VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HLUBOKÉ NEURONOVÉ SÍTĚ PRO ROZPOZNÁNÍ TVÁŘÍ VE VIDEU

DEEP NEURAL NETWORKS FOR FACIAL RECOGNITION IN VIDEO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN STRATIL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL HRADIŠ, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19665/2016/xstrat10

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Stratil Jan**
Obor: Informační technologie
Téma: **Hluboké neuronové sítě pro rozpoznání tváří ve videu**
Deep Learning for Facial Recognition in Video
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte základy teorie neuronových sítí, konvolučních neuronových sítí a zpětné propagace chyb.
2. Vytvořte si přehled o současných metodách pro rozpoznávání tváří pomocí konvolučních hlubokých neuronových sítí ve video sekvencích.
3. Vyberte konkrétní metody a aplikujte je na úlohu rozpoznání tváří ve video sekvencích.
4. Obstarejte si databázi vhodnou pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty nad datovou sadou.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Taigman et al.: DeepFace: Closing the Gap to Human-Level Performance in Face Verification. CVPR 2014.
- Parkhi, Omkar M., Andrea Vedaldi, and Andrew Zisserman. "Deep face recognition." Proceedings of the British Machine Vision 1.3 (2015): 6.
- Yang, Jiaolong, et al. "Neural aggregation network for video face recognition." arXiv preprint arXiv:1603.05474 (2016).

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Hradiš Michal, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá rozpoznáváním tváří ve videu pomocí hlubokých neuronových sítí. Tato úloha je rozdělena na 2 části. První část se zabývá trénováním sítě, která vytváří kompaktní příznakový vektor reprezentující identitu tváře ze snímku videa. Druhá část se zabývá trénováním agregační sítě, která vytvořené příznakové vektory agreguje v jeden. Tato agregace je rychlá a ukázala se být lepší než pooling metody. Výsledky jsou testovány na datasetu *LFW*, kde dosažená přesnost je 92.8% a na datasetu *YTF*, kde přesnost je 84.06%.

Abstract

This bachelor's thesis deals with facial recognition in video using deep neural networks. This task is split into 2 parts. The first part deals with training network that produces compact feature vector which represents the face identity from a video frame. The second part deals with training aggregation network that aggregates those feature vectors into one. This aggregation is fast and it has shown that its results are better than naive pooling methods. Results are tested on the *LFW* dataset, where it achieves 92.8% accuracy and on the *YTF* dataset, where the accuracy is 84.06%.

Klíčová slova

Rozpoznávání tváří, hluboké neuronové sítě, konvoluční neuronové sítě, strojové učení, příznakový vektor, agregace

Keywords

Facial recognition, deep neural networks, convolutional neural networks, machine learning, feature vektor, aggregation

Citace

STRATIL, Jan. *Hluboké neuronové sítě pro rozpoznání tváří ve videu.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Hluboké neuronové sítě pro rozpoznání tváří ve videu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Hradiše, Ph.D.. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Stratil
17. května 2017

Poděkování

Tímto bych chtěl velmi poděkovat mému vedoucímu práce Ing. Michalovi Hradišovi, PhD., za skvělé vedení, ochotu, cenné přípomínky a rady, které mi poskytl při řešení této práce.
Dále bych chtěl poděkovat firmě Innovatrics za poskytnutá data, anotace k datasetům a natrénovanou síť.
V neposlední řadě bych chtěl poděkovat výpočetnímu centru MetaCentrum. Velmi oceňují přístup k výpočetním a úložným zařízením, které vlastní skupiny a projekty přispívající k Národní Gridové Infrastruktuře MetaCentrum, který je poskytován pod programem "Projekty Velkého Výzkumu, Vývoje a Inovativních Infrastruktur" (CESNET LM2015042).

Obsah

1 Úvod **2**

2 Neuronové sítě a jejich trénování **3**
2.1 Model neuronu a konvoluce . . . 3
2.2 Učení neuronových sítí . . . 4
2.3 Vybrané vrstvy neuronových sítí . . . 6
2.4 Aktivační funkce . . . 8

3 Rozpoznávání tváří **10**
3.1 Zpracování dat . . . 11
3.2 Sítě pro extrakci příznakových vektorů . . . 11
3.3 Agregace příznakových vektorů . . . 13

4 Dostupné datasety a příprava dat **15**
4.1 Obrázkové datasety . . . 15
4.2 Video datasety . . . 16
4.3 Příprava dat pro trénování . . . 16

5 Návrh řešení a implementace **19**
5.1 Sítě pro extrakci příznakových vektorů . . . 19
5.2 Agregace . . . 21
5.3 Generování trénovacích dat . . . 21
5.4 Implementace . . . 22

6 Experimenty a výsledky **23**
6.1 Sítě bez batch normalizace . . . 23
6.2 Vliv batch normalizace . . . 24
6.3 Vyhodnocení na PFC . . . 25
6.4 Agregace příznakových vektorů . . . 25
6.5 Vyhodnocení na benchmark datasetech . . . 27

7 Závěr **33**

Literatura **34**

Přílohy **37**

A Obsah přiloženého DVD **38**

Kapitola 1

Úvod

V poslední době je zaznamenáván velký pokrok a růst konvolučních neuronových sítí, zejména jejich využití pro řešení úloh z oblasti počítačového vidění. Rozpoznávání tváří je jedna z nich a touto úlohou se tato práce zabývá.

Rozpoznávání tváří se v dnešní době používá hned na několika místech. Skoro každý někdy slyšel o odemykání telefonu či počítače pomocí obličeje, kde se verifikuje, zda-li se daný obličej snímaný kamerou shoduje s obličejem, který patří majiteli telefonu či počítače. Další typické využití může být u různých bezpečnostních složek, kdy je potřeba identifikovat osoby z bezpečnostní kamery, nebo v bezpečnostních systémech na letištích, kde se porovnávají obličeje cestujících procházející skrz turniket, jestli se neshodují s některým obličejem právě hledaných osob.

Pro rozpoznávání tváří existuje několik algoritmů a způsobů, jak k této problematice přistupovat. Dříve se využívaly metody bez neuronovových sítí. Mezi ně patří například *Eigen-Pep* [17], *Local Binary Pattern* [1] a mnoho dalších. Dnes ale dominují metody založené na neuronových sítích, díky čemuž se v této problematice dosáhlo lepších výsledků, které jsou srovnatelné s výsledky lidí [32]. Právě těmito metodami se tato práce zabývá.

Řešení je rozděleno na 2 části. V první části se trénuje konvoluční neuronová síť, která je schopná pro obrázky, jež už jsou zpracované (detekované, oříznuté a zarovnané tváře v obrázcích), vypočítat kompaktní příznakový vektor, který reprezentuje identitu obličeje z obrázku. Cílem trénování je, aby síť produkovala příznakové vektory co nejvíce podobné, pokud jsou tváře z obrázků stejného člověka a naopak co nejméně podobné, pokud tváře nejsou stejného člověka.

V druhé části jsou tyto příznakové vektory agregovány z jednotlivých snímků videa. Je zde snaha tyto příznakové vektory ohodnotit podle toho, jak jsou ve videu relevantní a pomocí tohoto ohodnocení vytvořit nový vektor váženým součtem.

V kapitole 2 je vysvětlen základní princip neuronových a konvolučních neuronových sítí, jakým způsobem fungují, jak se trénují, jaké obsahují vrstvy a další. Kapitola 3 pojednává o předchozích pracích, které se zabývaly rozpoznáváním tváří pomocí konvolučních neuronových sítí, jejich architekturách a trénování. Dostupné datasety a stručné informace o nich se nacházejí v kapitole 4. Stejně tak zde lze nalézt způsob, kterým byly předpřipraveny data před použitím při trénování. Implementační detaily, jako prostředí, ve kterém se trénovaly sítě a architektury sítí, které byly v této práci použity jsou popsány v kapitole 5. Stejně tak je zde popsán návrh řešení celého problému. Kapitola 6 je věnována experimentům, které byly provedeny. Dále zde jsou porovnány některé architektury a jsou zde vyhodnoceny sítě na benchmark datasetech. V závěru 7 jsou shrnuty výsledky práce a je zde diskutován budoucí vývoj.

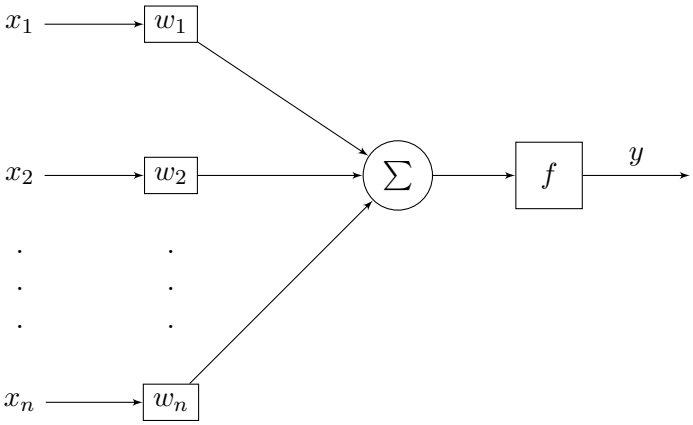
Kapitola 2

Neuronové sítě a jejich trénování

Neuronové sítě vycházejí z existujících biologických neuronových sítí v lidském mozku. Základním prvkem v mozku je biologický neuron. Podobně jako v mozku je základním prvekm neuronových sítí umělý neuron, což je určitá abstrakce biologického neuronu. Tyto neurony se dají považovat za výpočetní modely nebo také funkce, kdy je pro určitý vstup vypočítán výstup [21, 23].

2.1 Model neuronu a konvoluce

Dnes jeden z nejpoužívanějších modelů je model z roku 1943 navrhnutý panem **McCullochem** a panem **Pittsem** [19]. Na toto navázal pan **Rosenblatt**, který v roce 1958 pro neuron definoval učící metodu [27]. Více o historii a vývoji neuronových sítí lze nalézt v [21].



Obrázek 2.1: Model umělého neuronu

Každý neuron má definovaný vstupní vektor $x = (x_1, x_2, \ldots, x_n)$ a k němu připojený a jemu odpovídající vektor s váhami $w = (w_1, w_2, \ldots, w_n)$. Neurony mohou také obsahovat skalární hodnotu zvanou *bias*. Dále obsahují tzv. *aktivační funkci* (někdy také popisovaná jako přenosová funkce).

Na obrázku 2.1 lze vidět, že vstupní hodnoty, které vstupují do neuronu, se každá vynásobí váhou jí určenou. Tyto vynásobené hodnoty se poté sečtou a vstupují do aktivační

funkce. Výstup této *aktivační funkce* je totožný s výstupem neuronu. Matematicky lze závislost vyjádřit jako

$$y = f(\sum_{i=0}^{N}(x_i \cdot w_i) + b), \tag{2.1}$$

kde x_i jsou vstupy do neuronu, w_i jsou váhy neuronu odpovídající vstupům, b je bias daného neuronu, f je aktivační funkce (více o aktivačních funkcích v 2.4) a y je výstupní hodnota neuronu. Právě *bias* a *váhy* se trénují za účelem minimalizace chyby [23].

Konvoluce. Konvoluce je operace reprezentována znakem $*$ a jak z názvu vyplývá, je to základní operace v konvolučních neuronových sítích [7]. Obecně se jedná o operaci nad dvěma funkcemi a její definice je

$$s(t) = x(t) * w(t) = \int x(a)w(t-a)\mathrm{d}a. \tag{2.2}$$

V tomto případě je funkce $s(t)$ výstupní funkcí po konvoluci vstupních funkcí $x(t)$ a $w(t)$. Tato konvoluce je definována nad spojitými vstupy, ale v počítačovém vidění se pracuje s diskrétními hodnotami, proto je potřeba diskrétní konvoluce, která je definována jako

$$s(t) = x(t) * w(t) = \sum_{a=-\infty}^{\infty} x(t-a)w(a). \tag{2.3}$$

Obecně je vstupem x tensor (může být například obrázek) a vstupem w je také tensor zvaný *konvoluční jádro* (někdy také konvoluční filtr). Pro 2D konvoluci pro bod $[i, j]$ v obrázku platí

$$s[i,j] = x[i,j] * w[i,j] = \sum_{m}\sum_{n} x[i-m, i-n]w[m,n], \tag{2.4}$$

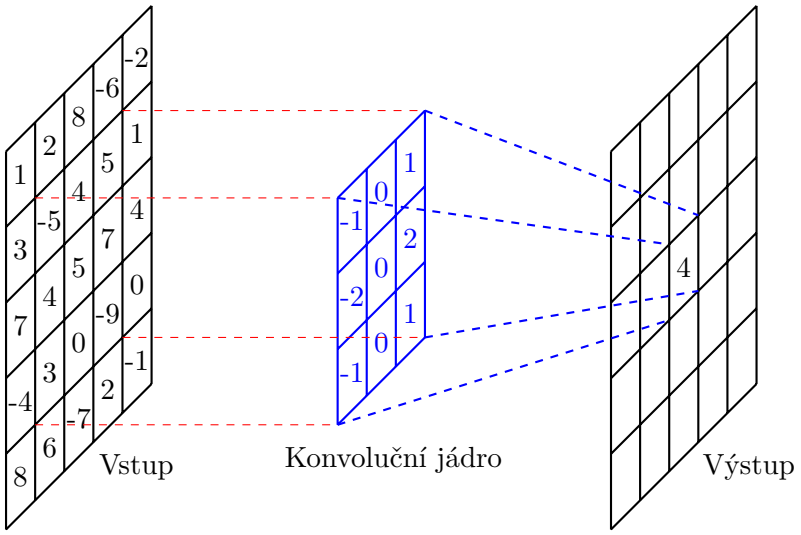
kde w je konvoluční jádro o velikosti $m \times n$ a x je vstupní obrázek. Pro ilustraci obrázek 2.2. Z obrázku lze vidět, že nastane problém, pokud se bude počítat konvoluce v bodech na hranicích obrázku a zároveň bude konvoluční jádro mít alespoň jeden z rozměrů větší jak 1. K tomuto problému se obecně přistupuje dvojím způsobem:

1. Konvoluce se nepočítá na těchto hraničních bodech, kde konvoluční jádro přesahuje hranice obrázku. Toto způsobí zmenšení obrázku tak, že se šířka obrázku zmenší o šířku konvolučního jádra - 1 $(m-1)$ a výška se zmenší o výšku konvolučního jádra - 1 $(n-1)$. Této variantě se říká úplná konvoluce.

2. Vstupní obrázek je zvětšen o stejné rozměry, o které by se obrázek zmenšil a hodnoty v těchto oblastech se nastaví například na 0, nebo se mohou pouze rozšířit hodnoty, které jsou v hraniční oblasti. Této variantě se říká stejná konvoluce.

2.2 Učení neuronových sítí

K tomu, aby neuronové sítě fungovaly, musí existovat nějaký způsob, kterým se mění váhy neuronů tak, aby dělaly to, co se po nich žádá. Využívá se k tomu standardní optimalizace, kde se minimalizuje chybová funkce na určitém datasetu. Dataset obsahuje trénovací data a očekáváný výsledek, který chceme, aby síť produkovala. Chybová funkce určuje jak moc se síť odchyluje od očekávané hodnoty.



Obrázek 2.2: Ukázka konvoluce

Mean Square Error. Mean Square Error je základní využívaná chybová funkce. Její definice je

$$MSE = E = \frac{1}{P}\sum_{p=1}^{P}\sum_{j=1}^{J}(t_{p,j} - y_{p,j})^2, \tag{2.5}$$

kde J je velikost výstupního vektoru, P je počet trénovacích vzorků, ze kterých se počítá chyba, $t_{p,j}$ je očekávaná výstupní hodnota (tzv. *ground truth* nebo *target*) a $y_{p,j}$ je výstupní hodnota vypočítaná neuronovou sítí [21].

Cross Entropy Loss. Cross Entropy Loss je chybová funkce používaná pro klasifikaci 1 z K. Její předpis je

$$E = \frac{1}{P}\sum_{p=1}^{P} -\log\left(\frac{e^{s_p}}{\sum_{j=0}^{J} e^{y_{j,p}}}\right), \tag{2.6}$$

kde P je celkový počet trénovacích vzorků, j je index ve výstupním vektoru o velikost J, s_p je predikovaná hodnota správně klasifikované třídy a $y_{j,p}$ je predikovaná hodnota s indexem j [12, 26].

Gradient Descent. *Gradientní sestup* je iterativní algoritmus, vymyšlený panem Cauchym v roce 1847 [2], který se používá při učení. Základním principem je výpočet parciálních derivací chybové funkce vůči jednotlivým parametrům sítě. Parciální derivace společně tvoří gradient (∇G), který udává směr, kterým chyba nejvíc roste. Opačný směr je právě ten, který se používá k minimalizaci chyby. Změna vah je tedy definována jako

$$w_{i+1} = w_i - \eta\nabla G(w_i), \tag{2.7}$$

kde w_i jsou váhy v dané iteraci i, $\eta > 0$ je učící konstanta (tzv. *learning rate*, která definuje jak rychle se budou měnit váhy) a ∇G je gradient vypočítaný z aktuální iterace. Tyto iterace se dále opakují, dokud chybová funkce není nižší než požadovaná hodnota, nebo dokud se chybová funkce nepřestane snižovat.

V praxi se používá *stochastický gradientní sestup* (*SGD*), kde rozdíl je v tom, že se nepoužívají všechna trénovací data, ale jenom jejich část (tzv. *mini-batch*). Tímto nedochází k přesnému výpočtu chyby, ale pouze k její aproximaci, což nevadí, protože i přesto se dojde k minimu. Více informací lze najít v [21, 23].

Backpropagation. *Zpětné šíření chyby* je algoritmus vymyšlený v roce 1970 v rámci diplomové práce panem Linnainmaaem [18], který se využívá pro efektivní výpočet parciálních derivací jednotlivých vah neuronů. Využívá se zde tzv. *chain rule* (řetězové pravidlo) pro parciální derivace. Nechť

$$z = f(y),$$
$$y = g(x),$$

pak pomocí chain rule lze zapsat

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x}. \tag{2.8}$$

Více o *backpropagation* lze nalézt v [16, 21, 23].

Inicializace vah. Inicializace vah je důležitým prvkem při trénování sítí. Špatná inicializace může někdy být důvodem pomalého trénování nebo i divergování. Naopak vhodná inicializace vah může vést k rychlejšímu trénování vah. V zásadě je vhodné a doporučované používat předtrénované vrstvy, pokud sítě, ze kterých se váhy využívají, byly trénovány pro podobný problém. Více o inicializaci lze nalézt v [22].

2.3 Vybrané vrstvy neuronových sítí

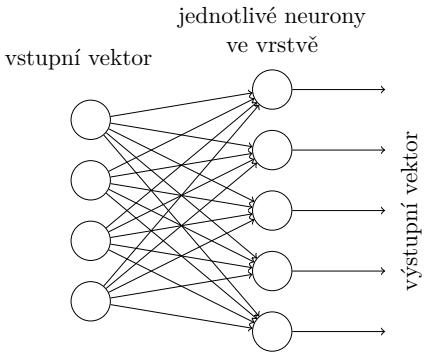
V této sekci jsou popsány některé základní vrstvy používané v neuronových sítí, které jsou zároveň využity v této práci. Data, která vstupují a vystupují do a z jednotlivých vrstev jsou obecně nazývány tensory (na který lze hledět jako na vícedimenzionální matici). Výstupním tensorům se také někdy říká aktivace.

Konvoluční vrstva. Princip konvoluční vrstvy byl poprvé použit v článku [5] v roce 1980, kdy se ještě nenazývala konvoluční vrstvou. Základní princip konvoluce byl popsán výše (sekce 2.1). Kromě *paddingu* (výběr úplné či stejné konvoluce), jehož princip byl vysvětlen také výše, se u této vrstvy nastavují vlastnosti jako je počet výstupních kanálů a velikost konvolučního jádra. Tyto informace spolu s počtem vstupních kanálů udávají, kolik parametrů daná vrstva obsahuje. Dále pak *stride* určuje s jakým prostorovým krokem se konvoluce bude počítat. Pokud je *stride* > 1 dochází ke zmenšování prostorové velikosti vstupu. Součin počtu vstupních a výstupních kanálů udává celkový počet konvolučních jader.

Konvoluce se zde počítá narozdíl od jednoduché konvoluce přes všechny vstupní kanály. Jednotlivé jednoduché konvoluce se dohromady sčítají. Rovnice pro bod $[i, j]$ je

$$s_r[i,j] = \sum_c \sum_m \sum_n (x_c[i-m, i-n] w_{c,r}[m,n]) + b_r, \tag{2.9}$$

kde r je index výstupního kanálu, i a j jsou souřadnice daného bodu, c je index kanálu, m a n jsou rozměry konvolučních jader, x_c je vstupní kanál, $w_{c,r}$ je konvoluční jádro pro daný vstupní kanál c a výstupní kanál r a b_r je bias pro daný výstupní kanál r.



Obrázek 2.3: Ukázka propojení plně propojené vrstvy

Plně propojená vrstva. Plně propojená vrstva se skládá z jednotlivých neuronů, kde všechny hodnoty vstupního vektoru jsou připojeny ke každému neuronu této vrstvy. Výstupem této vrstvy je taktéž vektor, jehož velikost je totožná s počtem neuronů v síti. Výstupy jednotlivých neuronů reprezentují společně výstupní vektor. Pro představu viz obrázek 2.3.

Pooling vrstvy. *Podvzorkovací vrstvy* jsou vrstvy, které agregují více hodnot z lokálního okolí do hodnoty jedné. V závislosti na nastavení prostorového kroku tato vrstva snižuje nebo nijak neovliňuje prostorovou velikost výstupu, toto nastavení je analogické jako v 2.3. To, kolik hodnot se agreguje do jedné je libovolné. Způsoby agregace jsou například výběr maximální nebo minimální hodnoty, nebo výpočet průměrné hodnoty [12].

Dropout vrstva. První zmínka o dropoutu je v článku [9]. Tato vrstva náhodně nastavuje některé aktivace na 0. Zde se nastavuje jaká procentuální část výstupů se bude nulovat. Tato vrstva se používá jako jeden ze způsobů regularizace při trénování neuronových sítí. Je nežádoucí, aby se neuronová síť rozhodovala například na základě výstupu jednoho neuronu nebo se naučila pouze na trénovací sadě a na validační či testovací sadě měla velkou chybovost. Tomu se říká přetrénovávání (tzv. *overfitting*).

BatchNormalization vrstva. Tato vrstva byla poprvé zmíněna v článku [11]. Vrstva normalizuje hodnoty napříč *mini-batchemi*. Zajišťuje, aby aktivace měly průměr 0 a standardní odchylku 1. Aktivace se tedy pohybují okolo 0. V průběhu trénování si vrstva počítá a ukládá statistiky, které využívá když se síť netrénuje. Tato vrstva se v praxi umisťuje mezi konvoluční nebo plně propojenou vrstu a aktivační vrstvu (typicky je to $ReLU$). Ukázalo se, že díky této vrstvě se nemusí dbát takový důraz na správnou počáteční inicializaci vah a také se může nastavit větší learning rate. Další vlastností je také regularizace. To je způsobeno vytvářením určitého šumu při normalizaci. V některých případech může zcela nahradit dropout vrstvu. Batch normalizace má však svoje praktické nevýhody a ty jsou v přidání další náročné vrstvy, což vede ke zpomalení samotného trénování.

Softmax vrstva. Tato vrstva, popsaná například v [12], je jedna z aktivačních vrstev, která se využívá například pro klasifikaci 1 z K. Tato vrstva převádí vstupní vektor na vektor pravděpodobností, kde každá hodnota reprezentuje pravděpodobnost, že vstup odpovídá

právě této třídě. Její definice je

$$y_k = \frac{e^{x_k}}{\sum_{n=1}^{N} e^{x_n}}, \tag{2.10}$$

kde indexy k a n jsou indexy vstupního vektoru x a N je velikost tohoto vektoru.

2.4 Aktivační funkce

Byl zde použit pojem aktivační funkce. Aktivační funkce je nelineární funkcí, kterých existuje několik. V dalším textu jsou uvedeny některé aktivační funkce, které se v praxi používají. Na obrázku 2.4 jsou znázorněny grafy jednotlivých aktivačních funkcí.

Binární aktivační funkce. Binární aktivační funkce neboli Heavisideova funkce zmíněná v [21, 23] je jedna z nejjednodušších aktivačních funkcí. Jedná se o skokovou aktivační funkci, jejíž předpis může vypadat například takto:

$$f(x) = \begin{cases} 1 & \text{pro } x >= t \\ 0 & \text{pro } x < t \end{cases} \tag{2.11}$$

kde x je vstupní hodnota aktivační funkce a t je práh (*tzv. threshold*, který znační hranici, kde se hodnota překlápí do hodnoty jiné). Definiční obor této funkce je $D(f) = \mathcal{R}$ a obor hodnot je diskrétní množina $H(f) = \{-1, 1\}$. Aktivační funkce je znázorněna na obrázku 2.4a. Tato funkce reprezentuje stavy neuronu, zda-li je zapnutý nebo vypnutý.

Sigmoida. Sigmoida, která je zmiňována v [21, 23], je definována předpisem

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{2.12}$$

kde x je vstupní hodnota aktivační funkce. Definiční obor této funkce je $D(f) = \mathcal{R}$ a obor hodnot je interval $H(f) = (0, 1)$. Aktivační funkce je znázorněna na obrázku 2.4c. Tato funkce nejvíce odpovídá tomu, jak se přenáší nervové vzruchy v biologickém neuronu.

Hyperbolický tangens. Hyperbolický tangens *tanh*, zmíněn v [21, 23], je definován předpisem

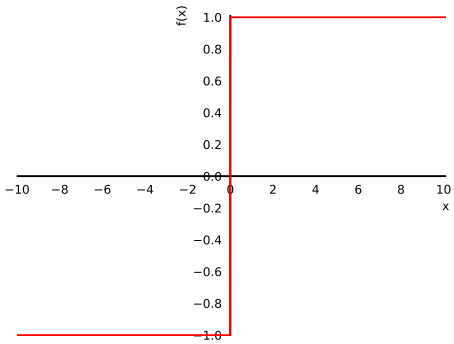
$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}, \tag{2.13}$$

kde x je vstupní hodnota aktivační funkce. Definiční obor této funkce je $D(f) = \mathcal{R}$ a obor hodnot je interval $H(f) = (-1, 1)$. Aktivační funkce je znázorněna na obrázku 2.4b.

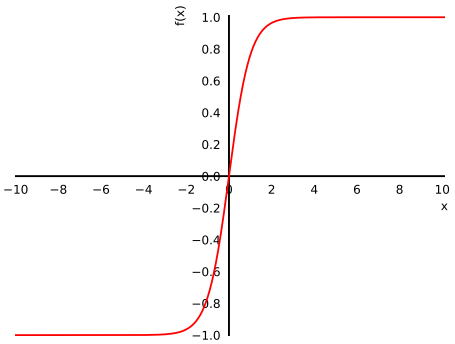
ReLU. *Rectified Linear Unit* [24] je definován předpisem

$$f(x) = max(0, x), \tag{2.14}$$

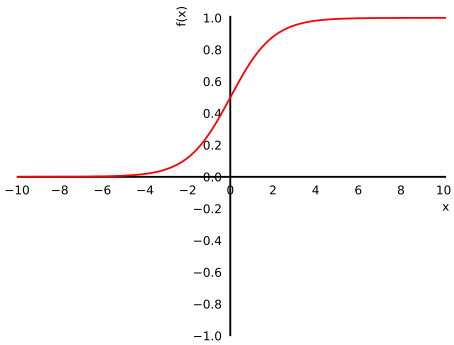
kde x je vstupní hodnota aktivační funkce. Definiční obor této funkce je $D(f) = \mathcal{R}$ a obor hodnot je interval $H(f) = \langle 0, \infty)$. Aktivační funkce je znázorněna na obrázku 2.4d. V článku [6] je poukázáno na to, že využití ReLU má lepší výsledky než *sigmoida* a *hyperbolický tangens*.



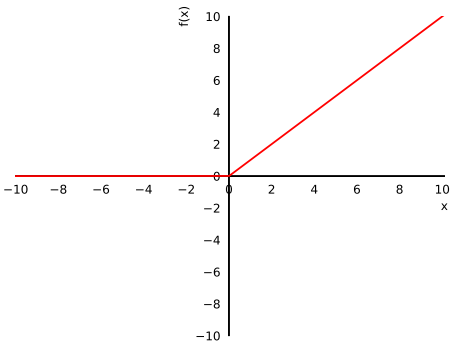
(a) Binární



(b) Hyperbolický tangens



(c) Sigmoida



(d) ReLU

Obrázek 2.4: Grafy aktivačních funkcí

Kapitola 3

Rozpoznávání tváří

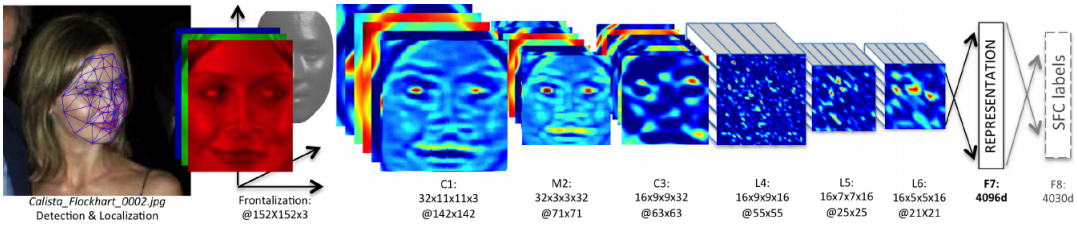
V této kapitole jsou popsány existující *state of the art* metody pro rozpoznávání tváří. Ze všeho nejdřív je ale potřeba definovat jaké úlohy se při rozpoznávání řeší. Tyto úlohy jsou obecně dvě – **identifikace** a **verifikace**.

Identifikace. Identifikace je proces určení identity člověka. Existuje-li nějaká databáze s reprezentacemi tváří, která obsahuje právě N reprezentací tváří, pak identifikace probíhá porovnáním každé dvojice. Dále můžeme seřadit reprezentace z databáze, od nejvíce podobných reprezentací po nejméně podobné. Můžeme například vzít nejpodobnější reprezentaci a v závislosti na určitém prahu (*threshold*) se rozhodnout, jestli tyto reprezentace jsou jedné a té samé identity. Tento práh je vždy jiný v závislosti na situaci, pokud je třeba identifikovat teroristu nebo hledaného člověka ve video záznamu, tento práh může být vyšší (méně striktní), aby byly upozorněny bezpečnostní složky a následně se provedla identifikace jiná než automatická. Pokud se naopak budou porovnávat reprezentace tváří za účelem vstupu do nějakého zabezpečeného sektoru, tento práh bude typicky nižší (více striktní). Výsledkem identifikace je získaní identity člověka nebo informace, že se daný člověk v databázi nenachází.

Verifikace. Verifikace je proces ověření, zda 2 reprezentace tváře náleží jedné a té samé identitě. Stejně jako v případě identifikace zde dochází k porovnání v závislosti na určitém prahu, který je vhodně zvolen podle situace, ve které se porovnává. Výsledkem verifikace je informace, zda tváře náleží stejné osobě či nikoliv.

Reprezentace tváře. Byla zde zmíněná reprezentace tváře, což je obecně n-rozměrný vektor, který je tvořen reálnými čísly. Porovnání se provádí například na základě vzdáleností těchto vektorů (euklidova nebo cosinova vzdálenost). Dále je tento vektor používán jako příznakový vektor.

V následujícím textu jsou shrnuty jednotlivé kroky pro rozpoznávání tváří. Postupně je popsána příprava dat pro trénování a obecně je zde popsán jaký formát musí mít tvář, která se má zpracovat. Jelikož tato práce pojednává o rozpoznávání pomocí hlubokých neuronových sítí, jsou zde zmíněny pouze tyto metody, jejich architektury a způsob trénování pro extrakci příznakových vektorů. Jako poslední jsou popsány agregační funkce, které tyto vytvořené příznakové vektory agregují v jeden. Ukázalo se, že vektor nemusí být nijak veliký, aby dosahoval dobrých výsledků. V článku [29] má výsledný vektor velikost 128.



Obrázek 3.1: Ukázka 3D zarovnání a architektury DeepFace, převzato z [32].

3.1 Zpracování dat

Obrázky, ze kterých se mají rozpoznávat identity musí obsahovat pouze jednu tvář. Musí tedy dojít k detekci a správnému oříznutí tváře. Pro to, aby síť dosahovala lepších výsledků musí být tváře nějakým způsobem zarovnané.

2D zarovnání. Jedním z jednoduších zarovnání je právě 2D zarovnání. Toto zarovnání je použito například v článku [26]. Jedná se o pouhou affinní transformaci, která podle vybraných významných bodů na tváři zarovná tvář. Těmito body mohou být třeba oči, nos, pusa a další.

3D zarovnání. Dalším o něco komplikovanějším způsobem je 3D zarovnání. Tento způsob je použit například v [32]. Pro obličej se vytvoří 3D model a pomocí něho se obličej transformuje tak, aby obličej byl frontální, tedy na něj bylo hleděno přímo zepředu. Díky tomu je zajištěno, že jednotlivé významné body na obličeji jsou vždy na stejné pozici, nezávisle na tom, jak je původní obličej otočený. Ukázka 3D zarovnání je na obrázku 3.1.

Dalším použitím vytvořeného 3D modelu může být syntetizování nových dat pro trénování. Obličej může být libovolně natočený nebo může být upravený jeho tvar. Tohoto přístupu se využívá v [28].

3.2 Sítě pro extrakci příznakových vektorů

Jak bylo výše popsáno, pro reprezentaci tváří jsou použity vektory. Tyto reprezentace by se vůbec nemusely používat a tváře by se mohly porovnávat přímo z obrázků, kde vstupem by byly 2 obrázky a pomocí binárního klasifikátoru by se určilo, zda-li jsou tváře stejné identity či nikoliv. Toto by ale vedlo k problému v případě, že by existovala velká databáze a úkolem by například bylo najít nejpodobnější obličej. Toto by bylo velmi neefektivní a z tohoto důvodu vznikla snaha reprezentovat tvář pomocí nějakého kompaktního vektoru. Pro jednotlivé tváře se daný vektor vytvoří pouze jednou a následně se pak například při vyhledávání nejpodobnějšího obličeje porovnávají pouze tyto vektory. V další části jsou popsány jaké existují architektury sítí a způsoby, kterými se trénují sítě, které tento příznakový vektor extrahují.

Architektury. Velký průlom byl zaznamenán publikováním článku [32]. Výsledky rozpoznávání tváří jsou zde srovnatelné s výsledky, kterých dosahovali lidé (Na *LFW* datasetu pomocí *unrestricted protokolu* [10] dosahovala síť přesnosti 97.35%, lidé dosáhli úspěšnosti 97.53%). Použitá architektura je ukázána na obrázku 3.1. Vzhledem k použitému 3D zarovnání v tomto článku jsou použité lokálně propojené konvoluční vrstvy. Tato vrstva je

podobná jako konvoluční vrstva s tím rozdílem, že pro každý pixel má jiný konvoluční filtr. Což znamená, že každá část tváře se bude zpracovávat jiným způsobem. Toto vede k velkému počtu parametrů ve vrstvě. V této architektuře je kolem 120 milionů parametrů a z toho 95% tvoří právě parametry této vrstvy. Pro natrénování této sítě bylo použito 4 milionů obrázků.

Další architektura, která dosahuje dobrých výsledků (na *LFW* datasetu [10] je dosaženo přesnosti 98.98%), je popsána v článku [26]. Tato architektura je identická s architekturou *VGG-16* [30] a obsahuje velké množství parametrů, protože obsahuje 16 konvolučních vrstev a plně propojené vrstvy s velikostí 4096-4096-1024. Tyto konvoluce obsahují konvoluční jádro o velikosti 3×3 a aktivaci *ReLU*. Tyto vrstvy jsou prokládány max pooling vrstvami a nakonci jsou umístěny plně propojené vrstvy. Pro natrénování této sítě bylo použito 2.6 milionu obrázků.

Výše uvedené architektury obě obsahovaly velké množství parametrů. Příkladem sítě, která obsahuje mnohem menší počet parametrů (7.5 milionu), je síť z [29]. Tato síť je založená na incepční vrstvě, která byla poprvé zmíněna v [31]. Tato vrstva provádí několik konvolucí, nejprve je využita konvoluce o velikosti 1×1. Pomocí této konvoluce je snižován počet vstupních kanálů pro následující konvoluce. Díky tomuto snižování počtu kanálů je výrazně redukován celkový počet parametrů. Ač je zde menší počet parametrů, přesnost této sítě na *LFW* datasetu [10] je 99.63%, což je lepší výsledek jak ve výše zmíněných sítích. Na druhou stranu pro natrénování této sítě byl použit větší počet obrázků (200 milionů).

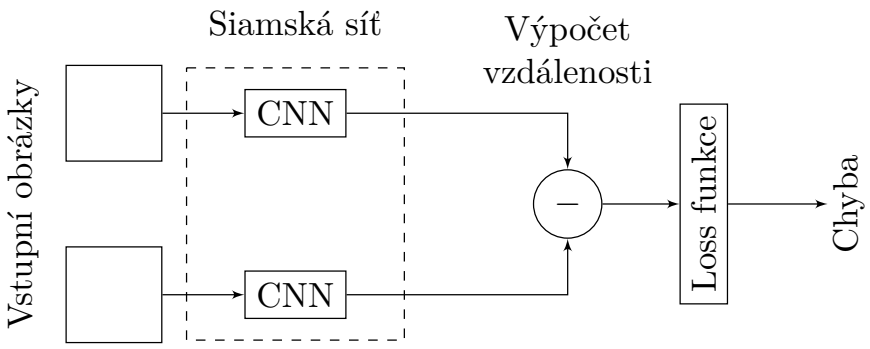
Trénování. Samotné trénování může být realizováno pomocí klasifikátoru 1 z K, kde K je počet identit a síť se snaží klasifikovat o jakou identitu se jedná. Poslední vrstva je softmax vrstva a pro trénování se používá cross-entropy loss. Takto je síť natrénována a pro získání příznakového vektoru se vezmou například aktivace z předposlední plně propojené vrstvy. Tento postup je využit v [32, 26].

Dalším způsobem trénování je pomocí siamských sítí (ukázka na obrázku 3.2). Tyto sítě mají mezi sebou sdílené váhy, tudíž se trénují společně. Pokud se použijí 2 siamské sítě, které počítají ze vstupních obrázků příznakové vektory, tak se při trénování cílí toho, aby vektory vypočtené z tváří stejných identit byly co nejvíce podobné a naopak vektory vypočtené z tváří patřící rozdílným identitám byly co nejvíce odlišné. Toho může být docíleno například pomocí *contrastive loss*, což je chybová funkce použitá například v [34]. Tato funkce je definována jako

$$E = \frac{1}{P} \sum_{p=1}^{P} y_{i,j_p} ||r_{1_p} - r_{2_p}||_2^2 + (1 - y_{i,j_p}) ||r_{1_p} - r_{2_p}||_2^2, \tag{3.1}$$

kde P je celkový počet vstupů, r_{1_p} a r_{2_p} jsou výstupní vektory reprezentující vstup, $y_{i,j_p} \in \{0, 1\}$ reprezentuje jestli jsou vstupy stejné třídy či nikoliv (1 značí, že jsou stejné, 0 rozdílné).

Jedním z dalších způsobů je trénování pomocí 3 siamských sítí. Zde jsou vstupem 3 obrázky. 2 rozdílné obrázky obsahující tváře náležící té samé identitě a 1 obrázek, který obsahuje tvář jiné idenity. Pro tyto tváře jsou vypočteny příznakové vektory. Cílem je síť natrénovat tak, aby 2 vektory stejných identit měly menší vzdálenost než vzdálenost vektoru jiné identity a jednoho vektoru stejné identity (tento vektor je označován jako pivot). Tohoto může být docíleno pomocí triplet-loss, což je chybová funkce použitá v [26, 29]. Její



Obrázek 3.2: Ukázka siamské sítě

definice je

$$E = \frac{1}{K}\sum_{k=1}^{K} max(0, \alpha - ||x_{a,k} - x_{n,k}||_2^2 + ||x_{a,k} - x_{p,k}||_2^2), \tag{3.2}$$

kde K je počet trojic, $\alpha \geq 0$ je fixní učící vzdálenost, $x_{a,k}$ je pivotující příznakový vektor, $x_{n,k}$ je vektor jiné identity než pivotující vektor a $x_{p,k}$ je vektor stejné identity jako identita pivotujícího vektor.

3.3 Agregace příznakových vektorů

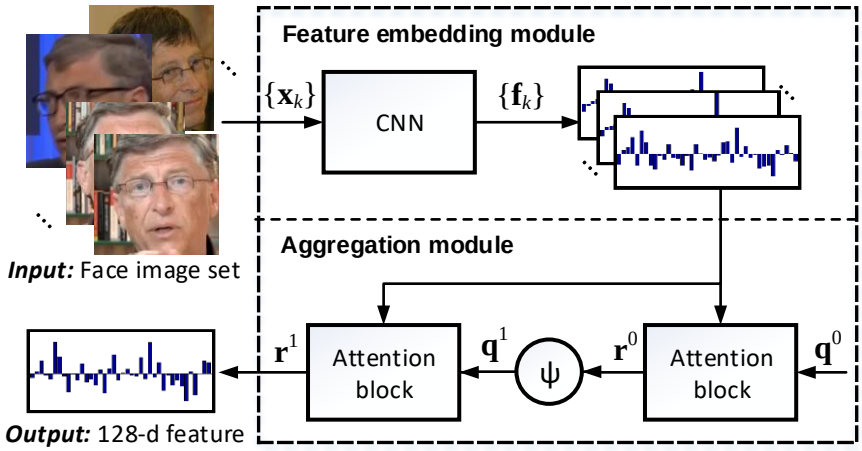
Při rozpoznávání ve videu nemusíme pracovat pouze se samostatnými obrázky, ale můžeme využívat sekvence obrázků. Tento přístup je žádoucí, protože ze sekvence obrázků jsme schopni získat více informací než pouze ze samostatných obrázků. Při rozpoznávání obličejů ve videích je možné vzít několik snímků zároveň a vypočítat z nich příznakové vektory. Poté tyto příznakové vektory agregovat v jeden příznakový vektor, se kterým se poté řeší jaké identitě náleží, nebo jestli náleží stejné identitě jako jiný příznakový vektor. Dále budou popsány metody, pomocí kterých se tato agregace provádí.

Pooling. Pooling je jeden z nejjednodušších způsobů, kterým lze agregovat více příznakových vektorů v jeden. Tento způsob je také použit v [34]. Standardně se vybírají maximální, minimální nebo průměrné hodnoty na stejných indexech napříč příznakovými vektory, které jsou agregovány. Tento přístup má výhody v tom, že je jednoduchý a výpočetně nenáročný.

Agregační neuronová síť. Agregační neuronová síť je další možnost, kterou lze využít pro agregaci příznakových vektorů. Obecně se může jednat o jakoukoliv neuronovou síť, jejíž vstupem je nějaký počet vektorů a výstupem je právě jeden.

Příkladem takové sítě může být síť z článku [34]. V tomto článku navrhnuli architekturu (znázorněno na obrázku 3.3), která je schopná ohodnotit jednotlivé vektory váhami a s jejich využitím vypočítat vážený součet jednotlivých vektorů. Tento postup vedl k lepším výsledkům, než byly výše popsané pooling metody.

Jak je vidět na obrázku 3.3 agregační modul provádí agregaci. Je složen z 2 modulů, jejichž účelem je nejdříve přidělení vah jednotlivým vektorům a poté pomocí těchto vah vypočtení váhového součtu a tím získání nového vektoru. Do tohoto modulu vedou 2 vstupy, jedním jsou vektory, které se agregují a druhým vstupem je vektor o stejné délce jako vstupní



Obrázek 3.3: Architektura agregační neuronové sítě, převzato z [34]

vektory. Váhy se počítají ve dvou fázích, nejprve se vypočte skóre vektoru pomocí skalárního součin jednotlivých vektorů, které se mají agregovat s druhým vstupním vektorem tak, že

$$s_k = q_0^T f_k, \tag{3.3}$$

kde q je vstupní vektor a f_k je příznakový vektor s indexem k. Dále je potřeba normalizovat toto skóre tak, aby součet přes všechny indexy byl 1. Toto je docíleno výpočtem

$$a_k = \frac{e^{s_k}}{\sum_{n=1}^{N} e^{s_n}}, \tag{3.4}$$

kde s_k je skóre z předchozí rovnice a N je velikost vstupních vektorů. Dále je pak pomocí váženého součtu vypočten vektor nový následovně

$$r_0 = \sum_{n=1}^{N} a_k f_k. \tag{3.5}$$

Tento vektor r_0 je poté vstupem do jedné plně propojené vrstvy, která z tohoto vektoru tvoří nový vektor q_1, který je vstupem do dalšího modulu pro výpočet finálního vektoru r_1.

Další možností je pro agregaci využít rekurentních sítí. Tohoto přístupu bylo využito v [20]. V tomto článku se řešil problém re-identifikace osob, ale stejná architektura může být využita i v případě rozpoznávání tváří. Jednotlivé obrázky vstupují do sítě, která extrahuje příznakové vektory. Tyto vektory jsou poté vstupem do rekurentní sítě, která napříč časem počítá průběžné příznakové vektory. Tyto průběžné vektory jsou poté agregovány pomocí max nebo avg poolingu do jednoho. Více informací lze nalézt v [20].

Kapitola 4

Dostupné datasety a příprava dat

Tato kapitola se věnuje veřejně dostupným datasetům, které lze využít pro rozpoznávání obličejů, a jejich základnímu popisu. Tyto datasety jsou jak statické obrázkové datasety, což znamená, že obsahují pouze obrázky vyfocené v různých situacích a různém čase, které na sebe nenavazují, tak i video datasety, které obsahují sekvence obrázků, které na sebe navazují. Taktéž je zde popsáno jakým způsobem dochází k přípravě dat pro jejich další použití.

4.1 Obrázkové datasety

FaceScrub dataset. Tento dataset [25] obsahuje 106 863 obrázků celkem 530 celebrit. Konkrétně 55 306 obrázků 265 mužů a 51 557 obrázků 265 žen. Obrázky byly automatizovaně stažené z internetu a za pomocí detektoru obličeje byly vyselektovány ty, které obsahovaly obličeje. Poté byly vyčištěny od těch, které byly špatně oanotované (tzn. nepatřily správné osobě). Součástí datasetu jsou souřadnice tzv. *bounding boxu*, tedy definice menšího výřezu, ve kterém se obličej vyskytuje. Tyto informace jsou důležité, protože se na obrázcích může vyskytovat více osob.

CACD2000 dataset. *Cross-Age Celebrity Dataset* [3] obsahuje 163 446 obrázků celkem 2 000 osob. Tento dataset se může využít pro rozpoznání pohlaví, odhad věku a rozpoznání obličeje. Dataset obsahuje informace o každé osobě jako je věk, jméno osoby, jestli se osoba nachází v LFW datasetu [10] a souřadnice 16 význačných bodů na obličeji (oči, nos, pusa, atp.) a další. Tento dataset nebyl kompletně celý manuálně kontrolován (pouze jeho část), proto obsahuje značné množství chyb a špatně detekovaných osob. Z tohoto důvodu nebyl tento dataset použit pro trénování sítě.

LFW dataset. *Labelled Faces in the Wild* dataset [10] obsahuje 13 233 obrázků celkem 5 749 osob. Fotografie osob jsou zachyceny v nekontrolovaných situacích, proto je tento dataset rozmanitý co se týče natočení hlavy, osvětlení, atd. Tento dataset se využívá pro úlohu rozpoznávání obličejů a dá se považovat za standardní dataset, na kterém se porovnávají dosažené výsledky. Vyhodnocování probíhá pomocí několika protokolů, v této práci je využit protokol *unrestricted with labeled outside data*. Samotné vyhodnocování probíhá pomocí *10-fold cross validation*, které je popsáno v sekci 6.5. Součástí datasetu nejsou anotace pro

Obrázek 4.1: Ukázka obrázků z *LFW* datasetu [10]

jednotlivé obrázky, proto jsem v této práci využil externích anotací[1]. Na obrázku 4.1 je ukázka z datasetu LFW.

CASIA WebFace Dataset. Dataset [35] obsahuje 494 414 obrázků celkem 10 575 osob. Tento dataset je jeden z největších veřejně dostupných datasetů. Stejně jako v *CACD2000* se zde nachází několik špatně označených obrázků. Díky panu Yongovi a jeho kolegům[2], kteří manuálně vyselektovali špatně označené obrázky, je možné získat korektnější dataset.

PubFig. *Public Figures* dataset [15] obsahuje 58 797 obrázků celkem 200 osob. Fotografie osob byly zachyceny v podobných podmínkách jako u *LFW* datasetu [10].

4.2 Video datasety

YouTube Faces dataset. Tento video dataset [33] obsahuje 3 425 videí celkem 1 595 osob. Tyto videa jsou navzorkované do celkem 6 070 framů (obrázků). Videa jsou, jak vyplývá z pojmenování datasetu, stažená z YouTube, což znamená, že videa mají podobný formát jako fotky v *LFW* a *PubFig* datasetu. Tento dataset se standardně využívá pro porovnávání úspěšnosti rozpoznání obličejů ve videu. Ukázku z datasetu lze vidět na obrázku 4.2.

IJB-A dataset. *IARPA Janus Benchmark A* dataset obsahuje jak obrázky, tak videa. Celkem obsahuje 5 397 obrázků a 2 042 videí navzorkovaných do 20 412 snímků patřící 500 osobám. Stejně jako *YTF* dataset může být využit pro porovnání úspěšnosti rozpoznání obličejů ve videu.

4.3 Příprava dat pro trénování

Za účelem trénování na co největším počtu dat jsem sloučil dohromady datasety *PubFig*, *FaceScrub* a *CASIA WebFace* a interně ho pojmenoval jako *PFC*. Pokud se vyskytovaly stejné osoby v různých datasetech, tyto osoby mají stejné identifikační číslo (tzv. *label*).

[1] http://www.vision.ee.ethz.ch/~mdantone/datasets/lfw_ffd_ann.txt
[2] https://groups.google.com/forum/#!topic/cmu-openface/Xue_D4_mxDQ

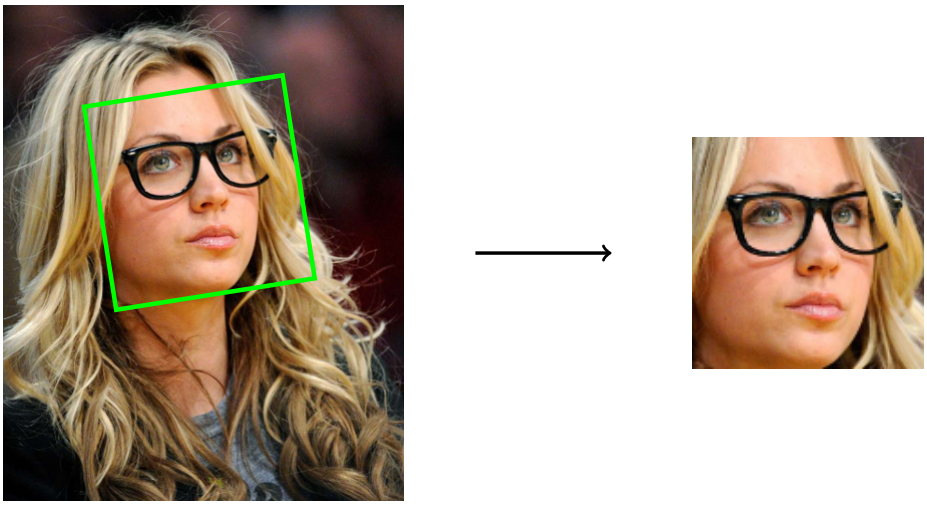
Obrázek 4.2: Ukázka z datasetu YTF

Osoby, které se nacházejí v *LFW* datasetu, byly odstraněny, aby výsledky nebyly nijak ovlivněny tím, že by se síť trénovala na identitách, na kterých by se poté vyhodnocovala. Odstraněny byly také osoby, které měly méně jak 15 různých obrázků (toto bylo provedeno z důvodu experimentů s agregací více obrázků viz 6.4). Sloučený dataset obsahuje 317 328 obrázků celkem 6 154 osob.

Firma Innovatrics mi poskytla anotace k obrázkovým datasetům (kromě *LFW*) a natrénovanou síť v Caffe, jejímž vstupem jsou obrázky s rozměry 128×128 px a oči jsou zarovnány na pozice (44,44) px – levé oko a (84,44) px – pravé oko. Z tohoto důvodu bylo vhodné připravit data stejným způsobem. Na obrázku 4.3 je ukázán princip zarovnání obličeje. Toto zpracování bylo provedeno za pomoci OpenCV knihovny (konkrétně knihovna cv2 v pythonu)[3]. Data byla následně náhodně promíchána po identitách a rozdělena na trénovací a testovací sadu. Trénovací sada obsahuje 260 000 obrázků a testovací sada 57 323.

Stejným způsobem byly zpracovány *YouTube Faces* dataset a *LFW* dataset, kde u *YTF* kromě informace jaké identitě náleží bylo nutné uchovávat i identifikační číslo videa, do kterého snímek patřil. Navíc zde byla data rozdělena do 10 částí (tzv. *foldů*), jelikož jak bylo zmíněno, tyto datasety se standardně vyhodnocují pomocí *10-fold cross validation*.

[3] http://opencv.org/



Obrázek 4.3: Ukázka zarovnání a oříznutí obrázku. Na obrázku se nachází herečka Kaley Cuoco z datasetu FaceSrub [25]

Kapitola 5

Návrh řešení a implementace

Jak je výše popsáno, princip všech metod pro rozpoznávání tváří je podobný. Je potřeba mít připravená data pro trénování, která jsou nějakým vhodným způsobem zarovnaná. Jak je popsáno v kapitole 4, obrázky z datasetu jsou zarovnány na fixní pozice očí pomocí 2D zarovnání.

Dále je vždy potřeba natrénovat síť, která extrahuje příznakové vektory, reprezentující identity z obrázků obsahující tváře. Architektury a způsoby, jakými lze tyto sítě trénovat byly taktéž uvedeny výše. Trénovat sítě je nutné tak, aby vektory byly pro stejné identity co nejméně vzdálené a pro rozdílné co nejvíce vzdálené. Bez natrénované sítě, která produkuje dobré příznakové vektory (může být ověřeno například na datasetu *LFW*), není možné pokračovat. Agregační síť by nedostávala vektory takové, které očekává a nemohla by se správně naučit vektory agregovat.

Jakmile má síť dobré výsledky extrahování vektorů, může se začít řešit agregační síť. Tato síť by měla produkovat z více vektorů jeden vektor takový, který opět splňuje podmínky uvedené výše. Agregace může probíhat ať už napříč obrázky nebo napříč videi.

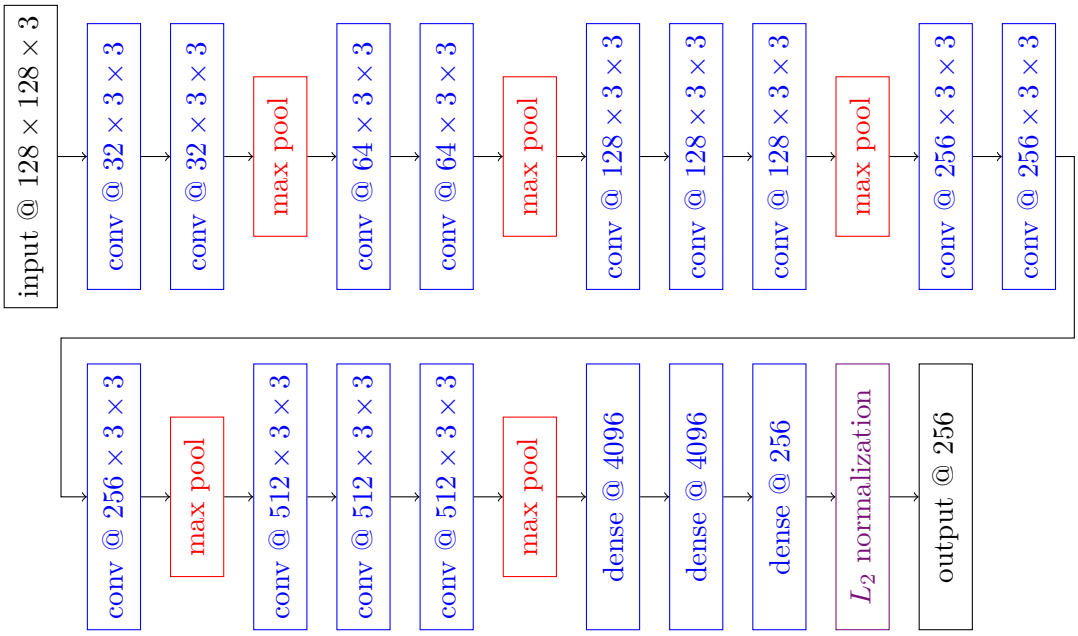
Toto je obecný postup, kterým jsem taktéž postupoval. V této kapitole jsou přesně popsané kroky, které jsem učinil pro řešení problému. Zároveň jsou na konci kapitoly implementační detaily, v jakém frameworku a na jakých strojích byly sítě trénovány.

5.1 Sítě pro extrakci příznakových vektorů

Jak již bylo řečeno, stěžejní části je natrénovat síť pro extrakci příznakových vektorů. Tato síť dostává jako vstup jednotlivé obrázky a produkuje příznakové vektory jako výstup.

Dále bylo potřeba vybrat některou z uvedených architektur sítí. Rozhodl jsem se využít jednoduchou (obsahuje pouze lineárně propojené vrstvy jako jsou konvoluční, max poolingové a plně propojené vrstvy) architekturu z článku [26], která je totožná s architekturou *VGG-16* [30]. Tato síť je velmi robustní (obsahuje 16 vrstev) a dosahuje velmi dobrých výsledků při rozpoznávání tváří (na *LFW* datasetu dosahuje přesnost 98.95% pomocí unrestricted protokolu [10]). Skutečnost, že je síť robustní a tím taktéž i pomalá, není problém, protože cílem zde nebylo natrénovat síť, která bude používána v reálném čase nebo v nějakém vestavném zařízení, které má málo dostupných prostředků.

Na obrázku 5.1 je znázorněna použitá architektura. Tato síť má velikost všech konvolučních filtrů 3×3 a konvoluce je zde *stejná* – konvoluce nemění prostorovou velikost. Po konvolučních a plně propojených vrstvách vstupují aktivace do *ReLU* aktivační vrstvy (kromě poslední plně propojené vrstvy). Lze si všimnout, že síť má menší počet kanálů



Obrázek 5.1: Architektura trénované sítě. Dense vrstvy označují plně propojené vrstvy.

než originální síť z článku [26], proto je síť dále pojmenována jako *VGG-16-small*. Max pooling vrstvy snižují prostorovou velikost 2× v obou směrech a jednotlivé lokální oblasti, ze kterých se vybírá maximální aktivace, se nepřekrývají).

Výstupní vektory jsou L_2 normalizovány, toto bylo z důvodu toho, aby vektory měly fixní velikost a tím se lépe trénovaly, protože jejich maximální vzdálenost je omezená. L_2 normalizace je definována rovnicí

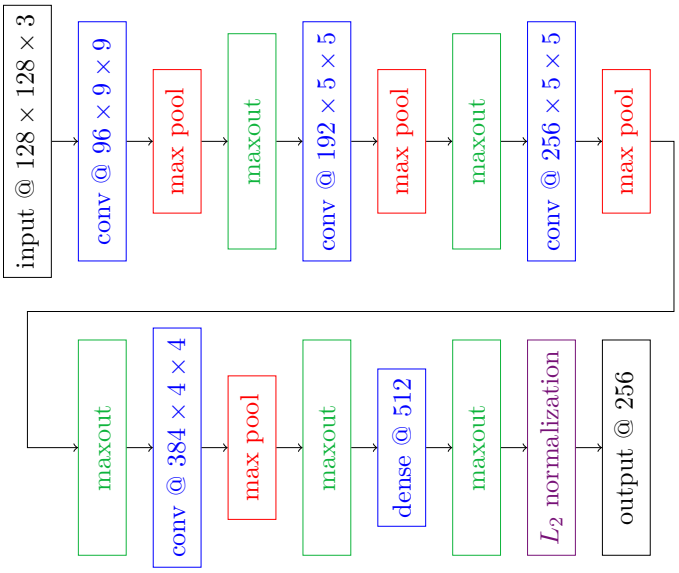
$$y = \frac{x}{||x||_2}, \tag{5.1}$$

kde x je vstupní velikost a $||x||_2$ je velikost vektoru.

Pro trénování jsem vybral chybovou funkci z článku [34] jménem *Contrastive Loss*. Z toho vyplývá, že bylo nutné využít siamské architektury (obrázek 3.2). Tudíž při trénování se vytvořily 2 sítě, které mají mezi sebou sdílené váhy. Tato siamská síť počítá euklidovu vzdálenost ze 2 produkovaných vektorů jednotlivými *CNN*. A z této vzdálenosti a informace, jestli se jedná o stejné/rozdílné identity se počítá chyba.

V průběhu experimentů, konkrétně 6.2, bylo zjištěno, že síť dosahuje lepších výsledků s batch normalizací před aktivační vrstvou. Toto je finální podoba natrénované sítě pro extrakci příznakových vektorů, která je v této práci nazvaná jako *VGG-16-small*.

Jak bylo výše řečeno, v rámci této práce mi byla poskytnuta natrénovaná síť v Caffe od firmy Innovatrics, která je dále pojmenována jako *fingera* síť. Vzhledem k tomu, že jsem svou síť trénoval v Kerasu a měl jsem pro to připravené podpůrné funkce, bylo potřeba převést tyto váhy z formátu pro Caffe do formátu pro Keras. Architektura této sítě je znázorněna na obrázku 5.2. S touto sítí jsem prováděl stejné experimenty, kromě trénování, jelikož tato síť již byla natrénovaná.



Obrázek 5.2: Architektura poskytnuté sítě je složena z klasických konvolučních, max pooling a plně propojených vrstev. Konvoluce je zde počítána úplná (snižuje tedy prostorovou velikost). Max pooling snižuje prostorovou velikost 2× v obou směrech. Výstupní vektory jsou taktéž L_2 normalizované. Dále je zde použita maxout vrstva, která vybírá ze 2 kanálů maximální aktivace a tím dochází ke zmenšování počtu kanálů.

5.2 Agregace

Z výše uvedených metod pro agregaci jsem použil agregační síť popsanou v článku [34]. Tato síť byla podrobně popsaná v sekci 5.2. Tato síť je jednoduchá, není výpočetně náročná. Zároveň další výhodou je, že se může natrénovat na jakémkoliv počtu obličejů a pak být využita pro agregaci jiného počtu obličejů. Základním principem sítě je naučit se ohodnotit vektory určitým skórem a váženým součtem produkovat nový vektor.

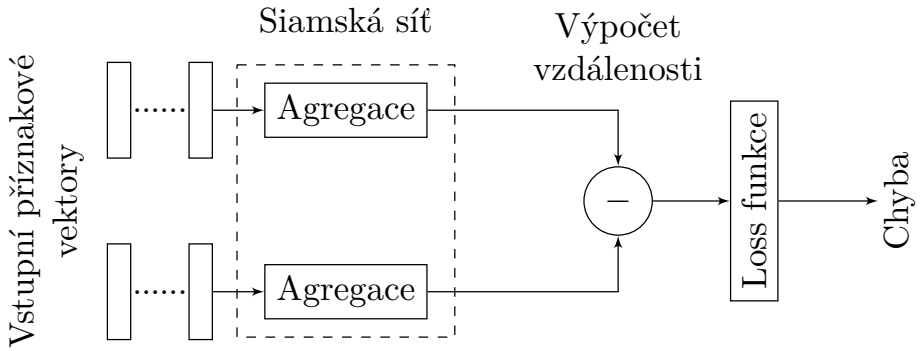
Tato síť byla trénována pouze z předpočítaných příznakových vektorů, čímž se urychlilo trénování sítě. Síť byla trénována taktéž pomocí *Contrastive Loss*. Tudíž zde byly opět 2 siamské sítě. Vstupem jim byly 2 n-tice příznakových vektorů a výstupem byla opět vzdálenost příznakových vektorů. Tato architektura je znázorněna na obrázku 5.3.

Stejně jako v originálním článku byla tato agregace porovnávána s agregací pomocí max poolingu a avg poolingu. Jelikož jednotlivé hodnoty z příznakových vektorů se pohybují v intervalu $\langle -1, 1 \rangle$, je v experimentech vyzkoušen i max-absolute pooling, což znamená, že se vybírají hodnoty, které mají maximální absolutní velikost.

Dalším alternativním způsobem může být vypočtení všech kombinací dvojic příznakových vektorů, kde každý příznakový vektor je z jiné n-tice. Z těchto dvojic vypočítat vzdálenosti a z těchto vzdáleností vybrat minimální vzdálenost a průměrnou vzdálenost, což je pojmenováno jako *min* l_2 a *avg* l_2. Tyto vzdálenosti určují výsledné vzdálenosti těchto 2 n-tic.

5.3 Generování trénovacích dat

Jak bylo v kapitole 4 popsáno, spojil jsem obrázkové datasety s obličeji. Tento spojený *PFC* dataset byl použit pro trénování sítě, která extrahuje příznakové vektory.



Obrázek 5.3: Ukázka siamské architektury pro trénování agregační sítě.

Za účelem vytěžení co největšího potenciálu z tohoto datasetu, docházelo při generování dat pro trénování k augmentaci těchto dat následujícími způsoby. Obrázky s obličeji byly náhodně horizontálně překlápěny, dále byl u obrázků náhodně mírně upravován jas a kontrast. Vzhledem k tomu, že detektor očí nebyl naprosto přesný, experimentálně jsem odhadl, že při posuvu do 3px ve obou směrech jsou oči zarovnané s podobnou odchylkou jako bez posuvů, tudíž dalším způsobem augmentace byly zvoleny tyto posuvy. Toto generování dat teoreticky zvýšilo několikanásobně celkový počet trénovacích dat. Při generování je dbáno, aby poměr stejných a rozdílných tváří byl 50 : 50.

Dále bylo potřeba vytvořit generátor dat, který generuje dvojice n-tic pro trénování agregační sítě. Stejně jako v článku [34] jsem se rozhodl trénovat jednotlivé sítě separátně. Pomocí natrénované sítě jsem vypočítal příznakové vektory a tyto vektory uložil. Při trénování agregační sítě se tyto vektory jednoduše načetli místo toho, aby byly tyto vektory byly počítány znovu z obrázků obličejů. Díky tomuto přístupu bylo trénování rychlejší.

5.4 Implementace

Samotná implementace probíhala na počítačích MetaCentra[1], což je organizace provozovaná v České Republice, poskytující studentům a akademickým pracovníkům možnost zdarma využívat výpočetní a úložné kapacity. V případě této práce byly hlavním cílem využití stroje, které obsahovaly grafické karty a které měly podporu CUDA a knihovny cuDNN[2], která se používá pro akceleraci výpočtů na grafických kartách pro hluboké neuronové sítě. Tyto podmínky splňovaly clustery *Doom* a *Zubat* obsahující grafiky nVidia Tesla K20.

Pro trénování bylo využito frameworku Keras [4], což je vysokoúrovňové API pro neuronové sítě. Keras je implementovaný v pythonu (podpora 2.7-3.5) a jako backend používá buď TensorFlow[3] nebo Theano[4]. Já jsem využil Keras ve verzi 1.1.1, spolu s TensorFlow ve verzi 0.10.0, jelikož MetaCentrum obsahovalo moduly s těmito verzemi.

Jak bylo výše zmíněno, bylo využito jazyka *python* pro podpůrnou funkcionalitu. Pro registrování úloh v MetaCentru a režii experimentů byl využit jazyk *bash*.

[1] https://metavo.metacentrum.cz/
[2] https://developer.nvidia.com/cudnn
[3] https://www.tensorflow.org/
[4] http://deeplearning.net/software/theano/index.html

Kapitola 6

Experimenty a výsledky

V této kapitole jsou popsány experimenty se sítěmi, které byly trénovány. Nejprve jsou trénovány sítě pro extrakci příznakových vektorů. Tyto sítě vycházejí z architektury uvedené v předchozí kapitole (viz obrázek 5.1). Mění se zde počet vrstev, velikosti kanálů a filtrů a výsledná velikost příznakového vektoru a další detaily, které jsou uvedeny u jednotlivých experimentů.

Dále, kromě zajímavých zjištění v průběhu trénování, zde jsou vyhodnoceny nejlepší natrénované sítě a firmou poskytnutá síť. Vyhodnocení probíhá na datasetu *LFW* [10], zde se jedná o vyhodnocení sítí extrahující příznakové vektory, tedy vyhodnocení pouze na obrázcích.

Pomocí natrénované sítě, která extrahuje příznakové vektory jsou předpočítány vektory z *YTF* datasetu, což je video dataset. Tyto vektory jsou poté použity pro trénování agregační sítě a zároveň je na tomto datasetu tato agregační síť vyhodnocena spolu s dalšími metodami.

6.1 Sítě bez batch normalizace

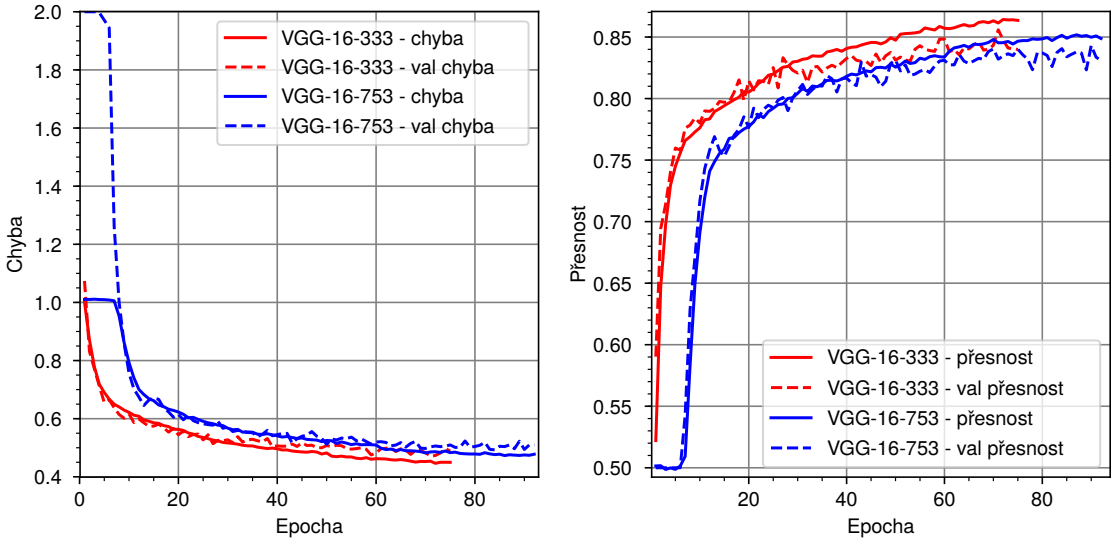
V tomto experimentu jsou porovnávány 2 sítě založené na architektuře definované v kapitole 5. Účelem je zjistit jakou přesnost dosáhnou dané architektury.

Následující 2 sítě byly trénovány cca 80 epoch. Každá epocha obsahovala cca 3 000 iterací, přičemž každá epocha trvala okolo 50 minut na natrénování. Trénování probíhalo s *learning rate* = 0.0008 (po 40. epoše se LR snížila na 0.0004) a pomocí *RMSprop* optimalizátoru [8]. Trénováno bylo pomocí siamských sítí a *contrastive loss*.

VGG-16-333. Tato síť má architekturu stejnou jako na obrázku 5.1. Jediný rozdíl je v tom, že velikost výstupního vektoru je 1024. Mezi plně propojenými vrstvami je dropout s pravěpodobností 0.4

VGG-16-753. Síť má stejnou architekturu jako síť *VGG-16-333*, až na to, že velikosti konvolučních filtrů v jednotlivých vrstvách jsou postupně 7-7-5-5-5-5-3-3-3-3-3-3. Velikost výstupního vektoru je taktéž 1024.

Jak lze z grafů vidět, výsledky na validační sadě se ustálily a už se více nezlepšují. Pokud by se trénovaly sítě dál, docházelo by pouze k zlepšování na trénovací sadě, neboli k přetrénovávání, což je nežádoucí. Pokud by trénovací sada obsahovala více dat, je možné,



Obrázek 6.1: Porovnání výsledků sítí bez batch normalizace na *PFC* datasetu. Vlevo je znázorněna chyba v průběhu epoch a vpravo přesnost v průběhu epoch.

že by s touto architekturou bylo dosáhnuto lepších výsledků. Z těchto výsledků ale vyplývá, že s touto architekturou a tímto způsobem trénování nelze dosáhnout lepších výsledků.

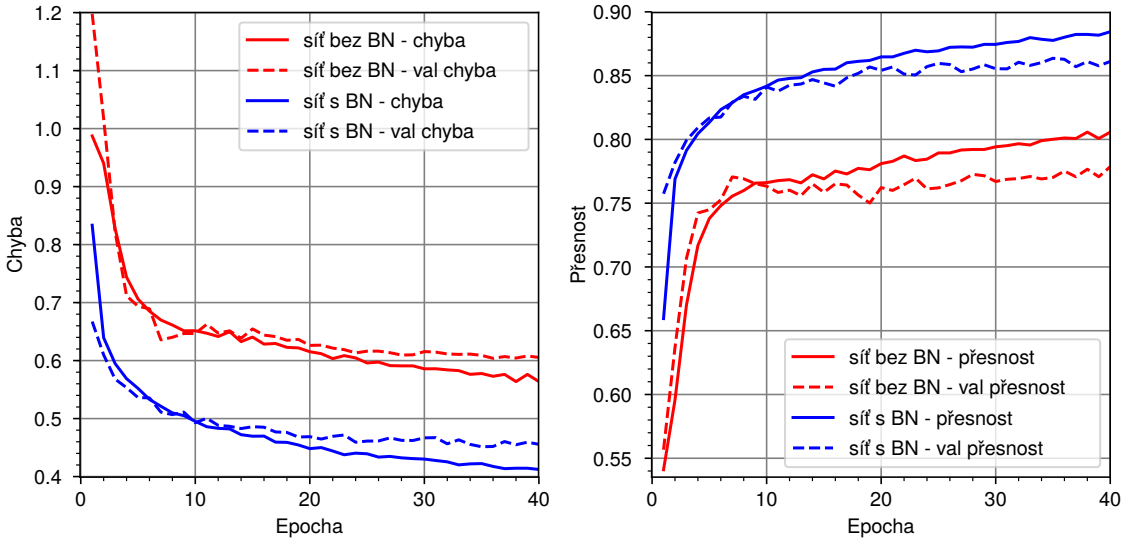
6.2 Vliv batch normalizace

V této sekci jsou popsány experimenty s batch normalizací. Batch normalizace byla popsána výše a její aplikací mezi konvoluční nebo plně propojenou vrstvu a aktivační vrstvu by mělo vést k rychlejšímu trénování a taktéž k jisté regularizaci, pokud nebude batch size veliká. Vzhledem k tomu, že architektury založené na *VGG* sítích obsahují velké množství aktivací, vyskytují se praktické problémy s pamětí. Tyto aktivace musejí být uložené v paměti GPU. Tudíž čím větší je batch size, tím větší jsou nároky na paměť. Z tohoto důvodu byla batch size použita okolo 30 (pokud byly velikosti kanálů dvakrát menší oproti *VGG-16*), nebo 20 (pokud by byly vrstvy naprosto totožné).

Architektury 2 sítí z obrázku 6.2 jsou naprosto totožné, s výjimkou, že *Síť s BN* obsahuje mezi konvoluční a aktivační vrstvou batch normalizaci. Sítě obsahovaly vždy pouze 1 konvoluční vrstvu následovanou max poolingem a tímto způsobem opakované 5×. Počet kanálů se vždy oproti předchozí konvoluční vrstvě zdvojnásobil, přičemž se začínalo na počtu 32 a končilo na 512. Velikost výstupního vektoru byla 1024. *Síť bez BN* navíc používala vrstvy předtrénované sítě *VGG-16* na ImageNetu [14].

Síť bez BN byla trénována s LR 0.0008 (s vyšší LR docházelo stále k divergenci) a jedna epocha, během které došlo k 2 000 iteracím, byla natrénována za 820 sekund. *Síť s BN* byla trénována s LR 0.001 a doba trénování jedné epochy se zvýšila na 1 062 sekund. Odtud dále byl jako optimalizátor používán *Adam* [13].

Na obrázku 6.2 je vidět znatelný rozdíl, který je důsledkem aplikované batch normalizace. *Síť s BN* se nejenom trénovala rychleji, ale s přesností 87% na validační sadě dosáhla lepších výsledků, než jakákoliv z předchozích testovaných sítí. Tato síť (ani s více parametry) po dalších trénovacích epochách nepřevršila přesnost na validační sadě 88%.



Obrázek 6.2: Porovnání sítí s batch normalizací a bez batch normalizace na *PFC* datasetu. Vlevo je znázorněna chyba v průběhu epoch a vpravo přesnost v průběhu epoch.

Proto byla přidána batch normalizace i mezi plně propojené vrstvy a aktivační vrstvy. Díky tomu přesnost na validační sadě dovršila 92% (viz obrázek 6.3).

Síť má konvoluční vrstvy stejné jako síť *VGG-16-333*, ale výstupní příznakový vektor má velikost 256. Síť byla trénována s počáteční LR 0.001 a pokud se 7 epoch nesnížila validační chyba o více jak 0.0005, došlo k 3× snížení LR.

Tato síť je zároveň nejlepší síť a je to finální síť použitá dále při agregaci a bude nazývaná *VGG-16-small*.

6.3 Vyhodnocení na PFC

V rámci tvorby *PFC* datasetu jsem vytvořil testovací sadu pro vyhodnocení. Tato testovací sada obsahuje 300 000 unikátních dvojic. Poměr stejných a rozdílných identit je taktéž 50 : 50.

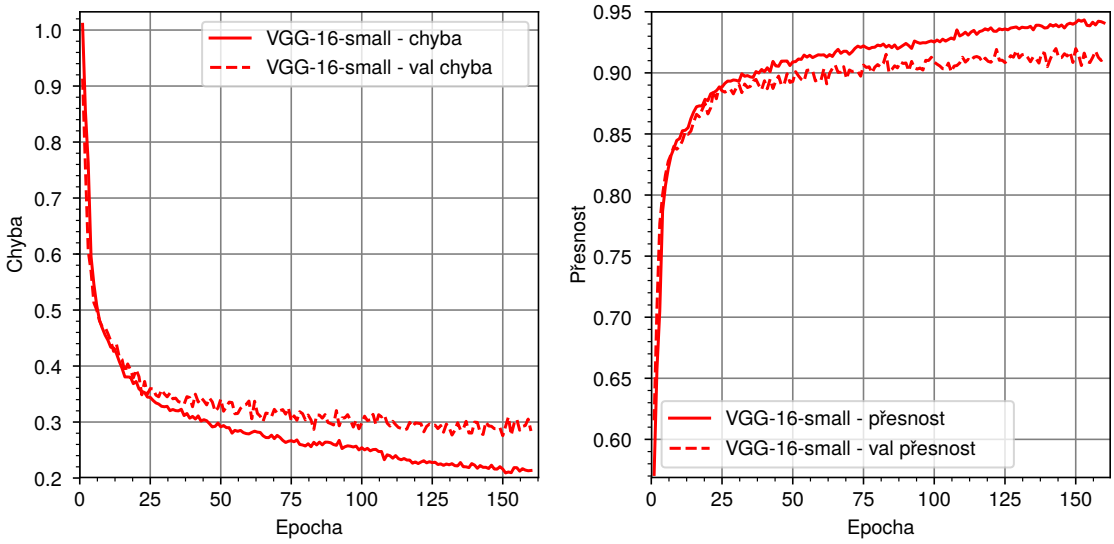
Fingera síť se pouze vyhodnocovala na testovací sadě, jelikož byla již natrénovaná. *VGG-16-small* byla trénována na *PFC* datasetu a následně vyhodnocena stejně jako *fingera* síť.

Výsledky *VGG-16-small* sítě a *fingera* sítě jsou prezentovány na obrázku 6.4. *VGG-16-small* síť má přesnost $1 - EER = 92.07\%$ a plochu pod křivkou $AUC = 0.9747$ a *fingera* síť má přesnost $1 - EER = 95.47\%$ a plochu pod křivkou $AUC = 0.9879$.

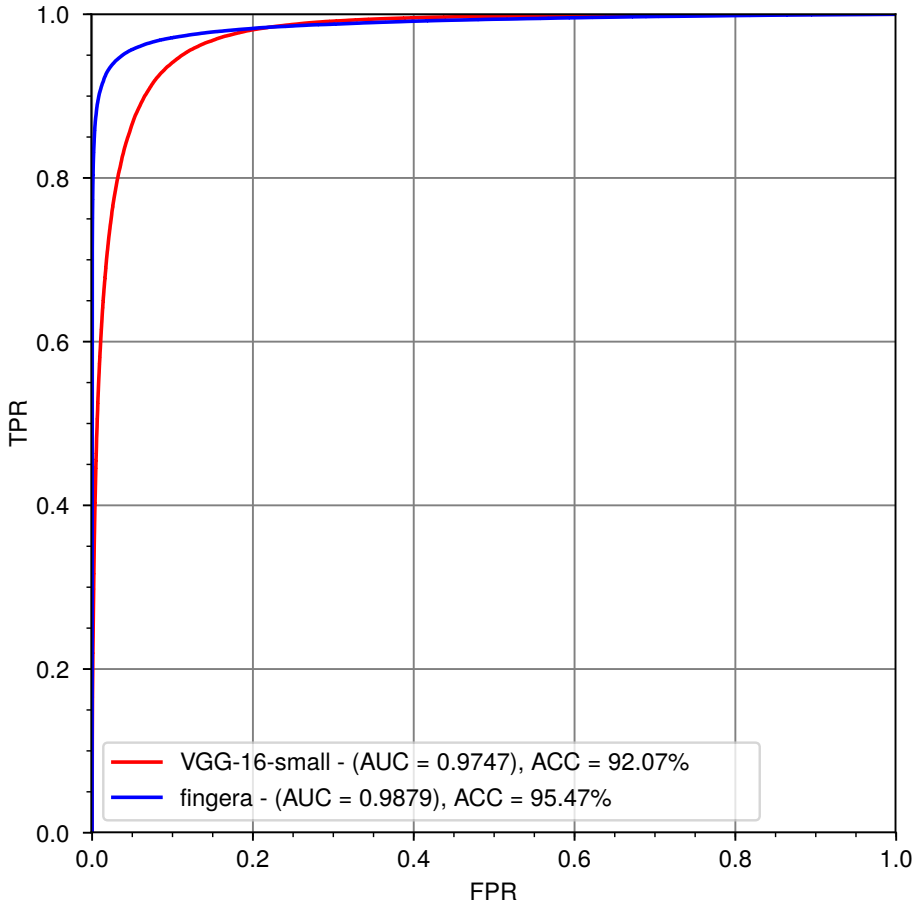
6.4 Agregace příznakových vektorů

Při agregování jednotlivých vektorů je zajímavou otázkou, jak moc velký vliv má počet agregovaných příznakových vektorů na výsledky. Logicky by měl větší počet znamenat větší množství informací a tím i vyšší přesnost. Experimenty byly provedeny na datasetu *PFC* a na *IJB* datasetu, ze kterých jsem použil pouze videa.

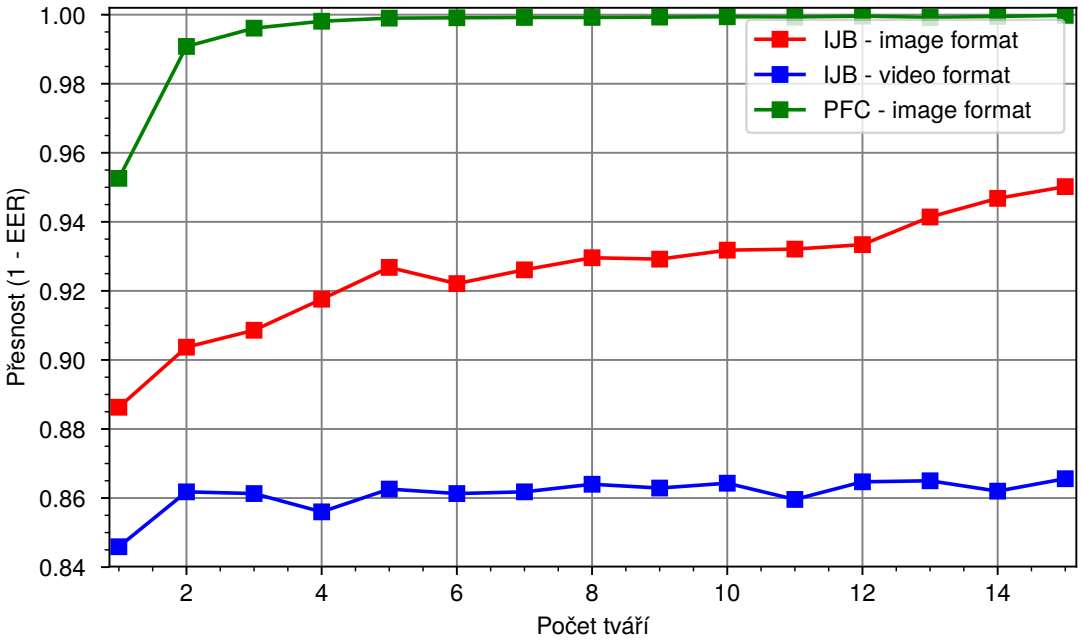
Agregace byla prováděna pomocí agregační sítě zobrazené na obrázku 5.3 a byla prováděna pouze na *fingera* síti. Trénovací a validační data byla generována jedním ze dvou



Obrázek 6.3: Průběh trénování sítě s batch normalizací i u plně propojených vrstev. Vlevo je znázorněna chyba v průběhu epoch a vpravo přesnost v průběhu epoch.



Obrázek 6.4: Porovnání výsledků na testovací sadě *PFC* datasetu pomocí ROC křivek.



Obrázek 6.5: Graf popisující vliv počtu obličejů na výslednou přesnost na validačním setu. Image format říká, že se obrázky generovaly pouze v závislosti na identitě. Video format říká, že se obrázky generovaly pouze z dané sekvence. Zde byla použita *fingera* síť.

způsobů. *Image format* znamená, že se s daty zacházelo jako s obrázky, vybraly se tedy náhodně příznakové vektory 2 různých lidí, nebo 2 příznakové vektory rozdílných obrázků stejného člověka. Druhým způsobem je *Video format*, což znamená, že se s daty zacházelo jako s videem. Tudíž se nevybíraly příznakové vektory náhodně podle identity, ale bylo nutné, aby daná n-tice byla z jednoho videa.

Jak lze z obrázku 6.5 vidět, zásadní rozdíl je u obou způsobů generování mezi použitím 1 a 2 příznakových vektorů. Což může být z toho důvodu, že 1 z nich mohl být špatně zarovnán, nebo ve špatné kvalitě, proto velmi roste přesnost s použitím 2 příznakových vektorů.

Dále se více mění přesnost v případě, kdy se data generovaly jako *Image format*. Toto je logické, protože pokud jsou na sobě obrázky časově nezávislé je větší pravděpodobnost, že budou úplně z jiné situace, jiného osvětlení apod. a tím si budou jednotlivé příznakové vektory méně podobné. Kdežto u *Video formatu* se pouze otáčí hlava (jak je z ukázky vidět, tak minimálně), ale ostatní věci jsou si hodně podobné, z toho důvodu si budou hodně podobné i výsledné příznakové vektory a tím se nezíská moc nových informací.

6.5 Vyhodnocení na benchmark datasetech

Vyhodnocení výsledných sítí je provedeno na datasetu *LFW*, kde je provedeno vyhodnocení sítě pro extrakci příznakových vektorů. Sítě, které se vyhodnocují jsou již předtrénované na jiných trénovacích datech – konkrétně síť *VGG-16-small* je předtrénovaná na *PFC* datasetu. Vyhodnocení je provedeno stejně, jako byly vyhodnoceny *state of the art* metody, aby bylo možné srovnat dosažené výsledky.

Na datasetu *YTF* jsou vyhodnoceny výše uvedené metody (5.2) a agregační síť, která agreguje více příznakových vektorů v jeden. Stejně jako u *LFW* datasetu jsou dosažené výsledky porovnány s dostupnými výsledky *state of the art* metod.

Oba benchmark datasety jsou standardně vyhodnocovány pomocí *10-fold cross validation*. To znamená, že jsou datasety předem rozděleny do 10 částí a provede se 10 trénování a 10 vyhodnocování na testovací sadě. V každé fázi se spojí 9 částí v jednu a ta tvoří trénovací sadu, zbylá poslední část tvoří testovací sada. Síť se tedy vyhodnocuje na $\frac{1}{10}$ a trénuje se na $\frac{9}{10}$, toto je provedeno $10\times$, aby testovací sada byla vždy jiná.

Účelem tohoto vyhodnocení je získání relevantních statistických informací i přesto, že pro testování existuje malé množství dat.

LFW. Vyhodnocení pomocí ROC křivek je na obrázku 6.6. *Fingera* síť na tomto datasetu má přesnost $1 - EER = 97.40\%$ se standardní odchylkou 0.79%. *VGG-16-small* síť má přesnost $1 - EER = 92.80\%$ se standardní odchylkou 0.81%. Výsledky *fingera* sítě jsou srovnatelné s uspěšnějšími metodami z [10]. Což se o *VGG-16-small* síti říci nedá. Síť *VGG-16-small* je lepší jako metody, které nevužívají konvoluční neuronové sítě a je srovnatelná pouze s nejhoršími metodami využívající konvoluční neuronové sítě.

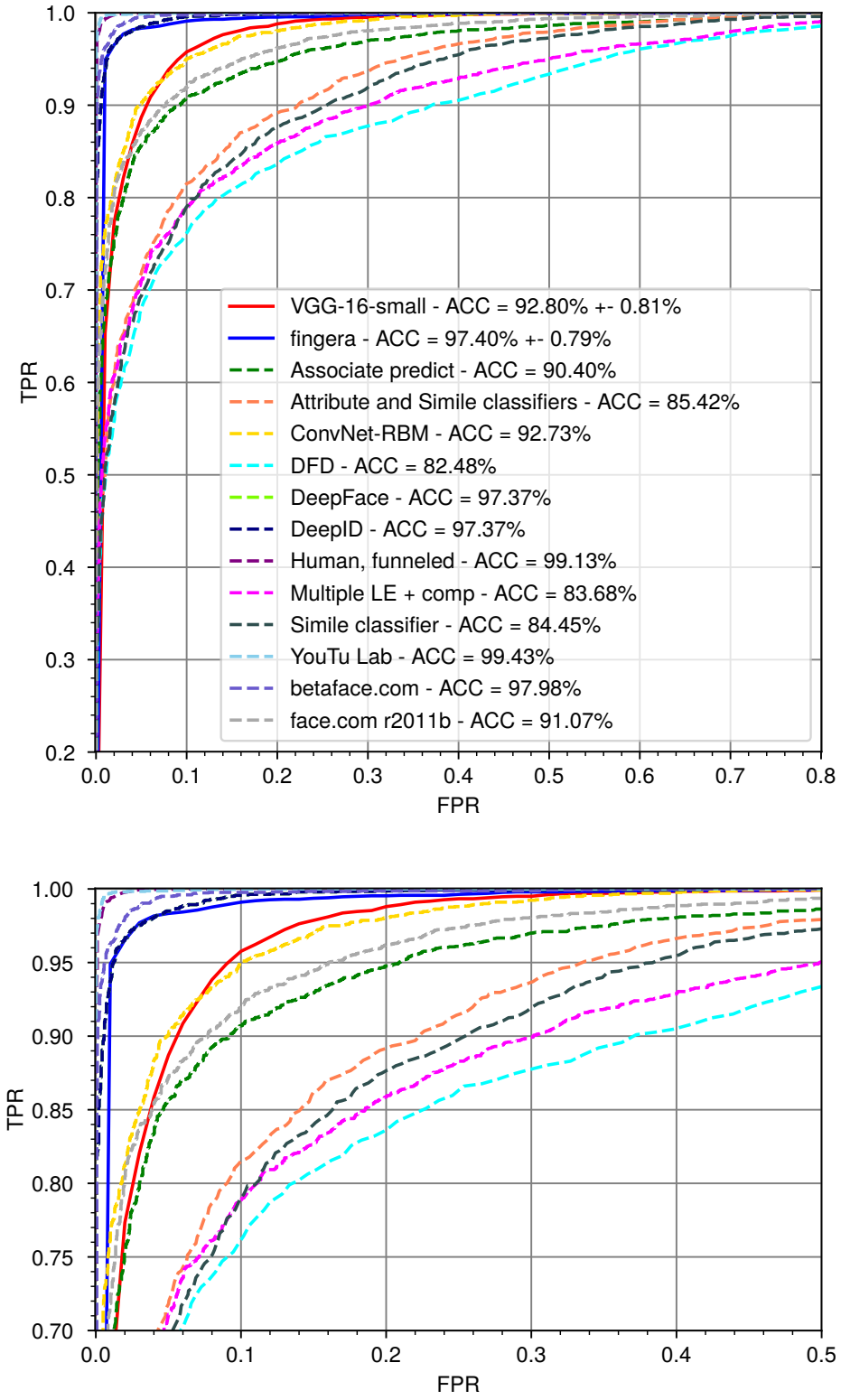
YTF. Vyhodnocení je pro přehlednost rozděleno na 2 obrázky. Jeden obrázek 6.7 obsahuje porovnání mezi mými použitými metodami, zatímco obrázek 6.8 obsahuje porovnání 4 nejlepších metod v porovnání s výsledky, které jsou publikované v [33]. Příznakové vektory byly agregovány z 10 obrázků, ale výsledky byly srovnatelné, pokud bylo agregováno ≥ 5 příznakových vektorů v 1. Přesnost při použití pouze jednoho obrázku byla s *fingera* sítí 88.44% a s *VGG-16-small* 79.26%.

Metody popsané jako *max*, *max abs*, *avg* jsou poolingové metody, které byly popsané v sekci 5.2. Dále metody *min* l_2 a *avg* l_2 počítají všechny kombinace vzdáleností a vybírají z ní minimální a průměrnou vzdálenost. Metoda *aggreg.* reprezentuje agregační síť.

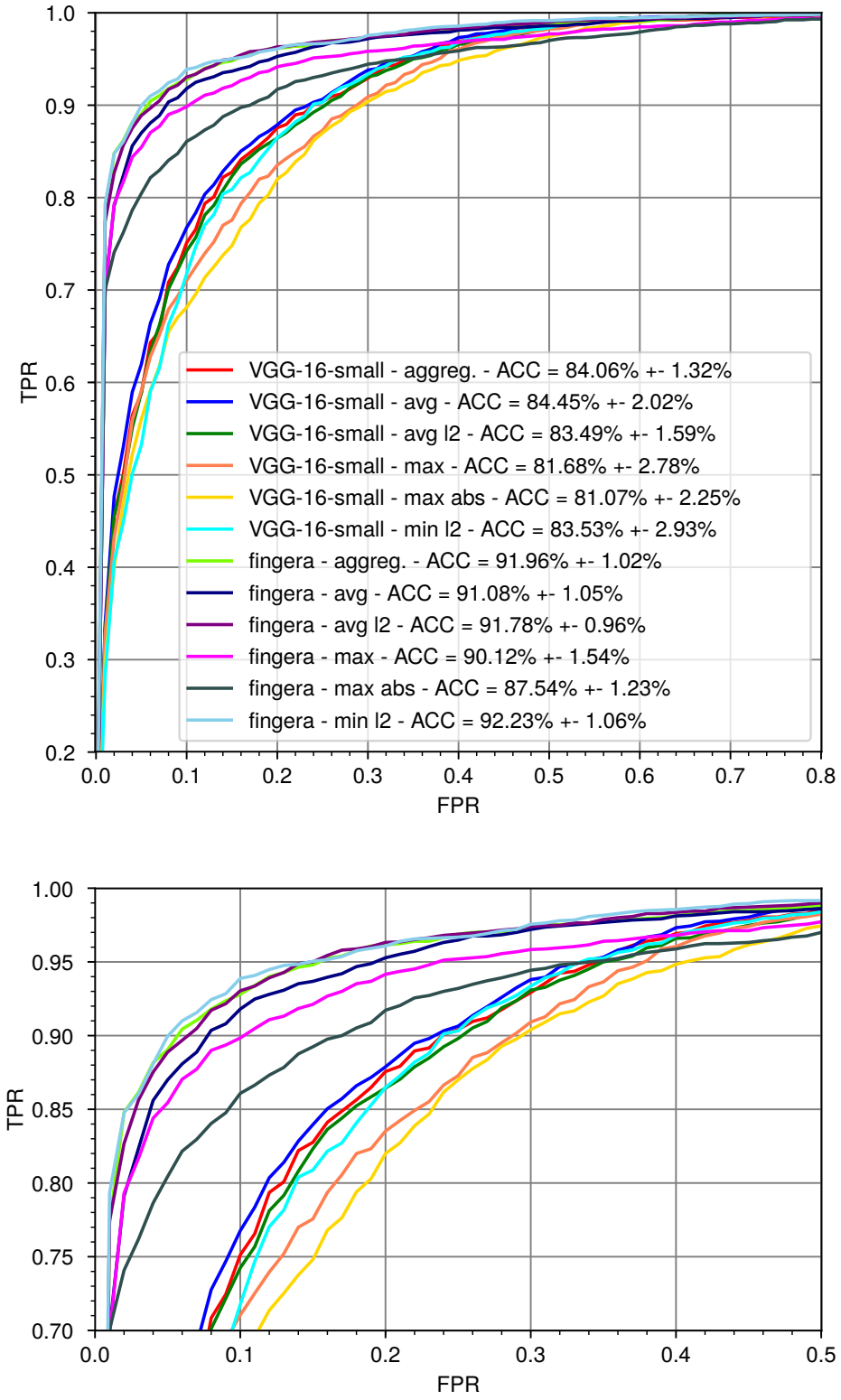
Trénování agregační sítě bylo poměrně rychlé – 10 000 iterací (500 000 příznakových vektorů s batch size 50) proběhlo za cca 90 sekund.

Na výsledcích z obrázku 6.7 nebo tabulky 6.1 je vidět, že v obou případech agregační síť není nijak výrazně lepší ani výrazně horší než *avg* l_2 a *min* l_2. Toto je pravděpodobně z toho důvodu, který byl již uveden výše, že tento dataset obsahuje obrázky, ve kterých se natočení hlavy nijak výrazně nemění a tím se nemění ani výsledné feature vektory. Z tohoto důvodu je úspěšnost agregace podobná těmto metodám.

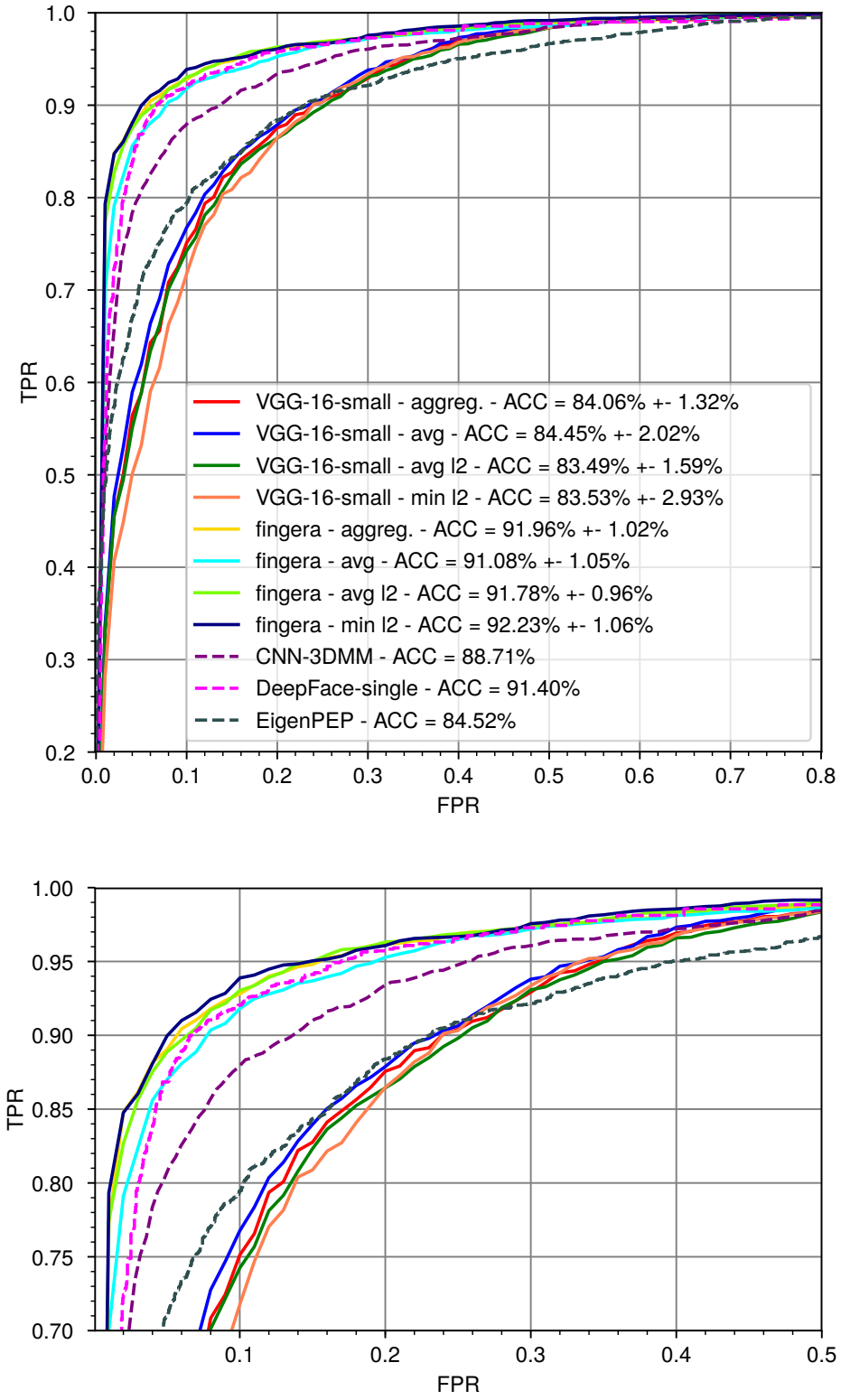
Je také vidět, že avg pooling je u *VGG-16-small* sítě nejúspěšnější metoda. Což může být z důvodu toho, že síť pro extrakci není dostatečně přesná. Kromě tohoto případu má agregační síť lepší výsledky než tyto poolingové metody.



Obrázek 6.6: Porovnání výsledků na *LFW* datasetu se state of the art metodami pomocí ROC křivek.



Obrázek 6.7: Porovnání výsledků metod agregací na YTF datasetu *fingera* sítě s *VGG-16-small* sítí. Značka *aggreg.* reprezentuje agregační síť.



Obrázek 6.8: Porovnání výsledků 4 nejlepších metod agregací na YTF datasetu se state of the art výsledky. Značka *aggreg.* reprezentuje agregační síť popsanou v 5.2.

Sítě a metody	**1 - EER (%)**	**AUC**
VGG-16-small - avg	84.45 ± 2.02	0.9243 ± 0.0154
VGG-16-small - avg l_2	83.49 ± 1.59	0.9172 ± 0.0138
VGG-16-small - max	81.68 ± 2.78	0.9081 ± 0.0159
VGG-16-small - max abs	81.07 ± 2.25	0.8992 ± 0.0162
VGG-16-small - min l_2	83.53 ± 2.93	0.9127 ± 0.0186
VGG-16-small - aggreg.	84.06 ± 1.32	0.9189 ± 0.0150
fingera - avg	91.08 ± 1.05	0.9674 ± 0.0042
fingera - avg l_2	91.78 ± 0.96	0.9729 ± 0.0044
fingera - max	90.12 ± 1.54	0.9587 ± 0.0087
fingera - max abs	87.54 ± 1.23	0.9461 ± 0.0069
fingera - min l_2	92.23 ± 1.06	0.9747 ± 0.0037
fingera - aggreg.	91.96 ± 1.02	0.9727 ± 0.0043
CNN-3DMM	88.71 ± 2.21	0.954
EigenPEP	84.52 ± 1.40	0.926
DeepFace-single	91.40 ± 1.10	0.963

Tabulka 6.1: Výsledky *fingera* sítě, *VGG-16-small* sítě a *state of the art* metod na *YTF* datasetu.

Kapitola 7

Závěr

V rámci této práce jsou popsány základní principy konvolučních neuronových sítí, jejich aktivační funkce a jakým způsobem se sítě trénují. Zároveň zde jsou popsány některé chybové funkce, které se pro trénování používají. Na toto navazuje shrnutí metod a předchozích prací pro rozpoznávání tváří ve videu.

Cílem práce bylo navrhnout a implementovat síť, která bude schopná rozpoznávat tváře ve videu. Toto je rozděleno do 2 logických bloků. Na síť, která počítá příznakové vektory z obrázků a na síť či metodu, která tyto příznakové vektory agreguje do jednoho výsledného vektoru. Tento vektor reprezentuje identitu v sekvenci snímků, neboli ve videu.

Povedlo se natrénovat síť, která produkuje příznakové vektory, jejíž přesnost testovaná na *LFW* je 92.8%. Tato síť obsahuje batch normalizaci, díky které se síť trénuje rychleji a dosahuje lepších výsledků. Trénována byla pomocí siamské sítě s chybovou funkcí *contrastive loss* na datasetu *PFC* obsahující 260 tisíc obrázků.

Dále se povedlo natrénovat agregační síť, která v případě agregace příznakových vektorů z *fingera* sítě má srovnatelné výsledky s jinými sítěmi testované na *YTF*. Agregované vektory ze sítě *VGG-16-small* mají prokazatelné horší výsledky, což je úměrné tomu, že síť extrahující tyto vektory nedosahuje srovnatelných výsledků s *fingera* síť.

Za důvod, proč *VGG-16-small* nedosahuje srovnatelných výsledků, považuji nedostatek trénovacích dat. Jak bylo zmíněno, *state of the art* sítě byly trénovány někdy i pomocí stovek miliónů obrázků, což je až o 3 řády více než obsahuje trénovací *PFC* dataset.

V rámci budoucího vývoje by bylo vhodné experimentovat s jinou základní architekturou pro extrakcí příznakového vektoru (například obsahující *Inception* vrstvu [29]). Tomuto se tato práce nevěnuje, jelikož samotné experimenty byly už tak časově náročné (v MetaCentru bylo počítáno celkem 448 dní procesorového času). Dále by bylo vhodné využít jinou agregační síť (například rekurentní neuronovou síť zmíněnou v [20]), získat více trénovacích dat nebo místo 2D zarovnání obličeje vyzkoušet 3D zarovnání.

Literatura

[1] Ahonen, T.; Hadid, A.; Pietikainen, M.: Face Description with Local Binary Patterns: Application to Face Recognition. Dec 2006, ISSN 0162-8828, s. 2037–2041, doi:10.1109/TPAMI.2006.244.

[2] Cauchy, A.-L.: Méthode générale pour la résolution des systèmes d'équations simultanées. *Compte Rendu des S'eances de L'Acad'emie des Sciences*, ročník 25, č. 2, 1847: s. 536–538.

[3] Chen, B. C.; Chen, C. S.; Hsu, W. H.: Face Recognition and Retrieval Using Cross-Age Reference Coding With Cross-Age Celebrity Dataset. *IEEE Transactions on Multimedia*, ročník 17, č. 6, June 2015: s. 804–815, ISSN 1520-9210, doi:10.1109/TMM.2015.2420374.

[4] Chollet, F.; aj.: Keras. 2015, [online], [cit. 2017-05-01].
URL https://github.com/fchollet/keras

[5] Fukushima, K.: Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, ročník 36, č. 4, 1980: s. 193–202, ISSN 03401200, doi:10.1007/BF00344251.

[6] Glorot, X.; Bordes, A.; Bengio, Y.: Deep sparse rectifier neural networks. *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, ročník 15, 2011: s. 315–323, ISSN 15324435, doi:10.1.1.208.6449.

[7] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press, 2016.

[8] Hinton, G.; Srivastava, N.; Swersky, K.: Overview of mini-batch gradient descent. [online], [cit. 2017-04-30].
URL
http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[9] Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; aj.: Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv e-prints*, 2012: s. 1–18, ISSN 9781467394673, doi:arXiv:1207.0580.

[10] Huang, G. B.; Ramesh, M.; Berg, T.; aj.: Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments. Technická Zpráva 07-49, University of Massachusetts, Amherst, October 2007.

[11] Ioffe, S.; Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Arxiv*, 2015: s. 1–11, ISSN 0717-6163, doi:10.1007/s13398-014-0173-7.2.

[12] Karpathy, A.: Convolutional Neural Networks for Visual Recognition. [online], [cit. 2017-05-02].
URL http://cs231n.github.io/linear-classify/

[13] Kingma, D. P.; Lei Ba, J.: Adam: A Method of Stochastic Optimization. 2015: s. 1–15.

[14] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, 2012: s. 1–9, ISSN 10495258, doi:http://dx.doi.org/10.1016/j.protcy.2014.09.007.

[15] Kumar, N.; Berg, A. C.; Belhumeur, P. N.; aj.: Attribute and Simile Classifiers for Face Verification. In *IEEE International Conference on Computer Vision (ICCV)*, Oct 2009.

[16] LeCun, Y.; Bengio, Y.; Hinton, G.: Deep learning. *Nature*, ročník 521, č. 7553, 2015: s. 436–444, ISSN 0028-0836, doi:10.1038/nature14539.

[17] Li, H.; Hua, G.; Shen, X.; aj.: Eigen-PEP for Video Face Recognition. 2015, ISBN 9783319168111, s. 17–33, doi:10.1007/978-3-319-16811-1.

[18] Linnainmaa, S.: The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, 1970: s. 6–7.

[19] McCulloch, W. S.; Pitts, W.: A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, ročník 5, č. 4, 1943: s. 115–133, ISSN 00074985, doi:10.1007/BF02478259.

[20] Mclaughlin, N.; del Rincon, J. M.; Miller, P.: Recurrent Convolutional Network for Video-based Person Re- Identification. *Computer Vision and Pattern Recognition (CVPR)*, 2016, ISSN 10636919, doi:10.1109/CVPR.2016.148.

[21] Mehrotra, K.; Mohan, C. K.; Ranka, S.: *Elements of Artificial Neural Networks*. The MIT Press, 1997, ISBN 0262133288, 344 s.

[22] Mishkin, D.; Matas, J.: All you need is a good init. *Iclr*, 2015: s. 1–8, ISSN 08981221, doi:10.1016/0898-1221(96)87329-9.

[23] Munakata, T.: *Fundamentals of the New Artificial Intelligence*. Springer-Verlag New York, Inc., druhé vydání, 2008, ISBN 978-1-84628-838-8, 255 s.

[24] Nair, V.; Hinton, G. E.: Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, , č. 3, 2010: s. 807–814, ISSN 1935-8237, doi:10.1.1.165.6419.

[25] Ng, H. W.; Winkler, S.: A data-driven approach to cleaning large face datasets. *2014 IEEE International Conference on Image Processing, ICIP 2014*, 2014: s. 343–347, doi:10.1109/ICIP.2014.7025068.

[26] Parkhi, O. M.; Vedaldi, A.; Zisserman, A.: Deep Face Recognition. *Procedings of the British Machine Vision Conference 2015*, , č. Section 3, 2015: s. 41.1–41.12, ISSN 00313203, doi:10.5244/C.29.41.

[27] Rosenblatt, F.: A probabilistic model for information storage and organization in the brain. *Psychological Review*, ročník 65, č. 6, 1958: s. 386–408, ISSN 1939-1471(Electronic);0033-295X(Print), doi:10.1037/h0042519.

[28] Rusk, N.: Do We Really Need to Collect Millions of Faces for Effective Face Recognition? *Nature Methods*, ročník 9905, č. 1, 2016: s. 35–35, ISSN 0302-9743, doi:10.1007/978-3-319-46448-0.

[29] Schroff, F.; Kalenichenko, D.; Philbin, J.: FaceNet: A unified embedding for face recognition and clustering. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, ročník 07-12-June-2015, 2015: s. 815–823, ISSN 10636919, doi:10.1109/CVPR.2015.7298682.

[30] Simonyan, K.; Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations (ICRL)*, 2015: s. 1–14, ISSN 09505849, doi:10.1016/j.infsof.2008.09.005.

[31] Szegedy, C.; Liu, W.; Jia, Y.; aj.: Going deeper with convolutions. *arXiv preprint arXiv: 1409.4842*, 2014: s. 1–9, ISSN 10495258, doi:10.1109/CVPR.2015.7298594.

[32] Taigman, Y.; Yang, M.; Ranzato, M.; aj.: DeepFace: Closing the gap to human-level performance in face verification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014: s. 1701–1708, ISSN 10636919, doi:10.1109/CVPR.2014.220.

[33] Wolf, L.; Hassner, T.; Maoz, I.: Face Recognition in Unconstrained Videos with Matched Background Similarity. 2011.

[34] Yang, J.; Ren, P.; Chen, D.; aj.: Neural Aggregation Network for Video Face Recognition. *Arxiv*, , č. October, 2016.

[35] Yi, D.; Lei, Z.; Liao, S.; aj.: Learning Face Representation from Scratch. *CoRR*, ročník abs/1411.7923, 2014.

Přílohy

Příloha A

Obsah přiloženého DVD

doc	obsahuje zdrojové dokumenty k technické práci a samotnou práci v PDF
examples	obsahuje ukázku použití a také video, které bylo součástí zadání
nets	obsahuje natrénované sítě
src	obsahuje zdrojové kódy, které byly použity k trénování a vyhodnocování
README.txt	obsahuje podrobnější popis jednotlivých souborů na přiloženém DVD