkcie je určiť, ako veľmi sa líši výstup siete od vstupu (alebo triedy, do ktorej patrí vstup). Trénovanie neurónovej siete je ňou vedené. Poznáme niekoľko rôznych chybových funkcií. Najznámejšie popíšem.

Stredná kvadratická chyba

Stredná kvadratická chyba [48] (Mean Squared Error) sa používa v regresných problémoch. Chyba je počítaná na základe vzdialenosti medzi výstupom siete a vstupom. Funkcia počíta štvorcové rozdiely medzi týmito hodnotami. Tento prístup umožňuje viacej penalizovať malé chyby. Na druhú stranu ešte viacej penalizuje okrajové hodnoty, takže je voči ním menej robustná. Matematické vyjadrenie:

$$L_{MSE} = \frac{\sum_{i=0}^{n}(y - y')^2}{n}, \tag{2.12}$$

kde y je aktuálny vstup, y' je výstup siete a n je počet odhadov.

Krížová entropia

Krížová entropia (Cross entropy) [28] sa používa pri klasifikácií. Funkcia hodnotí, ako ďaleko je rozdelenie pravdepodobnosti odhadovanej triedy od rozdelenia pravdepodobnosti triedy vstupu. Môžeme povedať, že funkcia meria vzdialenosť medzi dvoma rozdeleniami pravdepodobnosti. Matematické vyjadrenie:

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i), \tag{2.13}$$

kde t_i je pravdivostné označenie, p_i je odhad pravdepodobnosti príslušnosti k i-tej triede a n je počet tried.

2.2.3 Konvolučná neurónová sieť

Konvolučná neurónová sieť (KNS) je metóda hlbokého strojového učenia. KNS zoberie vstupný obrázok, priradí dôležitosť rôznym aspektom alebo objektom na ňom, a je schopná medzi nimi následne rozlišovať. Na rozdiel od klasickej neurónovej siete je KNS schopná zachytávať priestorové a prechodné závislosti aplikovaním rôznych filtrov. Informácie o konvolučných neurónových sietiach som čerpal zo zdroja [44].

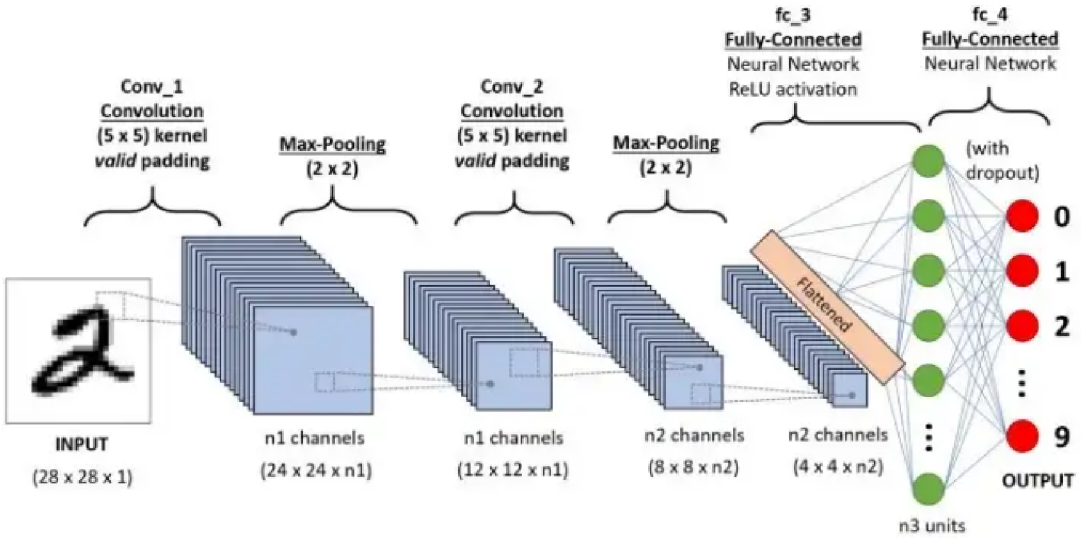
Konvolučná neurónová sieť sa skladá z niekoľkých vrstiev. Ich počet určuje ako je sieť hlboká. Poznáme rôzne druhy vrstiev ako napríklad konvolučná, pooling alebo plne prepojená vrstva. Tieto vrstvy môžu byť rôzne usporiadané, s rozličnými veľkosťami filtrov a ich počtom, alebo môžu obsahovať dokonca medzi sebou skratky. Poznáme preto niekoľko rôznych typov architektúr napríklad: LeNet, AlexNet, VGGNet, GoogleNet alebo ResNet. Príklad jednoduchej neurónovej siete je ilustrovaný na obrázku 2.10.

Konvolučná vrstva

Základom konvolučnej vrstvy je operácia *konvolúcie*, ktorá je matematicky definovaná následovne [10]:

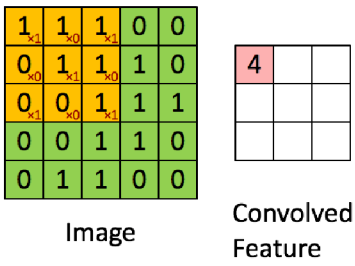
$$(f * h)(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x - i, y - j) \cdot h(i, j), \tag{2.14}$$

kde h je konvolučné jadro a f je vstup. Konvolučné jadro (alebo filter) obsahuje váhy, ktoré sú predmetom učenia. Jadro sa priloží k obrázku, jeho váhy sa vynásobia s pixelami obrázku, výsledok sa sčíta, zapíše a konvolučné jadro sa posunie o stanovenú hodnotu.



Obr. 2.10: Jednoduchá konvolučná neurónová sieť [44].

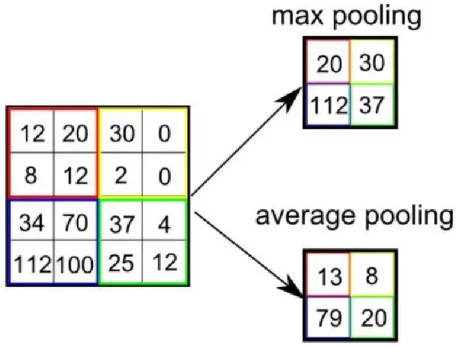
Výsledkom konvolúcie je zredukovaný výstup príznakov. Aby sa tomuto problému predišlo, tak sa na vstup aplikuje okraj. Ukážka jedného kroku konvolúcie je znázornená na obrázku 2.11. V praxi sa používa pri konvolúcií viacej *kanálov* (pre každú farbu RGB obrázku). Pre túto vrstvu sa zvykne používať ReLU aktivačná funkcia.



Obr. 2.11: Jeden krok konvolúcie na obrázku s rozmermi 5x5x1 s jadrom 3x3x1, kde dostávame príznaky o veľkosti 3x3x1 [44].

Pooling vrstva

Úlohou pooling vrstvy je extrahovať dominantné príznaky a znížiť výpočtovú náročnosť pomocou redukcie dimenzií. Poznáme dva druhy pooling vrstiev a to *max pooling* (vyberá najvyššiu hodnotu z časti obrázku pokrytú jadrom) a *average pooling* (vracia priemer z hodnôt zachytené jadrom). Pooling vrstva nemá učiace sa parametre. Rozdiel je znázornený na obrázku 2.12.



Obr. 2.12: Rozdiel medzi max a average pooling vrstvou [44].

Plne prepojená vrstva

Posledná vrstva býva plne prepojená. Používa sa na klasifikáciu. Vstup z konvolučných alebo pooling vrstiev je *sploštený* do stĺpcového vektoru a stáva sa vstupom do plne prepojenej neurónovej siete. Táto vrstva môže mať niekoľko skrytých vrstiev. Ako aktivačná funkcia pre skryté vrstvy sa využíva ReLU. Výstupná vrstva plne prepojenej vrstvy využíva aktivačnú funkciu Softmax, ktorá umožňuje viac-triednu klasifikáciu.

2.3 Algoritmy pre rozpoznávanie podľa tvárí

V tejto časti uvádzam známe algoritmy pre rozpoznávanie osôb podľa tvárí, ktoré sú založené na neurónových sieťach. Vybral som konkrétne tri prístupy a to DeepFace, FaceNet a nakoniec ArcFace, na ktorom je postavené moje riešenie.

2.3.1 DeepFace (Facebook)

DeepFace [51] je model vytvorený spoločnosťou Facebook v roku 2014. Algoritmus je určený pre rozpoznávanie tvárí osôb. Model DeepFace v kroku zarovnávania používa 3D modelovanie tváre. Algoritmus implementuje hlbokú neurónovú sieť so 120 miliónmi parametrami, používajúcu niekoľko lokálne prepojených vrstiev bez zdieľania parametrov. Sieť je trénovaná na obrovskom datasete so štyri miliónmi obrázkami tvárí s vyše 4 000 identitami. Sieť sa blíži svojou úspešnosťou výkonu ľudí.

Prínosy tohto modelu sú, že implementuje efektívnu hlbokú neurónovú sieť, metódu učenia, ktorá je schopná využiť obrovský dataset a dobre generalizuje pre iné dáta. Model zavádza efektívny systém zarovnania tvárí založený na 3D modelovaní tvárí.

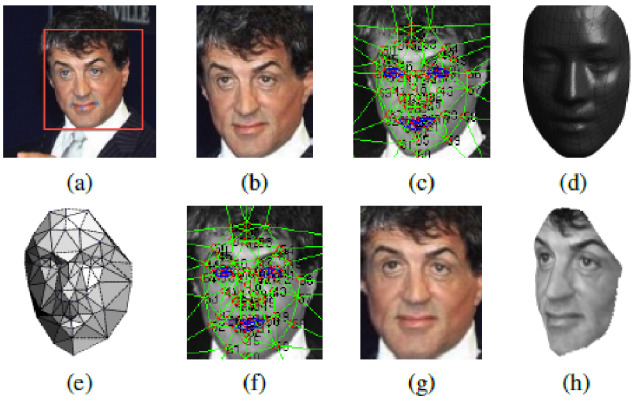
3D zarovnanie tváre

Proces sa začína 2D zarovnaním. Na začiatku algoritmus detekuje vo výreze tváre zarovnaného na stred 6 fiduciálnych bodov: stred očí, špičku nosa a ústa. Body sú znázornené na obrázku 2.13(a). Tieto body sa používajú na škálovanie obrázku a rotáciu pri zarovnaní. Výsledok zarovnania je ilustrovaný na obrázku 2.13(b). Tento prístup má ale problém s rotáciami mimo roviny.

Problém je riešený 3D zarovnaním. Základom je nájdenie 67 fiduciálnych bodov v 2D zarovnanom výreze, ako je ilustrovane na obrázku 2.13(c). Zároveň sa vygeneruje 3D model

s použitím generického algoritmu prevodu 2D na 3D model a manuálne sa aplikuje naň nájdené fiduciálne body. Následne sa skúsi aplikovať relácia medzi 2D a 3D modelom [40].

Posledným krokom je frontalizácia zarovnaného obrázku. Predtým je potrebné pridať residuálny komponent k súradniciam 3D modelu, aby sa znížila jeho deformácia. Frontalizácia je vykonaná aplikovaním afínej transformácie na jednotlivé časti riadenej Delaunayovou triangulációu generovanej na 67 fiduciálnych bodoch [40].



Obr. 2.13: Proces zarovnania tvárí použitý v modeli DeepFace [51].

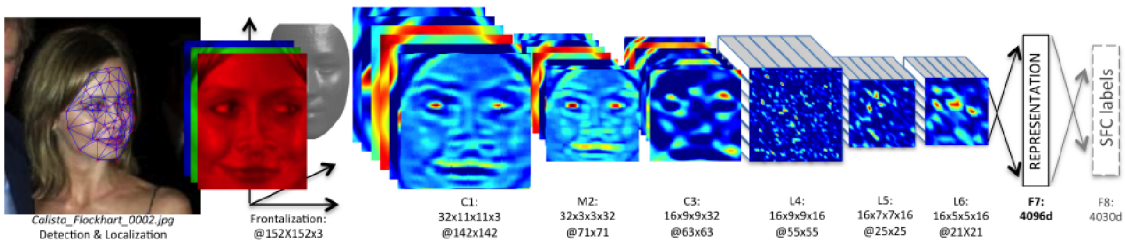
Neurónová sieť

Architektúra použitej neurónovej siete je znázornená na obrázku 2.14. Sieť pozostáva z ôsmich vrstiev. Prvé tri vrstvy sú predspracovacie. Vykonávajú najviac výpočtu, ale uchovávajú veľmi málo parametrov. Tieto vrstvy, len expandujú vstup na množinu jednoduchých lokálnych príznakov.

Vrstvy L4, L5 a L6 sú lokálne prepojené. Každé miesto v mape príznakov sa učí iná sada filtrov. Posledné dve vrstvy sú plne prepojené. Tieto vrstvy zachytávajú korelácie medzi príznakmi zachytenými vo vzdialenejších častiach tvárí (polohu a tvar očí, či úst).

Na výstupe siete je aktivačná funkcia softmax. Ako chybová funkcia sa používa cross-entropy loss. Pri učení sa využíva metóda stochastického gradientného zostupu.

Sieť bola trénovaná na datasete *Social Face Classification* (SFC), ktorý obsahuje 4.4 miliónov označených tvárí s 4 030 identitami. Na jedného človeka tak pripadá 800 až 1 200 tvárí.



Obr. 2.14: Architektúra neurónovej siete použitej v modeli DeepFace [51].

Výsledky v benchmark testoch

Testy boli vykonané na benchmark datasetoch ako LFW a SFC. Model DeepFace na datasete LFW dosiahol presnosť 97,00% a na datasete YTF presnosť 91,4%. Podrobnejšie výsledky je možné vidieť na obrázkoch 2.15 a 2.16.

Method	Accuracy ± SE	Protocol
Joint Bayesian [6]	0.9242 ±0.0108	restricted
Tom-vs-Pete [4]	0.9330 ±0.0128	restricted
High-dim LBP [7]	0.9517 ±0.0113	restricted
TL Joint Bayesian [5]	0.9633 ±0.0108	restricted
DeepFace-single	**0.9592** ±0.0029	unsupervised
DeepFace-single	**0.9700** ±0.0028	restricted
DeepFace-ensemble	**0.9715** ±0.0027	restricted
DeepFace-ensemble	**0.9735** ±0.0025	unrestricted
Human, cropped	0.9753	

Obr. 2.15: Výsledky modelu DeepFace na datasete LFW [51].

Method	Accuracy (%)	AUC	EER
MBGS+SVM- [31]	78.9 ±1.9	86.9	21.2
APEM+FUSION [22]	79.1 ±1.5	86.6	21.4
STFRD+PMML [9]	79.5 ±2.5	88.6	19.9
VSOF+OSS [23]	79.7 ±1.8	89.4	20.0
DeepFace-single	**91.4** ±1.1	96.3	8.6

Obr. 2.16: Výsledky modelu DeepFace na datasete YTF [51].

2.3.2 FaceNet (Google)

FaceNet je model určený pre verifikáciu, rozpoznávanie tvárí a zoskupovanie identít. Bol navrhnutý v roku 2015 výskumníkmi zo spoločnosti Google v práci [47]. Algoritmus dokáže rozoznávať tváre v nefrontálnych pozíciách.

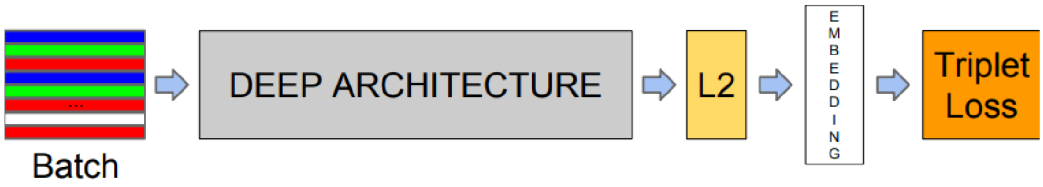
Model FaceNet sa učí mapovať obrázky tvárí do kompaktného euklidovského priestoru. Vzdialenosti priamo korešpondujú meraniu podobnosti tvárí. Metóda používa hlbokú konvolučnú neurónovú sieť, ktorá priamo optimalizuje embeddingy namiesto toho, aby systém používal špeciálne vrstvy na toto určené. Výhodou je, že sa tým odstráni možné slabé miesto v sieti.

Pre trénovanie systém FaceNet používa *trojice* dát s pozitívnymi a negatívnymi prípadmi. Tieto trojice sú generované novou *online* metódou. Tento prístup má veľký prínos vo forme reprezentačnej efektívnosti, keďže pre trénovanie je potrebných iba 128 bajtov na jednu tvár.

Klasifikácia je veľmi jednoduchá. Pri verifikácii je potrebné nájsť správny prah, ktorý nám dá správne výsledky. Rozpoznávanie je k-NN klasifikačný problém a zhlukovanie je možné dosiahnuť pomocou techník k-means alebo aglomeračného zhlukovania.

Popis algoritmu

Architektúru modelu je možné vidieť na obrázku 2.17. Konvolučná sieť je braná ako black-box. Ide o end-to-end učenie celého systému.



Obr. 2.17: Architektúra modelu FaceNet [47].

Model FaceNet používa hlbokú konvolučnú neurónovú sieť. Opiera sa o dve architektúry: Zeiler&Fergus a Inception. V pôvodnej práci autori experimentujú s viacerými variáciami týchto sietí. Príklad siete postavenej na architektúre Inception je ilustrovaný na obrázku 2.18.

type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

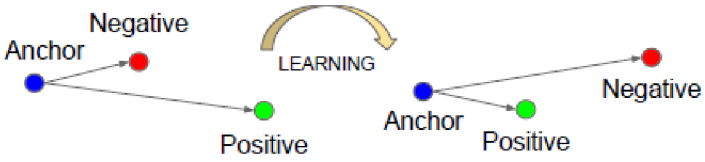
Obr. 2.18: Použitá konvolučná neurónová sieť v modeli FaceNet založená na architektúre Inception [47].

V algoritme sa používajú embeddingy. Tie sa nachádzajú na d-dimenzionálnej hypersfére.

Trojice sú reprezentované pomocou kotvy x_i^a (anchor), pozitívneho prípadu x_i^p (rovnaká osoba) a negatívneho prípadu x_i^n (iná osoba). Triplet loss minimalizuje vzdialenosť medzi kotvou a pozitívnym prípadom a maximalizuje vzdialenosť medzi kotvou a negatívnou identitou. Ilustrácia je na obrázku 2.19

Formálne to môžeme zapísať:

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2, \forall\, (f(x_i^a), f(x_i^p), f(x_i^n)) \in \tau, \tag{2.15}$$



Obr. 2.19: Znázornenie fungovania triplet loss [47].

$$L = \sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+ , \tag{2.16}$$

kde α je okraj medzi pozitívnymi a negatívnymi pármi, τ je množina všetkých možných trojíc v trénovacom datasete a N je veľkosť datasetu.

Aby sme pri učení dosiahli rýchlu konvergenciu, tak je dôležité vybrať trojice, ktoré porušujú jej obmedzenie definované v rovnici 2.15. To znamená, že sa snažíme maximalizovať vzdialenosť medzi kotvou a pozitívnym prípadom a súčasne minimalizovať vzdialenosť medzi kotvou a negatívnym prípadom.

Model FaceNet pri generácií trojíc používa *online* prístup. Znamená to, že algoritmus vyberá pozitívne a negatívne prípady len v rámci mini-dávky (obsahujúca tisíce prípadov).

Výsledky v benchmark testoch

Konvolučná neurónová sieť použitá v modelu FaceNet bola trénovaná pomocou *stochastického gradientného zostupu*, štandardným algoritmom *backpropagation* a algoritmom *AdaGrad*.

Sieť bola testovaná na štandardných datasetoch LFW a YTF. Model FaceNet dosahuje na datasete *LFW* presnosť *99,63%*. Na datasete *YTF* je to presnosť *95,12%*.

2.3.3 ArcFace

Additive Angular Margin Loss (ArcFace) je technika strojového učenia predstavená v článku [12]. Metódu je možné využiť v rozpoznávacích, ale aj generatívnych problémoch. Model dokáže rozoznávať tváre v nefrontálnych pozíciách.

Metóda ArcFace využíva geodetické vzdialenosti namiesto euklidovských. Model optimalizuje vzdialenosti medzi jednotlivými triedami na hypersfére. Algoritmus sa tak snaží zvýšiť kompaktnosť v rámci triedy a zvýrazniť rozdiely medzi jednotlivými triedami.

Algoritmus ArcFace je jednoduchý na implementáciu. Existuje už niekoľko implementácií ako napríklad InsightFace [20] alebo TensorFlow2 implementácia [24]. Algoritmus nie je potrebné kombinovať s inými chybovými funkciami, aby bolo možné dosiahnuť stabilnú konvergenciu.

Technika je taktiež efektívna. Autori rozširujú existujúce metódy strojového učenia o časti, ktoré majú zanedbateľnú výpočtovú náročnosť počas trénovania. Dosahuje tiež dobré výsledky na datasetoch, ktoré sa používajú ako benchmark.

Riešenie je robustné voči zašumeniu v prípadoch z reálneho sveta a to vďaka predstaveniu systému *podtried*.

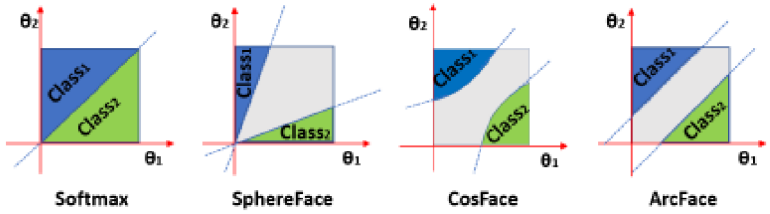
Porovnanie s ostatnými prístupmi

Pri problémoch rozpoznávania tvárí sa používajú dve hlavné prístupy. Prvým prístupom je trénovanie viactriedneho klasifikátora, ktorý dokáže rozlišovať medzi viacerými identitami. Tento prístup používa klasifikátor softmax. Druhým spôsobom je naučiť sa priamo *embeddingy* pomocou techniky *triplet loss*.

Tieto prístupy majú ale niekoľko nevýhod. Klasifikátor softmax funguje dobre pre uzatvorenú sadu tvárí. Veľkosť lineárnej transformačnej matice $W \in \mathbb{R}^{d \times N}$ sa lineárne zväčšuje s N. *Triplet loss* má problém s obrovským množstvom kombinácií, ktoré môžu vzniknúť vo veľkých datasetoch. Vzniká tak veľký nárast vo výpočtovej náročnosti.

Exitujú ďalšie prístupy podobné Arcface. Sú nimi *SphereFace* a *CosFace*. Model SphereFace predstavuje myšlienku *uhlového okraju*. Jeho chybová funkcia potrebuje radu aproximácií, ktoré spôsobujú nestabilné trénovanie siete. Aby sa tomuto problému predišlo, tak model SphereFace používa hybridnú chybovú funkciu, ktorá zahŕňa štandardnú chybovú funkciu softmax. Model ArcFace tento problém rieši pomocou *additive angular margin loss*.

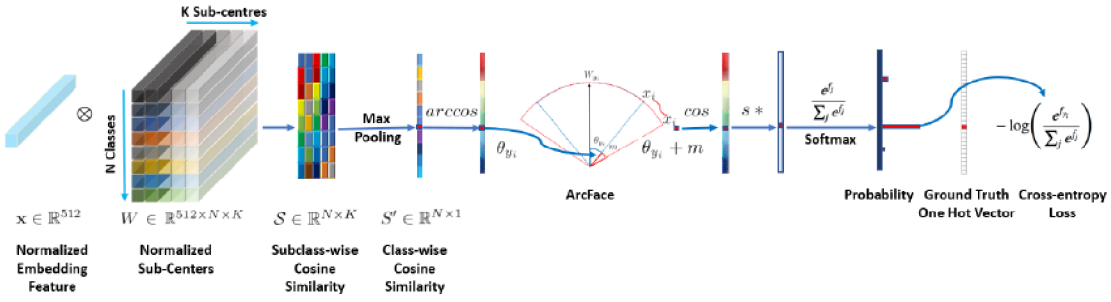
Medzi týmito modelmi môžeme pozorovať geometrický rozdiel. Spočíva v tvare rozhodovacej hranice a okraju. Rozdiely je možné vidieť na obrázku 2.20. Model ArcFace má konštantný lineárny uhlový okraj, zatiaľ čo modely *SphereFace* a *CosFace* majú nelineárny uhlový okraj. Tieto rozdiely môžu mať signifikantný vplyv na trénovanie modelu.



Obr. 2.20: Rozhodovacie okraje rôznych chybových funkcií. Prerušovaná modrá čiara značí rozhodovaciu hranicu a šedá časť rozhodovací okraj [12].

Popis algoritmu

Schéma návrhu algoritmu ArcFace je zobrazená na obrázku 2.21.



Obr. 2.21: Schéma architektúry metódy ArcFace [12].

Model ArcFace používa chybovú funkciu softmax:

$$L_1 = -\log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{N} e^{W_j^T x_i + b_j}}, \tag{2.17}$$

kde $x_i \in \mathbb{R}^d$ značí príznaky i-tého prípadu patriaceho do y_i-tej triedy. Dimenzia príznakového vektoru d je 512. $W_j \in \mathbb{R}^d$ značí j-ty stĺpec váhy $W \in \mathbb{R}^{d \times N}$, $b_j \in \mathbb{R}^N$ je bias a N je číslo triedy.

Pre jednoduchosť môžeme bias dať rovný nule, $b_j = 0$. Následne môžeme transformovať softmax logit do novej podoby, ktorá zohľadňuje geodetické vzdialenosti.

$$W_j^T x_i = \|W_j\| \|x_i\| \cos\theta_j \tag{2.18}$$

kde θ_j je uhol medzi váhou W_j a príznakom x_i. $\|W_j\|$ a $\|x_i\|$ normalizujeme, takže sa budú rovnať jednej. Zmeníme veľkosť na s, čo je polomer hypersféry. Dostaneme tak $W_j^T x_i = s\cos\theta_j$, čo môžeme dosadiť do chybovej funkcie softmax následovne:

$$L_2 = -\log \frac{e^{s\cos\theta_{y_i}}}{e^{s\cos\theta_{y_i}} + \sum_{j=1, j\neq y_i}^{N} e^{s\cos\theta_j}} \tag{2.19}$$

Keďže príznaky sú rozmiestnené okolo každého stredu príznakov na hypersfére, algoritmus ArcFace pridáva *additive angular margin penalty* m medzi x_i a W_y. Týmto zvyšuje kompaktnosť v rámci triedy a zväčšuje rozdiely medzi jednotlivými triedami. Toto je možné vidieť v nasledovnej rovnici:

$$L_3 = -\log \frac{e^{s\cos(\theta_{y_i}+m)}}{e^{s\cos(\theta_{y_i}+m)} + \sum_{j=1, j\neq y_i}^{N} e^{s\cos\theta_j}} \tag{2.20}$$

Podtriedy

Technika ArcFace dosahuje dobré výsledky na kontrolovaných dátach. Aby sa jej dobre darilo aj pri zašumených dátach reálneho sveta, metóda ArcFace je obohatená o podtriedy. Porovnanie s pôvodným modelom ArcFace a s inými metódami je ilustovaný na obrázku 2.22.

Každá trieda má K podcentier. Stačí, ak trénovací prípad bude blízko ktoréhokoľvek z týchto centier namiesto jediného centra. V pôvodnom algoritme ArcFace, by zašumené trénovacie dáta nepatrili do žiadnej triedy. Generovali by tak veľké chyby, ktoré by ovplyvnili trénovanie modelu. Systém podtried nevyžaduje, aby trénovacie dáta patrili všetkým podcentrám, ale stačí ak budú patriť len niektorým. Zašumené dáta tak vytvoria nedominantnú podtriedu.

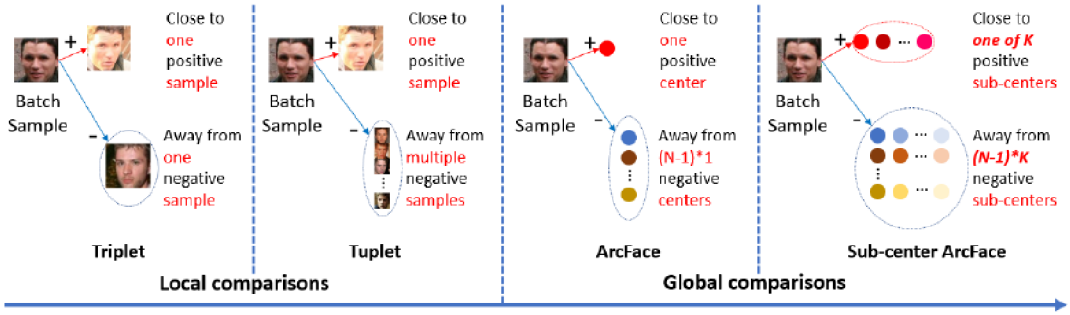
Túto myšlienku môžeme matematicky formulovať:

$$L_3 = -\log \frac{e^{s\cos(\theta_{y_i}+m)}}{e^{s\cos(\theta_{y_i}+m)} + \sum_{j=1, j\neq y_i}^{N} e^{s\cos\theta_j}}, \tag{2.21}$$

kde $\theta_j = \arccos\left(\max_k \left(W_{jk}^T x_i\right)\right), k \in \{1, \cdots, K\}$.

Výsledky v benchmark testoch

Model ArcFace bol testovaný na rôznych datasetoch ako LFW, YTF, CFP-FP, CPLFW, AgeDB, CALFW. Pre prehľadnosť v tejto práci uvediem verifikačné výsledky z datasetov



Obr. 2.22: Porovnanie techník Triplet, Tuplet, ArcFace a podtried ArcFace [12].

LFW a YTF, ktoré sú najviac používané pre benchmark testy. Úplné výsledky testov je možné nájsť v pôvodnej práci [12].

Model bol trénovaný na datasetoch MS1MV3 a IBUG-500K s ResNet100. Konkrétne dataset IBUG-500K obsahuje obrázky s rôznymi natočeniami tváre. Model dosahoval úspešnosť 99,8% na datasete LFW a 98,0% na datasete YTF. V tabuľke na obrázku 2.23 je možné vidieť, že prekonala modely SphereFace a CosFace na oboch uvedených datasetoch. Ako je možné vidieť z výsledkov, algoritmus ArcFace sa doťahuje na špičku najlepších modelov pre rozpoznávanie tvárí.

2.4 Datasety

Dataset v ponímaní strojového učenia predstavuje množinu dát, ktorá sa používa na trénovanie siete, jej evaluáciu a testovanie. Pri probléme rozpoznávania tvárí dataset obsahuje obrázky tvárí s ich identitami a potrebnými anotáciami (pohlavie, vek, príslušnosť k etniku, uhol natočenia tváre a ďalšie).

Dataset je možné podľa potreby rozdeliť na trénovacie, validačné a testovacie dáta (napríklad v pomere 80:10:10). Trénovacie dáta sa používajú na priame trénovanie modelu, validačné dáta slúžia pre vybratie čo najlepšieho modelu s najoptimálnejšími hyperparametrami a nakoniec testovacie dáta slúžia pre verifikovanie modelu na nevidených dátach.

Existuje obrovské množstvo datasetov. V následujúcej časti popíšem najznámejšie z nich. Okrem nich spomeniem aj datasety, ktoré sú vhodné pre učenie modelu pre nefrontálne rozpoznávanie tvárí, ktoré majú anotácie natočenia tváre na obrázku.

2.4.1 Labelled Faces in the Wild (LFW)

LFW [23] je používaný ako benchmarkový dataset pre modely. Dataset obsahuje 13 233 obrázkov s počtom 5 749 identít. Dve a viacej obrázkov má 1 680 ľudí. Obrázky tvárí boli zozbierané z internetu.

2.4.2 YouTube Faces DB (YTF)

Podobne ako dataset LFW aj YTF [58] slúži ako benchmarkový dataset. YTF obsahuje 3 425 videí 1 595 rôznych ľudí. Na každého človeka pripadajú približne dve videá. Dáta sú stiahnuté zo služby YouTube.

Method	#Image	LFW	YTF
DeepID [1]	0.2M	99.47	93.20
Deep Face [2]	4.4M	97.35	91.4
VGG Face [4]	2.6M	98.95	97.30
FaceNet [3]	200M	99.63	95.10
Baidu [95]	1.3M	99.13	-
Center Loss [72]	0.7M	99.28	94.9
Range Loss [73]	5M	99.52	93.70
Marginal Loss [17]	3.8M	99.48	95.98
SphereFace [13]	0.5M	99.42	95.0
SphereFace+ [84]	0.5M	99.47	-
CosFace [14]	5M	99.73	97.6
RegularFace [51]	3.1M	99.61	96.7
UniformFace [52]	6.1M	99.8	97.7
DAL [96]	0.5M	99.47	-
FTL [97]	5M	99.55	-
Fair Loss [98]	0.5M	99.57	96.2
Unequal-training [20]	0.55M	99.53	96.04
Noise-Tolerant [19]	1M noisy	99.72	97.36
AdaptiveFace [50]	5M	99.62	-
AFRN [99]	3.1M	**99.85**	97.1
PFE [100]	4.4M	99.82	97.36
DUL [101]	3.6M	99.78	96.78
RDCFace [102]	1.7M	99.80	97.10
HPDA [103]	5M	99.80	-
URFace [104]	5M	99.78	-
CircleLoss [105]	3.6M	99.73	96.38
GroupFace [55]	5.8M	**99.85**	97.8
BioMetricNet [106]	3.8M	99.80	**98.06**
BroadFace [107]	5.8M	**99.85**	98.0
IBUG500K,R100,BroadFace	11.96M	99.83	98.03
MS1MV3, R100, ArcFace	5.1M	99.83	98.02
IBUG500K, R100, ArcFace	11.96M	99.83	98.01

Obr. 2.23: Verifikačné výsledky (v %) rôznych metód na datasetoch LFW a YTF [12].

2.4.3 MS-Celeb-1M

MS-Celeb-1M [22] je dataset obsahujúci vyše 10 miliónov obrázkov (vo svojej verzii 1) so 100 000 identít celebrít zozbieraných z internetu. Dataset pochádza od výskumníkov zo spoločnosti Microsoft.

2.4.4 CelebFaces Attributes

CelebFaces Attributes [32] je dataset, ktorý obsahuje viac ako 200 tisíc obrázkov celebrít s viac ako 10 000 identitami. Každý obrázok má 40 anotácií rôznych vlastností. Osoby na obrázkoch sú v rôznych pózach, a majú za sebou rôzne pozadie.

2.4.5 DigiFace-1M

DigiFace-1M [3] je dataset obsahujúci vyše 1.2 milióna obrázkov so 110 tisíc identitami. Každá identita má k dispozícií 77 obrázkov s tvárami v rôznych prostrediach a pózach.

2.4.6 Head Pose Image Database

Head Pose Img Dataset [18] obsahuje 2 790 obrázkov tvárí s 15 identitami s uhlami natočenia a naklonenia od -90 do 90 stupňov. Pre každú osobu sú dostupné dve série 93

obrázkov (pre trénovanie a testovanie). Vďaka uhlovým anotáciám je tento dataset vhodný pre trénovanie nefrontálnej klasifikácie v tejto práci.

2.4.7 Robotics Lab dataset

Robotics Lab dataset [42] obsahuje 6 660 obrázkov s 90 identitami a anotáciou natočenia tváre od -90 do 90 stupňov. Pre každú osobu je dostupných 37 obrázkov. Vďaka uhlovým anotáciám je tento dataset vhodný pre trénovanie nefrontálnej klasifikácie v tejto práci.

Kapitola 3

Návrh riešenia a implementácia

V tejto kapitole popisujem návrh riešenia. Návrh obsahuje prehľad knižníc a voľbu implementačných nástrojov, ktoré som použil v mojej práci, architektúry, na základe ktorej je riešenie postavené a popis návrhu riešenia.

3.1 Knižnice

Existuje obrovské množstvo knižníc implementujúcich *state-of-the-art* algoritmy strojového učenia na rôznej úrovni. Niektoré knižnice ponúkajú základné algoritmy, iné umožňujú rýchle a efektívne vytváranie prototypov a modelov pre rozpoznávanie osôb podľa tvárí, a ďalšie implementujú komplexné riešenia detekcie a rozpoznávania tvárí. Veľa týchto knižníc využíva programovací jazyk Python alebo C++. Pre prehľadnosť a jednoduchosť zhrniem len niektoré z nich.

3.1.1 Vytváranie modelov

Najznámejšími knižnicami pre vytváranie modelov pre rozpoznávanie podľa tvárí patrí napríklad TensorFlow alebo PyTorch.

TensorFlow

TensorFlow [37] je end-to-end open source platforma pre strojové učenie. Knižnica používa programovací jazyk Python. Framework je vhodný pre začiatočníkov (jednoduché tvorenie modelov), ale aj pokročilých (budovanie modelov od základov). Poskytuje široké možnosti pre riešenie problémov počítačového videnia, textov, audia, generatívnych problémov a mnoho ďalších.

Výhodami tejto knižnice je, že podporuje prácu s viac-dimenzionálnymi poľami, GPU a distribuované spracovanie, automatickú diferenciáciu, vytváranie modelov, ich trénovanie, exportovanie, a ďalšie.

Viacdimenzionálne polia sú reprezentované objektom *tf.Tensor*. Objekt implementuje štandardné matematické operácie ako napríklad sčítanie, násobenie, transpozícia, konkatenácia a tak ďalej. Najdôležitejšími atribútmi sú *Tensor.shape* (veľkosť tensoru pozdĺž všetkých osí) a *Tensor.dtype* (typ elementov tensoru) [36].

Datasety je možné nahrať do špeciálnych súboroch s názvom *TFRecords*. Je to jednoduchý formát pre ukladanie sekvencií binárnych záznamov. Môže byť čítaný len sekvenčne.

Každý záznam obsahuje bajtový reťazec dát, ich dĺžku a hash pre kontrolu integrity. Tento súbor sa potom načíta a použije pri trénovaní [38].

Knižnica umožňuje stiahnutie vlastných datasetov, jednoduché načítanie obrázkov alebo audia, aj s ich triedami. Datasety je možné rozdeliť na trénovacie, validačné a testovacie dáta, podľa dopredu zvolených parametrov. Pri vytváraní modelu stačí zadefinovať jednotlivé vrstvy aj s ich parametrami. Platforma TensorFlow následne umožňuje model trénovať, evaluovať a testovať.

Keras

Keras [8] je knižnica, ktorá je postavená na platforme TensorFlow. Je to high-end knižnica, ktorá umožňuje veľmi rýchlo a jednoducho vytvárať modely strojového učenia. Knižnica Keras sa zameriava hlavne na hlboké učenie. Príklad využitia je na obrázku 3.1.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

Obr. 3.1: Definovanie konvolučnej neurónovej siete pomocou knižnice Keras [35].

PyTorch

PyTorch [13] je end-to-end framework pre strojové učenie, ktorý používa jazyk Python. Knižnica umožňuje rýchle experimentovanie, efektívne nasadenie do produkcie, distribuované trénovanie, podporuje mobilné zariadenia a rozhranie pre jazyk C++.

Framework PyTorch umožňuje stiahnutie a načítanie datasetov aj s ich príslušnými triedami. Užívateľ si vie jednoducho nadefinovať model: vrstvy konvolučnej neurónovej siete, hyperparametre učenia a priebeh samotného trénovania, evaluácie, a testovania. Podobne ako knižnica TensorFlow používa pre prácu s viac-dimenzionálnymi dátami objekt *Tensor*.

3.1.2 Implementácie algoritmov

V následujúcej časti ponúkam ukážku nástrojov, ktoré implementujú rôzne algoritmy a prístupy strojového učenia.

Dlib

Dlib [9] je knižnica implementujúca algoritmy strojového učenia v jazyku C++. Má široké možnosti. Implementuje hlboké neurónové siete, rôzne druhy techniky SVM, zhlukovacie algoritmy ako k-means alebo Chinese Whispers, siete s radiálnymi bázovými funkciami, numerické algoritmy, grafové modely (napr. joint tree), spracovanie obrázkov (HOG, frontálne detekovanie tváre, odhad pózy objektu, rozpoznávanie tvárí) a mnoho ďalších. Knižnica Dlib ponúka Python API, takže je ho možné použiť aj z tohto jazyka.

OpenCV

Open Source Computer Vision Library [53] je open-source knižnica, ktorá ponúka algoritmy pre počítačové videnie implementované v jazyku C++.

Knižnica poskytuje niekoľko modulov implementovaných ako zdieľané alebo statické knižnice. Medzi najvýznamnejšie moduly patria:

- základné funkcionality - základné dátové štruktúry ako multidimenzionálne polia Mat,
- spracovanie obrázkov - lineárne a nelineárne filtrovanie obrázkov, geometrické transformácie,
- analýza videa - odhad pohybu, odstránenie pozadia a algoritmy pre sledovanie objektov,
- kalibrácia kamery a 3D rekonštrukcia - odhad pózy objektu, elementy 3D rekonštrukcie,
- framework pre 2D príznaky,
- detekcia objektov,
- grafické užívateľské rozhranie,
- ostatné ako video I/O, FLANN, Google test wrappers ďalšie [52].

Face Recognition

Knižnica Face Recognition [16] je postavená na knižnici dlib. Knižnica umožňuje manipulovať s tvárovými rysmi, detekovať a identifikovať tváre v obrázkoch. Knižnica je spustiteľná z príkazového riadku alebo pomocou jazyku Python.

InsightFace

InsightFace [20] je open source 2D&3D nástroj pre hlbokú analýzu tvárí. Nástroj je založený hlavne na knižniciach Pytorch a MXNet. Nástroj InsightFace implementuje bohatú paletu štandardných algoritmov pre rozpoznávanie, detekciu tvárí a ich zarovnanie.

V rámci nástroja InsightFace je množstvo projektov, ktoré sa špecializujú na jednotlivé časti procesu rozpoznávania tvárí. Pri detekcii sú to RetinaFace a SCRFD. Algoritmy SDUNet a CoordinateReg pre zarovnanie. Pre rozpoznávanie sú implementované ArcFace, SubCenter-ArcFace, VPL a Partial-FC.

DeepFace

DeepFace [56] je knižnica pre rozpoznávanie tvárí a analýzu vlastností tváre, ako vek, pohlavie, emócie alebo rasa. Knižnica je implementovaná v jazyku Python. DeepFace zahŕňa viacej *state-of-the-art* modelov ako VGG-Face, Google FaceNet, OpenFace, Facebook DeepFace, DeepID, ArcFace, Dlib a SFace. Užívateľ si môže vybrať ktorýkoľvek z týchto modelov. Predvolený je model VGG-Face.

Knižnica pre rozpoznávanie tvárí poskytuje štandardné funkcie pre detekovanie, zarovnanie, normalizáciu, reprezentácie a verifikáciu. Všetky potrebné kroky sa dejú na pozadí. Užívateľ si tieto funkcie vie spustiť jednoduchým volaním. Príklad takéhoto volania uvádzam vo výpise číslo 3.1.

```
from deepface import DeepFace
result = DeepFace.verify(img1_path = "img1.jpg", img2_path = "img2.jpg")
```

Výpis 3.1: Verifikovanie osoby pomocou knižnice DeepFace

Knižnica taktiež zahŕňa rôzne modely pre detekciu tvárí. Medzi ne patria OpenCV, SSD, Dlib, MTCNN, RetinaFace a MediaPipe. Užívateľ si môže podobne ako pri rozpoznávaní tvárí vybrať jeden z nich. Model OpenCV je nastavený ako predvolený. Model RetinaFace a MTCNN je odporúčaný, ak je kľúčovou požiadavkou presnosť detekcie. Nevýhodou týchto modelov je, že sú pomalšie. Ak užívateľovi záleží na rýchlosti, odporúčanými modelmi sú OpenCV alebo SD.

Knižnica DeepFace ponúka možnosť real-time video analýzy. Pre tieto účely sa používa dostupná webkamera. Pri tejto analýze sa zároveň aplikuje rozpoznávanie tvárí a analýza vlastností tváre.

Knižnicu DeepFace je možné štandardne importovať do kódu. Knižnica je k dispozícií taktiež skrz API alebo príkazového riadku.

3.2 Výber nástrojov

V tejto časti uvádzam implementačné nástroje, algoritmy a datasety použité pri vytváraní vlastného riešenia.

3.2.1 Implementačné nástroje a knižnice

Pre implementáciu modelu som si vybral knižnicu TensorFlow2 a Keras. Mám s nimi najväčšie skúsenosti, keďže som ich už viackrát využil. Vytváranie modelov je v nich jednoduché a rýchle. Knižnica ponúka aj možnosť trénovania na GPU.

Model bude postavený na základe algoritmu ArcFace. Jeho implementácií existuje veľmi veľa. Oficiálnu implementáciu ponúka knižnica Insightface, ktorú som spomínal v tejto práci v kapitole 3.1.1. Existujú rôzne riešenia tretích strán, ktoré sú pekne zhrnuté v oficiálnom repozitári tejto knižnice [20].

Keďže chcem využiť knižnicu TensorFlow2 bolo odporúčané využiť nasledovnú implementáciu algoritmu ArcFace [24].

3.2.2 Datasety

Na trénovanie siete plánujem využiť datasety *Head Pose Image* a *Robotics Lab*, ktoré sú bližšie pripísané v časti 2.4. Datasety obsahujú potrebné anotácie rotácie a natočenia tváre.

Ak by tieto datasety neboli dostatočné plánujem využiť iné, ktoré som uvádzal v práci napríklad *MS1-Celeb-M*. Obrázky by bolo potrebné dodatočne anotovať o pozíciu tváre. Toto by bolo možné vykonať napríklad nástrojom *Deepgaze* [39].

3.3 Návrh modelu

Model bude postavený na algoritme ArcFace ako som spomínal vyššie. Budem sa pokúšať upraviť chybovú funkciu softmax 2.20, tak aby dávala čo najlepšie výsledky.

Na vstupe budú obrázky z datasetov, ktoré budú obsahovať dve anotácie: natočenie tváre pozdĺž horizontálnej osi (pan) a vertikálnej osi (tilt). S týmito vstupmi avšak knižnica [24] nevie pracovať, preto ju budem musieť rozšíriť.

V práci sa budem snažiť využiť parameter m (additive angular margin penalty) z chybovej funkcie softmax, ktorú model ArcFace implementuje. Uvádzam ju ešte raz:

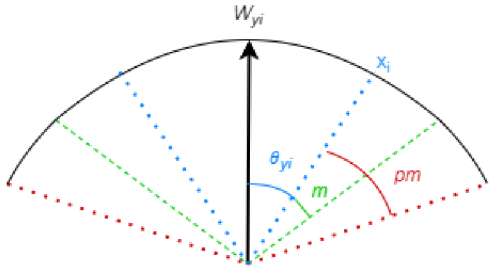
$$L_3 = -\log \frac{e^{s\cos(\theta_{y_i}+m)}}{e^{s\cos(\theta_{y_i}+m)} + \sum_{j=1,j\neq y_i}^{N} e^{s\cos\theta_j}} \tag{3.1}$$

Tento parameter je pre metódu ArcFace veľmi dôležitý, ako uvádza práca [12]. Zvyšuje kompaktnosť v rámci triedy a zväčšuje rozdiely medzi jednotlivými triedami. Penalizácia je rovná geodetickej vzdialenosti na normalizovanej hypersfére. To spôsobuje, že má lepšie geometrické vlastnosti, čo sa týka rozhodovacej hranice. Ako je vidieť na obrázku 2.20, model ArcFace má konštantný lineárny uhlový okraj na celom intervale oproti ostatným prístupom.

Vzhľadom na dôležitosť tohto parametru m pre model ArcFace, budem preto tento parameter modifikovať dodatočnými úpravami a penalizáciami a skúmať vplyv na úspešnosť modelu. Uvediem tu spôsoby, akými by sa mohol meniť. Navrhol som dve veľké druhy experimentov a to s multiplikatívnou a aditívnou penalizáciou parametru m. Tieto popíšem v následujúcich častiach. Ďalšie experimenty by záviseli na dosiahnutých výsledkoch.

3.3.1 Multiplikatívna penalizácia

Experimenty s multiplikatívnou penalizáciou sa snažia priamo penalizovať parameter m. Najprv uvádzam experiment s jednoduchšou úpravou modelu ArcFace, a to penalizáciu nefrontálnych snímkov. Následne popisujem zložitejšiu, dynamickú penalizáciu, ktorá rôzne penalizuje rôzne stupne odklonenie od frontálnej pozície (čím väčší odklon, tým väčšia penalizácia). Vizualizácia všeobecnej multiplikatívnej penalizácie je zobrazená na obrázku 3.2.



Obr. 3.2: Vizualizácia vlastného návrhu multiplikatívnej penalizácie parameteru m. Na obrázku je možné vidieť stred triedy W_{y_i}, príznak x_i, modrou zvýraznený uhol medzi stredom triedy a príznakom θ_{y_i}, zelenou zvýraznený uhol s prídavnou uhlovou penalizáciou (additive angular margin penalty) m a nakoniec červenou zvýraznený uhol s multiplikatívnou penalizáciou pm.

Penalizácia nefrontálnych snímkov

Pri tomto druhe experimentov budem penalizovať len nefrontálne pozície (frontálne zostanú zachované). Upravená rovnica pre ArcFace by vyzerala takto:

$$L_{NFP} = -\log \frac{e^{s\cos(\theta_{y_i}+pm)}}{e^{s\cos(\theta_{y_i}+pm)} + \sum_{j=1,j\neq y_i}^{N} e^{s\cos\theta_j}}, \tag{3.2}$$

kde p je penalizácia. Na začiatok by penalizácia pre frontálne pozície mohla mať veľkosť 1 a pre nefrontálne 1.5 alebo 2. Veľkosť penalizácie bude predmetom experimentov.

Dynamická penalizácia

V tomto prípade by rovnica 3.2 zostala rovnaká bez úprav. Zmenila by sa hodnota penalizácie p. Pre frontálne pozície by sa okraj nepenalizoval. Penalizácia pre nefrontálne pozície by závisela od stupňa odklonu od frontálnej pozície. Vyššie by boli penalizované pozície s väčším stupňom odklonu (najviac teda tváre, kde natočenie pozdĺž horizontálnej a aj vertikálnej osi by dosahovalo 90°). Predmetom experimentovania by mohli byť dve rôzne prístupy k dynamickej penalizácií.

Prvý prístup by mohol sčítavať oba uhly charakterizujúce natočenie tváre, súčet normalizovať a zdvihnúť o jedničku. Dostali by sme tak dynamickú penalizáciu medzi 1 až 2:

$$p_m = \frac{|tilt| + |pan|}{180} + 1 \tag{3.3}$$

Druhý prístup by pracoval s nejakou formou tried. Spočítali by sa absolútne hodnoty uhlov a následne by sa podľa určeného intervalu hodnote priradila trieda. Rovnica pre interval 60 stupňov by mohla vyzerať následovne:

$$trieda(x) = \begin{cases} 0 & \text{keď } 0 <= x < 60 \\ 1 & \text{keď } 60 <= x < 120 \\ 2 & \text{keď } 120 <= x < 180 \\ 3 & \text{keď } 180 <= x \end{cases} \tag{3.4}$$

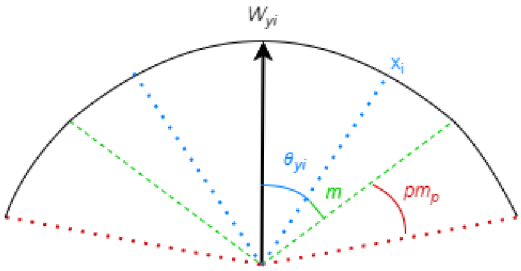
kde x je súčet absolútnych hodnôt vertikálneho a horizontálneho uhlu. Dostali by sme tak 4 triedy. Každá z nich by sa následne normalizovala medzi hodnotu 1 až 2. Aby sme zvýraznili a ešte viac penalizovali tváre s najväčším odklonom od frontálnej pozície, tak by som hodnoty normalizoval kvadratickým spôsobom. Normalizácia by mohla vyzerať nasledovne:

$$p_{trieda} = \frac{trieda^2}{max^2} + 1 \tag{3.5}$$

kde trieda je výsledok z rovnice 3.4 a max je číslo najväčšej triedy (pri štyroch triedach ako je uvedené vyššie by to bolo číslo 3).

3.3.2 Aditívna penalizácia

Rozšírenie o aditívnu penalizáciu by nepenalizovalo priamo okraj m. Namiesto toho by som sa ho snažil zachovať a pridať k nemu penalizácie alebo nový penalizovaný okraj. Pri tomto druhu experimentov by sa naskytla široká škála možností. Penalizáciu by som mohol napríklad aplikovať aj pre každý uhol zvlášť. Navrhoval by som využitie dynamickej penalizácie. Druh by závisel od výsledkov experimentov. V návrhu uvediem ako by mohli vyzerať takto upravené chybové funkcie. Vizualizácia všeobecnej aditívnej penalizácie je zobrazená na obrázku 3.3.



Obr. 3.3: Vizualizácia vlastného návrhu aditívnej penalizácie parameteru m. Na obrázku je možné vidieť stred triedy W_{y_i}, príznak x_i, modrou zvýraznený uhol medzi stredom triedy a príznakom θ_{y_i}, zelenou zvýraznený uhol s prídavnou uhlovou penalizáciou (additive angular margin penalty) m a nakoniec červenou zvýraznený uhol aditívnej penalizácie pm_p.

Jednoduchá aditívna penalizácia

Rovnica 3.6 popisuje prípad, kedy pôvodný prídavný uhlový okraj zostane zachovaný a bude k nemu pripočítaná len penalizácia. Penalizácia môže byť dynamická, podobne ako uvádza rovnica 3.3, len by bola medzi hodnotami 0 až 1. Rovnica by vyzerala takto:

$$L_{JAP} = -\log \frac{e^{s\cos(\theta_{y_i}+p_m+m)}}{e^{s\cos(\theta_{y_i}+p_m+m)} + \sum_{j=1,j\neq y_i}^{N} e^{s\cos\theta_j}}, \tag{3.6}$$

kde p_m je dynamická penalizácia a m je prídavný uhlový okraj.

Jednoduchý aditívny penalizovaný okraj

V tomto prípade pomocou dynamickej penalizácie penalizujeme nový okraj:

$$L_{JAPO} = -\log \frac{e^{s\cos(\theta_{y_i}+p_m m_p+m)}}{e^{s\cos(\theta_{y_i}+p_m m_p+m)} + \sum_{j=1,j\neq y_i}^{N} e^{s\cos\theta_j}}, \tag{3.7}$$

kde p_m je dynamická penalizácia, m_p je nový penalizovaný okraj a m je prídavný uhlový okraj.

Zložený aditívny penalizovaný okraj

Tento prípad je rozšírením experimentu s jednoduchým prídavným penalizovaným okrajom. Pri tomto experimente by som penalizáciu rozložil zvlášť pre každý druh natočenia tváre (horizontálne a vertikálne):

$$L_{ZAPO} = -\log \frac{e^{s\cos(\theta_{y_i}+p_{tilt}m_{p1}+p_{pan}m_{p2}+m)}}{e^{s\cos(\theta_{y_i}+p_{tilt}m_{p1}+p_{pan}m_{p2}+m)} + \sum_{j=1,j\neq y_i}^{N} e^{s\cos\theta_j}}, \tag{3.8}$$

kde p_{tilt} je dynamická penalizácia závislá na vertikálnom natočení tváre, p_{pan} je dynamická penalizácia závislá na horizontálnom natočení tváre, m_{p1} a m_{p2} sú nové penalizované okraje a m je prídavný uhlový okraj.

3.4 Implementácia

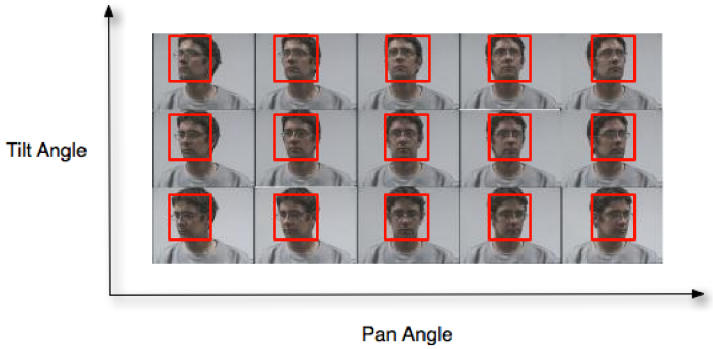
V tejto časti uvádzam implementáciu riešenia. Jedná sa hlavne o rozšírenie existujúcej knižnice [24], ktorá implementuje algoritmus ArcFace využitím knižnice TensorFlow2 (Keras).

Postupne rozoberám rozšírenia pre načítavanie datasetov s anotáciami natočenia tváre, rozšírenie modelu, aby s týmito dátami vedel pracovať. Ďalej uvádzam, akým spôsobom s týmito anotáciami pracujem vo vnútri modelu a nakoniec opisujem spôsob trénovania a testovania.

Dokumentácia repozitára s inštaláciou, spôsobom spustenia a s použitím je priložená v koreňovom priečinku s kódom.

3.4.1 Spracovanie a príprava dát

Pre prácu s modelom potrebujem dva druhy anotácií a to naklonenie v horizontálnom (span) a vertikálnom (tilt) smere. Toto je ilustrované na obrázku 3.4.



Obr. 3.4: Naklonenie tváre pozdĺž dvoch osí [17].

Datasety *Head Pose Image* a *Robotics lab* obsahujú jednu alebo obe z týchto informácií. Problém nastáva pri datasete *MS1-Celeb-M*, ktorý tieto informácie neposkytuje. Pre extrakciu tohto typu anotácií som použil už zmieňovaný nástroj *Deepgaze* [39]. Anotácie spracovávam v súbore `arcface-tf2_convert_train_tfrecord.py`. V nasledovných častiach by som popísal spôsob spracovania a prípravy dát pre jednotlivé datasety.

Head Pose Image

V datasete *Head Pose Image* je informácia o natočení hlavy reprezentovaná v názve obrázku alebo v textovom súbore s rovnakým názvom. Dáta sú uložené klasicky v priečinkoch, kde má každá osoba vlastný priečinok a názov priečinka je identifikátorom osoby. Pre *Head Pose Image* dataset som upravil názvy priečinkov na čísla od 0 po 15, aby program s nimi vedel pracovať a správne ich usporiadal do tried.

Názov obrázku má v *Head Pose Image* datasete následujúci formát:

$$person[Id][Serie][Number][Tilt][Pan].jpg$$

kde

- $Id = \{01, ..., 15\}$ je identifikátor osoby,
- $Serie = \{1, 2\}$ je označenie pre trénovaciu a testovaciu sadu,

- $Number = \{00, 01, ..., 92\}$ je číslo súboru v priečinku,
- $Tilt = \{-90, -60, -30, -15, 0, +15, +30, +60, +90\}$ je vertikálny uhol,
- $Pan = \{-90, -75, -60, -45, -30, -15, 0, +15, +30, +45, +60, +75, +90\}$ je horizontálny uhol.

Robotics lab

Pre dataset *Robotics lab* je táto informácia reprezentovaná len v názve obrázku. Dáta sú uložené podobne ako v prípade datasetu *Head Pose Image*. Názvy tried som ale upravil z pôvodných, ktoré obsahovali text (napríklad Subject01) na čisto číselné (01) a zdvihol som ich o číslo sto (101), aby pri kombinovanom trénovaní (s datasetom *Head Pose Image*) boli jasne rozlíšiteľné.

Názov obrázku má v *Robotics lab* datasete následujúci formát:

$$[Serie][Id][Pan].jpg$$

kde

- $Serie = \{A, B\}$ je označenie pre trénovaciu a testovaciu sadu,
- $Id = \{101, ..., 190\}$ je identifikátor osoby,
- $Pan = \{-90, ..., +90\}$, kde interval je 5 stupňov a značí horizontálny uhol.

MS1-Celeb-M

Knižnica *Deepgaze* umožňuje jednoducho extrahovať z obrázku s tvárou jej natočenie. Vytvoril som skript `extract_head_pose.py`, ktorý využíva túto knižnicu. Skript vytvára súbor, ktorý obsahuje záznamy tohto typu:

$$[Person\_id][Img\_name][Tilt][Pan]$$

kde

- $Person\_id = \{0, ..., 85741\}$ je identifikátor osoby,
- Img_name je názov obrázku,
- $Tilt$ je vertikálny uhol,
- Pan je horizontálny uhol.

Obrázky a ich anotácie pre všetky datasety sú ďalej spracovávané v súbore `convert_train_tfrecord.py`. Tento súbor som rozšíril o vlastný kód. Každý dataset je spracovávaný vlastnou funkciou (napríklad `parse_file_name_headpose()`). Pri datasetoch *Head Pose Image* a *Robotics lab* sa z názvov súborov parsujú anotácie naklonenia tváre. Keďže tváre v *Robotics lab* datasete sa líšia len v horizontálnom smere, tak sa pre vertikálny uhol používa základná hodnota 0. Pri datasete *MS1-Celeb-M* sa anotácie získavajú z dodaného anotačného súboru, ktorý bol vygenerovaný za použitia knižnice *Deepgaze*.

Knižnica *arcface-tf2* umožňuje prednačítať obrázky v binárnej podobe. Dataset potom už nie je potrebný. Umožňuje to rýchlejšie trénovanie a efektívnejšiu prácu s datasetom.

Funkcionalita je pôvodná (implementovaná knižnicou). Ja som ju pre jednoduchšiu prácu refaktoroval a umožnil spúšťať príznakom (pôvodne je implementovaná v inom súbore a bol by potrebný iný príkaz na spustenie).

Výsledkom parsovania je súbor typu `tfrecord`, ktorý sa následne využíva pri trénovaní. Tento súbor je možné vygenerovať príkazom (je potrebné byť v pracovnom priečinku `arcface-tf2`) uvedenom vo výpise 3.2.

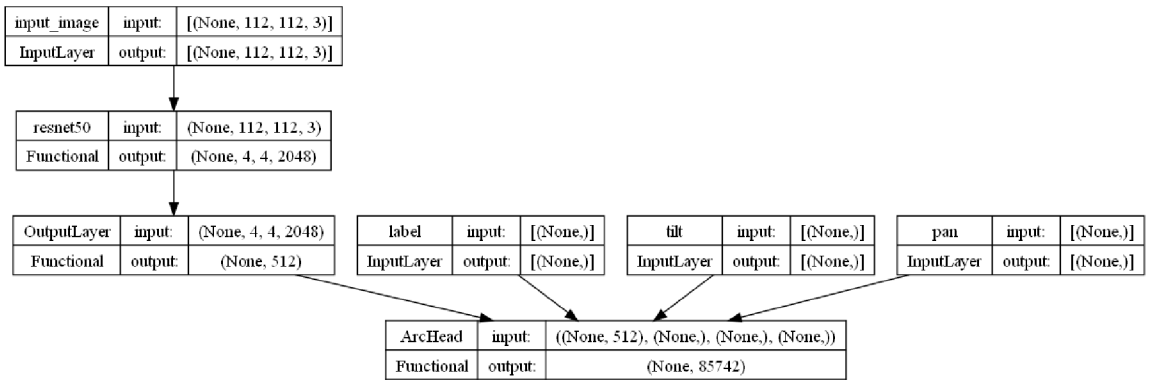
```
python3 convert_train_tfrecord.py --dataset_path="cesta-k-datasetu" --output_path="data/
    head_pose.tfrecord" --convert_image_to_binary
```

Výpis 3.2: Príkaz pre prípravu dát na trénovanie s ich prednačítaním pre dataset *Head Pose Image* (Linux).

3.4.2 ArcFace model

Knižnica implementujúca algoritmus ArcFace [24] má v základe jednoduchú štruktúru, ktorá je zobrazená na obrázku 3.5. Má vstupnú vrstvu pre obrázky. Veľkosť obrázkov je možné konfigurovať. Pre *backbone vrstvu* ponúka na výber ResNet50 alebo MobileNetV2. Rozhodol som sa pre sieť ResNet50 pretože dosahovala o niečo lepšie priemerné výsledky na rozdiel od siete MobileNetV2 [24].

Model ďalej obsahuje výstupnú vrstvu z ResNet50 a vstupné vrstvy. Medzi ne som pridal vstupy pre horizontálne a vertikálne naklonenie tváre. Tieto vrstvy sú vstupom do vrstvy ArcFace pomenovanej *ArcHead*.



Obr. 3.5: ArcFace model.

Natočenie tváre je extrahované zo súboru typu *tfrecord*. Implementácia sa nachádza v súbore `modules/dataset.py`. Existujúcu implementáciu som rozšíril o extrakciu dotyčných dát (vertikálneho a horizontálneho uhlu natočenia tváre). Implementácia je rozdelená pre načítavanie súboru typu *tfrecord* s prednačítanými obrázkami a bez nich. Knižnica to rieši jednoduchým prepínačom. Rozšíriť kód som tak musel pre obe varianty.

Knižnica implementuje všetky modely v súbore `modules/models.py`. Vyššie opisovaný model ArcFace je implementovaný pomocou triedy *ArcFaceModel*. Existujúci kód som rozšíril o vlastnú implementáciu, ktorá zabezpečuje, aby sa informácie o naklonení hlavy dostali správnym spôsobom do vnútra modelu a predali príslušnej vrstve ArcHead.

Vrstva ArcHead implementuje samotný algoritmus ArcFace. Na vstupe má vstupné dáta s označeniami tried a naklonením tváre, ktoré je vyjadrené v uhloch. O túto informáciu bol kód mnou rozšírený. Tieto dáta sa propagujú do triedy *ArcMarginPenaltyLogists*. Táto trieda má na starosti výpočet, ktorý vykonáva algoritmus ArcFace.

V triede *ArcMarginPenaltyLogists* vykonávam zmeny na algoritme ArcFace. Zmeny sú založené na matematickom návrhu uvedenom vyššie. Vstupné hodnoty vchádzajú do triedy vo vektoroch. Je to spôsobené tým, že trénovanie pracuje s dávkami. Moja úprava preto počíta s tým a využíva funkcie pre prácu s vektormi poskytnutými knižnicou TensorFlow. Implementácia pre každý návrh je v samostatnej vetve.

Vypichol by som implementáciu návrhu dynamickej penalizácie, ktorá triedi absolútny súčet horizontálneho a vertikálneho do tried. Definoval som hranice pre jednotlivé triedy pomocou vektoru hodnôt float. S týmto vektorom pracuje funkcia z knižnice TensorFlow `searchsorted`, ktorá vráti index miesta (v zoradenom poli), pred ktorý keby bola hodnota vložená, tak bude poradie zachované. Funkcia tak veľmi ľahkým spôsobom umožní nájsť interval do ktorého hodnota patrí. Index v poli reprezentuje triedu. Trieda sa nakoniec zoberie a normalizuje kvadratickým spôsobom ako som uviedol v rovnice 3.5.

3.4.3 Trénovanie modelu

Knižnica implementuje trénovanie v súbore `train.py`. Tento súbor som rozšíril o radu vecí, ktoré podrobnejšie opíšem. Jedná sa hlavne o prispôsobenia trénovania na viacerých GPU súčasne. Knižnica umožňuje nastavenie trénovania pomocou konfiguračného súboru, poskytuje debugovací a normálny mód spustenia, alebo ukladanie checkpointov a logov z trénovania. Dáta pri trénovaní neboli nijako upravované a augmentované.

Parametre trénovania

Trénovanie je nastaviteľné pomocou konfiguračného súboru uloženého v priečinku `configs`. Užívateľ má možnosť si tak vytvoriť vlastnú konfiguráciu trénovania, prípadne upraviť už tie existujúce. V konfiguračnom súbore je možné nastaviť základné hyperparametre trénovania ako počet epoch, rýchlosť učenia, alebo veľkosť trénovacej dávky. V tomto súbore je možné ďalej definovať veľkosť vstupných obrázkov, veľkosť embeddingov, typ backbone siete, typ vrchnej vrstvy (ArcHead, NormHead), cestu k trénovaciemu (a aj testovaciemu) datasetu alebo frekvenciu ukladania modelu počas trénovania (checkpoints).

Pre trénovanie som upravil už existujúci konfiguračný súbor pre sieť ResNet50. Súbor má názov `configs/arc_res50.yaml` a použil som ho ako základ pre vlastné konfigurácie. Veľkosť trénovacej dávky (batch) som nastavil na 128. Model som nechal trénovať v základe na 5 epoch. Číslo sa líšilo pri jednotlivých experimentoch. Bolo ovplyvnené schopnosťou modelu konvergovať k nejakej ustálenej chybovosti a dĺžkou trénovania modelov (niektorým zabrala jedna epocha aj 20 hodín). To druhé bolo dôvodom ukončenia trénovania pri väčšine modelov. Modely som trénoval maximálne 2 až 3 dni, vzhľadom na dostupnú kapacitu strojov pre trénovanie. Preto bude možné vidieť vo výsledkoch malý počet epoch, na ktorom boli modely trénované. Rýchlosť učenia som nastavil na hodnotu 0,01. Model som ukladal na každej stej dávke.

Pri trénovaní bolo využitá metóda stochastického gradientného zostupu (SGD), ktorá je súčasťou knižnice Keras. Metóda bola definované knižnicou, ktorá implementuje model ArcFace. Pri využití SGD je nastavené momentum na hodnotu 0,9, ktoré urýchľuje gradientný zostup v príslušnom smere a tlmí oscilácie. Taktiež bolo aplikované Nestorovo momentum.

Spôsob trénovania

Všetky modely boli natrénované pomocou služby MetaCentrum [33]. Model bol trénovaný na rôznych strojoch s grafickými kartami. Väčšinou boli využívané stroje *adan*, kde uzol obsahoval dve grafické karty NVIDIA *Tesla T4*.

Používanú knižnicu som musel preto rozšíriť, aby dokázala využívať viacej GPU súčasne. Rozšírenie je implementované pomocou modulu `tf.distribute`. Nastavil som stratégiu trénovania na *MirroredStrategy*, ktorá umožňuje trénovať model na viacerých GPU v rámci jedného uzlu. Vytváranie modelu alebo jeho načítanie musí následne prebiehať v rámci rozsahu platnosti tejto stratégie.

Na zadávanie úloh pre službu MetaCentrum som vytvoril niekoľko skriptov, ktoré sa využívajú v rámci príkazu `qsub`. Medzi tieto skripty patria:

- `train-headpose.sh` - trénovanie modelu na datasete *Head Pose*,
- `train-robotics.sh` - trénovanie modelu na datasete *Robotics lab*,
- `test-combined.sh` - trénovanie modelu na datasetoch *Head Pose* a *Robotics lab*,
- `test-ms1m.sh` - trénovanie modelu na datasete *MS1-Celeb-M*.

Skripty väčšinou pozostávajú z definície úlohy pre MetaCentrum, ako napríklad čas rezervácie stroja, počet GPU, veľkosť pamäte RAM, veľkosť dočasnej pamäte určenej na výpočet nazývanej *scratch* a druh čakacej fronty. Skript následne uloží identifikátor vytvorenej práce a cestu k *scratch* priečinku (pre prípad ak by nastala chyba a bolo by ho potrebné ručne vymazať). Skript potom načíta `python36-modules`, čo obsahuje všetky potrebné knižnice pre spustenie programu. Ďalej premiestňuje dáta a program do *scratch* adresára, spracováva dataset a vytvára súbor typu `tfrecord` (alebo pracuje už len s vytvoreným súborom pre ušetrenie času). Skript potom spúšťa trénovanie alebo testovanie modelu a skopíruje výsledky naspäť do úložiska.

3.4.4 Testovanie modelu

Knižnica implementuje testovanie modelu na benchmarkových datasetoch a to *LFW*, *CFP-FP* a *AgeDB-30*. Vývojári poskytujú výsledky ich natrénovaneho modelu v GitHub repozitári [24]. Pre účely tejto práce som vytvoril rozšírenie testovania na testovacích sadách datasetov. Rozšírenie je implementované v súbore `arcface-tf2/test.py`.

Riešenie stavia na existujúcom kóde, ktorý umožňuje načítanie jedného obrázku, jeho úpravu a vytvorenie embeddingov pomocou natrénovaného modelu. Tento kód rozširujem o to, aby vedel načítať obrázky z testovacieho datasetu, tým istým spôsobom vytvoril embeddingy a následne ich otestoval.

Pre testovacie účely som vygeneroval z testovacích datasetov dvojice obrázkov s označením, či tieto obrázky spadajú do jednej triedy (je na nich rovnaká osoba) alebo nie. Pre každý dataset je vygenerovaná vlastná sada týchto dvojíc. Dvojice sú zapísané v súbore obsahujúce názov `pairs_<dataset_name>.txt` (napríklad `pairs_headpose.txt`). Súbor je pri testovaní načítaný, z obrázkov ktoré obsahuje sa vytvoria embedingy, a tie sa podľa uvedených párov navzájom otestujú na euklidovskú vzdialenosť. Následne sa otestuje rozdiel medzi nimi oproti rôznym prahom a konečný výsledok je vyhodnotený na základe najlepšieho prahu (podobne ako je implementované autormi knižnice pri benchmarkových testoch). Na základe toho sa vypočítajú metriky *accuracy*, *precision*, *recall* a *F1-score* a uvedie prah, pre ktorý boli vypočítané.

Moje rozšírenie umožňuje taktiež testovať presnosť modelov vzhľadom na rôzne naklonenia tváre. Vytvoril som dvojice obrázkov, kde každý patrí do iného intervalu naklonenia (tie sú popísané presnejšie v kapitole 4). Výsledky testovania sú zobrazené na štandardnom výstupe a taktiež je vytvorený súbor typu *csv* so súhrnom výsledkov.

Kapitola 4

Experimenty a výsledky

V tejto kapitole popisujem vykonané experimenty s modelom ArcFace. Uvádzam zmeny, ktoré som spravil v chybovej funkcii a výsledky na testovacích, a benchmarkových datasetoch. Na záver porovnávam moje výsledky s výsledkami, ktoré boli dosiahnuté na pôvodne natrénovanom modeli ArcFace vývojármi knižnice.

4.1 Základné informácie k experimentom

Ku každému návrhu modifikácie chybovej funkcie uvedenej v sekcii 3.3 som vykonal niekoľko experimentov na rôznych datasetoch a ich rôznych veľkostiach. Chcel som tak skúmať chovanie modelu trénovaného na rôznych dátach, o rôznych veľkostiach. Ďalším dôvodom bolo, že dataset MS1-Celeb-M je obrovský. Mal som naplánovaný väčší počet experimentov, ale obmedzené dostupné výpočtové možnosti služby Metacentra.

Pri datasete MS1-Celeb-M som používal prvých 1000, 2000, 5000 alebo 10000 identít. Pre ušetrenie času boli pre trénovanie modelov s 10000 identitami, využité predtrénované modely na 5000 identitách. Pri experimentoch som využíval hlavne backbone sieť ResNet-50. Dôvody som uviedol vyššie. Za účelom porovnania som pre vybrané varianty experimentov použil aj druhú ponúkanú sieť MobileNetV2.

Pri každom experimente uvádzam celkové výsledky, kde je možné vidieť úspešnosť modelu ako takého. Následne uvádzam výsledky pre jednotlivé natočenia tváre vo vertikálnom a horizontálnom smere rozdeleného do tried. Použil som rozdelenie do tried hlavne kvôli datasetu MS1-Celeb-M, ktorý neobsahuje snímky fotené v kontrolovaných podmienkach (na rozdiel od datasetov Head Pose Images a Robotics Lab. Chcel som tak zoskupiť podobné pozície k sebe.

Triedy sú rozdelené do troch kategórií s numerickým označením 0 až 3, kde trieda 0 obsahuje frontálne pozície alebo pozície blízke k frontálnym (interval $\langle 0, 5)$ stupňov), trieda 1 zahŕňa pozície v intervale $\langle 5, 30)$ stupňov, trieda 2 obsahuje pozície v intervale $\langle 30, 60)$ stupňov a v poslednej triede 3 sú pozície patriace do intervalu $\langle 60, 90)$ stupňov. Jedná sa o súčet absolútnych hodnôt naklonenia tváre (uhlov) pozdĺž vertikálnej a horizontálnej osi.

Definícia umožňuje pracovať až so siedmimi triedami, ale dataset MS1-Celem-M neobsahoval medzi vybranou vzorkou väčšie náklony tváre. Výsledky podľa mňa aj napriek tomu vytvárajú celkom dobrý obraz o fungovaní modelu pri bežných pozíciách hlavy. Je taktiež možné už aj pri týchto nakloneniach tvárí dobre identifikovať ako sa jednotlivé modely dokážu s nimi vysporiadať.

Testovací dataset pre MS1-Celeb-M bol vytvorený z dvojíc náhodne vybratých identít s označením 10 000 až 10 499 (takže 500 identít). Počet testovacích dvojíc pre celkové testovanie modelov je 1000. Pre testovanie modelov s rozdelením do tried uhlov bolo vygenerovaných 960 dvojíc s rovnomerným zastúpením pre jednotlivé triedy.

Všetky výsledky sú zhrnuté v súbore `results/results-summary.xlsx`.

4.2 Základný model

Na začiatku som natrénoval model bez akýchkoľvek modifikácií na datasetoch, ktoré budem využívať pri ďalších experimentoch. Model bude slúžiť ako základ pre ďalšie porovnávania. Celkovo som natrénoval 8 modelov na štyroch datasetoch: Headpose Image, Robotics Lab, ich kombinácií a MS1-Celeb-M. Výsledky experimentov je možné vidieť v tabuľke 4.1.

Dataset	**Veľkosť datasetu**	**Typ backbone siete**	**Epochy**	**Chyba**	**Testovací dataset**	**Benchmarkový dataset**				**Kombinovaná presnosť**
						lfw	**aged**	**cfp**	**avg**	
Headpose Image	16	ResNet50	5	0.34	0.96	0.58	0.55	0.57	0.57	0.76
Robotics Lab	90	ResNet50	2	0.01	0.97	0.59	0.53	0.56	0.56	0.77
Combined	106	ResNet50	5	0.35	0.81	0.54	0.53	0.53	0.53	0.67
MS1-Celeb-M	1000	ResNet50	13	15.37	0.73	0.81	0.63	0.68	0.71	0.72
	2000	ResNet50	5	18.21	0.78	0.89	0.65	0.69	0.74	0.76
	5000	ResNet50	4	18.19	0.80	0.92	0.72	0.57	0.74	0.77
	10000	ResNet50	16	17.95	0.88	0.95	0.78	0.85	0.86	0.87
	10000	MobileNetV2	3	14.12	0.86	0.97	0.84	0.88	0.90	0.88

Tabuľka 4.1: Výsledky testovania základného modelu. V tabuľke je uvedený názov datasetu, jeho veľkosť, typ backbone siete, počet epoch na ktorom bol model trénovaný, dosiahnutá chyba a presnosť, ktorá vznikla testovaním na testovacom alebo benchmarkovom datasete. Na záver uvádzam kombinovanú presnosť, čo je aritmetický priemer presnosti z testovacieho datasetu a priemernej presnosti z benchmarkových datasetov.

Na začiatku som sa snažil využiť datasety obsahujúce anotácie natočenia hlavy a ich kombinácie. Presnosť modelov na benchmarkových datasetoch bola malá. Pre dataset Headpose Image priemerná presnosť bola 57%, pre Robotics Lab 56% a pre ich kombináciu ešte horšia, a to 53%. Tieto datasety sú malé, dokopy obsahujú 106 rôznych identít. Toto nestačilo na natrénovanie veľkého modelu.

Preto som sa rozhodol použiť dataset MS1-Celeb-M, ktorý bol využitý pri trénovaní modelu vývojármi knižnice. Výsledky boli neporovnateľne lepšie už na najmenšom datasete s tisíckou identít, kde dosiahol presnosť 71%. Na datasete s 2000 a 5000 identitami model dosiahol o niečo väčšiu presnosť 74%. Pri datasete s 10000 identitami dosiahol moddel skokové zlepšenie a to presnosť 86% pri použití siete ResNet50 a až 90% pri sieti MobileNetV2.

Pre ďalšie experimenty som si vybral dataset s 5000 a 10000 identitami (niekedy sa môžu vyskytnúť aj iné kvôli testovacím dôvodom). Datasety ukazuje dobré výsledky a odhadujem, že pri ďalších experimentoch budú lepšie generalizovať vďaka svojej veľkosti.

Výsledky testovania modelu so zameraním sa na vplyv naklonenia tváre je zobrazený v tabuľke 4.2. Pri tomto druhe testov som vynechal modely natrénované na datasetoch Headpose Image a Robotics Lab, pretože dosahovali zlé výsledky a kvôli tomu nebudú použité pri ďalších experimentoch.

Testovanie ukázalo, že model dosahuje vyššej presnosti pri rovnakých triedach (t. j. s podobným naklonením tváre). Zaujímavé je že pri rovnakých triedach ale nefrontálnych pozíciách (dvojice 2_2, 3_3), model ArcFace vykazuje lepšie výsledky ako pri frontálnych pozíciách. Pre dvojice frontálnych a nefrontálnych obrázkov (dvojice 0_x) vo všeobecnosti platí, že čím je rozdiel väčší, tým sa zmenšuje presnosť modelu. Prekvapivé je, že ak sa

Dataset	Veľkosť datasetu	Typ backbone siete	Dvojice uhlových tried									
			0_0	0_1	0_2	0_3	1_1	1_2	1_3	2_2	2_3	3_3
MS1-Celeb-M	1000	ResNet50	0.82	0.82	0.70	0.67	0.83	0.70	0.72	0.72	0.72	0.87
	2000	ResNet50	0.82	0.83	0.78	0.75	0.82	0.73	0.73	0.68	0.82	0.92
	5000	ResNet50	0.85	0.85	0.75	0.75	0.92	0.72	0.75	0.75	0.8	0.93
	10000	ResNet50	0.90	0.95	0.85	0.88	0.92	0.78	0.85	0.90	0.87	0.95
	10000	MobileNetV2	0.83	0.90	0.85	0.80	0.90	0.85	0.82	0.83	0.80	0.98

Tabuľka 4.2: Výsledky testovania základného modelu vzhľadom na triedy naklonenia tváre. Tabuľka obsahuje názov datasetu, jeho veľkosť a typ backbone siete. Pri dvojiciach uhlových tried prvé číslo značí triedu pre prvý obrázok a druhé triedu druhého obrázku. Napríklad pri dvojici 0_1 prvý obrázok patrí do triedy 0 a druhý do triedy 1. Význam tried som uviedol v sekcii 4.1.

jedná o nefrontálne pozície s rôznymi triedami medzi dvojicami obrázkov, tak to už naplatí. Tendencia je skôr opačná: čím je vychýlenie väčšie, tým model dosahuje lepšej presnosti. Výnimkou je v tomto model používajúci sieť MobileNetV2.

4.3 Multiplikatívna penalizácia

Pokračoval som experimentami s multiplikátívnou penalizáciou parameteru m. Uvádzam výsledky testovania modelov s penalizáciou nefrontálnach snímkov a dynamickou penalizáciou, ktoré som popísal v sekcii 3.3.1.

4.3.1 Penalizácia nefrontálnych snímkov

Penalizácia nefrontálnych snímkov som popísal rovnicou 3.2. Experimentoval som s rôznymi hodnotami penalizácie p ako 0 alebo 1,0 pre frontálne pozície a 1,0, 1,5 alebo 2,0 pre nefrontálne. Model som trénoval už len na datasete MS1-Celeb-M s rôznymi veľkosťami. Výsledky experimentov sú uvedené v tabuľke 4.3.

Dataset	Veľkosť datasetu	Typ backbone siete	Penalizácia (frontálna|nefrontálna)	Epochy	Chyba	Testovací dataset	Benchmarkový dataset				Kombinovaná presnosť
							lfw	aged	cfp	avg	
MS1-Celeb-M	1000	ResNet50	1.0|2.0	3	0.09	0.75	0.87	0.68	0.73	0.76	0.76
	2000	ResNet50	1.0|2.0	2	18.09	0.76	0.86	0.64	0.72	0.74	0.75
	5000	ResNet50	1.0|2.0	3	18.79	0.78	0.90	0.66	0.74	0.77	0.77
	10000	ResNet50	1.0|2.0	8	18.50	0.85	0.93	0.74	0.76	0.81	0.83
	5000	ResNet50	0|1.0	1	0.90	0.52	0.50	0.50	0.50	0.50	0.51
	5000	ResNet50	1|1.5	2	19.06	0.77	0.89	0.69	0.69	0.76	0.76

Tabuľka 4.3: Výsledky testovania modelu s penalizáciou nefrontálnych snímkov.

Najlepší výsledok s priemernou presnosťou 81% na benchmarkových datasetoch, dosiahol model natrénovaný na datasete s počtom identít 10000 pre penalizácie 1,0 (frontálna pozícia) a 2,0 (nefrontálna pozícia). Najhorší výsledok dosiahol model používajúci penalizáciu 1,0 (nefrontálna pozícia) s priemernou presnosťou 50%.

Pri podrobnejšom pohľade na výsledky vidíme, že pri penalizácií 2,0 pre dataset o veľkosti 1000 je možné vidieť zlepšenie o 5% oproti základnému modelu (na 76%). Pre dataset o veľkosti 2000 som nezaznamenal žiadne zlepšenie (74%). Model natrénovaný na datasete o veľkosti 5000 ukázal menšie zlepšenie o 3% oproti základnému modelu (na 77%). Pre penalizáciu 1,5 môžeme tiež pozorovať mierne zlepšenie o 2% na presnosť 76%. Model trénovaný na datasete o veľkosti 10000 dosiahol najlepšiu presnoť (81%). Oproti základnému modelu došlo ale k zhoršeniu o 5%. Ak neberieme do úvahy najhorší a najlepší výsledok, tak ostatné modely boli aspoň tak úspešné ako základný model.

Výsledky testovania modelu s vplyvom naklonenia tváre na výsledok je zobrazený v tabuľke 4.4.

Dataset	Veľkosť datasetu	Typ backbone siete	Penalizácia (frontálna|nefrontálna)	Dvojice uhlových tried									
				0_0	0_1	0_2	0_3	1_1	1_2	1_3	2_2	2_3	3_3
MS1-Celeb-M	1000	ResNet50	1.0|2.0	0.85	0.78	0.85	0.75	0.87	0.70	0.78	0.70	0.73	0.90
	2000	ResNet50	1.0|2.0	0.82	0.80	0.68	0.80	0.78	0.73	0.67	0.73	0.75	0.93
	5000	ResNet50	1.0|2.0	0.83	0.78	0.70	0.77	0.85	0.75	0.73	0.68	0.80	0.92
	10000	ResNet50	1.0|2.0	0.90	0.87	0.80	0.80	0.95	0.80	0.85	0.85	0.83	0.98
	5000	ResNet50	0|1.0	0.57	0.55	0.53	0.57	0.53	0.50	0.50	0.58	0.50	0.53
	5000	ResNet50	1|1.5	0.83	0.78	0.72	0.68	0.88	0.75	0.70	0.70	0.80	0.93

Tabuľka 4.4: Výsledky testovania modelu s penalizáciou nefrontálnych snímkov vzhľadom na triedy naklonenia tváre.

Výsledky sú podobné základnému modelu. Pri penalizácií 2,0 pre dataset o veľkosti 1000 je možné vidieť zlepšenie alebo porovnateľné výsledky naprieč všetkými triedami až na dvojice 0_1 a 2_2. Najvýraznejšie zlepšenia dosahovala dvojica tried 0_2 (až o 15%), dvojica 0_3 (8%) a dvojica 1_3 (6%). Pre dataset o veľkosti 2000 je vidieť zhoršenie pre nefrontálne pozície. Model natrénovaný na datasete o veľkosti 5000 ukázal tak isto zhoršenie (niekde aj o 7%). Podobné výsledky dosiahol aj model s penalizáciou 1,5. Pre penalizáciu 1,0 (nefrontálne pozície) sa výsledky hýbu medzi presnosťou 50% až 60% čo je zhodné s celkovými výsledkami, ktoré tiež nedopadli dobre.

Je potrebné mať na pamäti, že tento druh testov používa testovací dataset extrahovaný z datasetu MS1-Celeb-M. Výsledky je potrebné preto skôr porovnávať s celkovými výsledkami pre testovací dataset ako benchmarkový.

4.3.2 Dynamická penalizácia

Pri týchto experimentoch som pracoval s dvoma druhmi: s penalizáciou popísanou rovnicou 3.3 a penalizáciou založenou na triedach. Pri prvom prípade som vytvoril dva experimenty, kde som skúšal rôzny interval penalizácie a to od 0 po 1 alebo od 1 po 2. Pre triedy ako je znázornené v rovnici 3.4 som použil tri druhy rozdelenia a to intervaly 15, 30 a 60. Dostal som tak počet tried 12, 6 a 3. Výsledky experimentov sú zobrazené v tabuľke 4.5.

Dataset	Veľkosť datasetu	Typ backbone siete	Váhy	Epochy	Chyba	Testovací dataset	Benchmarkový dataset				Kombinovaná presnosť
							lfw	aged	cfp	avg	
MS1-Celeb-M	5000	ResNet50	(|t|+|p|) / 180 + 0	2	0.00	0.51	0.50	0.50	0.50	0.50	0.51
	5000	ResNet50	(|t|+|p|) / 180 + 1	1	18.23	0.77	0.90	0.72	0.79	0.80	0.79
	10000	ResNet50	(|t|+|p|) / 180 + 1	8	5.36	0.71	0.88	0.70	0.80	0.79	0.75
	10000	MobileNetV2	(|t|+|p|) / 180 + 1	7	9.08	0.75	0.82	0.73	0.77	0.77	0.76
	5000	ResNet50	15 stupňov	2	5.12	0.89	0.97	0.82	0.87	0.89	0.89
	10000	ResNet50	15 stupňov	8	0.36	0.91	0.96	0.83	0.89	0.90	0.90
	10000	MobileNetV2	15 stupňov	11	6.42	0.82	0.97	0.83	0.86	0.89	0.85
	5000	ResNet50	30 stupňov	2	1.95	0.89	0.96	0.82	0.87	0.88	0.89
	10000	ResNet50	30 stupňov	11	0.28	0.92	0.97	0.83	0.89	0.89	0.91
	10000	MobileNetV2	30 stupňov	7	7.57	0.84	0.97	0.83	0.86	0.89	0.86
	5000	ResNet50	60 stupňov	2	19.79	0.74	0.84	0.65	0.54	0.68	0.71

Tabuľka 4.5: Výsledky testovania modelu s dynamickou penalizáciou.

V tomto prípade môžeme vidieť značné zlepšenie oproti predchádzajúcim experimentom. V prvom type, kde uhly sčítavame a normalizujeme, sme dostali v lepšom prípade priemernú presnosť na benchmarkových datasetoch 80%. Toto je viac ako dosiahli modely v základných testoch a pri binárnej penalizácií. Avšak experimenty s rozdelením uhlov do tried dopadli ešte lepšie. Najlepšie výsledky dosiahlo rozdelenie do tried s intervalom 15 (89%) a 30 (88%) stupňov. Môžeme povedať, že výsledky boli veľmi podobné. Výhodu mal ale druhý model (interval 30 stupňov), pretože pri učení rýchlejšie konvergoval k nižšej chybovosti. Preto som tento model využil pri ďalších experimentoch.

Pri podrobnejšom pohľade na výsledky vidíme, že pri penalizácií s normalizovaním uhlov a pričítaním jednotky pre dataset o veľkosti 5000 je možné vidieť zlepšenie o 6% oproti základnému modelu (na 80%). Pri väčšom datasete s 10000 identitami je naopak vidieť zhoršenie: pre sieť ResNet50 o 7% (79%) a pre sieť MobileNetV2 skoro dvojnásobné zhoršenie o 13% (77%).

Pre penalizácie založené na triedach s intervalom 15 stupňov pre dataset s 5000 identitami je možné vidieť význačné zlepšenie oproti základnému modelu, a to o 15% (na 89%). Taktiež je možné vidieť zlepšenie pre dataset s 10000 identitami používajúci sieť ResNet50, a to o 4% (90%). Pre sieť MobileNetV2 nedošlo k zlepšeniu. Dosiahol presnosť 89%.

Pre interval 30 stupňov na datasete s 5000 identitami zaznamenávam porovnateľné zlepšenie a to o 14% (88%), a tak isto aj pre dataset s 10000 identitami a sieťou ResNet50 (o 3% na presnosť 89%). Sieti MobileNetV2 sa darilo o niečo málo horšie oproti základnému modelu. Zhoršenie bolo na úrovni 1% (89%).

Interval so 60 stupňami ukazoval zhoršenie oproti základnému modelu (o 6% na 68%).

Výsledky testovania modelu s vplyvom naklonenia tváre na výsledok sú zobrazené v tabuľke 4.6.

Dataset	Veľkosť datasetu	Typ backbone siete	Váhy	Dvojice uhlových tried									
				0_0	0_1	0_2	0_3	1_1	1_2	1_3	2_2	2_3	3_3
MS1-Celeb-M	5000	ResNet50	(|t|+|p|) / 180 + 0	0.68	0.53	0.65	0.62	0.58	0.53	0.52	0.53	0.58	0.70
	5000	ResNet50	(|t|+|p|) / 180 + 1	0.83	0.88	0.72	0.73	0.82	0.75	0.73	0.70	0.65	0.93
	10000	ResNet50	(|t|+|p|) / 180 + 1	0.77	0.80	0.65	0.73	0.82	0.72	0.72	0.67	0.77	0.90
	5000	MobileNetV2	(|t|+|p|) / 180 + 1	0.78	0.85	0.70	0.70	0.83	0.73	0.75	0.75	0.80	0.87
	5000	ResNet50	15 stupňov	0.93	0.97	0.90	0.90	0.95	0.87	0.88	0.88	0.92	0.97
	10000	ResNet50	15 stupňov	0.88	0.98	0.87	0.87	0.97	0.95	0.93	0.97	0.87	0.98
	10000	MobileNetV2	15 stupňov	0.82	0.90	0.82	0.77	0.87	0.88	0.77	0.75	0.77	0.92
	5000	ResNet50	30 stupňov	0.85	0.97	0.85	0.85	0.90	0.87	0.88	0.88	0.83	0.97
	10000	ResNet50	30 stupňov	0.93	0.98	0.93	0.88	0.97	0.92	0.95	0.92	0.92	0.95
	10000	MobileNetV2	30 stupňov	0.85	0.92	0.83	0.78	0.90	0.90	0.82	0.78	0.73	0.92
	5000	ResNet50	60 stupňov	0.80	0.72	0.68	0.67	0.78	0.65	0.78	0.70	0.83	0.90

Tabuľka 4.6: Výsledky testovania modelu s dynamickou penalizáciou vzhľadom na triedy naklonenia tváre.

Prvý model s normovanou penalizáciou pre dataset s počtom identít 5000, nedosiahol dobré výsledky. Boli o niečo lepšie ako dosiahol jeho ekvivalent pri penalizácií nefrontálnych snímkov (bez jednotkového posunu). Druhý model (s jednotkovým posunom) mal podobné výsledky ako základný model. Zaznamenal som tam ale prepad u dvojíc tried 1_1 a 2_3. Pri datasete o veľkosti 10000 identít sa potvrdili výsledky (modelu sa darilo naprieč triedami horšie).

Výrazné zlepšenie je možné vidieť pri následujúcich experimentoch. Model s veľkosťou 5000 identít s intervalom rozdelenia 15 prekonal základný model a vidieť výrazné zlepšenie hlavne pre nefrontálne pozície. Najlepší výsledok dosiahol pre dvojice tried 0_1 a 3_3, kde dosiahol presnosť až 97%. Najlepšie zlepšenie zaznamenal pre dvojice tried 0_2 a 0_3, kde som zaznamenal zlepšenie až o 15% (na presnosť 90%). V ostatných dvojiciach obsahujúcich nefrontálne pozície sa držalo zlepšenie väčšinou v okolí 10%. To isté platí pre model s intervalom 30 stupňov, ktorý trošku zaostáva, ale ukazuje porovnateľné zlepšenia oproti základnému modelu.

Model s veľkosťou 10000 identít a sieťou ResNet50 (interval 15 stupňov), ukázal tak isto zlepšenie oproti rovnakému základnému modelu. Zlepšenia sú ale menšie oproti menšiemu datasetu. Zlepšenie je znova veľmi viditeľné pre nefrontálne pozície. Najviditeľnejšie je to pre mierne natočené tváre pri dvojici tried 1_2, kde som dosiahol zlepšenie až o 17% (na 95%), čo prekonáva predchádzajúce modely. Model s intervalom rozdelenia 30 stupňov dosiahol podobné výsledky.

Model používajúci sieť MobileNetV2 dosiahol podobné výsledky ako základný model.

4.4 Aditívna penalizácia

Pri tomto druhe experimentov som znova pracoval s datasetom MS1-Celeb-M s veľkosťou 5000 a 10000 prvých tried. Ako penaltu som využil, ako bolo už popísané vyššie, prístup rozdelenia uhlov do tried v intervale 30 stupňov. Výsledky sú zhrnuté v tabuľke 4.7. Výsledky testovania modelu s vplyvom nakloneniа tváre sú zobrazené v tabuľke 4.8.

Dataset	Veľkosť datasetu	Typ backbone siete	Váhy	Epochy	Chyba	Testovací dataset	Benchmarkový dataset				Kombinovaná presnosť
							lfw	aged	cfp	avg	
MS1-Celeb-M	5000	ResNet50	JAP (30 stupňov)	2	1.05	0.70	0.80	0.57	0.66	0.68	0.69
	5000	ResNet50	JAPO (30 stupňov)	2	19.83	0.78	0.86	0.64	0.65	0.72	0.79
	5000	ResNet50	ZAPO (30 stupňov)	2	1.95	0.69	0.77	0.55	0.63	0.65	0.67
	10000	ResNet50	ZAPO (30 stupňov)	5	0.35	0.72	0.81	0.59	0.67	0.69	0.71
	10000	MobileNetV2	ZAPO (30 stupňov)	8	0.27	0.74	0.79	0.61	0.68	0.69	0.72

Tabuľka 4.7: Výsledky testovania modelu s aditívnou penalizáciou.

Dataset	Veľkosť datasetu	Typ backbone siete	Váhy	Dvojice uhlových tried									
				0_0	0_1	0_2	0_3	1_1	1_2	1_3	2_2	2_3	3_3
MS1-Celeb-M	5000	ResNet50	JAP (30 stupňov)	0.82	0.77	0.70	0.63	0.83	0.67	0.75	0.67	0.62	0.88
	5000	ResNet50	JAPO (30 stupňov)	0.82	0.82	0.83	0.72	0.83	0.73	0.75	0.72	0.73	0.97
	5000	ResNet50	ZAPO (30 stupňov)	0.78	0.72	0.68	0.62	0.73	0.67	0.65	0.70	0.67	0.90
	10000	ResNet50	ZAPO (30 stupňov)	0.83	0.77	0.77	0.70	0.82	0.65	0.65	0.70	0.67	0.88
	10000	MobileNetV2	ZAPO (30 stupňov)	0.83	0.75	0.72	0.70	0.83	0.65	0.72	0.72	0.72	0.90

Tabuľka 4.8: Výsledky testovania modelu s aditívnou penalizáciou vzhľadom na triedy nakloneniа tváre.

4.4.1 Jednoduchá aditívna penalizácia

V testoch som vychádzal z rovnice 3.6. Výsledky dopadli horšie ako v predchádzajúcich modifikáciách. Priemerná presnosť na benchmarkových datasetoch dosiahla 68%, čo je horšie ako výsledky v základných testoch pre rovnaký dataset (zhoršenie o 6%).

Model taktiež dosahoval horšie výsledky pre nefronálne pozície. V najhoršom prípade sa jednalo o zhoršenie o 18% (62%) pre dvojicu tried 2_3. Tento model tak nepriniesol žiadne zlepšenie.

4.4.2 Jednoduchý aditívny penalizovaný okraj

Pri experimentoch som vychádzal z rovnice 3.7. Výsledky boli znova horšie ako pri predchádzajúcich modifikáciách a dosahovali hodnoty skôr podobné tým, čo vyšli pri základných testoch. Priemerná presnosť na benchmarkových datasetoch bola 72% (zhoršenie o 2%). Nevýhodou bolo, že tento model veľmi pomaly konvergoval na rovnakom počte epoch oproti ostatným experimentoch s prídavnou penalizáciou.

Model dosahoval podobných presností aj pri pohľade na rôzne nakloneniа tváre. Pre dvojicu tried 0_2 priniesol dokonca zlepšenie o 8% (83%). Tento model tiež nepriniesol žiadne zlepšenie.

4.4.3 Zložený aditívny penalizovaný okraj

Toto bol posledný druh experimentov, ktorý som vykonával nad chybovou funkciou použitou v modeli ArcFace. Vychádzal som z rovnice 3.8. Výsledky dopadli horšie ako v predchádzajúcich modifikáciách.

Priemerná presnosť na benchmarkových datasetoch pre model trénovaný na 5000 identitách dosiahla 65% (pokles o 9% oproti základnému modelu). Pre model trénovaný na 10000 identitách to dopadlo ešte horšie. Presnosť zostala podobná (69% pre obe typy sietí), ale vzhľadom na lepšie výsledky pre základný model som zaznamenal zníženie presnosti pre sieť ResNet50 o 17% a pre sieť MobileNetV2 až o 21%.

Čo sa týka výsledkov vzhľadom na triedy naklonenia tváre, tak modely ukázali zhoršenie naprieč všetkými dvojicami tried.

4.5 Vyhodnotenie

Vykonal som niekoľko druhov experimentov na rôznych datasetoch s rôznou veľkosťou. Na začiatok som natrénoval model bez modifikácií, aby som mal základ oproti ktorému uvidím, či daná modifikácia pomohla lepšie klasifikovať dáta.

Zistil som, že datasety ktoré obsahovali potrebné anotácie boli však príliš malé a dosahovali veľmi zlé výsledky, v priemere presnosť 55%. Preto som sa rozhodol použiť väčší dataset, ktorý bol použitý pri vývoji danej knižnice. Dataset bol obrovský , preto som zobral prvých 1000, 2000, 5000 a 10000 osobností na ktorých som natrénoval základný model. Výsledky boli oveľa lepšie a ich priemer dosahoval presnosť 73% pre menšie datasety a presnosť 86% až 90% pre dataset s 10000 identitami.

Modely, ktoré pracovali s penalizáciou nefrontálnych snímkov a dynamickou penalizáciou zlepšili výsledky modelu. Pri tej prvej najlepší výsledok zlepšil presnosť modelu o 3% a pri dynamickej penalizácií to bolo až o 15%. Naopak experimenty s prídavnou penalizáciou neukázali zlepšenie, ba v niektorých prípadoch zhoršenie (v najlepšom prípade s presnosťou 72% a v najhoršom 65%).

Najlepším modelom, ktorý dosahoval priemernú presnosť na benchmarkovom datasete 90% bol model používajúci dynamickú penalizáciu s rozdelením uhlov naklonenia hláv do tried s intervalom 15 stupňov a sieťou ResNet50, natrénovaný na datasete MS1-Celeb-M s počtom identít 10000. Zlepšenie oproti základnému modelu bolo malé, len 4%. Avšak druhým najlepším modelom, ktorý dosahoval priemernú presnosť na benchmarkovom datasete 89% bol model s tými istými parametrami, ale natrénovaný na rovnakom datasete s počtom identít 5000. Tento dosiahol spomínané zlepšenie o 15% oproti modelu bez modifikácií.

Zaujímavým výsledkom týchto experimentov je práve to, že moje zlepšenia výrazne pomáhali modelom, ktoré boli trénované na menšom objeme dát. Pri väčšom datasete som dokázal vylepšiť presnosť modelu, ale to nebolo také viditeľné, resp. pri niektorých prípadoch nebolo žiadne.

Nakoniec uvediem porovnanie s pôvodne natrénovaným modelom od vývojárov knižnice. Výsledky majú evedené na svojom GitHub repozitári [24] a uviedol som ich taktiež pre ukážku na obrázku 4.1.

Pre porovnanie je relevantný prvý riadok v tabuľke na obrázku 4.1, pre sieť ResNet50 bez nastavenej možnosti CCrop, kde model dosahoval veľmi vysokých presností. Pre dataset LFW to bolo 99,42%, pre AgeDB-30 95,32% a pre CFP-FP 92,56%.

Môj najlepší model dosiahol presnosť pre dataset LFW 96,37%, čo sa veľmi blížilo modelu od vývojárov. Podobne aj výsledok 89,10% pre CFP-FP, ktorý ešte naviac na rozdiel od modelu od vývojárov predbehol svojou úspešnosťou výsledky na datasete AgeDB-30, kde som dosiahol presnosť 83,05%.

Môj druhý najlepší model dosiahol podobné výsledky. Presnosť pre dataset LFW bola 97,03%, čo je lepšie ako výsledok prvého modelu. Pre ďalšie datasety bol rozdiel väčší a to presnosť 82,02% pre AgeDB-30 a 86,99% pre CFP-FP.

Backbone	Head	Loss	CCrop	LFW	AgeDB-30	CFP-FP
ResNet50	ArcFace	Softmax	False	99.42	95.32	92.56
MobileNetV2	ArcFace	Softmax	False	99.13	91.62	91.50
ResNet50	ArcFace	Softmax	True	99.38	95.13	94.87
MobileNetV2	ArcFace	Softmax	True	98.88	91.58	93.19

Obr. 4.1: Verifikačné výsledky pôvodných modelov poskytnutých vývojármi knižnice arcface-tf2. Výsledky sú uvedené pre rôzne typy backbone sietí a augmentácií [24].

Keď sa pozrieme na priemery, tak model od vývojárov dosahoval v priemere na benchmarkových datasetoch presnosť 95,77% a môj najlepší model 89,51%. Rozdiel medzi mojím modifikovaným modelom na menšom počte dát a ich modelom natrénovanom na celom datasete činí približne 7%, čo pokladám za dobrý výsledok vzhľadom na spôsob trénovania.

Kapitola 5

Záver

Cieľom diplomovej práce bolo zoznámiť sa s existujúcimi riešeniami rozpoznávania tvárí so zameraním sa na nefrontálne rozpoznávanie, preskúmať možnosti existujúcich datasetov pre frontálne a nefrontálne rozpoznávanie, navrhnúť vlastné rozšírenie algoritmu ArcFace, implementovať ho pomocou knižnice Keras alebo PyTorch a porovnať s výsledkami pôvodného algoritmu ArcFace.

V úvode práce som opísal proces rozpoznávania tvárí od detekcie až k rozpoznávaniu identít. Popísal som *state-of-the-art* algoritmy používajúce sa v tomto procese. Uviedol som, ktoré algoritmy sú vhodné aj pre nefrontálne rozpoznávanie, aké majú výhody a nevýhody.

Spomenul som základné techniky strojového učenia ako neurónové siete a konvolučné neurónové siete. Ony tvoria základ existujúcich modelov pre rozpoznávanie osôb podľa tvárí. Následne som podrobne popísal algoritmy rozpoznávania tvárí ako ArcFace, FaceNet a DeepFace. Tieto modely majú rôzny prístup k rozpoznávaniu a aj s vysporiadaním sa s tvárami v nefrontálnych pozíciách. Preto som sa na nich zameral a podrobnejšie ich popísal.

Uviedol som existujúce datasety, ktoré sa používajú pre benchmarkové testovanie a trénovanie modelov frontálneho alebo nefrontálneho rozpoznávania tvárí. Nakoniec som uviedol knižnice pre vývoj modelov strojového učenia a existujúce riešenia implementujúce *state-of-the-art* algoritmy rozpoznávania osôb podľa tvárí.

Následne som navrhol riešenie postavené na algoritme ArcFace, konkrétne na jednej z jeho existujúcich implementácií. Riešenie pracuje s penalizáciou aditívneho uhlového okraja a snaží sa ju rôzne upravovať v závislosti či sa jedná o frontálnu, alebo nefrontálnu pozíciu. Postupne som navrhol penalizáciu nefrontálnych pozícií, dynamickú penalizáciu a nakoniec prídavnú penalizáciu nového okraja.

Riešenie som implementoval. V práci som uviedol miesta knižnice, ktoré som rozširoval o vlastný kód. Taktiež som pridal implementáciu testovania na testovacom datasete, ktorá nebola dostupná. Slúžila mi na dodatočnú verifikáciu, aby som mal aj iné dáta ako z benchmarkovch testov. Kvôli nedobrým výsledkom experimentov na datasetoch obsahujúcich anotácie natočenia hlavy som implementoval skript pre extrakciu tohto druhu dát z datasetu bez anotácií.

Nakoniec som vykonal experimenty pre jednotlivé modifikácie knižnice. Najlepším modelom, ktorý dosahoval priemernú presnosť na benchmarkovom datasete 90%, bol používajúci dynamickú penalizáciu s rozdelením uhlov náklonu tváre do tried s intervalom 15 stupňov. Model bol trénovaný na podmnožine datasetu MS1-Celeb-M obsahujúceho 10000 prvých identít. Dosiahol zlepšenie o 4% oproti modelu ArcFace bez modifikácií.

Druhým najlepším modelom (presnosť 89%) bol model s rovnakými parametrami natrénovaný na 5000 prvých identitách datasetu MS1-Celeb-M. Zaujímavosťou je, že som dosiahol pri ňom zlepšenie o 15% presnosti oproti modelu bez rozšírenia. Tento výsledok bol stále horší o 7% oproti modelu natrénovanom vývojármi knižnice. Považujem to ale za dobrý výsledok vzhľadom na to, že som mal dostupné obmedzené zdroje pre trénovanie.

Ďalšie rozšírenie môjho riešenia môže spočívať napríklad v augmentácií vstupných obrázkov. Jednak knižnica obsahuje nejaké základné orezanie obrázkov, bolo by však zaujímavé pozerať sa na to, ako by sa model správal, ak by dostal rôzne natočené obrázky, prípadne s nejakým šumom, či zakrytím. Ďalej by sa dalo viac pracovať s aditívnou penalizáciou. Výsledky tam nedosahovali dobrú presnosť, ale možno nejakými úpravami by sa mohli zlepšiť. Za pokus by stálo vyskúšanie využitie penalizácie nefrontálnych pozícií namiesto dynamickej penalizácie.

Literatúra

[1] Antoniadis, P. *Activation Functions: Sigmoid vs Tanh* [online]. Baeldung [cit. 2023-01-21]. Dostupné z: https://www.baeldung.com/cs/sigmoid-vs-tanh-functions.

[2] avcontentteam. *PCA: A Practical Guide to Principal Component Analysis in R Python* [online]. Analytics Vidhya [cit. 2023-01-21]. Dostupné z: https://www.analyticsvidhya.com/blog/2016/03/pca-practical-guide-principal-component-analysis-python/.

[3] Bae, G., La Gorce, M. de, Baltrušaitis, T., Hewitt, C., Chen, D. et al. DigiFace-1M: 1 Million Digital Face Images for Face Recognition. In: IEEE. *2023 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2023.

[4] Behera, G. S. *Face Detection with Haar Cascade* [online]. Towards Data Science [cit. 2023-01-07]. Dostupné z: https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08.

[5] Campos, N. *Integral Image in Hardware* [online]. sistenix.com [cit. 2023-05-09]. Dostupné z: https://sistenix.com/integral.html.

[6] Cheng, C. *Principal Component Analysis (PCA) Explained Visually with Zero Math* [online]. Towards Data Science [cit. 2023-01-08]. Dostupné z: https://towardsdatascience.com/principal-component-analysis-pca-explained-visually-with-zero-math-1cbf392b9e7d.

[7] Chu, L. *Human Neuron vs. Artificial Neuron: Similarities and Discrepancies* [online]. The Gradient [cit. 2023-01-08]. Dostupné z: https://lanstonchu.wordpress.com/2021/09/06/human-neuron-vs-artificial-neuron-similarities-and-discrepancies/.

[8] Contributors. *About Keras* [online]. Keras [cit. 2023-01-08]. Dostupné z: https://keras.io/about/.

[9] Contributors. *Dlib Homepage* [online]. Dlib [cit. 2023-01-08]. Dostupné z: http://dlib.net/.

[10] Contributors. *Konvoluce* [online]. Wikipedia [cit. 2023-01-21]. Dostupné z: https://cs.wikipedia.org/wiki/Konvoluce.

[11] Dalal, N. a Triggs, B. Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005, sv. 1, s. 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[12] Deng, J., Guo, J. a Zafeiriou, S. ArcFace: Additive Angular Margin Loss for Deep Face Recognition. *CoRR*. 2018, abs/1801.07698. Dostupné z: http://arxiv.org/abs/1801.07698.

[13] Foundation, T. L. a Contributors. *PyTorch Homepage* [online]. PyTorch [cit. 2023-01-08]. Dostupné z: https://pytorch.org/.

[14] Furnieles, G. *Sigmoid and SoftMax Functions in 5 minutes* [online]. Towards Data Science [cit. 2023-01-08]. Dostupné z: https://towardsdatascience.com/sigmoid-and-softmax-functions-in-5-minutes-f516c80ea1f9.

[15] Gandhi, R. *Support Vector Machine — Introduction to Machine Learning Algorithms* [online]. Towards Data Science [cit. 2023-01-08]. Dostupné z: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[16] Geitgey, A. *Face Recognition* [online]. GitHub [cit. 2022-11-29]. Dostupné z: https://github.com/ageitgey/face_recognition.

[17] Gourier, N. a Crowley, J. *Head Pose Image Database* [online]. International Workshop on Visual Observation of Deictic Gestures, Cambridge, UK [cit. 2023-04-02]. Dostupné z: http://crowley-coutaz.fr/Head%20Pose%20Image%20Database.html.

[18] Gourier, N. a Crowley, J. Estimating Face orientation from Robust Detection of Salient Facial Structures. *FG Net Workshop on Visual Observation of Deictic Gestures*. Január 2004.

[19] Gradilla, R. *Multi-task Cascaded Convolutional Networks (MTCNN) for Face Detection and Facial Landmark Alignment* [online]. Medium [cit. 2023-01-07]. Dostupné z: https://medium.com/@iselagradilla94/multi-task-cascaded-convolutional-networks-mtcnn-for-face-detection-and-facial-landmark-alignment-7c21e8007923.

[20] Guo, J., Deng, J., An, X., Yu, J., Gecer, B. et al. *InsightFace: 2D and 3D Face Analysis Project* [online]. GitHub [cit. 2023-01-08]. Dostupné z: https://github.com/deepinsight/insightface.

[21] Guo, J., Deng, J., Xue, N. a Zafeiriou, S. Stacked dense u-nets with dual transformers for robust face alignment. *ArXiv preprint arXiv:1812.01936*. 2018.

[22] Guo, Y., Zhang, L., Hu, Y., He, X. a Gao, J. MS-Celeb-1M: A Dataset and Benchmark for Large-Scale Face Recognition. *CoRR*. 2016, abs/1607.08221. Dostupné z: http://arxiv.org/abs/1607.08221.

[23] 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.

[24] Huang, K.-Y. *Arcface-tf2* [online]. GitHub [cit. 2023-01-08]. Dostupné z: https://github.com/peteryuX/arcface-tf2.

[25] Ippolito, P. P. *Feature Extraction Techniques* [online]. Towards Data Science [cit. 2023-01-08]. Dostupné z: https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be.

[26] Jaadi, Z. *A Step-by-Step Explanation of Principal Component Analysis (PCA)* [online]. builtin [cit. 2023-01-21]. Dostupné z: https://builtin.com/data-science/step-step-explanation-principal-component-analysis.

[27] Koech, K. E. *The Basics of Neural Networks (Neural Network Series) — Part 1* [online]. Towards Data Science [cit. 2023-01-08]. Dostupné z: https://towardsdatascience.com/the-basics-of-neural-networks-neural-network-series-part-1-4419e343b2b.

[28] Koech, K. E. *Cross-Entropy Loss Function* [online]. Towards Data Science [cit. 2023-01-08]. Dostupné z: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e.

[29] Kurama, V. *A Guide to AdaBoost: Boosting To Save The Day* [online]. PaperspaceBlog [cit. 2023-01-21]. Dostupné z: https://blog.paperspace.com/adaboost-optimizer/#:~:text=AdaBoost%20is%20an%20ensemble%20learning,turn%20them%20into%20strong%20ones.

[30] Lab, B. *Face Recognition Pipeline Clearly Explained* [online]. Medium [cit. 2023-01-07]. Dostupné z: https://medium.com/backprop-labs/face-recognition-pipeline-clearly-explained-f57fc0082750.

[31] Li, D. S. Z. a Jain, D. A. K. *Handbook of Face Recognition*. 2. vyd. Springer, 2011. ISBN 978-0857299314.

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

[33] MetaCentrum. *MetaCentrum (MetaVO) - virtuální organizace pro celou akademickou obec* [online]. MetaCentrum [cit. 2023-04-07]. Dostupné z: https://metavo.metacentrum.cz/.

[34] Mihir. *Locally Linear Embedding (LLE) | Data Mining and Machine Learning* [online]. Analytics Vidhya [cit. 2023-01-08]. Dostupné z: https://medium.com/analytics-vidhya/locally-linear-embedding-lle-data-mining-b956616d24e9.

[35] organization, G. M. I. R. a Contributors. *Basic classification: Classify images of clothing* [online]. TensorFlow [cit. 2023-01-08]. Dostupné z: https://www.tensorflow.org/tutorials/keras/classification.

[36] organization, G. M. I. R. a Contributors. *TensorFlow basics* [online]. TensorFlow [cit. 2023-01-08]. Dostupné z: https://www.tensorflow.org/guide/basics.

[37] organization, G. M. I. R. a Contributors. *TensorFlow Homepage* [online]. TensorFlow [cit. 2023-01-08]. Dostupné z: https://www.tensorflow.org/.

[38] ORGANIZATION, G. M. I. R. a CONTRIBUTORS. *TFRecord and tf.train.Example* [online]. TensorFlow [cit. 2023-05-07]. Dostupné z: https://www.tensorflow.org/tutorials/load_data/tfrecord#tfrecords_format_details.

[39] PATACCHIOLA, M. a CONTRIBUTORS. *What is Deepgaze?* [online]. GitHub [cit. 2023-04-15]. Dostupné z: https://github.com/mpatacchiola/deepgaze/tree/2.0.

[40] PAWANGFG. *Deep Face Recognition* [online]. Geeks for geeks [cit. 2023-01-08]. Dostupné z: https://www.geeksforgeeks.org/deep-face-recognition/.

[41] RAMAN257. *ML | Independent Component Analysis* [online]. Geeks for geeks [cit. 2023-01-08]. Dostupné z: https://www.geeksforgeeks.org/ml-independent-component-analysis/.

[42] ROBOTICS LABORATORY, D. o. C. S. a INFORMATION ENGINEERING, N. C. K. U. *Databases for Face Detection and Pose Estimation* [online]. National Cheng Kung University [cit. 2023-01-08]. Dostupné z: http://robotics.csie.ncku.edu.tw/Databases/FaceDetect_PoseEstimate.htm.

[43] ROSEBROCK, A. *OpenCV Eigenfaces for Face Recognition* [online]. pyimagesearch [cit. 2023-05-07]. Dostupné z: https://pyimagesearch.com/2021/05/10/opencv-eigenfaces-for-face-recognition/.

[44] SAHA, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* [online]. Towards Data Science [cit. 2023-01-08]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[45] SAINI, D. *BERT Embedding for Classification* [online]. Analytics Vidhya [cit. 2023-01-21]. Dostupné z: https://medium.com/analytics-vidhya/bert-embedding-for-classification-7c51aead26d9.

[46] SAINI, Y. *Face Recognition using Fisherfaces* [online]. OpenGenus [cit. 2023-05-07]. Dostupné z: https://iq.opengenus.org/face-recognition-using-fisherfaces/.

[47] SCHROFF, F., KALENICHENKO, D. a PHILBIN, J. FaceNet: A Unified Embedding for Face Recognition and Clustering. *CoRR*. 2015, abs/1503.03832. Dostupné z: http://arxiv.org/abs/1503.03832.

[48] SHAH, S. *Cost Function is No Rocket Science!* [online]. Analytics Vidhya [cit. 2023-01-08]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/02/cost-function-is-no-rocket-science/.

[49] SHARMA, S. *Activation Functions in Neural Networks* [online]. Towards Data Science [cit. 2023-01-08]. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[50] SIROVICH, L. a KIRBY, M. Low-Dimensional Procedure for the Characterization of Human Faces. *Journal of the Optical Society of America. A, Optics and image science*. Apríl 1987, zv. 4, s. 519–24. DOI: 10.1364/JOSAA.4.000519.

[51] Taigman, Y., Yang, M., Ranzato, M. a Wolf, L. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, s. 1701–1708. DOI: 10.1109/CVPR.2014.220.

[52] team, O. a Contributors. *Introduction* [online]. OpenCV [cit. 2023-01-08]. Dostupné z: https://docs.opencv.org/4.7.0/d1/dfb/intro.html.

[53] team, O. a Contributors. *OpenCV Homepage* [online]. OpenCV [cit. 2023-01-08]. Dostupné z: https://opencv.org/.

[54] Tyagi, M. *HOG (Histogram of Oriented Gradients): An Overview* [online]. Towards Data Science [cit. 2023-01-07]. Dostupné z: https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f#:~:text=What%20is%20it%3F,the%20purpose%20of%20object%20detection.

[55] Viola, P. a Jones, M. Rapid Object Detection using a Boosted Cascade of Simple Features. In:. Február 2001, sv. 1, s. I–511. DOI: 10.1109/CVPR.2001.990517. ISBN 0-7695-1272-0.

[56] viso.ai a Contributors. *Deepface* [online]. GitHub [cit. 2022-12-01]. Dostupné z: https://github.com/serengil/deepface.

[57] Wang, C. The Development and Challenges of Face Alignment Algorithms. *Journal of Physics: Conference Series*. 2019.

[58] Wolf, L., Hassner, T. a Maoz, I. Face recognition in unconstrained videos with matched background similarity. In: *CVPR 2011*. 2011, s. 529–534. DOI: 10.1109/CVPR.2011.5995566.

[59] Yiu, T. *Understanding Neural Networks* [online]. Towards Data Science [cit. 2023-01-08]. Dostupné z: https://towardsdatascience.com/understanding-neural-networks-19020b758230.

Príloha A

Obsah pamäťového média

- `src` - zdrojové kódy aplikácie
  - `annotation_extraction` - implementácia extrakcie anotácií
  - `arcface-tf2` - implementácia modelu ArcFace
  - `datasets` - anotácie so súbormi tfrecord a testovacími dvojicami
  - `metacentrum_scripts` - skripty pre spúšťanie úloh na službe MetaCentrum
  - `documentation.md` - dokumentácia k zdrojovým kódom
  - `requirements_arcface.txt` - závislosti pre arcface-tf2
  - `requirements_deepgaze.txt` - závislosti pre extrakciu anotácií
- `results` - výsledky experimentov s natrénovanými modelmi
- `thesis.zip` - zdrojové kódy textu práce
- `Rospoznavani-osob-podle-obliceje-v-nefrontalnich-pozicich.pdf` - pdf verzia práce