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

ODHAD OBLIČEJE Z ŘEČOVÉHO SIGNÁLU

LEARNING THE FACE BEHIND A VOICE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **PETR ZUBALÍK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. OLDŘICH PLCHOT, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání diplomové práce

Student: **Zubalík Petr, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Strojové učení
Název: **Odhad obličeje z řečového signálu**
Learning the Face Behind a Voice
Kategorie: Zpracování řeči a přirozeného jazyka
Zadání:

1. Prostudujte statistické techniky pro modelování řeči a řečníka, soustřeďte se na neuronové sítě a extrakci embedingů.
2. Prostudujte techniky pro zpracování obrazu (obličejů), seznamte se s dostupnými modely, které umožňují extrakci embedingů z obrazových dat.
3. Seznamte se s databázemi Voxceleb a AVS Speech a proveďte rešerši dalších dostupných databází vhodných k řešení problematiky odhadu obličeje z řečové nahrávky.
4. Proveďte rešerši problematiky odhadu obličeje z řeči a prostudujte současné přístupy a architektury používaných modelů.
5. Navrhněte a natrénujte systém, který bude generovat obličej na základě vstupní řeči.
6. Vyhodnoťte úspěšnost a robustnost natrénovaného modelu a analyzujte jeho případné nedostatky.

Literatura:

- Tae-Hyun Oh, Tali Dekel, Changil Kim, Inbar Mosseri, William T. Freeman, Michael Rubinstein, Wojciech Matusik, "Speech2Face: Learning the Face Behind a Voice", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019

- A. Ephrat, I. Mosseri, O. Lang, T. Dekel, K. Wilson, A. Hassidim, W. T. Freeman, and M. Rubinstein, "Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation", ACM Transactions on Graphics (SIG-GRAPH), 2018

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Plchot Oldřich, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 18. května 2022
Datum schválení: 2. listopadu 2021

Abstrakt

Hlavním cílem této diplomové práce bylo navrhnout a implementovat systém, který bude schopný odhadnout obličej na základě řeči daného člověka. Tento problém je vyřešen pomocí systému složeného ze tří modelů konvolučních neuronových sítí. První z nich je založen na architektuře ResNet a slouží pro extrahování příznaků z hlasových nahrávek. Druhým modelem je plně konvoluční neuronová síť, která převádí tyto příznaky na styly, na základě kterých bude upravován výsledný obrázek obličeje. Získané styly jsou poté předávány na vstup generátoru StyleGAN pro vygenerování výsledného obličeje. Navržený systém je implementován v programovacím jazyce Python s využitím frameworku PyTorch. V poslední kapitole práce je rozebráno a vyhodnoceno několik důležitých experimentů prováděných v rámci ladění a testování vytvořeného systému.

Abstract

The main goal of this thesis is to design and implement a system that will be able to generate a face based on the speech of a given person. This problem is solved using a system composed of three convolutional neural network models. The first one is based on the ResNet architecture and is used to extract features from speech recordings. The second model is a fully convolutional neural network which converts the extracted features into the styles which form a base for the final facial image. These styles are then passed as an input to the StyleGAN generator, which creates the resulting face. The proposed system is implemented in the Python programming language using the PyTorch framework. The last chapter of the thesis discusses some of the most significant experiments performed to fine-tune and test the developed system.

Klíčová slova

konvoluční neuronové sítě, ResNet, GAN, zpracování řeči, umělá inteligence, generativní adverzní sítě, zpracování obrazu, Python, PyTorch, odhad obličeje, StyleGAN

Keywords

convolutional neural networks, ResNet, GAN, speech processing, artificial intelligence, generative adversarial networks, image processing, Python, PyTorch, face estimation, StyleGAN

Citace

ZUBALÍK, Petr. *Odhad obličeje z řečového signálu*. Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Oldřich Plchot, Ph.D.

Odhad obličeje z řečového signálu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Oldřicha Plchota, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Zubalík
16. května 2022

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé práce panu Ing. Oldřichu Plchotovi, Ph.D. za cenné připomínky a odborné vedení při tvorbě této diplomové práce. Dále bych chtěl poděkovat organizaci MetaCentrum VO patřící pod českou Národní Gridovou Infrastrukturu (NGI), která mi umožnila přístup k výpočetním a úložným kapacitám potřebným pro mé výpočty.

Obsah

Úvod **2**

1 Zpracování zvukových a obrazových dat **3**
1.1 Extrakce příznaků z dat 3
1.2 Hluboké neuronové sítě 6
1.3 Vybrané architektury neuronových sítí 17

2 Datové sady **21**
2.1 VGGFace 21
2.2 VoxCeleb 22
2.3 AVSpeech 24

3 Návrh modelu pro odhad obličeje z řečového signálu **25**
3.1 Současné přístupy 25
3.2 Návrh vlastního systému 27
3.3 Trénování systému 32

4 Implementace **35**
4.1 Předzpracování dat 35
4.2 Implementace modelu 38
4.3 Trénovací algoritmus 40

5 Experimenty **43**
5.1 Prostředí experimentů 43
5.2 Data 44
5.3 Prvotní experimenty na malém datasetu 45
5.4 Experimenty s VoxCeleb2 47
5.5 Zhodnocení a návrh na pokračování 55

Závěr **57**

Literatura **59**

A Výsledky systému **64**

B Obsah datového média **66**

Úvod

Cílem mé práce je převod řeči na obrázek obličeje, který patří člověku mluvícímu na zvukové nahrávce. Tento převod by měl být uskutečněn pomocí jednoho nebo více modelů neuronových sítí spojených do jednoho systému, který na svém vstupu vezme zvukový signál, nebo jeho reprezentaci, a na výstupu vrátí obrázek obličeje. Cílem však není, aby výsledný obrázek obsahoval stoprocentní kopii obličeje daného řečníka, spíše by měl vyobrazovat obličej, který bude obsahovat stejné rysy ovlivňující hlas mluvčího – velikost úst a nosu, tvar obličeje, struktura lícních kostí nebo například plnost rtů. Do výsledného obrázku by se měla promítnout i etnická příslušnost daného člověka, protože tento fakt se také projevuje na způsobu, jakým daný jedinec mluví.

V rámci první části této práce jsem nastudoval metody zpracování řeči a obrazu, které by mohly být vhodné pro využití na převod zvukového signálu obsahující řeč na korespondující obrázek obličeje daného člověka. Soustředil jsem se na neuronové sítě, hlavně na typ konvolučních neuronových sítí, které se v posledních letech staly velmi oblíbené na poli zpracování zvuku a obrazu. Pro zpracování řeči a obrazu se dají použít obdobné modely, protože oba typy těchto dat jsou vhodné pro zpracování pomocí filtrů, ze kterých jsou složeny jednotlivé konvoluční vrstvy v konvolučních neuronových sítích.

V další fázi jsem se soustředil na sběr dat pro trénování modelů, které budou převod audio signálu na obraz obličeje vykonávat. Vybral jsem datasety VoxCeleb a AVSpeech. VoxCeleb je hotová datová sada, kterou stačilo po částech stáhnout, a poté spojit dohromady. Autoři AVSpeech dali k dispozici pouze metadata, protože jejich dataset obsahuje několik terabytů dat. Pro tuto datovou sadu jsem tedy implementoval program, který ji postupně stahuje a zároveň připravuje pro trénování. Dataset VoxCeleb výchází z datasetu VGGFace, který obsahuje fotky obličejů lidí, jejichž nahrávky hlasu jsou v datové sadě VoxCeleb.

Ve třetí kapitole popisuji návrh systému, který se skládá z několika modelů neuronových sítí. První z nich je konvoluční enkodér hlasových nahrávek založený na architektuře ResNet. Dalším modelem je mapovací plně konvoluční neuronová síť převádějící extrahované příznaky na tzv. styly, pomocí kterých bude tvořen výsledný obrázek. Poslední součástí mého systému je generativní model *StyleGAN2*, který na základě dříve získaných stylů vytváří obrázek obličeje. Ve čtvrté kapitole je popsána implementace všech potřebných částí. Před samotným systémem jsem nejdříve implementoval program pro stahování datasetu AVSpeech a normalizaci fotek obličejů ze získaného datasetu. Pro implementaci modelů neuronových sítích jsem využil programovací jazyk Python a framework PyTorch.

V poslední části této práce jsou uvedeny nejdůležitější experimenty a jejich výsledky, pomocí kterých jsem svůj systém ladil a testoval. Ke většině experimentů jsou uvedeny výsledky ve formě obrázků. V rámci fáze experimentování jsem nad systémem prováděl změny, které jsou v této kapitole také popsány. Po experimentech je uvedeno krátké zhodnocení výsledků mé práce a návrh, jak bych ve své práci pokračoval, pokud bych měl k dispozici více času a výpočetních prostředků.

Kapitola 1

Zpracování zvukových a obrazových dat

V této kapitole jsou uvedeny teoretické informace vztahující se k metodám použitým k řešení mé práce. Celá moje práce se pohybuje kolem zpracování řeči a obrazu pomocí neuronových sítí. V rámci teorie jsem se zaměřil hlavně na ně. V první části popisuji extrakci příznaků hlavně ze zvukových dat, které jsou posléze dávány na vstup neuronových sítí. V druhé sekci jsou popsány právě neuronové sítě – jejich základní princip a učení. Důraz je zde kladen také na konvoluční neuronové sítě, které jsou hlavním nástrojem v oblasti zpracování obrazu ale používané jsou také na zpracování zvukových signálů.

1.1 Extrakce příznaků z dat

V dnešní době je k úspěšnému natrénování hlubokých modelů neuronových sítí a zpracování velkého množství dat potřeba vysoký výpočetní výkon, aby samotné trénování netrvalo příliš dlouhou dobu. Tato časová náročnost může být úspěšně redukována pomocí správné úpravy našich trénovacích dat pomocí již zmíněné extrakce příznaků. Je to tedy první, a velmi důležitý, krok ve zpracování dat. Jedná se o proces, kdy je hlavním cílem z dat o mnoho dimenzích vytáhnout pouze jejich relevantní části pro danou úlohu.

V oblasti zpracování řeči a zvukových signálů se používají techniky pro extrakci příznaků, které využívají práci se spektrem daného signálu. Toto spektrum obsahuje kombinaci frekvencí, ze kterých se daný zvukový signál skládá. Obecně, spektrum ukazuje magnitudu a fázi signálu jako funkci frekvence a času. I když je dokázáno, že použití původního audio signálu na vstupu, může mít stejné [54] nebo dokonce lepší [49] výsledky, než použití spektra, je spektrum využíváno stále ve většině přístupů ke zpracování zvuku.

Prvním z uvedených příznaků jsou MFCC (angl. *Mel Frequency Cepstral Coeficients*) [18], které byly představeny už v 80. letech minulého století, ale stále jsou hojně používány i v dnešní době. Další skupinou jsou spektrogramy, jichž existuje hodně druhů, mezi které patří i Mel-spektrogram. Postup jeho výpočtu je o něco jednodušší než u MFCC. Tento druh spektrogramu není však vhodný vždy. Například v [48][23] používají autoři místo mel-spektrogramů komplexní spektrogramy, protože by je převod do Mel-spektra připravil o část potřebných frekvencí a fázi. Stupnice Mel-spektra je totiž založena na nelineárním lidském vnímáni zvuku, kdy je člověk schopen lépe rozlišit změnu v tónu daného signálu při nízké frekvenci než při vysoké.

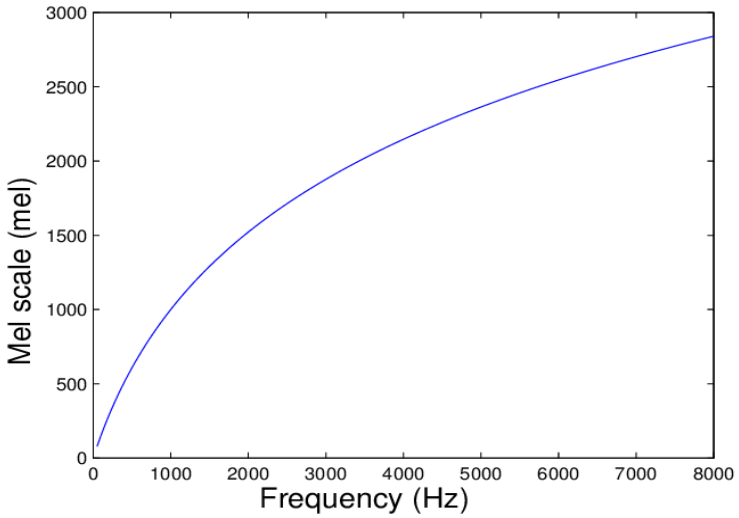
V této sekci budou popsány tyto dva způsoby extrakce příznaků ze zvukových dat. V rámci mé diplomové práce budu experimentovat hlavně se spektrogramy a původním nezpracovaným audio signálem, ale zde jsem uvedl i MFCC, jelikož se jejich výpočet hodně podobá výpočtu spektrogramů.

1.1.1 MFCC

MFCC jsou tvořeny několika-krokovým procesem, který je vykreslen na obrázku 1.2. Na samotném začátku musí být signál rozdělen na rámce. Cílem tohoto rozdělení je získat krátké části signálu, které jsou v daném rámci stacionární. Každý rámec má délku asi 20-40 ms a obsahuje n vzorků. Další rámec začíná m vzorků po počátečním vzorku a překrývá se s předchozím rámcem na $(n - m)$ vzorcích. Je důležité pohlídat, aby rámec nebyl příliš krátký, pak by obsahoval málo vzorků pro výpočet, ani moc dlouhý, protože pak by se signál obsažený v tomto rámci moc měnil.

Druhým krokem je použití tzv. *Hammingova okna*. Tato funkce se využívá na vyhlazení signálu na konci a na začátku rámce. Když se tvoří nový rámec, má na začátku konci signál ostře useknutý, skákající z 0 na nějakou jinou hodnotu a naopak. Tato diskontinuita může způsobit problémy ve výsledné frekvenční analýze v podobě šumu.

Třetí krok spočívá v aplikaci Diskrétní Fourierovy Transformace na každý rámec. Ta převádí daný rámec z domény času na frekvenční doménu.



Obrázek 1.1: Graf zobrazuje vztah mezi frekvenční stupnicí a stupnicí Mel. Obrázek je převzat z práce [25].

Dalším krokem v tomto procesu je převod frekvenční stupnice na stupnici Mel, která je vyjádřena ve stejnojmenné jednotce Mel (jejich vztah můžete vidět na obrázku 1.1). Tento převod může být zapsán následovně:

$$F_{\text{MEL}} = 2959 \log_{10}\left(\frac{F_{\text{Hz}}}{700}\right) \tag{1.1}$$

V posledním kroku zbývá převést signál zpátky do časové domény pomocí diskrétní cosinové transformace, pomocí které se dá udělat inverzní fourierova transformace. Tento převod je vyjádřen následujícím vztahem:

$$c_{\mathrm{mf}}(n) = \sum_{k=1}^{K} \log\, m_k cos\left[n(k-0.5)\frac{\pi}{K}\right], \tag{1.2}$$

jehož výsledkem jsou Mel-frekvenční cepstrální koeficienty. Tuto část práce o vytvoření MFCC jsem čerpal ze studijní opory [64].

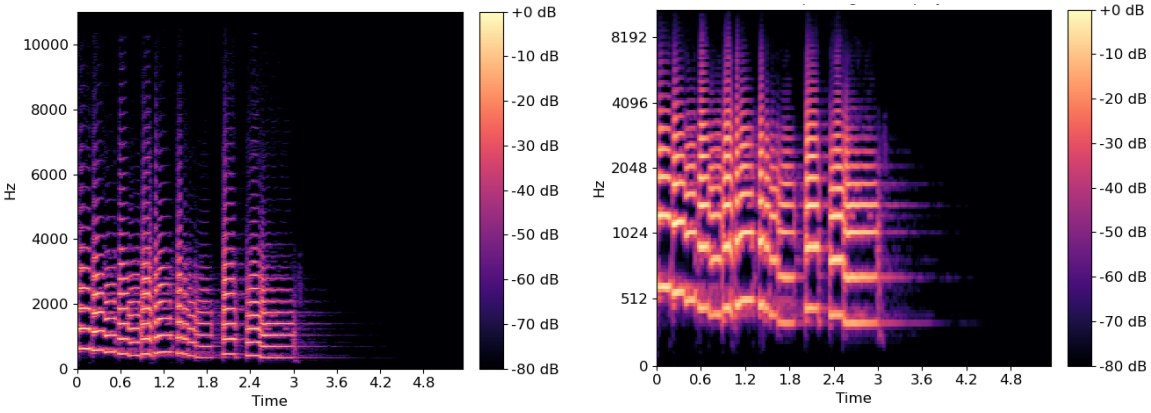
Obrázek 1.2: Diagram popisující proces vytváření MFCC koeficientů ze zvukového signálu. Nejdříve se daný signál rozdělí na rámce o určité velikosti, na které se poté aplikuje Hammingovo okno. Nad těmito rámci se počítá Diskrétní Fourierova Transformace. V posledních dvou krocích je stupnice frekvencí převedena na stupnici Mel a následně je aplikovaná diskrétní kosinová transformace.

1.1.2 Spektrogram

Spektrogram je vizuální reprezentace spektra audio signálu, které se skládá z frekvencí signálu v čase. Obecně se dá říct, že vodorovná osa spektrogramu reprezentuje čas, svislá osa frekvenci, a barvy ve spektrogramu určují úroveň energie signálu.

Pro vytvoření výše zmíněného mel-spektrogramu je využit podobný postup jako pro vytvoření mel-frekvenčních cepstrálních koeficientů. Jedinou změnou u počítání spektrogramu je ta, že jsou vynechány poslední kroky výpočtu MFCC, kdy se počítá převod frekvencí do logaritmické domény a diskrétní kosinová transformace.

Mel-spektrogram, který je převedený do Mel škály, nemusí být vždy vhodný pro reprezentaci zvukového signálu. Pokud je zapotřebí zachovat všechny frekvence bez úprav, může být spektrogram vytvořen pouze pomocí výpočtu diskrétní fourierovy transformace nad rámci, které byly také upraveny pomocí Hammingova okna. Po provedení této transformace se ještě můžou hodnoty z ní získané umocnit pomocí určitého koeficientu.



Obrázek 1.3: Na tomto obrázku je vidět rozdíl mezi spektrogramem (vlevo) a mel-spektrogramem (vpravo). Obrázky jsou převzaty z [6].

1.1.3 Extrakce příznaků z obrazu

Pro extrakci příznaků z obrazových dat existuje několik způsobů. Většina těchto metod využívá různé filtry pro modifikaci hodnot pixelů v závislosti na jejich okolí. Nejběžnější typ filtrů jsou lineární filtry. Lineární se jim říká, protože počítají novou hodnotu pixelu pomocí lineární kombinace hodnot pixelů z jeho okolí. Filtrování je založeno na operaci konvoluce, která je více rozebrána v sekci o konvolučních neuronových sítích 1.2.3. Nová hodnota daného pixelu se počítá pomocí tzv. *jádra*. To je reprezentováno maticí pevně nastavených čísel. Pomocí této matice se pak počítá lineární kombinace čísel z okolí pixelu. Výstupem filtrování je tzv. *mapa příznaků*, která obsahuje extrahované příznaky.

Asi nejvíce známým využitím lineárních filtrů je detekce hran v obraze. Pro tento úkol vznikla celá řada filtrů používaných v praxi. Za vyjmenování stojí filtr Sobelův, Laplaceův a nebo Prewittův [57]. Pro extrakci příznaků se dále může použít například filtr pro segmentaci obrazu, která může být buď kontextuální nebo nekontextuální. Kontextuální segmentace sdružuje pixely s podobnými vlastnostmi, které jsou navíc blízko u sebe. Nekontextuální segmentace sdružuje pixely s podobnými vlastnostmi globálně, z celého obrázku. Jednoduchou aplikací segmentace může být *prahování*. Je to technika, která nastavuje hodnotu pixelu na jednu z binárních hodnot 1 nebo 0 podle předchozí hodnoty pixelu. Pokud byla tato hodnota větší než námi zvolený práh, je nová hodnota pixelu rovna 1, pokud menší, je nová hodnota rovna 0. Výsledkem tohoto prahování je binární mapa příznaků [26].

Dalšími možnostmi aplikace lineárních filtrů je rozmazání, nebo naopak zaostření, obrazu. Dalšími z této kategorie filtrů jsou filtry pro odstranění šumu z obrazu. Nejedná se už o extrakci příznaků jako takovou, ale pořád může jít o úpravu obrázku pro jeho jednodušší následné zpracování [1].

Pro extrakci příznaků z obrazu existuje i další možnost, z hlediska mé práce nejdůležitější, a to jsou konvoluční neuronové sítě. Modely konvolučních neuronových sítí také používají filtry zvané *konvoluční jádra*, jejichž velkou výhodou je, že parametry tohoto filtru nejsou nastaveny napevno, ale jsou optimalizovány v průběhu trénování neuronové sítě. Neuronová síť si tedy sama určí parametry těchto filtrů podle učícího algoritmu a hodnot chybové funkce. Princip neuronových sítí a způsob jejich učení je popsán v následující části této kapitoly.

1.2 Hluboké neuronové sítě

Umělé neuronové sítě vznikly jako napodobenina biologických neuronových sítí, které tvoří mozek lidí a zvířat. Biologické neuronové sítě se skládají z bilionů jednotlivých výpočetních jednotek, tzv. *neuronů*. Každý z těchto neuronů dokáže přijímat elektrický signál, který mu je předán buď jinými neurony nebo smyslovými receptory. Jakmile daný neuron tyto signály zpracuje, předává je dalším neuronům, které s ním budou opět pracovat. Právě díky propojení a efektivní komunikaci těchto neuronů je biologický mozek schopen tak velkého výpočetního výkonu. Propojení jednotlivých neuronů je realizováno pomocí *synapsí*. Upravováním existujících synaptických vazeb, a vytvářením nových, je dáno, že se lidé dokáží učit. Propojování výpočetních jednotek a učení se snaží napodobit i umělé neuronové sítě.

V dnešní době mají umělé neuronové sítě značné uplatnění ať už v komerčních nebo nekomerčních aplikacích. V této kapitole se budu hodně zabývat konvolučními neuronovými sítěmi. Ty se používají jak na zpracování řeči, tak na zpracování obrázků. Dá se říci, že hlavní využití modelů konvolučních neuronových sítí je dvojí. Buď se extrahují z dat příznaky, které se posléze využijí (rozpoznání řeči, detekce obličejů a jejich rozpoznání, segmentace řeči

či obrazu nebo třeba klasifikace obrázků), nebo se generují nová data s určitými vlastnostmi (generování obrázků obličejů, které mají zadané rysy, generování obrázků podle textového popisu nebo např. generování falešné hlasové nahrávky, kterou ale daný člověk nenamluvil). Oba dva způsoby – tedy extrakce příznaků i generování nových dat – najdou v této práci využití, a proto se jimi budu v této kapitole zabývat.

1.2.1 Základní princip neuronových sítí

Nejčastěji používaný typ umělých neuronových sítí je označován jako *dopředné neuronové sítě*. U dopředných sítí (někdy nazývaných také *vícevrstvé perceptrony*) se informace šíří od vstupní vrstvy postupně přes každou skrytou vrstvu až na výstup. Neexistují zde žádná zpětná spojení, která by vedla informaci z výstupu skryté vrstvy zpátky na její vstup. Takto koncipované neuronové sítě se nazývají *rekurentní*. Ty se také využívají ve zpracování řeči, v této práci je však nevyužívám, tak je zde nebudu dále rozebírat. Kvalitní vhled do rekurentních neuronových sítí poskytuje kniha Deep Learning od pana Goodfellowa a spol. [27].

Každá dopředná neuronová síť aproximuje určitou matematickou funkci f. U neuronové sítě pro klasifikaci s parametry $\boldsymbol{\theta}$ tato funkce mapuje vstupní vektor hodnot $\boldsymbol{x}$ na výstupní vektor tříd $\boldsymbol{y}$ ($\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$). V průběhu učení pak optimalizuje hodnoty parametrů $\boldsymbol{\theta}$, pro které dostává nejlepší aproximaci mapující funkce f. Pokud má daný model sítě více vrstev, může být mapující funkce f rozdělena na jednotlivé funkce pro každou vrstvu. Má-li model tři vrstvy, vzniknou tři funkce f_1, f_2, f_3. f_1 pro první vrstvu, f_2 pro druhou vrstvu a f_3 pro třetí vrstvu. Tyto funkce nyní můžou být zřetězeny za sebe, což dává výslednou funkci $f(\boldsymbol{x} = f_3(f_2(f_1(\boldsymbol{x}))))$. Vrstva reprezentovaná funkcí f_1 se tedy nazývá *vstupní*, protože jako první zpracovává vektor hodnot $\boldsymbol{x}$. Výstupní vrstvou je pak ta, která je reprezentovaná funkcí f_3, protože její výsledek je výsledkem konečným, který je dán na výstup celé sítě. Druhá vrstva v pořadí se nazývá skrytá, protože není vidět ani z jedné strany neuronové sítě.

Pojem hluboké učení se začal používat, protože se hloubka neuronových sítí – tedy počet jejich skrytých vrstev – neustále zvyšoval. Na počátku existovaly pouze modely s několika skrytými vrstvami. V nynějších modelech jsou skrytých vrstev i tisíce s obrovským počtem optimalizovaných parametrů.

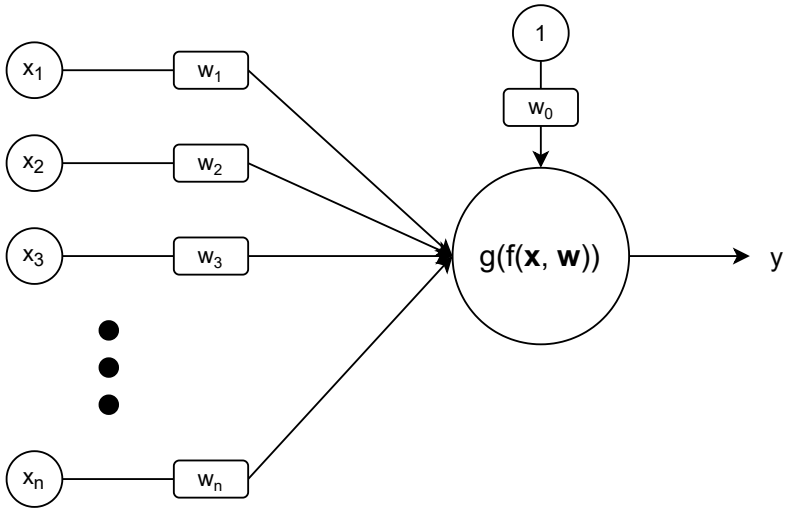
Perceptron. Jako předchůdci dnešních neuronových sítí jsou brány lineární modely, které abstrahovaly funkci jednoho neuronu z lidského mozku. Prvním dobře použitelným z nich byl *Perceptron* [53]. Jeho strukturu můžete vidět na obrázku 1.4. Tento model se touto strukturou výrazně nelišil od svých předchůdců, velmi je ale předehnal v možnostech počítání. Perceptron už dokázal pracovat i se vstupními hodnotami, které nejsou binární, váhy pro jednotlivé vstupy mohly také nabývat různých hodnot, a největším posunem kupředu bylo učící pravidlo, pomocí kterého se mohly jednotlivé hodnoty vah měnit už v průběhu výpočtu. Na vstupu tedy bral vektor reálných čísel $\boldsymbol{x}$, ke kterému přiřadil vektor vah w. Dál v tomto modelu figuruje tzv. práh (angl. *bias*), který je speciálním vstupem do tohoto modelu neuronu, jehož hodnota je pořád $x_0 = 1$. U předchozích modelů neuronu byla váha w_0 pro tento práh vždy nastavena na pevnou hodnotu, u perceptronu mohla být její hodnota měněna v průběhu výpočtu pomocí trénovacího pravidla jako u jiných vah z vektoru $\boldsymbol{w}$ [30]. Po přijetí všech vstupů se v těle neuronu počítají dvě funkce – bázová $f(\boldsymbol{x})$ a aktivační $g(f(\boldsymbol{x}))$.

$$f(\boldsymbol{x}) = \sum_{i=0}^{n} x_i w_i \tag{1.3}$$

$$g(f(\boldsymbol{x})) = \begin{cases} 0 & \text{pro } f(\boldsymbol{x}) < 0 \\ 1 & \text{pro } f(\boldsymbol{x}) \geq 0 \end{cases}, \tag{1.4}$$

kde x_i je jedna hodnota z vektoru $\boldsymbol{x}$, w_i je hodnota z vektoru vah $\boldsymbol{w}$ a n je celkový počet hodnot ve vektoru $\boldsymbol{x}$ [30].

Aktivační (také nazývaná jako *přenosová*) funkce $g(f(\boldsymbol{x}))$ použitá u perceptronu se jmenuje *skoková aktivační funkce* [27]. Tato aktivační funkce byla hojně používána u modelů, které reprezentovaly osamocený neuron. Výběr aktivační funkce má výrazný vliv na chování modelu, a proto budou v další sekci popsány hojně používané typy aktivačních funkcí.



Obrázek 1.4: Na tomto obrázku můžete vidět model perceptronu. x_i značí jednotlivé vstupní hodnoty a w_i jsou k nim korespondující váhy, w_0 je váha udávající hodnotu prahu pro aktivační funkci $g(f(\boldsymbol{x}, \boldsymbol{w}))$ a y je výstupní hodnota.

Aktivační funkce

Aktivační funkce, v rovnici 1.4 označená jako g, udává, jaký výstup půjde z daného neuronu. Tato funkce má velký vliv na chování jednotlivých neuronů i celé neuronové sítě, a proto je tedy nutné popsat ty nejznámější a nejpoužívanější z nich.

Skoková funkce. Jak již bylo zmíněno výše, skoková aktivační funkce byla používána hlavně u modelů, které reprezentovaly osamocený neuron. Definiční obor této funkce je definován na celé množině reálných čísel. Obor hodnot je však definován pouze jako množina $H(f) = 0; 1$. Hodnota funkce tudíž skokově přechází z hodnoty 0 na hodnotu 1 a naopak. To ukazuje vlastnost funkce zvanou *all-or-none*[1] [30].

Logistická sigmoida. Jednou z původních aktivačních funkcí je logistická sigmoida. Jedná se o esovitě zahnout funkci, jejíž obor hodnot je definován na intervalu $(0; 1)$. Její definiční obor je opět definován na celé množině reálných čísel. Pro její obor hodnot se tato funkce

[1] všechno nebo nic

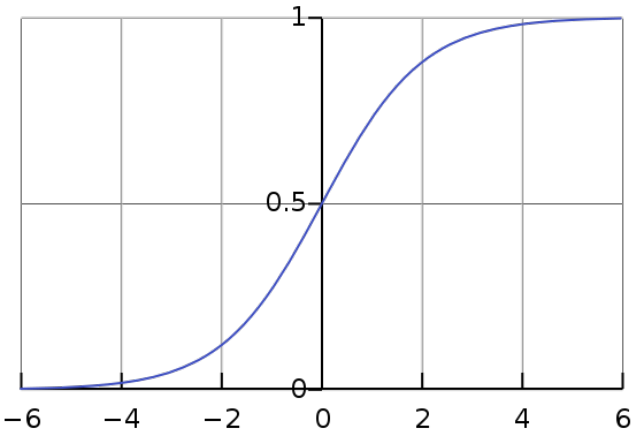
často používá jako výstupní funkce z neuronové sítě, která má udávat pravděpodobnost určitého jevu. Funkční předpis této funkce je následovný:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}, \tag{1.5}$$

kde L je maximální hodnota funkce, x_0 je střední hodnota definičního oboru, e je Eulerovo číslo a k je strmost křivky (nebo také stupeň růstu) [45][44]. Nejpoužívanější hodnoty jsou však $L = 1, k = 1, x_0 = 0$ [30]. Pro ně má pak rovnice tvar:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{1.6}$$

Průběh této logistické sigmoidy je zobrazen na obrázku 1.5.



Obrázek 1.5: Ukázka průběhu aktivační funkce logistická sigmoida. Obrázek převzat z [5].

Hyperbolický tangens. Alternativou logistické sigmoidy se stala aktivační funkce hyperbolický tangens. Tyto funkce mají velmi podobný průběh, hlavní rozdíl je však ten, že při použití hyperbolického tangentu se model rychleji učí a konverguje k maximu [27]. Její průběh můžete vidět na obrázku 1.6. Předpis této aktivační funkce je:

$$f(x) = tanh(x), \tag{1.7}$$

kde

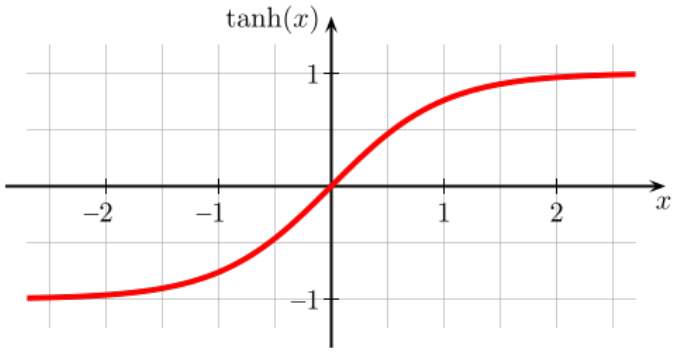
$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}. \tag{1.8}$$

Hyperbolický tangens se dá vyjádřit i pomocí funkce logistické sigmoidy $\sigma(x)$ [27]:

$$tanh(x) = 2\sigma(2x) - 1. \tag{1.9}$$

ReLU. ReLU neboli *Rectified linear unit* [47] je v dnešní době jednou z nejvyužívanějších aktivačních funkcí. Je dobrou volbou pro vrstvy neuronové sítě, u kterých si člověk není jistý, jakou aktivační funkci zvolit. Na kladné části definičního oboru je tato funkce lineární, což má za následek, že je dobře optimalizovatelná pomocí metod využívajících gradient. V záporné části definičního oboru je pak rovna 0. Její předpis vypadá takto:

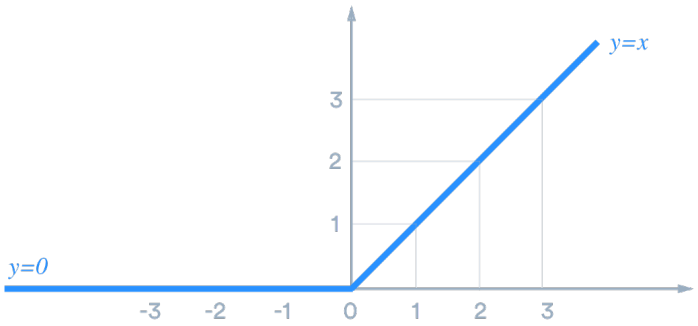
$$f(x) = max(0, x). \tag{1.10}$$



Obrázek 1.6: Ukázka průběhu aktivační funkce hyperbolický tangens. Obrázek převzat z [3].

Derivace této funkce nabývá buď hodnoty 0 pro záporná čísla, nebo hodnoty 1 pro kladná čísla. V bodě $x = 0$ nemá definovanou derivaci. V praxi se ale uvádí, že derivace pro hodnotu $x = 0$ je také rovna hodnotě 1:

$$\frac{\delta}{\delta x} f(x) = \begin{cases} 0 & \text{pro } x < 0 \\ 1 & \text{pro } x \geq 0 \end{cases}. \tag{1.11}$$



Obrázek 1.7: Ukázka průběhu aktivační funkce ReLU. Obrázek převzat z [42]

1.2.2 Učení neuronových sítí

Neuronová síť se skládá z velkého množství neuronů. Vlastnosti spojení jednotlivých neuronů jsou dány pomocí vah, kdy každé spojení má jednu váhu. Učením neuronové sítě se myslí úprava vah spojení neuronů tak, aby byla minimalizována chybová funkce – neboli, aby měla neuronová síť co nejlepší odezvu na trénovací data. Chybová funkce určuje, jak moc se liší očekávaný výsledek a reálný výstup neuronové sítě. Jakým způsobem se tato odlišnost určuje je dáno typem chybové funkce. Další důležitou součástí je samotný učící algoritmus, který má za úkol měnit váhy spojení, aby byla chybová funkce minimalizována. V průběhu let bylo představeno mnoho chybových funkcí i učících algoritmů. Některé z nich v této sekci představím.

Chybové funkce

Jelikož chybová funkce (angl. *cost function* nebo *loss function*), udává, jak moc se liší hodnota výstupu neuronové sítě od očekávaného výsledku, výběr typu této funkce má velký vliv na úspěšnost samotného učení.

Mean squared error. Je to jedna z nejpoužívanějších chybových funkcí a může být definována jako

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}(y_{p,j}^{*} - y_{p,j})^2, \tag{1.12}$$

kde N je počet vzorků z trénovací sady, M je počet výstupních neuronů, $y_{i,j}$ je výstup neuronové sítě a $y_{i,j}^{*}$ je *ground truth* (očekávaný výsledek). Pokud by se podařilo dosáhnout rovnosti $y_{i,j}^{*} = y_{p,j}$, platilo by $MSE = 0$ [27].

Cross entropy. Tato chybová funkce se nejčastěji používá při klasifikaci 1 z K tříd. Její výstup je K čísel z intervalu $(0; 1)$, které se dají považovat za normalizovanou pravděpodobnost, se kterou patří daný vzorek dat do odpovídající třídy. Cross entropy pro dávku N vzorků z trénovací sady se počítá jako průměr hodnot cross entropy pro jednotlivé vzorky:

$$E = \frac{1}{N}\sum_{n=1}^{N} -log\left(\frac{e^{\boldsymbol{f}_{y_n}}}{\sum\limits_{j=0}^{J} e^{\boldsymbol{f}_j}}\right), \tag{1.13}$$

kde N je počet vzorků z trénovací sady, y_n je správně klasifikovaná třída pro daný vstup uložená ve výstupním vektoru $\boldsymbol{f}$ s pravděpodobnostmi pro všechny třídy, j jsou indexy všech pravděpodobností pro třídy uložených ve vektoru $\boldsymbol{f}$ [27][2][50].

Optimalizační algoritmy

Neuronové sítě obvykle používají iterativní učící algoritmy založené na výpočtu gradientu chybové funkce a postupu proti směru jeho růstu. Tak vznikl taky jejich název – algoritmus gradientního sestupu [10]. Cílem těchto algoritmů je najít hodnoty vah, které minimalizují chybovou funkci. Od algoritmu gradientního sestupu byl odvozen další algoritmus – *Adam*. Oba tyto algoritmy budou popsány v této sekci.

Gradientní sestup. Jak již bylo řečeno, tento algoritmus minimalizuje chybovou funkci $f(\boldsymbol{x})$. Této minimalizace je dosaženo pomocí počítání parciálních derivací $\left(\frac{\delta y}{\delta x_i}\right)$, kde x_i je vzorek trénovacích dat na vstupu do neuronové sítě. Pomocí této derivace je získán směr růstu dané funkce $f(\boldsymbol{x})$ v bodě x_i. Gradient je pak tvořen jednotlivými parciálními derivacemi v bodě x_i ze vstupního vektoru dat $\boldsymbol{x}$. Jelikož gradient udává směr růstu funkce $f(\boldsymbol{x})$, pro minimalizaci nám stačí vzít jeho zápornou hodnotu a jít proti směru jeho růstu. Nové hodnoty vah dané neuronové sítě se pak upravují následovně:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \epsilon\nabla_{\boldsymbol{w}_i} f(\boldsymbol{w}_i), \tag{1.14}$$

kde $\boldsymbol{w}_{i+1}$ je vektor nově vypočtených vah pro následující iteraci učícího algoritmu, $\boldsymbol{w}_i$ je vektor vah ze stávající iterace, ϵ je učící krok (angl. *learning rate* – jeden z nejdůležitějších hyperparametrů při učení neuronové sítě) a $\nabla_{w_i} f(\boldsymbol{w}_i)$ je gradient vypočtený ve stávající iteraci i [27]. Pokud by bylo úkolem funkci $f(\boldsymbol{x})$ maximalizovat, stačí jít po směru gradientu:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i + \epsilon\nabla_{\boldsymbol{w}_i} f(\boldsymbol{w}_i), \tag{1.15}$$

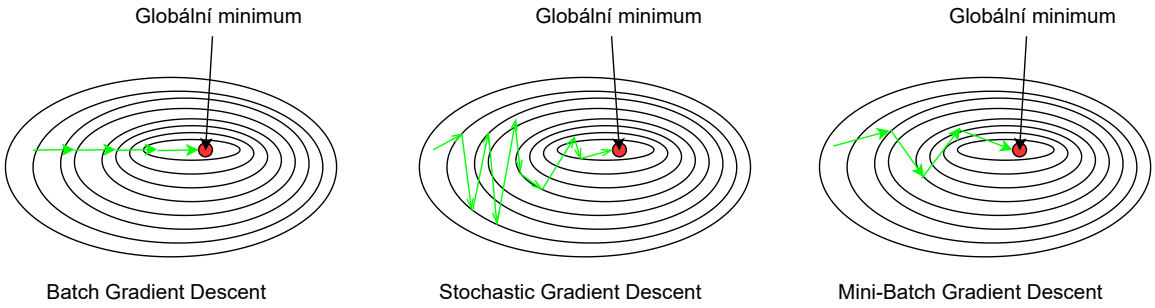
Adam. Celým názvem *Adaptive moment estimation* [39]. Tento algoritmus funguje na stejném principu jako algoritmus gradientního sestupu. Velkou výhodou tohoto algoritmu ale je, že používá pro každý parametr vlastní koeficient učení, který se navíc v průběhu trénování mění. Algoritmus gradientního sestupu používá pouze jeden koeficient učení pro všechny váhy.

Oba dva algoritmy můžou být spouštěny třemi různými způsoby. Každý způsob se liší velikostí dávky vzorků dat, pro kterou se počítá chybová funkce a upravují váhy neuronové sítě. Základní z nich, počítající nové hodnoty vah po zpracování dávky z celého datasetu, není moc optimální, protože v dnešní době se modely trénují na velkém množství dat, a učení je pomalé. Výsledek chybové funkce počítá jako průměr hodnot chybové funkce jednotlivých vzorků dat. Tato metoda se anglicky nazývá *batch gradient descent* [27].

Druhou metodou je tzv. *stochastic gradient descent*, který chybovou funkci a nové hodnoty parametrů počítá pro každý jeden vzorek dat z datasetu [27]. Největší výhodou tohoto způsobu je v rychlosti učení a rychlé konvergenci, což je způsobeno tím, že nemusí procházet celý dataset pro jednu úpravu parametrů. Další výhodou je fakt, že tento způsob přidává do učícího algoritmu trošku šumu, což zlepšuje výslednou generalizaci modelu.

Poslední, třetí, metoda je zlatou střední cestou. Tzv. *mini batch gradient descent* počítá chybovou funkci a úpravu parametrů neuronové sítě z dávky vzorků dat o předem dané velikosti [27]. Je rychlejší než gradientní sestup z dávky dat obsahující celý dataset, stále však přidává šum do učícího procesu, čím zlepšuje generalizaci. Občas se může stát, že tento šum zapříčiní, že bude algoritmus skákat tam a zase zpátky a nebude konvergovat do minima. Tomu se dá předejít pomocí pomalého snižování učícího koeficientu.

Nové hodnoty gradientů jsou počítány pomocí algoritmu zpětného šíření chyby (angl. *backpropagation*) [27]. Tento algoritmus je často zaměňován za kompletní učící algoritmus, např. gradientní sestup. Backpropagation má však za úkol pouze propagovat informaci získanou vypočítáním chybové funkce zpětně do neuronové sítě, aby se mohly počítat gradienty pro všechny parametry ve všech vrstvách. Princip algoritmu backpopagation a výpočet gradientů bude popsán v následující sekci.



Obrázek 1.8: Tento obrázek popisuje, jaký je rozdíl v průběhu učení a nalezení globálního minima pomocí jednotlivých typů gradientního sestupu – Batch Gradient Descent, Stochastic Gradient Descent a Mini-Batch Gradient Descent.

Backpropagation

Učení neuronové sítě pomocí gradientního sestupu a backpropagation sestává ze 4 fází:

1. **dopředné šíření vstupních hodnot** – výpočet výstupu sítě skrze propagace vstupní hodnoty neuronovou sítí,
2. **výpočet chybové funkce** – pomocí výstupu z neuronové sítě je spočítána hodnota chybová funkce a její derivace,
3. **zpětná propagace chyby** – pomocí zpětného šíření chyby a řetězového pravidla jsou vypočítány hodnoty gradientů pro všechny parametry sítě,
4. **gradientní sestup** – aktualizace hodnot vah na všech spojení mezi neurony pomocí vypočítaných gradientů v předešlém kroku.

Řetězové pravidlo. Řetězové pravidlo (angl. *chain rule*) je metoda, která slouží pro výpočet derivace chybové funkce v závislosti na konkrétním parametru neuronové sítě pomocí derivací chybové funkce v závislosti na parametrech, které se nacházejí blíže výstupu neuronové sítě – tedy pomocí derivací, jejichž hodnoty jsou už známé.

V dopředné neuronové síti počítá každý neuron váhovanou sumu vstupů a jejich vah:

$$y_i = \sum_j w_{i,j} z_j, \tag{1.16}$$

kde z_j je výstup j-tého neuronu předchozí vrstvy, který je použit jako vstup do i-tého neuronu stávající vrstvy. $w_{i,j}$ je pak váha přiřazená k danému propojení. Řetězové pravidlo pak se poté dá pro vyhodnocení derivace chybové funkce v závislosti na parametru $w_{i,j}$ definovat takto:

$$\frac{\delta E_n}{\delta w_{i,j}} = \delta_i \frac{\delta y_i}{\delta w_{i,j}}, \tag{1.17}$$

kde δ_i je chyba, která může být zapsána jako:

$$\delta_i = \frac{\delta E_n}{\delta y_i}. \tag{1.18}$$

Rovnice zpětného šíření chyby pro k-tou vrstvu může být zapsána takto:

$$\delta_j^{(k)} = f'(a_j^{(k)}) \sum_i w_{i,j}^{(k+1)} \delta_i^{(k+1)}, \tag{1.19}$$

kde $w_{i,j}^{(k+1)}$ je váha daného spojení, $\delta_i^{(k+1)}$ je gradient z vrstvy $k+1$ a f' je derivace aktivační funkce.

1.2.3 Konvoluční neuronové sítě

Konvoluční neuronové sítě (známé pod zkratkou CNN z anglického *Convolutional Neural Networks*), jsou speciálním typem dopředných neuronových sítí, které vynikají ve zpracování dat ve tvaru n-rozměrných vektorů. Nejoblíbenější je jejich využití ve zpracování obrazu, který se jim předává jako 3-dimenzionální matice většinou se 3 kanály, kde každý kanál reprezentuje jednu z RGB složek. Hojně se ale používají také ve zpracování zvuku.

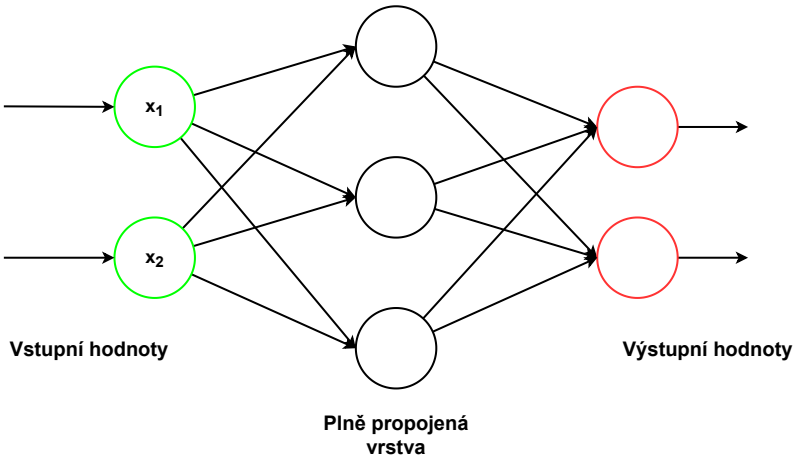
Konvoluční neuronové sítě přijímají zvuková data ve formě 1D vektorů, které zpracovávají sekvenčně a vždy berou několik vzorků zároveň.

Konvoluční neuronové sítě představil ve svém článku Yann LeCun už v roce 1989, kdy se mu povedlo natrénovat model pro klasifikaci ručně psaných číslic [41]. Za uplynulých 30 let udělal výzkum v této oblasti velký pokrok a konvoluční neuronové sítě si našly velké využití v komerční i akademické sféře. Obrovským milníkem byl rok 2012, kdy Alex Krizhevsky a spol. představili architekturu *AlexNet* [40], která porazila všechny dosavadní způsoby klasifikace na rozsáhlé datové sadě ImageNet [19], která obsahuje na 15 milionů obrázků z asi 22 000 tříd. Tato architektura se skládala z 8 vrstev. V dnešní době existují architektury, které obsahují stovky až tisíce skrytých vrstev. První takovou architekturou byla síť ResNet [31]. Ta opět získala 1. místo v soutěži klasifikace na datové sadě ImageNet. Architektura sítě ResNet bude rozebrána na konci této kapitoly.

V této kapitole popíšu jednotlivé části – vrstvy – ze kterých se konvoluční neuronové sítě typicky skládají. Dále uvedu princip operace konvoluce, od které je odvozen název konvolučních neuronových sítí, a na které stojí celé jejich fungování. Popis jednotlivých vrstev je inspirován z [27].

Plně propojená vrstva

Plně propojená vrstva (angl. *fully connected layer*) je základním stavebním kamenem všech dopředných neuronových sítí. Jak už z názvu vyplývá, všechny její neurony jsou spojeny se všemi výstupy z vrstvy předchozí. To sebou nese velkou nevýhodu, a tou je vysoký počet trénovatelných parametrů. V architekturách konvolučních neuronových sítí je většinou umístěna na samotném konci, kde zpracovává příznaky, které byly získany pomocí konvolučních vrstev. Princip této vrstvy je uveden na obrázku 1.9.



Obrázek 1.9: Příklad plně propojené vrstvy, kde jsou všechny neurony plně propojené vrstvy napojeny na všechny neurony vrstvy předešlé. Nevýhodou této vrstvy je velký počet parametrů, které musí být optimalizovány při učení.

Konvoluční vrstva a operace konvoluce

Tato sekce je inspirována knihou Deep Learning [27]. Operace konvoluce je ve své nejobecnější formě operací nad dvěma funkcemi čísel x a w z reálné domény. Mapa příznaků (angl. *feature map*), jak je výstup konvoluce běžně označován, může být vypočítána jako:

$$s(t) = \int x(a)w(t-a)da, \tag{1.20}$$

kde x je funkce nazývaná *vstup*, w je funkce označovaná jako *konvoluční jádro* a $s(t)$ je mapa příznaků.

Konvoluce se může zapisovat pomocí operátoru $*$ následovně:

$$s(t) = (x * w)(t). \tag{1.21}$$

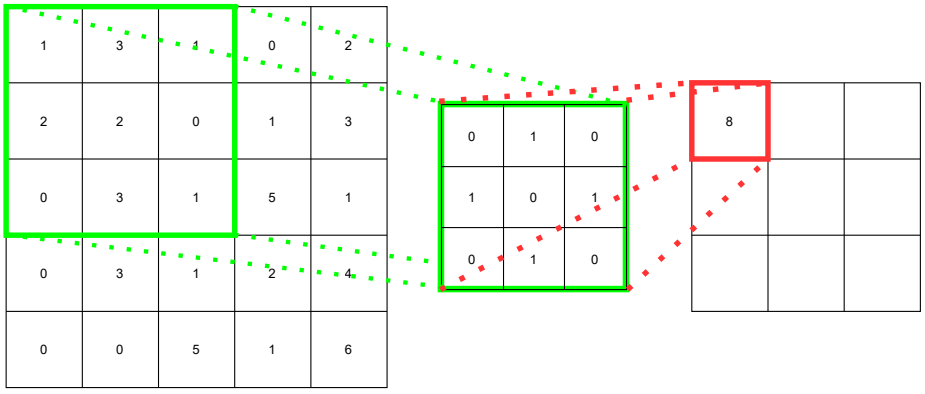
V oboru zpracování obrazu a řeči je zapotřebí pracovat nad množinou diskrétních hodnot. Například pro černobílý obrázek se dá konvoluce zapsat jako:

$$S(i,j) = (I * K)(i,j) = \sum_k \sum_l I(k,l)K(i-k, j-l), \tag{1.22}$$

kde I je černobílý obrázek, K je 2D jádro konvoluce, i a j jsou indexy bodu v I a k, l jsou rozměry jádra.

Plně propojená vrstva je pro zpracování obrazu nevhodná, protože pracuje s absolutní pozicí pixelů a jejich hodnot. To má za následek problémy u zpracování obrázků, které jsou posunuty, otočeny nebo nějakým způsobem deformovány. Plně propojené vrstvy nedokáží reagovat na tyto rozdíly mezi jednotlivými obrázky.

Konvoluční vrstva tento problém elegantně řeší tím, že pracuje i s pixely v okolí právě zpracovávaného pixelu. Toto okolí je tak velké jako konvoluční jádro. Pokud je cílem zpracovávat vetší okolí aktuálního pixelu, stačí zvětšit konvoluční jádro. Jelikož konvoluční vrstva pracuje i s tímto okolím, je imunní vůči mírným transformacím obrázku. Další velkou výhodou je nízký počet parametrů. U plně propojených vrstev je každý neuron napojen na všechny výstupy předchozí vrstvy. To znamená mnoho trénovatelných parametrů pro tato spojení. U konvoluční vrstvy jsou trénovány pouze parametry, které přísluší danému konvolučnímu jádru.



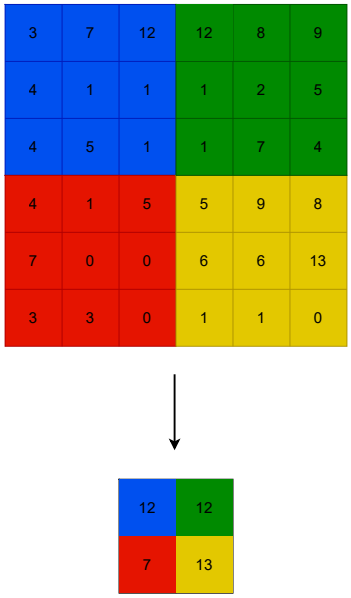
Obrázek 1.10: Na tomto obrázku můžete vidět příklad konvoluční vrstvy. Hodnota v nové mapě příznaků se počítá pomocí okolí daného pixelu na základě konvolučního jádra.

Aktivační vrstva

Aktivační vrstva počítá aktivace pro výstup neuronů z předchozí konvoluční vrstvy podle zvolené aktivační funkce. V dnešní době se pro konvoluční vrstvy nejvíce používá aktivační funkce ReLU (popisovaná v této kapitole v sekci 1.2.1) a její aktualizované verze *Leaky ReLU* a *Parametric ReLU* [27].

Sdružovací vrstva

Vrstvou, která je zahrnuta v téměř každé konvoluční neuronové síti, je vrstva sdružovací, spíše známá pod anglickým názvem *pooling layer*. Hlavním úkolem těchto vrstev je podvzorkování (tzv. *downsampling*) map příznaků, které byly získány pomocí konvolučních vrstev. Sdružovací vrstva je tedy umístěna vždy až za vrstvou konvoluční a vrstvou aktivační. Podvzorkování mapy příznaků je realizováno postupným posunem po mapě s krokem (angl. *stride*) o velikosti alespoň 2. V každém kroku vezme všechny hodnoty s určitého okolí a vypočítá z nich pouze 1 hodnotu s určitými vlastnostmi. Sdružovací vrstvy mají několik typů, kde 2 hlavní z nich jsou *max-pooling* a *average-pooling*. První zmíněný vybírá z okolí maximální hodnotu a druhý počítá průměr ze všech hodnot v tomto okolí. Princip max-pooling vrstvy můžete vidět na obrázku 1.11. Sdružovací vrstvy zároveň přidávají drobnou invarianci proti posunutí ve vstupních datech.



Obrázek 1.11: Příklad použití max-pooling vrstvy. Tato vrstva vybírá maximální hodnotu z dané oblasti mapy příznaků.

Normalizace dávek

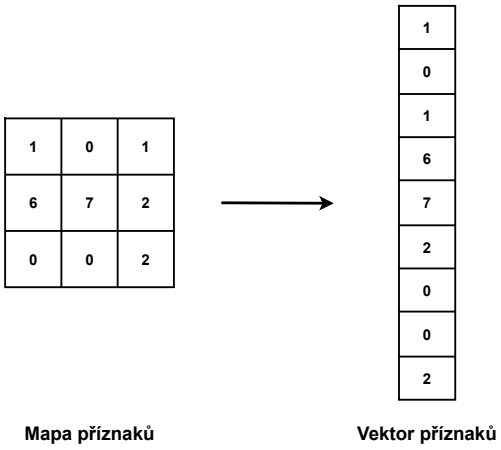
Tato vrstva je nejvíce známá pod svých anglickým názvem *batch normalization* [34]. U konvolučních neuronových sítí s velkým počtem vrstev je občas těžké odrazit se na začátku trénování. Jedním z důvodů může být, že rozložení hodnot vstupů do vrstev, které jsou hluboce zanořeny v síti, se mění s aktualizací parametrů po každé *mini-batch* dávce. Tato aktualizace parametrů dané vrstvy předpokládá, že předchozí vrstva dává hodnoty pořád ze stejného rozložení. To ale není pravda, protože při učení neuronové sítě se aktualizují parametry všech vrstev zároveň, a tak se s velkou pravděpodobností změní i rozložení hodnot vycházejících z vrstvy, která předchází naší právě aktualizované vrstvě. Tento problém je také označován jako vniřní kovariační posun (angl. *internal covariate shift*).

Vrstva normalizace dávek pomáhá tento problém odstranit tím, že normalizuje výstup aktivační vrstvy tak, aby hodnoty měly střední hodnotu 0 a směrodatnou odchylku 1 – tedy aby měly Gaussovské rozložení. Tato normalizace nám zajistí, že výstupy z předchozích

vrstev budou vždy ze stejného rozložení, což může razantně urychlit a stabilizovat trénování neuronové sítě.

Flattening

Flattening neboli zploštění map příznaků není samostatná vrstva konvolučních neuronových sítí, je to ale velmi důležitá část, ve které se převádí mapy příznaků na 1D vektor příznaků, který je poté předán na vstup plně propojené vrstvy, která tento vektor dále zpracovává. Za zploštěním může následovat celá dopředná neuronová síť sestavená z plně propojených vrstev, které nejdříve zpracují vektor příznaků, a nakonec z nich vytvoří výstup sítě (např. klasifikaci vstupních dat do tříd).



Obrázek 1.12: Příklad operace flattening. Tato operace se používá v konvolučních neuronových sítích pro převod mapy příznaků na 1D vektor, který může být vložen na vstup plně propojených sítí.

Předposlední vrstva

Předposlední vrstva (angl. *penultimate layer*) je vrstva, která se nachází před výstupní vrstvou. U konvolučních neuronových sítí se většinou jedná o plně propojenou vrstvu, která na svém výstupu vrací N-dimenzionální vektor. Tento vektor reprezentuje příznaky, které byly ze vstupních dat vytaženy předcházející částí neuronové sítě. Tato vrstva má velké využití při použití již předučených modelů neuronových sítí, kdy z předučeného modelu je extrahován tento vektor příznaků, který je následně využit dle aktuální potřeby.

1.3 Vybrané architektury neuronových sítí

V praxi se v posledních několika letech začaly neuronové sítě využívat opravdu hodně. Konvoluční neuronové sítě našly největší využití ve zpracování obrazu i zvuku. Na začátku se používaly hlavně k extrakci informací z obrazových dat, v poslední době však našly využití i v úkolech jako je generování fotek obličejů nebo i videí. Ve zvukové doméně se konvoluční neuronové sítě používají hlavně pro extrakci příznaků z řeči. Tyto příznaky jsou poté dále zpracovávány například pomocí tzv. transformerů. V této části v krátkosti popíšu architektury neuronových sítí, které mám v plánu využít i ve své práci. Jedná se o architekturu *ResNet* [31],

která slouží pro extrakci příznaků. Dále bude popsána architektura *GAN*, která naopak slouží ke generování umělých dat, a která přinesla nový způsob učení.

1.3.1 ResNet

Architektura ResNet [31] byla první, která dokázala plně využít sílu mnoha zanořených konvolučních vrstev na náročné úkoly. Předchozí architektury nedokázaly hluboce zanořené konvoluční vrstvy efektivně trénovat. ResNet to dokázal pomocí vytvoření tzv. ResNet bloků a reziduálních spojení. ResNet blok je kaskáda několika vrstev – konvoluční, aktivační, normalizační – v různém počtu a kombinaci. Ze vstupu každého ResNet bloku vede reziduální spojení, které přeskakuje zmiňovanou kaskádu vrstev, a přičítá vstupní hodnoty k těm výstupním. Autoři práce toto spojení nazvali *mapování identity*. Díky tomuto kroku se trénování modelů s velkým počtem vrstev stává opět stabilním a umožňuje extrakci detailních informací z obrázků. S použitím této architektury dokázali autoři natrénovat až 152-vrstvou konvoluční neuronovou síť, která dokázala dosáhnout state-of-the-art výsledků v soutěži *ImageNet* [19].

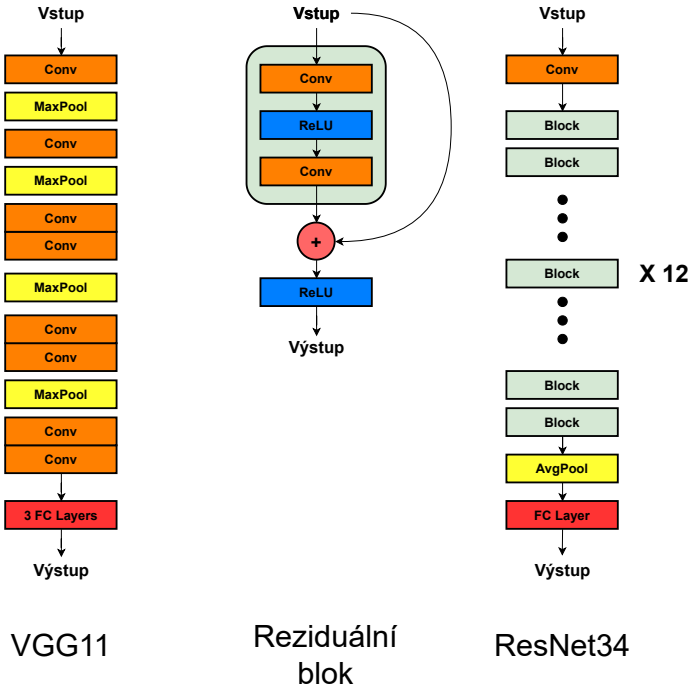
Nejlepší vysvětlení této architektury bude pomocí obrázku. Na obrázku 1.13 můžete vidět, jak taková síť ResNet vypadá. Na začátku má jako každá jiná konvoluční neuronová síť vstupní vrstvu, která ale hned následuje kaskádou reziduálních bloků. V sítích s architekturou ResNet se uprostřed sítě moc nevyskytují sdružovací vrstvy pro redukci dimenzionality dat, protože se tato redukce provádí automaticky díky vysokému počtu konvolučních vrstev.

1.3.2 GAN

Do roku 2014 měly v rámci hlubokého učení úspěch hlavně modely diskriminativní. Tyto modely se například využívaly v problematice klasifikace, ve které je hlavním úkolem mapování vysoce dimenzionálních vstupů na třídy objektů. Diskriminativní model má za úkol naučit se vypočítat pravděpodobnost třídy y, pokud dostane na vstupu vzorek x. Naopak, generativní model se učí funkci rozložení pravděpodobnosti, která říká, jak mohla být trénovací data generována. Pomocí této funkce poté může například klasifikovat data do tříd. Dá se říct, že diskriminativní model se učí přímo funkci rozložení pravděpodobnosti $p(y|x)$. Generativní model se učí funkci společné pravděpodobnosti $p(x, y)$, ze které poté dopočítává podmíněnou pravděpodobnost $p(y|x)$.

V roce 2014 představili Goodfellow a spol. [28] práci s názvem *Generative Adversarial Nets*. Přeložit do češtiny se tento název dá jako *Generativní adverzní sítě* nebo také *Generativní kompetitivní sítě*. Při učení těchto GAN sítí se totiž netrénuje pouze jeden samostatný model ale modely dva. První z modelů, *generátor*, je generativní model, jehož úkolem je generovat data. Druhý model je diskriminativní a také je podle toho pojmenován – *diskriminátor*. Diskriminátor obdrží generovaná data od generátoru a jeho úkolem je rozpoznat, v porovnání s referenčními daty, která data jsou falešná, a která pravá. Cílem generátoru je generovat data tak, aby je diskriminátor nebyl schopný rozpoznat od originálů. Diskriminátor by naopak měl rozpoznat všechny uměle generované vzorky.

Následující část vychází z knihy *Deep Learning* [27] od stejného autora jako architektura GAN. Cílem generativních kompetitivních sítí je naučit se pravděpodobnostní rozložení p_g, ze kterého bude generátor generovat nové vzorky dat. U základního konceptu trénování modelu s architekturou GAN musí být zadefinovány pravděpodobnostní rozložení $p_z(\boldsymbol{z})$, které bude sloužit pro generování vstupů $\boldsymbol{z}$ do generátoru. Generátor poté generuje výstupní data na základě vstupu $\boldsymbol{z}$. Jinými slovy, mapuje vstupní $\boldsymbol{z}$ na výstupní vzorek z rozložení p_g. Funkce generátoru může být zapsána jako $G(\boldsymbol{z}, \boldsymbol{\theta}_g)$, kde G je model neuronové



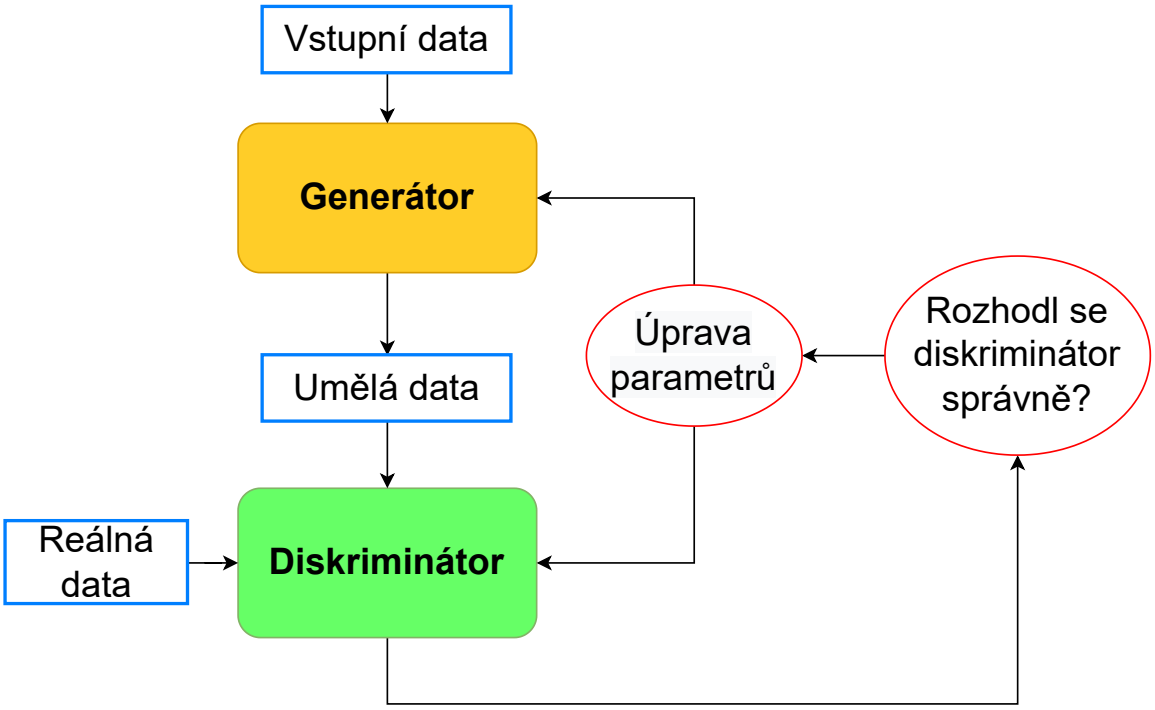
Obrázek 1.13: Porovnání architektury *ResNet34* s architekturou *VGG11*. Architektura VGG se skládá z kaskády konvolučních vrstev, kde data proudí vždy pouze z výstupu předchozí vrstvy na vstup vrstvy aktuální. Po každé konvoluční vrstvě následuje ještě aktivační vrstva ReLU, což v obrázku není vyobrazeno. Resnet34 se naopak skládá z tzv. *reziduálních bloků*. Tato architektura vypadá podobně jako VGG11. Velkým rozdílem je ale aplikace tzv. *mapování identity*, které dovádí vstup do reziduálního bloku na jeho konec, kde jej přičítá k výstupu tohoto bloku. Toto mapování stabilizuje trénování modelu a umožňuje využití velmi hlubokých neuronových sítí. Dalším rozdílem jsou tři plně propojené vrstvy na konci architektury VGG11, kdežto ResNet34 má pouze jednu plně propojenou vrstvu. Tento fakt výrazně redukuje počet trénovatelných parametrů v modelu, protože s počtem plně propojených vrstev roste velmi rapidně počet parametrů, které se v nich musí trénovat.

sítě s trénovatelnými parametry $\boldsymbol{\theta}_g$. Druhý model, diskriminátor $D(\boldsymbol{x}, \boldsymbol{\theta}_d)$, je také model neuronové sítě, jehož výstupem je pravděpodobnost $D(\boldsymbol{x})$, která udává, jestli $\boldsymbol{x}$ je jeden ze vzorků z p_g nebo je to originální vzorek dat. Právě tuto pravděpodobnost správného rozpoznání má diskriminátor maximalizovat při trénování. Generátor je naopak trénován tak, aby minimalizovat pravděpodobnost úspěšného rozhodnutí diskriminátoru $log(1 - D(G(\boldsymbol{z})))$. Pak se dá říct, že generátor a diskriminátor hrají hru Mini-Max:

$$\min_G \max_D V(D, G) = E_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}(logD(\boldsymbol{x})) + E_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}(log(1 - D(G(\boldsymbol{z})))). \tag{1.23}$$

Ukázalo se však, že nevýhodou předchozí rovnice je, že se diskriminátor naučí rozpoznávat falešná data příliš brzy, ještě než bude generátor dostatečně kvalitní, a tím pádem bude generátor saturovat vlivem ztráty gradientů. Proto byla zavedena tzv. *non-saturating loss*, která na místo toho, aby generátor minimalizoval pravděpodobnost rozpoznání jeho vzorku diskriminátorem, bude maximalizovat fakt, že dokáže diskriminátor oklamat – $log(D(G(\boldsymbol{z})))$.

Tato změna zajistí, že na začátku trénování, kdy není generátor dostatečně natrénovaný, budou dostupné pořád kvalitní hodnoty gradientů.



Obrázek 1.14: Tento obrázek v jednoduchosti popisuje princip trénování modelů s architekturou GAN. Nejdříve jsou vstupní data zpracována generátorem, který na jejich základě generuje data umělá. Ty jsou spolu s reálnými daty předány na vstup diskriminátoru, a ten rozhoduje, která z nich jsou pravá, a která jsou falešná. Po tomto rozhodnutí se vypočítá chybová funkce generátoru i diskriminátoru a jsou upraveny parametry obou modelů.

Kapitola 2

Datové sady

Pro správné naučení modelu generovat fotku obličeje ze zvukové nahrávky hlasu, je zapotřebí mít kvalitní trénovací data. Potřebný dataset musí obsahovat jak hlasové nahrávky, tak fotky obličejů stejných identit, přičemž oba druhy dat musí dosahovat určité kvality – nezašuměné nahrávky lidské řeči a fotky obličejů, na kterých jde jasně rozeznat identita daného člověka. Dataset, který by obsahoval všechna taková data bohužel neexistuje. Jsou k dispozici však datasety obsahující buď fotky obličejů, anebo zvukové nahrávky hlasu, a to dokonce s překrývajícími se množinami identit. Databáze zvukových nahrávek hlasu se nazývá VoxCeleb [46]. Dataset s fotkami obličejů, které korespondují s databází VoxCeleb se nazývá VGGFace [50]. Posledním z existujících datasetů, které jsem mohl pro svou práci využít, je AVSpeech, představený v článku pojednávajícím o separaci řeči [23]. Tato databáze se skládá z množiny anotací pro videa z platformy YouTube[1], ze kterých je možné stáhnout audio nahrávku hlasu a fotku obličeje člověka, který na nahrávce mluví.

Každý z těchto datasetů bude rozebrán níže v této kapitole. Dále bude v této kapitole uvedeno, jaké úpravy jsem s daty musel provést, abych je mohl předložit mému modelu pro trénování.

2.1 VGGFace

Datová sada VGGFace je rozdělena na 2 verze – VGGFace1 [50] a VGGFace2 [9]. První verze vznikla v roce 2015, protože její autoři potřebovali dataset pro trénování modelu rozpoznání obličejů postaveného na konvoluční neuronové síti [50]. Druhá verze datasetu byla sesbírána v roce 2018. S rozmachem konvolučních neuronových sítí přišla i potřeba opravdu rozsáhlých datových sad. Jelikož neexistovala žádná datová sada s obrázky obličejů takových rozměrů jako VGGFace2, rozhodli se její autoři vytvořit vlastní dataset, který by byl použitelný pro nejrůznější aplikace v rámci práce s fotkami obličejů – rozpoznání a identifikace, určení pózy obličeje nebo určení věku člověka.

VGGFace1 obsahuje $2,6$ milionů obrázků pro 2622 identit [50]. Druhá verze tohoto datasetu, VGGFace2, se skládá z $3,31$ milionů obrázků pro 9131 identit [9].

[1] https://www.youtube.com/

Sběr dat

Obrázky z obou verzí této datové sady byly sesbírány podobným způsobem. Jelikož VGGFace2 vznikal později, používali při jeho tvoření autoři pokročilejší techniky. V této sekci je dále popsáno, jakými fázemi musela data projít, než se stala součástí tohoto datasetu.

Sestavení seznamu kandidátních identit. Prvním krokem muselo být vytvoření seznamu identit, jejichž fotky budou stahovány. Plánem bylo soustředit se na celebrity a veřejné osoby, aby byl dostupný velký počet jejich snímků online. Tento krok zahrnoval i lidskou práci. Na začátku se vytvořil početný seznam identit – pro VGGFace1 5000, pro VGGFace2 dokonce 500000. Pro každou z těchto identit bylo vyhledáno 100 obrázků, které potom prochází lidští anotátoři, a vybírají identity, pro které minimálně 90% obrázků patří dané identitě. Ve VGGFace1 zůstalo po tomto filtrování 3250 identit, které byly ještě zredukovány, protože se některé z nich vyskytovaly i v datesetech LFW (Labeled Faces in the Wild) [33] nebo YTF (Youtube Faces Database) [60]. Zbylo tak pouze výsledných 2 622 identit. Pro VGGFace2 byl tento poměr ještě horší, po stejném filtrování zůstalo v tomto datasetu pouze výsledných 9131 z půl milionu identit.

Získání obrázků. V druhé fázi došlo k samotnému stahování obrázku. Pro každou identitu bylo staženo 1000 obrázků z vyhledávání obrázků na Google[2]. Dále bylo pro každou identitu staženo 200 obrázků s příponou *sideway* a 200 obrázků s příponou *very young* pro zajištění, že v datové sadě budou obličeje s odlišnou pózou a věkem.

Detekce obličeje a filtrování pomocí klasifikace. Tato fáze probíhá pouze při sběru dat do datasetu VGGFace2. Obličeje jsou detekovány pomocí konvoluční neuronové sítě MTCNN [61]. Poté, co proběhne detekce obličejů, dochází k jejich rozpoznání a klasifikaci. Pokud obličej není klasifikován pro správnou identitu, je vyřazen z datasetu.

Odstranění duplicit. Stejně jako u datasetu VoxCeleb 2.2 se i v datech pro VGGFace1 i VGGFace2 hledají duplicity na základě podobnosti. Podobnost se zjišťuje shlukováním pomocí tzv. VLAD deskriptorů (Vector of LocallyAggregated Descriptors). Pokud se obrázky liší pouze např. ve vyvážení barev nebo JPEG artefakty jsou zařazeny do jednoho shluku. Z každého shluku je poté vybrán pouze jeden obrázek.

2.2 VoxCeleb

VoxCeleb je audio-vizuální datová sada, která obsahuje krátké videoklipy s lidskou řečí. Tyto klipy byly extrahovány z rozhovorů slavných osobností, které byly nahrány na platformu YouTube. Tento dataset vznikl v roce 2017 [46] a byl prvním ve své kategorii. Ne, že by neexistovaly podobné datasety již dříve, tento ale vznikl plně automatizovaně, bez nutnosti lidské anotace.

Jeho autoři potřebovali dostatek dat pro trénování modelu pro identifikaci a verifikaci mluvčího v reálném prostředí. Reálným prostředím jsou zde myšleny zvukové nahrávky, které obsahují nejen hlas mluvčího, ale také nějaký šum – zvuky na pozadí, lidský smích, hudba, nebo třeba nejsou dostatečně kvalitní, kvůli hodně vzdálenému mikrofonu. Takový úkol vyžaduje velké množství dat, která odpovídají této podmínce, a většina vyhovujících datasetů z této oblasti byla nedostupná pro akademické účely. Jediným volně dostupným datasetem byl tzv. *The Speakers in the Wild (SITW) Speaker Recognition Database* [43], který ale obsahoval data jen od 299 mluvčích, a byl manuálně anotovaný, takže jeho škálování bylo velmi časově a finančně náročné. Autoři datové sady VoxCeleb se tedy rozhodli

[2] https://www.google.cz/imghp

vytvořit vlastní automatizovanou pipeline pro sběr audio-vizuálních dat, která bude lehce škálovatelná, a nebude u pořizování potřeba lidského zásahu ve formě anotace.

VoxCeleb se dělí na 2 verze. První z nich je VoxCeleb1 [46], která vznikla právě v roce 2017. Tato verze obsahuje přes 100 000 nahrávek od 1 251 identit. Jelikož měl tento dataset velký úspěch, rozhodli se autoři v roce 2018 vytvořit druhou verzi VoxCeleb2 [13]. Ta obsahuje přes milion nahrávek od více než 6 000 identit. Velkou výhodou je, že se žádné nahrávky ani identity nepřekrývají s verzí VoxCeleb1, a tak jde oba datasety spojit do jedné obrovské datové sady. Autoři však uvádějí, že rozložení národností v datové sadě neodpovídá rozložení v celkové lidské populaci.

Sběr dat

Jak již bylo zmíněno, sběr audio-vizuálních dat pro tuto datovou sadu byl plně automatizován. Toho autoři docílili pomocí pipeline, která byla rozdělena do 5 hlavních fází popsaných níže.

Výběr kandidátních identit. Nejdříve bylo potřeba vybrat určitý počet identit (angl. *Persons of Interest, POIs*), které se budou v datasetu vyskytovat. Tyto identity byly získány z datasetu VGGFace 2.1. První verze VoxCeleb1 vychází z VGGFace1 a VoxCeleb2 vychází z datasetu VGGFace2. Tento seznam identit obsahuje asi polovinu žen a polovinu mužů.

Stažení videí z YouTube. Jakmile byl hotový seznam daných identit, bylo potřeba stáhnout všechna videa pro tyto identity z YouTube. Pro všechny identity bylo vyhledáno nejlepších 50 až 100 videí pomocí YouTube vyhledávání. Jediným filtrem v této fázi bylo jméno dané identity s příponou *interview*, která měla zvýšit pravděpodobnost, že na videu bude daná osoba mluvit.

Detekce a sledování obličejů. V každém staženém videu se v každém snímku detekuje obličej pomocí HOG (Histogram orientovaných gradientů) detektoru [38] v VoxCeleb1. Ve VoxCeleb2 se detekce provádí pomocí detektoru založeném na konvoluční neuronové síti natrénované na datasetu VGGFace2. Detekce obličejů z každého snímku videa jsou sloučeny do stop, které určují pohyb obličeje mezi jednotlivými snímky. Systém pro tvoření stop pohybu obličejů autoři převzali z článků [24][14] a optimalizovali, aby lépe zvládal zpracovat vetší množství dat. Ve VoxCeleb1 se pomocí těchto stop verifikuje, že na nahrávce je opravdu námi požadovaná identita. Ve VoxCeleb2 do této fáze připadá ještě rozpoznání obličeje, zda se jedná o námi hledanou identitu. Toto rozpoznání je prováděno pomocí ResNet modelu [31] natrénovaného také na VGGFace2 datasetu.

Ověření mluvčího. V této fázi musí systém ověřit, zda na nahrávce mluví opravdu ta osoba, která je na ní také vyobrazená. Toto ověření má za úkol konvoluční neuronová síť SyncNet [15][16], která porovnává pohyb úst na videu se zvukovou nahrávkou.

Odstranění duplicit a přidání metadat. Jelikož na YouTube může nahrávat videa kdokoliv, je celkem běžné, že je stejné video nahráno vícekrát. Pro nalezení duplicit je z každého segmentu řeči extrahován embedding reprezentován 1024-dimenzionálním vektorem, a ze všech dvojic těchto embeddingů je poté vypočítána euklidovská vzdálenost. Pokud je mezi nějakou dvojicí menší threshold, než předem určená hodnota, je považována za duplicitu. Dále jsou ke každému videu přidána metadata, která obsahují informace jako např. pohlaví nebo národnosti mluvčích.

Hlavně u datasetu VoxCeleb1 se stávalo, že daná nahrávka neprošla přes pipeline i kvůli nedostatkům jednotlivých částí. Např. HOG detektor obličejů nedokázal moc dobře detekovat špatně osvícené nebo natočené obličeje, a tak velké množství dat nebylo zpracováno [13].

2.3 AVSpeech

AVSpeech (Large-scale Audio-Visual Speech Dataset) [23] je velká audio-vizuální datová sada, která obsahuje videoklipy mluvících lidí bez zasahujících zvuků z okolí. Jednotlivé segmenty řeči jsou 3 až 10 sekund dlouhé a na každém mluví pouze 1 osoba, která je zároveň vyobrazená na videu. Dataset vznikl jako součást vývoje systému pro segmentaci řeči [23]. Takový systém dokáže separovat řeč více řečníků, mluvících zároveň, na záznamy řeči, připadající pouze jednomu řečníkovi. Dohromady obsahuje dataset přibližně 4 700 hodin video segmentů pocházejících z 290 000 YouTube videí. Zdrojová videa se skládají hlavně z přednášek a jiných naučných videí, kde mluví co nejčastěji pouze jedna osoba. Autoři tohoto datasetu neposkytují data ke stažení, ale jsou k dispozici pouze metadata. Celkově totiž dataset obsahuje několik TB dat, takže by ani nebylo vhodné distribuovat ho vcelku. Metadata obsahují ID daného YouTube videa, start a konec segmentu v sekundách a X, Y souřadnice obličeje, kde se nachází na začátku segmentu.

Sběr dat

Stejně jako u datové sady VoxCeleb, i AVSpeech byl vytvořen automaticky, bez nutnosti lidských zásahů. V tomto případě se však pipeline pro vytvoření datasetu sestává pouze ze 2 hlavních kroků

Sledování mluvčího. Nejdříve se nad každým videem provede sledování mluvčího (angl. *speaker tracking*), aby se mohly posléze detekovat segmenty videa, kde daný člověk aktivně mluví. Metodu pro sledování mluvčího autoři převzali z článku od Hoovera a spol. z roku 2017 [32]. Pokud byl obličej na daném snímku špatně osvícený, hodně natočený nebo rozmazaný, byl tento snímek zahozen. Pokud bylo takových snímků více než 15%, byl zahozen celý segment videa.

Výběr čistých dat. Autoři datasetu pro svůj úkol potřebovali pouze čistá data, která neobsahují řeč znečištěnou nějakými jinými zvuky. Výběr čisté řeči probíhal pomocí výpočtu SNR (*Signal to noise ratio*) daného segmentu řeči s odšuměným segmentem. Odšumění bylo provedeno pomocí před-učeného modelu neuronové sítě, kterou vytvořili samotní autoři [23]. Neuronová síť pro odšumění signálu řeči byla natrénována na datech ze stránky LibriVox [4], kde jsou dostupné nahrávky audio knih, které představují ideální data čisté řeči.

Kapitola 3

Návrh modelu pro odhad obličeje z řečového signálu

Cílem této diplomové práce bylo navrhnout a hlavně implementovat model, který dokáže na základě řečového signálu generovat obrázek obličeje, který bude mít stejné rysy, jako obličej řečníka mluvícího na nahrávce řeči. Tento model se bude skládat ze dvou hlavních částí – enkodéru a dekodéru. Enkodér bude mít za úkol správně extrahovat příznaky, které budou popisovat hlavní rysy obličeje, z řeči. Tento enkodér jsem založil na konvoluční neuronové síti ResNet. Pro druhou část, dekodér, jsem využil již hotový předtrénovaný model neuronové sítě *StyleGAN*. Jedná se o neuronovou síť s architekturou GAN, která dosáhla state-of-the-art výsledků na poli generování obrázků obličejů. Stejně důležitou částí této práce je chybová funkce, kterou jsem se snažil navrhnout na míru mým modelům. V první části této kapitoly projdu také současné přístupy k problematice odhadu obrázku obličeje podle hlasu řečníka.

3.1 Současné přístupy

Problematice odhadu obličeje z řečového signálu se do dnešního dne moc pozornosti nevěnovalo. Jeden z prvních pokusů byl publikován v roce 2019 v práci nazvané *WAV2PIX* [22]. Největšího úspěchu se však podařilo dosáhnout v práci *Speech2Text* [48] také z roku 2019, která byla publikována asi o 2 měsíce později. Autoři této práce dosáhli velmi dobrých výsledků v odhadu rysů obličeje podle hlasu mluvčího. Narozdíl od prvního článku, jejich systém dokázal generalizovat i na data, která neviděl při trénování, čehož se v práci *WAV2PIX* nepodařilo dosáhnout. Princip fungování těchto dvou prací popíšu v první části této kapitoly.

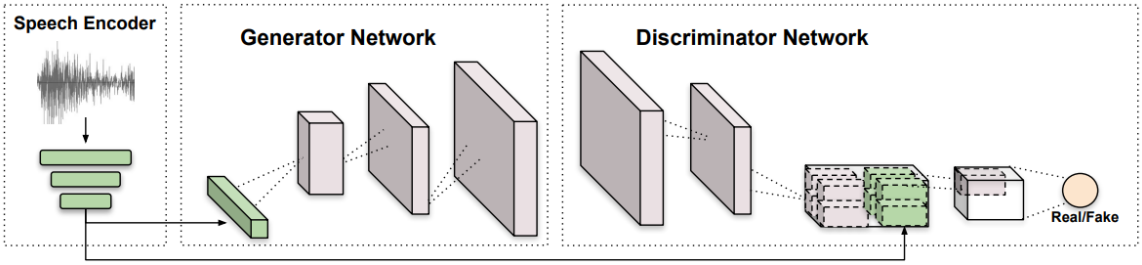
3.1.1 Wav2Pix

Autoři systému *WAV2PIX* implementovali model založený na architektuře GAN, který na vstupu bere zvukový signál obsahující řeč a na výstupu generuje odhadnutý obrázek obličeje řečníka. Na první pohled vypadají výsledky v článku dobře. Problém ale je, že tento model nedokáže generalizovat mimo trénovací množinu, a všechny výsledky, které jsou prezentovány, byly dosaženy na identitách viděných při trénování.

Ve *WAV2PIX* se autoři vydali cestou, kde nepoužívají ručně extrahované příznaky ze signálu řeči – např. spektrogram nebo MFCC – ale pro extrakci příznaků navrhnuli enkodér, který se učí společně s celou GAN architekturou. Z tohoto enkodéru jde potom extrahovaný embedding na vstup generátoru GAN architektury. Z generátoru na výstupu

vypadne odhadnutý obrázek obličeje, který je v diskriminátoru porovnáván s referenčním obrázkem daného řečníka.

Více o této práci si můžete přečíst ve článku [22]. Zde se jí už nebudu zabývat, jelikož má vlastní práce z ní téměř nečerpá a nevychází.



Obrázek 3.1: Architektura systému *WAV2PIX*. Na úplném začátku se nachází hlasový enkodér, který extrahuje vektor příznaků z nahrávky řeči. Tento vektor je poté vložen na vstup generátoru, který má na starosti vytvoření obrázku podle informací obsažených ve vstupním vektoru. Výsledný obrázek poté vstupuje do diskriminátoru, který rozhoduje o jeho kvalitě a pravosti. Obrázek je převzat z článku WAV2PIX [22].

3.1.2 Speech2Face

V již zmiňované práci *Speech2Face* [48] chtěli její autoři vyzkoušet, jak moc ovlivňuje vzhled osoby její hlas. Z jiného úhlu pohledu, zda je možné z krátké nahrávky řeči odhadnout obličej v tzv. normalizovaném tvaru – zachycený zepředu, rovnoměrně osvětlený a s neutrálním výrazem. V článku autoři kladou velký důraz na to, že jejich cílem, stejně jako u mé práce, není vytvořit systém, který na výstupu dá naprosto přesnou kopii obrázku obličeje mluvčího, ale vytvoří obrázek obličeje, který obsahuje stejné rysy, jako obličej daného mluvčího z nahrávky. Mezi tyto rysy patří např. tvar lícních kostí, plnost rtů, velikost úst nebo velikost nosu.

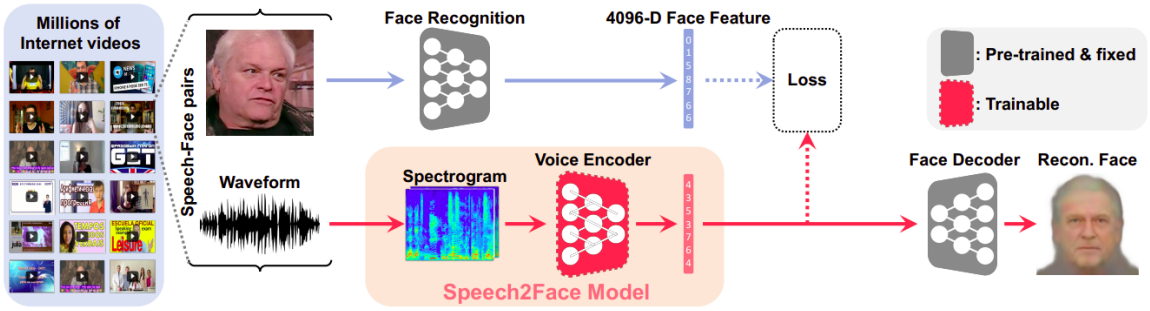
Výše zmíněného cíle se autorům podařilo dosáhnout pomocí modelu neuronové sítě, která převádí komplexní spektrogram, vytvořený z nahrávky řeči, na reprezentaci korespondujícího obličeje v podobě 4096-dimenzionálního vektoru. Tento vektor je poté předán na vstup předtrénovanému generátoru obličejů, který jej převede na zmiňovaný obrázek obličeje.

Trénování celého systému se skládalo z několika kroků. Nejdříve byl natrénován dekodér obličejů z práce autorů Cole a spol. [17], který na svém vstupu bere 4096-dimenzionální vektor reprezentující embedding obličeje získaný z předtrénované neuronové sítě určené pro rozpoznávání obličejů [50] – konkrétně z její předposlední vrstvy – a na výstupu vrací obrázek obličeje v normalizovaném tvaru. Tato síť byla natrénována na datasetu VGGFace2, a v této práci již její učení neprobíhalo. Byla zde využita jednak pro získávání vstupů pro dekodér obličejů, ale také pro trénování enkodéru hlasových nahrávek, který bude popsán dále.

Nejdůležitější součástí tohoto systému je určitě hlasový enkodér. Tím je konvoluční neuronová síť typu VGG s 9 konvolučními vrstvami, která se při trénování učí namapovat extrahované embeddingy hlasu na embeddingy získané z enkodérů obličejů zmiňovaného v předchozím odstavci. Důvod tohoto mapování je ten, že jakmile dokáže enkodér z hlasu extrahovat korespondující embeddingy k těm obličejovým, může je vkládat do dekodérů

obličejů a získávat důvěryhodné obrázky mluvčího z hlasové nahrávky. Trénování probíhalo na datasetu AVSpeech, o kterém si můžete více přečíst v kapitole 2.3.

Oproti předchozímu modelu *WAV2PIX* se autorům podařilo vytvořit systém pro odhad obličeje, který dokáže z nahrávky hlasu vyextrahovat relevantní informace o vzhledu mluvčího. Tento regresní model, který se pouze snaží namapovat enkodér hlasu na enkodér obrázků obličejů, dosahuje velmi uspokojivých výsledků i pro mluvčí, které neviděl při trénování, a dokáže tak generalizovat mimo trénovací datovou sadu.



Obrázek 3.2: Architektura systému *Speech2Face*. Obrázek je převzat z práce [48].

3.2 Návrh vlastního systému

Při návrhu mého systému jsem se nenechal inspirovat ani jednou z prací zmíněných na začátku této kapitoly. Na těchto přístupech se mi nelíbilo, že je v obou případech trénován jak enkodér hlasu, tak dekodér obličejů. V případě práce *Speech2Face* se sice dekodér trénuje zvlášť, i tak je to ale výpočetně náročný úkon. Pro trénování kvalitního generátoru obrázků obličejů je zapotřebí obrovský výpočetní výkon a velká datová sada, aby se model nepřetrénoval na trénovací data. Proto si myslím, že je v mém případě lepší, když použiji již předtrénovaný model, který je schopný generovat kvalitní snímky obličejů na základě vstupních informací. Jak již bylo zmíněno v úvodu této kapitoly, pro svou práci jsem si jako dekodér obličejů vybral *StyleGAN* [36], který bude více popsán v následujících částech této práce. Tomuto generátoru jsou v průběhu inference vkládány na vstup tzv. styly, které určují, jak bude výsledný obličej vypadat. Styly jsou reprezentovány pomocí 512-dimenzionálních vektorů, které budou získány z mého enkodéru hlasových nahrávek. Tento enkodér bude konvoluční neuronová síť založená na architektuře ResNet, ze které bude v průběhu inference extrahována pyramida map příznaků. Z těchto map příznaků budou následně tvořeny styly pro *StyleGAN*. Každý extrahovaný vektor příznaků, než se stane stylem, projde mapovací plně konvoluční neuronovou sítí, která má za úkol zmenšit jeho dimenzionalitu postupnou redukcí pomocí konvolučních vrstev s krokem (*stride*) o velikosti 2.

V dalších částech této kapitoly popíšu jednotlivé části mého systému. Nejdůležitější z nich bude enkodér hlasových nahrávek, který jsem musel navrhnout tak, aby z něj šly během inference extrahovat jednotlivé vektory stylů o různých dimenzionalitách. Po extrakci budou tyto vektory mapovány pomocí jednoduchých konvolučních sítí na vstup *StyleGANu*, který na jejich základě vygeneruje výsledný obrázek obličeje.

3.2.1 Enkodér

O možnosti napojení enkodéru přímo na vstup *StyleGANu* jsem se dozvěděl z práce *Encoding in Style* od Richardsona a spol. [52]. V této práci autoři ukázaly, že pomocí jejich enkodéru obrázků obličejů, který je přímo napojen na *StyleGAN*, lze získat takové styly, pomocí kterých lze dostat z generátoru téměř totožný obličej s tím vstupním. Záměrně zde zdůrazňuji, že enkodér je připojen přímo na generátor. V průběhu roku 2020 vzniklo několik prací, které ukázali, že *StyleGAN* pracuje nad latentním prostorem W, který poskytuje schopnost kontrolovat, jaký výstup se dá z generátoru dostat [35][56][29][59]. Každá z těchto prací však převádí vstupní obrázek na vektor $\boldsymbol{w} \in W$, nad tímto vektorem provádí dané transformace a poté jej vkládá na vstup generátoru, který na jeho základě vygeneruje obrázek. Ukázalo se však, že převod vstupního obrázku pouze na 1 vektor $\boldsymbol{w} \in W$ není dostačující na kvalitní rekonstrukci obrazu. Další práce se tedy pokoušely převádět vstupní obraz na 18 různých vektorů $\boldsymbol{w}$, jeden pro každou ze vstupních vrstev *StyleGANu* [8][7][63]. Tyto vektory byly vytvořeny v tzv. rozšířeném latentním prostoru W^+. Nevýhodou však bylo, že obrázek byl nejdříve převeden na vektory z W^+ a až nad těmito vektory byly prováděny optimalizace a úpravy. Tento přístup byl velmi neefektivní pro využití při trénování neuronových sítí, protože úpravy pro 1 obrázek mohly vzít až několik minut času. Efektivního a přesného převodu do W^+ se povedlo dosáhnout právě až v práci *pSp* [52].

Právě díky tomuto efektivnímu převodu do W^+ jsem se rozhodl zvolit podobný přístup jako autoři práce *pSp* s tím rozdílem, že oni enkodují obrazy obličejů, a já v mé práci zpracovávám hlasové nahrávky. Jestliže lze extrahovat z obrázku obličeje důležité informace, tyto informace poté konvertovat na vektory stylů a podle těchto vektorů vygenerovat totožný obličej, určitě by mělo jít podobné informace extrahovat i z řeči. Jak již bylo zmíněno dříve, jisté rysy v obličeji člověka mají vliv na to, jaký má daná osoba hlas. Můj model by však neměl rekonstruovat totožný obličej jako v *pSp*, ale obličej právě s takovými rysy, jaké má mluvčí na dané nahrávce.

Mým enkodérem je konvoluční neuronová síť, která postupně z nahrávky získává vektory přiznáků. Čím hlouběji se v síti nachází, tím více mají vektory příznaků redukovanou dimenzionalitu kvůli konvolučním sítím s krokem (*stride*) větším než 1. Takovémuto postupu se občas říká *pyramida příznaků*, jelikož na začátku jsou vektory s největší dimenzionalitou, která se postupně snižuje. Enkodér v průběhu inference extrahuje 3 typy vektorů reprezentující různé úrovně detailů propsaných do výsledného obrázku:

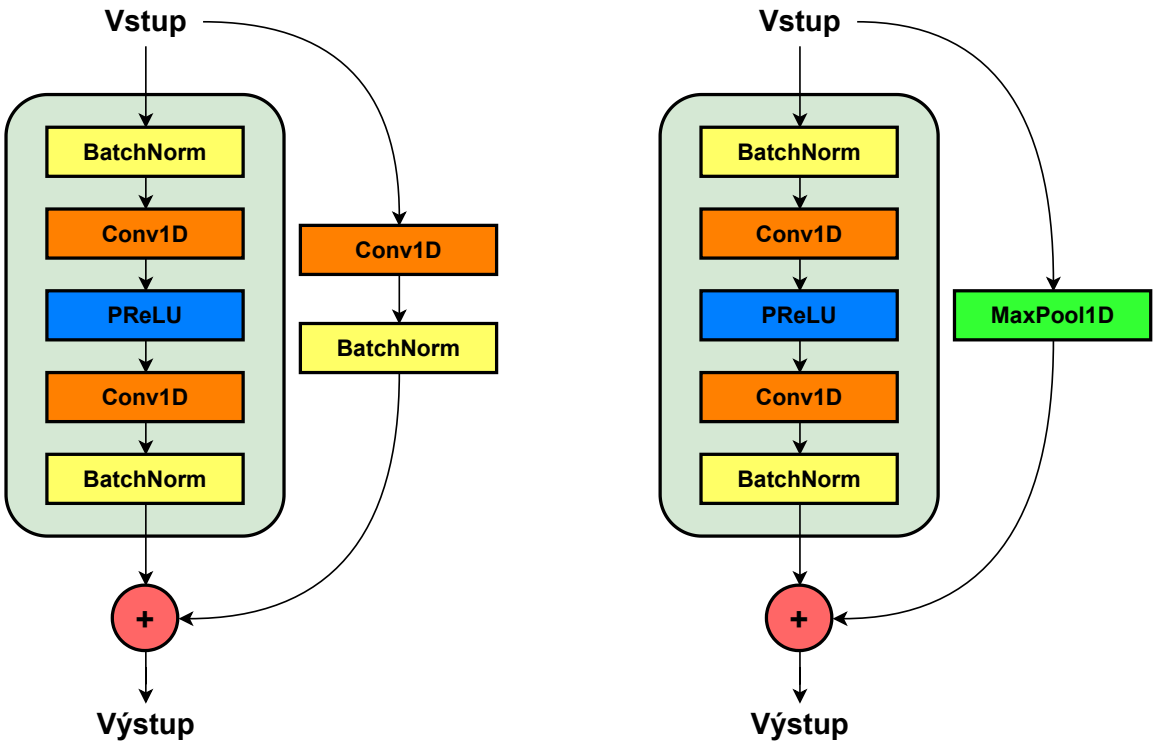
1. **Hrubé** – Tyto vektory jsou získány z příznaků nejblíže vstupní vrstvě. Zachycují nejnižší úroveň detailů v audio nahrávce. Mají největší dimenzionalitu a do modelu *StyleGAN* jsou vkládány jako poslední.

2. **Střední** – Z enkodéru jsou extrahovány uprostřed sítě. Tyto vektory reprezentují střední úroveň detailů z nahrávky a do *StyleGANu* jsou také vkládány pomocí vstupních vrstev uprostřed modelu.

3. **Jemné** – Poslední úroveň extrahovaných vektorů je získávaná na konci inference enkodéru. Styly vytvořené z těchto vektorů jsou vkládány na vstup *StyleGANu* v počátečních fází generování obrázku.

Enkodér je postaven na architektuře ResNet [31]. Na vstupu do enkodéru je konvoluční vrstva s jádrem o velikosti 25, která bere 1D vektor obsahující jednotlivé vzorky audio nahrávky, a na výstupu dává 64 kanálů s příznaky. Po vstupní vrstvě následuje vrstva normalizace dávek a aktivační vrstva PReLU. Po této vstupní kaskádě následuje série tzv. ResNet

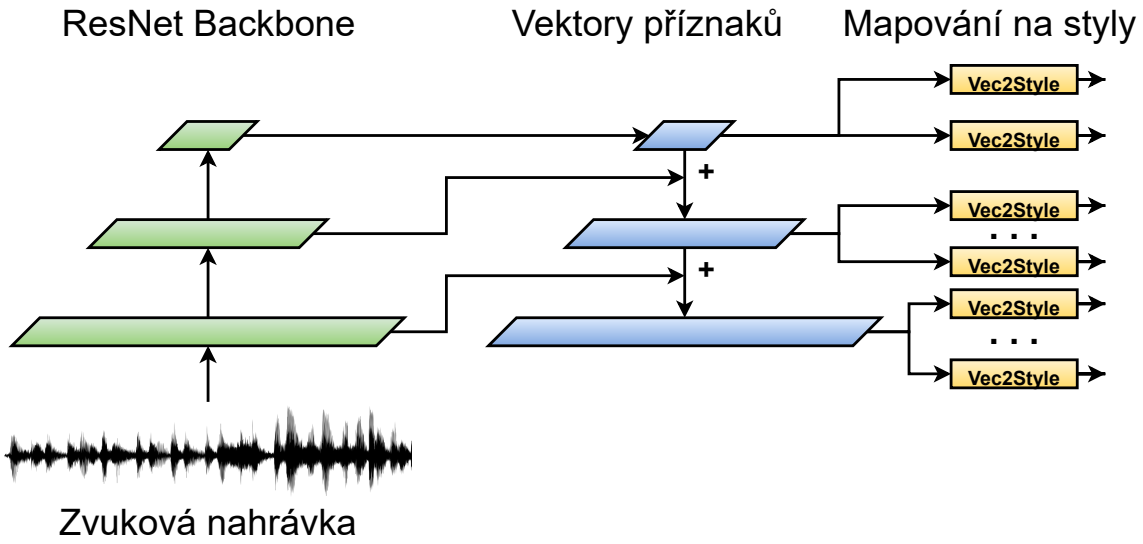
bloků. Každý blok má na svém začátku vrstvu normalizace dávek, kterou následuje vrstva konvoluční, a hned za ní aktivační vrstva PReLU. Dále data prochází další konvoluční vrstvou a vrstvou normalizace dávek. Před výstupem z ResNet bloku dochází ještě k sečtení výsledků z ResNet bloku s hodnotou přivedenou pomocí reziduálního spojení. Strukturu mých ResNet bloků můžete vidět na obrázku 3.3. Nalevo je ResNet blok, který se používá, pokud je na vstupu odlišný počet kanálů než na výstupu. Pak musí být do reziduálního spojení zakomponována konvoluční vrstva, která právě mění počet kanálů, a snižuje dimenzionalitu dat, následovaná normalizací dávek. Pokud má být na vstupu a výstupu stejný počet kanálů, je v reziduálním spojením zakomponován max-pooling, který snižuje dimenzionalitu dat.



Obrázek 3.3: Na tomto obrázku můžete vidět návrh reziduálních bloků, které chci použít ve svém modelu. Na levé straně je reziduální blok, který se bude využívat v případě, že bude na vstupu a výstupu odlišný počet kanálů. Pro tuto potřebu je součástí reziduálního propojení konvoluční vrstva, která má na starosti právě převod počtu kanálů. Zároveň tato vrstva redukuje dimenzionalitu dat pomocí kroku (*stride*) s hodnotou 2. Pokud se nemusí měnit počet kanálů, používá se reziduální blok vyobrazený napravo. Součástí reziduálního spojení není vrstva konvoluční ale sdružovací – max-pooling. Ta také redukuje dimenzionalitu dat tak, aby se před výstupem z bloku sčítala zpracovaná data se vstupními daty o stejných dimenzích.

Jak již bylo zmiňováno výše, v průběhu inference se budou z enkodéru postupně extrahovat vektory pro jednotlivé styly. Princip extrakce těchto vektorů je vidět na obrázku výsledného modelu 3.4. Po extrakci musí každý z vektorů ještě projít tzv. mapovací neuronovou sítí, která z každého vektoru vytvoří konkrétní styl. Úkolem této sítě je jednak redukce dimenzionality daného vektoru a úprava hodnot výsledného stylu. Architekturu mapovací sítě můžete vidět na obrázku 3.5.

U extrakce stylů z nahrávky zvuku jsem se nechal inspirovat enkodérem z již zmiňované práce *Encoding in Style* [52], kde autoři řešili obdobný problém, jako já. Jejich vstupem však nebyly nahrávky hlasu ale obrázky obličejů.

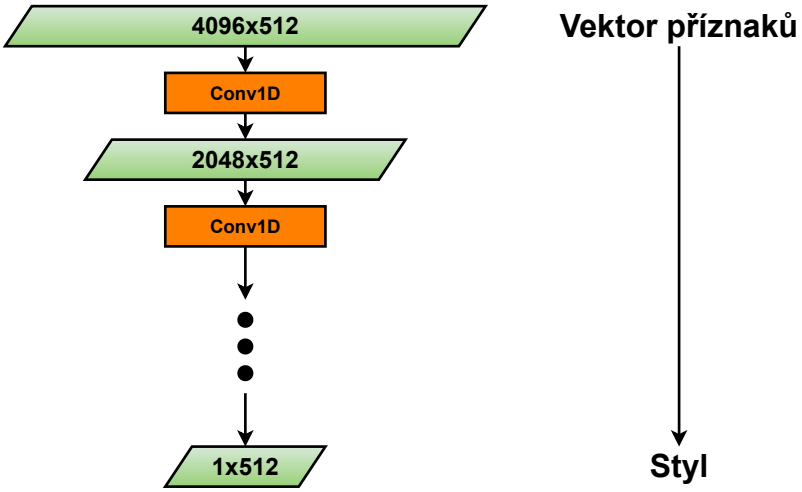


Obrázek 3.4: Na obrázku výše můžete vidět návrh mého enkodéru zvukových nahrávek. Je založen na extrakci tzv. *pyramidy příznaků*, která je extrahována pomocí konvoluční neuronové sítě s architekturou ResNet. Vektory příznaků jsou extrahovány ve 3 úrovních, kde každá úroveň nese jiné informace o nahrávce a řečníkovi samotném. Z těchto příznaků jsou poté pomocí mapovacích neuronových sítí generovány styly, které se budou vkládat na vstup generátoru *StyleGan*, a které budou mít vliv na podobu výsledného obrázku obličeje. Vektory příznaků jsou, před mapováním na styly, navíc po úrovních sčítány, protože nejvyšší úroveň nese detailnější informace o nahrávce, ale zase nemá informace o kontextu nahrávky, protože už je hodně zredukovaná její dimenzionalita. Sečtením se tyto informace propagují i do příznaků z první a druhé úrovně, které to mají naopak.

3.2.2 Dekodér

Jako dekodér jsem se tedy rozhodl zvolit *StyleGAN*. Použití již natrénovaného modelu je velkou úsporou času, jelikož natrénovat vlastní hodně kvalitní generátor obličejů zabere velké množství času, dat a výpočetního výkonu. U kvalitního natrénovaného modelu budu mít záruku, že u něj v průběhu trénování systému nenastane nějaká neočekávaná situace a trénink na ní nezhavaruje. Pokud se mi podaří systém správně natrénovat, *StyleGAN* by neměl mít problém generovat vizuálně kvalitní obličeje s korespondujícími rysy ve tváři.

Typický GAN generátor měl doposud pouze jednu vstupní vrstvu, do které se vkládal latentní vektor $\boldsymbol{z} \in Z$, který definuje, jaký obrázek má být vygenerován. Autoři modelu *StyleGAN* však ve své práci [36] představili možnost, jak generátoru předávat latentní vektory i v průběhu inference a upravovat tak výsledný obrázek. Na obrázku 3.6 můžete vidět rozdíl mezi architekturou klasického GAN generátoru a generátoru *StyleGAN*. U *StyleGAN* modelu se latentní vektor $\boldsymbol{z} \in Z$ propaguje mapovací sítí, která z něj vytvoří vícedimenzionální vektor $\boldsymbol{w} \in W$, který obsahuje styly pro *StyleGAN*. Z je označení pro vstupní latentní prostor a W zde označuje latentní meziprostor, ve kterém dokáže *StyleGAN* praco-



Obrázek 3.5: Návrh plně konvoluční neuronové sítě pro převod vektoru příznaků na styl, který bude sloužit pro předávání informací o podobě řečníka. Vstupní vektor příznaků s 512 kanály se postupně redukuje pomocí konvolučních vrstev s různým krokem (*stride*), až zůstane pouze 1 hodnota pro každý z 512 kanálů. Tyto hodnoty jsou poté převedeny do jednoho vektoru s jedním kanálem o délce 512, který reprezentuje daný styl.

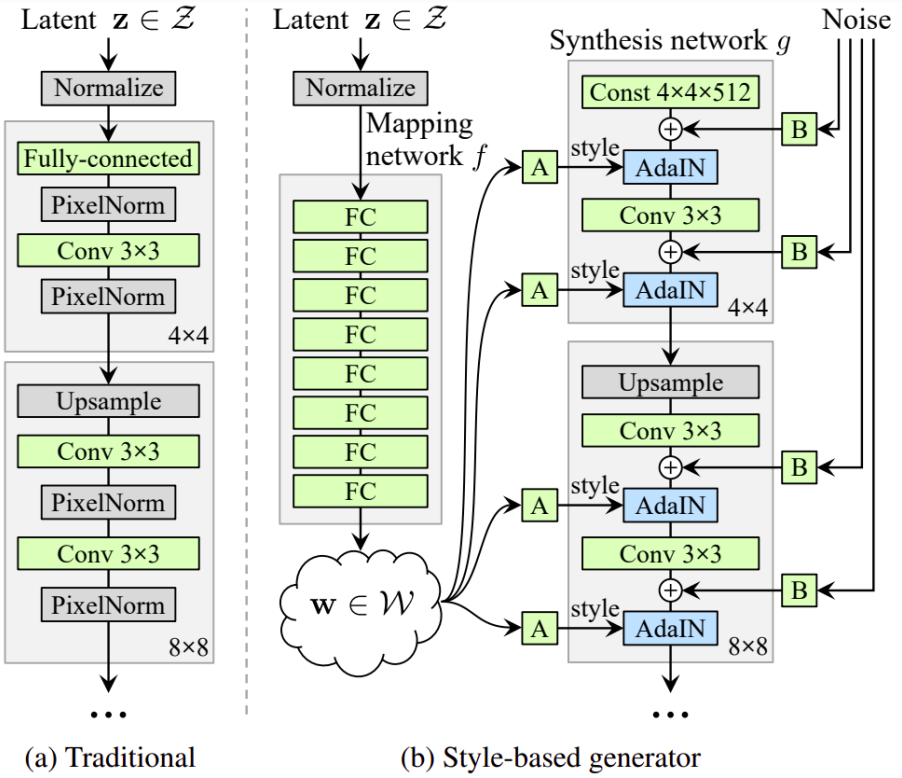
vat. V základní verzi je těchto stylů 18 [36]. Syntéza obrázku pomocí *StyleGAN* vždycky začíná z předučené konstanty, ke které jsou postupně přidávány jednotlivé styly a šum. Model sestává z několika bloků, kdy do každého bloku vedou 2 styly. Každý blok obsahuje několik konvolučních vrstev, po kterých vždy následuje operace *AdaIN* (angl. Adaptive instance normalization). Ta má na starosti míchání dosavadních map příznaků s přidaným šumem a daným stylem. Před zpracováním daným blokem dochází ještě ke dvojnásobnému zvýšení rozlišení syntetizovaného obrázku pomocí vrstev obrácené konvoluce. *StyleGAN* tedy obsahuje 9 těchto bloků, kdy každý zvětšuje vstupní obrázek dvakrát. Má-li tedy na začátku konstantu o rozměrech 4×4, na výstupu dostane obrázek s rozlišením 1024×1024. To je pro uměle generovaný obrázek obličeje opravdu vysoké rozlišení, aby na něm nebyly vidět různé nedostatky a artefakty.

Mé využití generátoru *StyleGAN* spočívá v tom, že místo mapovací sítě, která vytváří z vektoru $\boldsymbol{z} \in Z$ styly $\boldsymbol{w} \in W$ bude můj enkodér hlasových nahrávek, který bude dodávat styly na vstup místo mapovací sítě.

Více o použitém dekodéru se můžete dozvědět v původním článku [36] a nebo v článku [37], kde se autoři pokusili původní model ještě vylepšit.

3.2.3 Spojení modelů do jednoho systému

Výše bylo vysvětleno, jak budou fungovat jednotlivé modely. V této části ve zkratce popíšu na obrázku 3.7, jak bude můj systém fungovat po spojení všech modelů do jedné části. Nejdříve bude z hlasové nahrávky extrahován určitý počet vektorů příznaků s různou úrovní detailu informace o řečníkovi z nahrávky. Vektor poslední úrovně je pomocí interpolační techniky zvětšen na dimenzionalitu vektoru z 2. úrovně a tyto vektory jsou sečteny. Tím se docílí toho, že i do druhého vektoru se dostanou informace z posledního kroku extrakce. Ze stejného důvodu je tento výsledný vektor opět zvětšen a dále je proveden součet s vektorem příznaků z 1. úrovně extrakce. Tento krok dá 3 vektory příznaků, ze kterých se budou tvořit styly

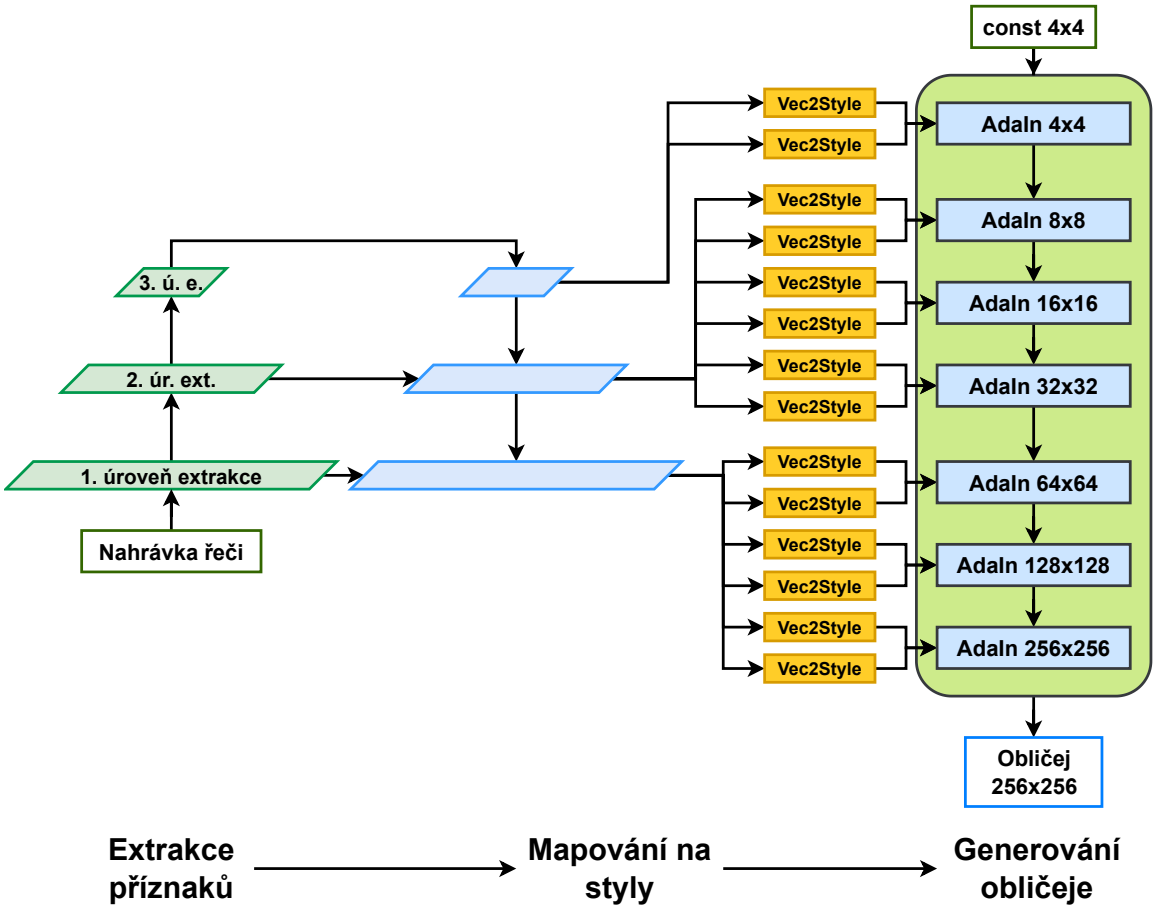


Obrázek 3.6: Obrázek architektury generátoru *StyleGAN*. (a) Na levé straně je pro porovnání tradiční generátor, který bere vstup pouze jednou vstupní vrstvou na začátku sítě. (b) Na pravé straně je generátor, který jako vstup bere vektory reprezentující dané styly, které udávají výslednou podobu generovaného obrázku. Tyto styly jsou získány tak, že vstup $\boldsymbol{z}$ je prohnán mapovací sítí složenou z plně propojených vrstev, která z něj vytvoří požadovaný počet stylů. Ty jsou pak vkládány na vstup generátoru pomocí vstupních vrstev v průběhu celé inference.

pro *StyleGAN*. Tyto vektory jsou pomocí mapovací sítě převedeny na 14 stylů, které jsou poté předány na vstup generátoru *StyleGAN*. Ještě před vstupem do generátoru prochází styly speciální afinní transformací, kterou definovali autoři generátoru. Generátor poté začne s generováním obličeje řečníka ze vstupní předučené konstanty o rozměrech 4×4. Postupně mu jsou na vstupní vrstvy vkládány dané styly, které pomocí operace *AdaIn* (*adaptive instance normalization*) zapracovává do výsledného obrázku. Finální obrázek má rozlišení 256×256 pixelů a měl by obsahovat obličej se stejnými rysy jako je obličej řečníka.

3.3 Trénování systému

Ani dobrý návrh a sofistikovanost modelu nezabezpečuje dobrý výsledek. Nejkritičtější částí mé práce bude určitě kvalitní natrénování modelu pomocí vhodných dat. Pro správné učení modelu jsem musel vybrat vhodný učící algoritmus a data. Jako učící algoritmus jsem zvolil *ADAM*, který je popsán výše v části o učení neuronových sítí 1.2.2. Pro trénování budu používat data z datasetů *VoxCeleb*, *VGGFace* a *AVSpeech*, kterým patří kapitola 2. V další části této kapitoly popíšu chybovou funkci, která bude použita pro trénování modelu.



Obrázek 3.7: Na tomto obrázku je návrh celého systému seskládaného z jednotlivých modelů neuronových sítí. Na prvním místě je enkodér řeči, který z hlasové nahrávky extrahuje vektory příznaků s informacemi o daném řečníkovi. Tyto vektory jsou poté pomocí mapovacích sítí převáděny na styly, pomocí kterých generátor *StyleGAN* generuje výslednou podobiznu obličeje řečníka z nahrávky.

Model budu trénovat pomocí dvojic audio nahrávek řeči a snímků obličejů daných řečníků. Více o zpracování dat bude uvedeno v kapitole o experimentech.

3.3.1 Chybová funkce

Stejně důležitou součástí systému, jako enkodér a dekodér, je chybová funkce, která bude řídit učení. Pokud by byly zmiňované 2 modely sebelepší, bez správného cíle v podobě chybové funkce, by se nedokázaly naučit vykonávat správnou úlohu. Chybová funkce použitá pro učení mého modelu je váženým součinem několika částí, které se násobí příslušným koeficientem. Chybová funkce počítá, jak kvalitně dokázal dekodér vygenerovat nový obrázek obličeje $\hat{y}$ vůči očekávanému výsledku z datasetu y. Složení chybové funkce jsem převzal z práce *pSp* [52].

MSE Loss. Prvním dílem mé chybové funkce je tzv. *Mean squared error*. Tato funkce počítá sumu rozdílů jednotlivých pixelů mezi obrázky y a $\hat{y}$. Rovnice pro výpočet *MSE* 1.12 je popsána v kapitole o učení neuronových sítí.

LPIPS Loss. Další důležitou součástí je *LPIPS* loss [62]. *LPIPS* je zkratka pro *Learned Perceptual Image Patch Similarity*. Tato metrika se počítá pomocí extrahovaných embeddingů z konvolučních neuronových sítí natrénovaných na velké datové sadě *ImageNet* [19]. Ukázalo se totiž, že klasické percepční chybové funkce – PSNR nebo SSIM – úplně nesouhlasí s percepčním rozpoznáním podobnosti okem člověka. Pokud bychom zprůměrovali hodnoty pixelů v obrázku pomocí průměrovacího filtru, klasické percepční metriky nebudou hlásit velkou odchylku od původního obrázku. Pokud ale dojde k menší změně objektu na obrázku, například vlivem otočení nebo jiné deformace, pro oko člověka to není příliš velká změna, kdežto klasické metriky zahlásí velkou odchylku. Autoři práce *LPIPS* přišli na to, že metriky počítané pomocí konvolučních neuronových sítí mají podobné hodnocení podobnosti jako lidské oko a dají se tak využít při ohodnocování chybové funkce.

Výpočet *LPIPS loss* se řídí rovnicí:

$$L_{LPIPS}(y, \hat{y}) = \|F(y) - F(\hat{y})\|_2 \,, \tag{3.1}$$

kde F je extraktor příznaků z obrázků a y a $\hat{y}$ jsou *ground-truth* obrázek a výstup generátoru.

ID Loss. V tomto specifickém případě odhadu obličejů není zapotřebí generovat 100% kopie obrázků. Důležité je, aby byla zachována identita mluvčího a důležité rysy v jeho obličeji. K tomu slouží takzvaná *ID Loss*. K jejímu výpočtu se používá předtrénovaný model *ArcFace* [20] s architekturou ResNet. Pomocí modelu se z obrázků extrahují embeddingy obsahující reprezentaci obličejů s důrazem na rysy, podle kterých lze určit identitu osoby. *ID Loss* se potom počítá podle následující rovnice:

$$L_{ID}(y, \hat{y}) = 1 - cs(R(y), R(\hat{y})), \tag{3.2}$$

kde R značí předtrénovaný model *ArcFace*, *cs* reprezentuje cosinovu podobnost 2 vektorů, y je tzv. *ground-truth* obrázek a $\hat{y}$ je generovaný obrázek pomocí mého modelu.

Regularizace. V rámci stabilizace trénování enkodéru doporučují autoři práce *pSp* použít tzv. *regularizační loss*. Ta má za úkol donutit enkodér, aby extrahoval styly podobné průměrnému stylu, který je spočítán ze stylů, na kterých byl trénován generátor *StyleGAN*. Rovnice pro tuto loss vypadá následovně:

$$L_{reg}(\boldsymbol{x}) = \|E(\boldsymbol{x}) - \bar{\boldsymbol{w}}\|_2 \tag{3.3}$$

Celková chybová funkce po složení jednotlivých součástí pak vypadá následovně:

$$L = \lambda_1 L_{MSE} + \lambda_2 L_{LPIPS} + \lambda_3 L_{ID} + \lambda_4 L_{reg}, \tag{3.4}$$

kde L_{MSE} je *mean squared error* a λ_1 je její koeficient, L_{LPIPS} je percepční loss a λ_2 její koeficient, L_{ID} je ID loss a λ_3 její koeficient a nakonec L_{reg} je regularizační loss pro vytváření správných stylů a λ_4 je její koeficient.

Kapitola 4

Implementace

Stejně důležitým úkolem jako návrh modelu a trénovacího algoritmu je i jejich implementace. Než jsem se ale mohl pustit do implementace samotného systému, musel jsem nejdříve implementovat několik modulů, pomocí kterých jsem připravil data pro trénování. Prvním z těchto modulů byl skript pro stahování dat z datasetu AVS. Dalším z nich bylo převádění fotek obličejů do normalizovaného stavu. V poslední řadě jsem pak v rámci příprav musel implementovat načítání dat ve správném formátu, který dokáže vzít na vstupu výsledný model. Pro veškerou implementaci jsem zvolil programovací jazyk Python 3[1] společně s hlavním frameworkem PyTorch[2], který poskytuje API pro vytváření a trénování modelů neuronových sítí a všech potřebných věcí kolem. PyTorch umožňuje optimalizaci výpočtů na grafických kartách značky NVIDIA díky podpoře speciální platformy NVIDIA CUDA[3]. Konkrétně u mé práce bych se bez této optimalizace neobešel, jelikož můj výsledný model je velmi výpočetně náročný. Vše o implementaci mého systému je popsáno v této kapitole. Následné experimenty s implementovaným modelem jsou popsány v kapitole Experimenty 5.

4.1 Předzpracování dat

Před samotnou implementací modelu a trénování bylo nejdříve potřeba získat a správně předzpracovat data. První z implementovaných části mé práce byl skript, který měl za úkol stahovat data z datasetu AVS. Podrobnější popis tohoto datasetu můžete najít v kapitole 2.3. Skládá se z anotací YouTube videí, kde každá anotace obsahuje:

- ID konkrétního YouTube videa,
- start a konec časového segmentu, kde mluvčí mluví, v sekundách,
- souřadnice obličeje, kde se nachází na začátku segmentu řeči.

Soubor s anotacemi obsahující tyto 3 údaje je naprosto dostačující jako vstup programu, který dokáže dataset postupně stahovat.

Stahování audio nahrávek je implementováno pomocí volání nástrojů *ffmpeg*[4] a *youtube-dl*[5]. *Youtube-dl* je nástroj pro stahování multimediálních souborů z různých internetových

[1] https://www.python.org/
[2] https://pytorch.org/
[3] https://developer.nvidia.com/cuda-zone
[4] https://ffmpeg.org/
[5] http://ytdl-org.github.io/youtube-dl/

platforem, ne jenom z YouTube. Je implementován v Pythonu a já jsem využil jeho rozhraní pro příkazovou řádku v linuxové distribuci *Ubuntu*. *Youtube-dl* stačí na vstupu parametry definující formát výsledného souboru a URL adresu videa, které chce uživatel stáhnout. Po stažení vrací na výstupu konečný soubor. Tento soubor je následně zpracován pomocí nástroje *ffmpeg*. Jedná se o open-source nástroj, který sestává ze sady knihoven a programů pro práci se zvukem, videem a dalšími multimediálními soubory a datovými proudy. Opět využívám rozhraní nástroje pro příkazovou řádku. Programu *ffmpeg* předávám na vstupu parametry s informacemi, že chci z videa extrahovat pouze audio, jaký časový rámec chci z nahrávky vyříznout, a nakonec v jakém formátu chci nahrávku uložit.

Obrázky obličejů, také potřebné k trénování, jsou stahovány pomocí knihovny OpenCV[6]. Ta poskytuje ve svém Python API třídu `VideoCapture`, pomocí které lze stáhnout video z platformy YouTube. Jakmile je dané video staženo, jej z něj extrahován rámec, který se vyskytuje v čase začátku segmentu řeči. Z toho rámce je poté vyřezána část podle anotovaných souřadnic obličeje. K ujištění se o poloze obličeje na extrahovaném snímku z videa slouží i model složený ze tří konvolučních neuronových sítí pro detekci obličejů *MTCNN* [61]. Jeho implementaci jsem získal z Python balíčku `facenet-pytorch`[7], který si lze jednodušše nainstalovat pomocí balíčkovacího systému `pip`. Detekovaná a vyříznutá fotka obličeje je poté uložena k dříve stáhnuté audio nahrávce řeči.

Snímky obličejů musely být před načítáním do mých modelů ještě normalizovány. Tento proces bude popsán v následující části této kapitoly. Pokud bych při trénování použil nenormalizované snímky obličejů, extrémně bych tím ztížil mému systému trénování. Normalizace musela proběhnout u všech snímků, i u těch z datasetů VGGFace, ne jenom u stahovaných snímků z datasetu AVS.

4.1.1 Normalizace fotek

Snímky obličejů jsem musel normalizovat do tvaru, kdy každý obličej je snímán ze předu bez výrazného výrazu ve tváři. Pokud by při tréninku byly obličeje různě natočené s různými výrazy, musel by se model učit tyto variace ignorovat, což by výrazně zvýšilo náročnost už tak obtížného úkolu.

Na začátku jsem vybíral ze dvou modelů. První z nich vznikl v rámci práce *Synthesizing Normalized Faces from Facial Identity Features* [17]. Tento model využili i autoři modelu *Speech2Face*. Hlavní myšlenkou tohoto modelu je využít kvalitních příznaků reprezentujících identitu člověka z obrázku extrahovaných modelem *FaceNet* [55]. Tyto příznaky jsou invariantní vůči póze, osvětlení a výrazu ve tváři, takže se jedná o mapovací úkol 1×1, kdy se mapují konkrétní příznaky na korespondující obličej. Pro toto mapování autoři práce natrénovali dekodér příznaků na obrázek obličeje, který je založen na konvolučních neuronových sítích. Ještě před vstupem příznaků do konvolučního modelu jsou příznaky propagovány vícevrstvým perceptronem pro vygenerování tzv. orientačních bodů a textur. Textury a orientační body, anglicky nazývané *landmarks*, jsou poté předány na vstup hluboké konvoluční neuronové síti, na jejímž výstupu je očekávaný obrázek obličeje v normalizovaném stavu. Nevýhodou tohoto modelu ale je, že se musí trénovat pomocí datové sady obsahující dvojice příznaků identity a normalizovaného obrázku obličeje. Datová sada, na které autoři svůj článek trénovali, obsahovala 12 000 normalizovaných fotek obličejů, což pro učení generátoru obličejů není příliš mnoho. Model tedy pro různé vstupní příznaky generuje

[6] https://opencv.org/
[7] https://github.com/timesler/facenet-pytorch

snímky obličejů s různými artefakty nebo s tzv. problémem černých brýlí, kdy má obličej místo očí černé fleky, jakoby měl nasazeny sluneční brýle.

Druhým modelem je již v minulé kapitole zmiňovaný model *pSp* [52]. Využití tohoto modelu bylo prvotně pro zakódování obrázku obličeje na vstupní styly pro model *StyleGAN* [36], který by podle nich měl vygenerovat totožný obličej. Tohoto se jim podařilo velmi úspěšně dosáhnout. Dalším experimentem s využitím síly generátoru *StyleGAN* bylo generování obrázků obličejů, které jsou snímány zepředu s konstantním osvětlením. Tento cíl se jim taky podařilo splnit velmi dobře, a to pouze s minimálními změnami v trénovacím procesu. První změnou bylo náhodné horizontální otočení obrázku v průběhu trénovaní. To nutilo model generovat obrázky, které byly co nejblíže oběma variantám otočení – tedy v pozici zepředu. Jedinou nevýhodou je, že takto generované obličeje nemají většinou neutrální výraz ale jsou generovány s mírným úsměvem. Další změnou v trénovacím procesu byla úprava loss funkce, kdy se tato funkce zaměřovala na generování obličejů, které mají totožné rysy podle modelu *ArcFace* [20], a nezajímalo ji tolik porovnávání obrázků pomocí chybové funkce *MSE* a *LPIPS* [62]. Velkou výhodou tohoto modelu je, že se umí trénovat metodou učení bez učitele (angl. *unsupervised learning*). To má za následek to, že nepotřebuje datovou sadu vstupních obrázků a očekávaných obrázků na výstupu, ale pro správné natrénování mu stačí pouze obrázky vstupní. Tento model byl trénován na datasetu *FFHQ* [36], který obsahuje 70 000 snímků obličejů stažených z webové stránky *Flickr*[8].

Pro normalizaci mých fotek obličejů jsem si tedy vybral druhý model. Normalizace byla implementována jako jedna z prvních věcí, protože jsem věděl, že zpracovat všechny fotky z datasetů, které jsem chtěl použít, bude velmi výpočetně náročné. To se nakonec ukázalo jako pravda a celková doba normalizace všech fotek zabrala asi 1 500 hodin na grafické kartě NVIDIA Tesla T4 s 16 GB pamětí. Prostředí, na kterém jsem normalizaci fotek prováděl, je stejné jako prostředí experimentů s implementovaným modelem, a bude popsáno v kapitole Experimenty 5.

Implemenace modelu *pSp* je dostupná na GitHubu[9], kde se autoři odkazují i na místo, odkud lze stáhnout natrénovaný model pro normalizaci fotek obličejů. Autoři poskytují také skript `inference.py` pro inferenci modelem. Ten bere jako parametry cestu k datům, které má zpracovat, dále pak cestu k tzv. *checkpointu* s natrénovanými parametry modelu, a velikost jedné dávky dat (*batch size*). Pomocí tohoto skriptu jsem dokázal jednoduše zpracovat fotky obličejů, které jsem potřeboval k následnému trénování.

4.1.2 Načítání datasetu

Již před implementací samotného modelu bylo jasné, v jaké podobě mu budu muset předkládat data. Modely neuronových sítí implementované pomocí frameworku PyTorch berou na vstupu data v podobě více dimenzionálních polí. Pro mé audio nahrávky to byl konkrétně formát `[s, c, w]`, kde `s` je velikost jedné dávky, `c` je počet vstupních kanálů a `w` je počet vzorků dané audio nahrávky. Pro obrazová data je tento formát trošku odlišný, protože daný model pracuje ve více dimenzích. Jedná se o formát `[s, c, w, h]`, kde `s` je opět velikost jedné dávky, `c` je počet kanálů (pro RGB obrázek je roven 3) a `w` a `h` jsou šířka a výška obrázku.

PyTorch poskytuje pro načítání dat jednoduché API obsahující několik stěžejních tříd. Stěžejní z nich jsou třídy `Dataset` a `DataLoader`. Pomocí třídy `Dataset` může každý vytvořit vlastní třídu reprezentující dataset potřebný k trénování, a to tak, že vlastní třída dědí

[8] https://www.flickr.com/
[9] https://github.com/eladrich/pixel2style2pixel

z `Dataset` třídy. Důležité je pak implementovat metody `__getitem__` a `__len__` nově vytvořené třídy. Metoda `__getitem__` se používá pro indexování vzorků dat z datasetu. Například pomocí `dataset[i]` lze dostat i-tý vzorek dat z datasetu. Já jsem pro svůj dataset vytvořil třídu `FaceSpeechDataset`, která si načítá cesty k audio nahrávkám a obrázkům obličejů buďto z předpřipraveného CSV souboru, nebo, pokud ještě tento CSV seznam neexistuje, z adresářové struktury definované v konfiguraci systému. Načtené cesty audio nahrávek a fotek obličejů se párují podle ID daného řečníka. Při tréninku totiž potřebuji vždycky audio nahrávku i referenční obrázek obličeje. Metodu `__getitem__` má třída `FaceSpeechDataset` implementovanou tak, že podle parametru `index` vrací načtenou dvojici dat `(audio_nahrávka, fotka_obličeje)` na daném indexu. Načítání audio nahrávek je implementováno pomocí balíčku `torchaudio`[10], který v základu obsahuje sadu jednoduchých funkcí pro načítání a práci se zvukovými daty. Pomocí metody funkce `torchaudio.load` načítám audio nahrávku ve formátu `wav` jako 1D tensor. Dále je nahrávka převzorkována pomocí funkce `torchaudio.transforms.resample` na požadovanou vzorkovací frekvenci. V rámci experimentů poté upravuji nahrávce různě délku, to je implementováno pouze vybráním určitého počtu vzorků z načteného a upraveného tensoru. Obrazová data načítám pomocí třídy `Image` z balíčku `Pillow`[11]. Pro načtení obrazových dat používám konkrétně metodu `Image.open`, který načte obrázek jako strukturu z balíčku Numpy[12] `ndarray`. Jedinou úpravou této struktury je převod na třídu `torch.Tensor` a následná normalizace hodnot pixelů. Metoda `__len__` pak vrací počet dvojic v načteném datasetu. Pokud se cesty načítají přímo z adresářové struktury, jsou poté dvojice cest uloženy do CSV souboru, aby bylo jejich načtení jednodušší a hlavně rychlejší. Celý dataset obsahuje přes milion dvojic a načítání dvojnásobného počtu cest z adresářové struktury je zbytečně časově náročné.

Jakmile mám cesty k souborům uložené ve třídě `FaceSpeechDataset` a implementované načítání dat v rámci metody `__getitem__`, můžu celou třídu `FaceSpeechDataset` předat do třídy `DataLoader`, která má na starosti vytváření dávek dat o předem definované velikosti. Toto vytváření dávek dat probíhá až za běhu, při samotném trénování modelu.

4.2 Implementace modelu

Stejně jako návrh modelu, i jeho implementace je rozdělena do podobných částí. Sám jsem však všechny neimplementoval. Implementaci generátoru *StyleGan* jsem převzal z veřejného repozitáře na GitHubu. Vlastní implementaci jsem realizoval pro enkodér řeči a mapovací síť, která převádí extrahované vektory příznaků na styly, které jsou poté předávány na vstup *StyleGANu*. V této sekci bude popsána implementace těchto tří částí a jejich propojení. Mému celkovému modelu jsem dal pracovní název *Voice Styled Face*. Do češtiny bych to přeložil jako *Hlasem stylizovaný obličej*.

4.2.1 Enkodér

Implementace enkodéru a mapovacích sítí je k nalezení v souboru `vsf_encoder.py`. Tento soubor obsahuje dvě třídy. První z nich je `VoiceEncoder`, která reprezentuje již zmiňovaný enkodér řeči. Tato třída dědí z třídy `nn.Module`, kterou poskytuje balíček `nn` z frameworku PyTorch, a ze které by měla dědit každá třída, která reprezentuje buď komponentu neuronové sítě nebo síť samotnou. V konstruktoru této třídy se nejdříve dle parametrů převzatých

[10] https://pytorch.org/audio/stable/index.html
[11] https://python-pillow.org/
[12] https://numpy.org/

při inicializaci rozhodne, kolik a jaké reziduální bloky má enkodér obsahovat. Jakmile je to rozhodnuto, jsou postupně do těla enkodéru přidávány reziduální bloky reprezentované třídou `ResNetBlock1D`, která je implementovaná v souboru `utils.py` ve stejném modulu jako `vsf_encoder.py`. Jak již bylo zmíněno v kapitole o návrhu enkodéru 3.2.1, reziduální bloky obsahují konvoluční, aktivační a normalizační vrstvy. Ty jsou implementovány pomocí PyTorch tříd `Conv1d`, `PReLU` a `BatchNorm1d` opět importovaných z balíčku `nn` z PyTorch. Pokud se jedná o reziduální blok, který nemění počet kanálů zpracovávaných dat, je v jeho reziduálním spojení umístěna jedna max-pooling vrstva implementovaná pomocí třídy `MaxPool1d` ze stejného balíčku. Každá třída, která dědí z třídy `Module`, musí mít implementovanou metodu `forward`, která definuje, jak se mají zpracovat data při průchodu daným modulem neuronové sítě. V případě reziduálních bloků to je zpracování dat pomocí těla bloku a poté sečtení se vstupními daty přivedenými na výstup sítě pomocí reziduálního spojení.

Jakmile mám vytvořeno tělo enkodéru, musím realizovat extrakci vektorů příznaků a následné mapování příznaků na styly. Vektory jsou extrahovány ve třech úrovních. Každá úroveň extrakce je umístěna po sérii několika reziduálních bloků. Implementace je realizována pomocí `for` cyklu, kdy v každé iteraci projde dávka dat jedním reziduálním blokem. Jakmile se cyklus dostane za reziduální blok, kdy má být extrahována daná úroveň, uloží se aktuální vektory příznaků do proměnné. Všechny vektory pro danou úroveň jsou stejné hodnoty. Když skončí inference enkodérem, mám uloženy tři vektory příznaků ze tří úrovní extrakce. Tyto vektory jsou poté propagovány mapovacími sítěmi a už jako styly uloženy do seznamu stylů. Tento seznam je poté převeden na jeden tenzor pomocí metody `stack`, která vezme seznam tenzorů, v mém případě 512-dimenzionálních stylů, a podle definované dimenze je spojí do jednoho tenzoru. Tím končí práce enkodéru a tento tenzor stylů je předán na vstup generátoru *StyleGAN*.

4.2.2 Změna v návrhu mapovací sítě a její implementace

Mapovací síť je implementována jako druhá třída v souboru `vsf_encoder.py`, a to konkrétně `Vec2Style`. Podle inspirace z práce *pSp* [52] jsem v každé z těchto sítí měl v plánu mít určitý počet konvolučních vrstev, které budou postupně redukovat dimenze vstupních vektorů s 512 kanály, až zbude každý kanál s pouze 1 hodnotou. Pro audio nahrávky o délce několika sekund s vzorkovací frekvencí 16 000, které mají na vstupu celkově desítky tisíc vzorků, by to však byl příliš velký počet konvolučních vrstev. Proto jsem byl nucený změnit mapovací síť tak, aby dokázala redukovat dimenze i bez tolika konvolučních vrstev. Na začátek této mapovací sítě jsem vložil tři konvoluční vrstvy, které nejdříve zpracují vstupní data. Za tuto sekvenci konvolučních vrstev jsem dále vložil vrstvu adaptivního max-poolingu, ze které vystoupí data s 512 kanály a dimenzí o hodnotě 1. Konvoluční vrstvy jsou stejně jako u enkodéru implementovány pomocí tříd `Conv1d` z modulu `nn`. Adaptivní max-pooling je implementován pomocí třídy `AdaptiveMaxPool1d`. Rozdíl mezi normálním a adaptivním poolingem je ten, že adaptivní pooling si sám určí, jaké má použít velikosti konvolučního jádra a kroku při posunu konvolučního jádra po datech, aby dosáhl požadované dimenzionality dat na výstupu. Metoda `forward` této třídy tedy vezme vstupní data a propaguje je třemi konvolučními vrstvami, následně je pošle do adaptivního poolingu, který je redukuje na velikost 1. Jakmile má data s 512 kanály o velikosti 1, změní rozložení těchto dat a vytvoří z nich tenzor s jedním kanálem o velikosti 512. Tato změna na tenzor je implementována pomocí metody `view` ze třídy `Tensor`, která reprezentuje data pro výpočet v PyTorch. Úplně posledním krokem je aplikace afinní transformace nad daty, kterou je zapotřebí provést před

vstupem stylů do *StyleGAN*. Implementaci této transformace jsem převzal z implementace *StyleGANu*, která bude popsána v další části této kapitoly.

4.2.3 Dekodér

Implementaci dekodéru, tedy generátoru *StyleGAN*, jsem převzal z veřejného repozitáře[13] z GitHubu od autora *rosinality*[14]. V rámci mého projektu je umístěna v modulu `stylegan2` v souboru `model.py`. Generátor samotný je implementován třídou `Generator`. V tomto zdrojovém souboru je ale implementováno daleko více tříd, které jsou využity třídou `Generator`. Stejně tak jako mnou implementované třídy pro neuronové sítě, i tato dědí z PyTorch třídy `Module`. Některé potřebné výpočetní operace jsou implementovány v jazyce C++ pro jejich optimalizaci. Pro ně je poté přítomný vždy jeden zdrojový soubor v Pythonu, který slouží jako tzv. *wrapper* nad C++ implementací, abychom ji mohli využívat v ostatních Python zdrojových souborech. Více o této implementaci se můžete dozvědět ze souboru `README.md` ze zmiňovaného GitHub repozitáře[15].

4.2.4 Propojení modelů

Propojení všech doposud implementovaných modelů jsem realizoval v třídě `VoiceStyledFace`. V konstruktoru této třídy probíhá instanciace objektů enkodéru a dekodéru společně s případnou vrstvou average-poolingu, pokud by chtěl uživatel generovat menší fotku než v původním rozlišení *StyleGANu*. Pokud je třídě `VoiceStyleFace` předána cesta k tzv. *checkpointu*, tedy souboru, který obsahuje uloženou konfiguraci již trénovaných parametrů modelů, jsou enkodér a dekodér inicializovány s těmito parametry. Propojení těchto modelů je realizováno až v metodě `forward`. Nejdříve jsou vstupní data zpracována enkodérem, který je převede na tenzor stylů, tento tenzor je poté předán na vstup dekodéru. Ten vrací seznam obrázků generovaných pro předané styly. Jakmile jsou vygenerovány obrázky, metoda `forward` je vrací o úroveň výše, kde pomocí nich bude počítána chybová funkce systému pro danou dávku dat.

4.3 Trénovací algoritmus

Posledními kroky implementace byly trénovací algoritmus a chybová funkce. Tato funkcionalita je implementována v modulu `train`. Implementace trénovacího algoritmu v PyTorch není vůbec složitá. PyTorch poskytuje několik tříd, které řeší velkou většinu problémů za uživatele. Trénovací algoritmus je implementován ve třídě `Trainer` v souboru `trainer.py`. V konstruktoru této třídy je nejdříve vytvořen objekt třídy `VoiceStyleFace`, který reprezentuje můj systém pro převod hlasu na fotku obličeje. Dále probíhá instanciace tříd, které reprezentují jednotlivé části chybové funkce. Tyto třídy a jejich implementace bude popsána v následující části této kapitoly. Jakmile má třída `Trainer` hotovou inicializace chybové funkce, může přejít na vytvoření optimalizátoru trénovatelných parametrů. Parametry generátoru *StyleGAN* jsou zamraženy na již natrénovaných hodnotách a dále se netrénují. Optimalizátor bude tedy pracovat pouze s parametry enkodéru a mapovacích sítí. Jako trénovací algoritmus jsem zvolil *Adam*, který je implementován v PyTorch frameworku v balíčku *optim* třídou `Adam`. Na vstupu bere jako parametr učící koeficient (angl. *learning rate*).

[13] https://github.com/rosinality/stylegan2-pytorch
[14] https://github.com/rosinality
[15] https://github.com/rosinality/stylegan2-pytorch/blob/master/README.md

Dále je pomocí optimalizátoru iniciován *tzv. scheduler*, který s každým krokem snižuje učící koeficient, aby optimalizátor nezůstal zaseknutý v jednom místě kvůli příliš vysokému učícímu koeficientu. Tento *scheduler* je implementován pomocí PyTorch třídy `ExponentialLR`, která je implementována také v balíčku `optim` v submodulu `scheduler`. Tento *scheduler* s každým jeho krokem vynásobí učící koeficient podle zadaného parametru. Po inicializaci optimalizačního algoritmu a *scheduleru* zbývá vytvořit 2 objekty třídy `FaceSpeechDataset`. První pro trénovací množinu dat a druhý pro testovací množinu. Tyto objekty datasetů jsou poté předány jako parametr při instanciaci tříd `DataLoader`, které slouží pro vytváření a načítání dávek dat.

Trénovací cyklus je implementován v metodě `train`, obsahuje jeden velký `for` cyklus, který reprezentuje trénování systému po předem určený počet epoch. V tomto cyklu je vnořený další `for` cyklus, který reprezentuje trénování systému na jednom průchodu trénovací sadou. Jakmile je hotový průchod trénovací sadou, přejde se na průchod sadou testovací, který vyhodnocuje aktuální kvalitu modelu.

4.3.1 Chybová funkce

Pro každou část chybové funkce jsem vytvořil jednu třídu, která má na starosti výpočet chyby. Nejjednodušší na implementaci byly chybové funkce *MSE* a *L1*. Pro výpočet *MSE* chybové funkce jsem využil již hotové implementace v PyTorch, kde je tato funkce implementována funkcí `mse_loss` uvnitř balíčku `nn` v submodulu `functional`. Tato funkce bere na vstupu generovaný obrázek z mého systému a referenční obrázek z datasetu a počítá chybu podle rovnice 1.12. Taktéž chybová funkce *L1* je implementována ve stejném submodulu, a to konkrétně funkcí `l1_loss`. Tato funkce bere stejné parametry jako funkce `mse_loss`, akorát že počítá chybu podle pomocí sumy kvadrátů rozdílu jednotlivých pixelů ale pomocí sumy rozdílu daných pixelů.

Další částí je v kapitole 3.3 zmiňovaná *LPIPS* chybová funkce. Její implementaci jsem převzal z práce *pSp* [52]. Základem této implementace je třída `LPIPS`, která si v konstruktoru instanciuje třídu reprezentující konvoluční neuronovou síť pro extrakci embeddingu reprezentujícího informace o obrázku. Na výběr má ze tří architektur – *AlexNet*, *SqueezeNet*, *VGG16* – a vybírá podle parametru, který dostane na vstupu konstruktoru. V metodě `forward` této třídy se nejdříve extrahují embeddingy z generovaného a referenčního obrázku, ze kterých se počítá kvadrát rozdílu, který se poté ještě propaguje několika konvolučníma vrstvama, nad jejichž výstupem se provádí normalizace a sumarizace do jedné hodnoty. Na detailnější implementaci se můžete podívat na GitHubu autora práce *pSp*[16].

Následující součást chybové funkce jsem také převzal z práce *pSp*. Jedná se o regularizační komponentu, která má zaručovat, aby se styly extrahované z řeči příliš moc nelišily od průměrného stylu, počítaného ze stylů, na kterých byl trénován generátor *StyleGAN*. Tato část chybové funkce je implementována třídou `WNormalizationLoss`. Její jediný výpočet spočívá v tom, že vezme průměrný styl, který vzešel z trénovaní generátoru, a odečte jej od stylů, které vzešli z enkodéru řeči. Tento rozdíl je poté sumarizován do jedné hodnoty a navrácen jako výsledek.

Poslední částí chybové funkce je takzvaná *ID loss*. Implementaci této chybové funkce jsem také převzal z práce *pSp*. K výpočtu chybové funkce je potřeba model *FaceNet*, který je postavený na konvoluční neuronové síti s architekturou ResNet. *FaceNet* je natrénovaný k extrakci příznaků z obličeje, které reprezentují identitu daného člověka na fotce. Na vstupu tento model bere obrázky obličejů s rozlišením 112×112 a na výstupu dává vektor příznaků.

[16] https://github.com/eladrich/pixel2style2pixel

Pro zmenšení obrázků z mého generátoru na požadované rozlišení používám adaptivní average-pooling. Implementace modelu je umístěna ve třídě `BackboneLoss`. V metodě `forward` třídy `IDLoss` se nejdříve extrahují příznaky pomocí modelu `FaceNet` z vygenerovaného obrázku a z obrázku referenčního, které jsou mezi sebou porovnány pomocí skalárního součinu daných vektorů příznaků. Výsledek skalárního součinu je poté odečten od hodnoty 1 a vrácen jako výsledek chybové funkce.

Kapitola 5

Experimenty

V poslední kapitole této práce budu popisovat nejzajímavější experimenty, které jsem se svým systém prováděl, a jakých se mi podařilo dosáhnout výsledků. Pro experimentování jsem měl na začátku připravenou velkou datovou sadu, pomocí které jsem chtěl trénovat svůj systém. Již při prvních experimentech se však ukázalo, že pro trénování na takto velké datové sadě nebudu mít k dispozici dostatečnou výpočetní kapacitu. V následující kapitole budou tedy popsány experimenty na různě velkých datových sadách s různým nastavením. Dále popíšu, jak jsem v průběhu experimentování změnil samotný enkodér pro extrahování příznaků z řeči a mapovací sítě pro převod příznaků na styly. Ještě před samotnými experimenty je ve zkratce popsáno prostředí, na kterém jsem experimenty prováděl a své neuronové sítě trénoval. Za popis prostředí je vložena ukázka dat, se kterými jsem pracoval. Pro vyhodnocení výsledků provedených experimentů jsem nepoužíval žádnou standardní metriku používanou pro posuzování podobnosti obrazu. Metriky jako SSIM, PSNR nebo MSE mezi sebou porovnávají obrázky pomocí rozdílů jednotlivých pixelů. Mým úmyslem však nebylo generovat naprosto stejné obrázky obličejů jako jsou ty referenční. Cílem mé práce je generovat obličeje se stejnými rysy jako má obličej mluvčího z dané nahrávky řeči. To ale neznamená, že se musí jednat o totožný obrázek.

5.1 Prostředí experimentů

Veškeré experimenty byly prováděny na výpočetní strojích, které jsou součástí české Národní Gridové Infrastrukury Metacentrum[1]. Všechny výpočetní zdroje byly poskytnuty z projektu *e-Infrastruktura CZ* (e-INFRA CZ LM2018140) podporovaných Ministerstvem školství, mládeže a tělovýchovy České republiky.

Metacentrum poskytuje několik možností, jak experimenty spouštět. První varianta je pomocí tzv. dávkových skriptů (*batch jobs*). V tomto skriptu uživatel definuje, jaké chce využívat zdroje, jaké chce využívat balíčky, připraví si prostředí a spustí danou úlohu. Do běhu této úlohy však již nejde zasahovat, a pokud obsahuje dávkový skript chybu nebo je potřeba udělat jiná změna, musí se aktuální běh zrušit, dávkový skript změnit a poslat znova do fronty na spuštění. Tomuto předchází druhá varianta spuštění – interaktivní úloha. V rámci interaktivní úlohy je uživateli přiřazen stroj s definovanými výpočetními zdroji, a ten je pak schopný s těmito zdroji pracovat pomocí příkazové řádky dle potřeby. Já osobně jsem využíval hlavně interaktivní úlohy, protože jsem v průběhu experimentování potřeboval

[1] https://metavo.metacentrum.cz/

různě měnit nastavení spouštěných programů a předávat jim různé parametry. Dané zdroje jsou uživateli přiřazeny maximálně na 24 hodin, po této době si o ně musí žádat znova.

Prostředí Metacentra poskytuje nepřeberné množství hotových modulů, které může uživatel libovolně využívat. Mezi ně patří i modul s připraveným Python prostředím, které obsahuje i balíčky, které potřebuji pro trénování mého modelu. Já jsem se vydal však cestou přípravy vlastního Python virtuálního prostředí pomocí Python distribuce Anaconda[2]. Předpřipravený modul totiž obsahoval staré verze potřebných balíčků. Já jsem pro všechny experimenty používal PyTorch verze 1.10.2 a CUDA verzi 10.2.

Veškeré experimenty v rámci Metacentra byly spouštěny na grafické kartě NVIDIA A100 s 40 GB paměti. Grafická karta s takto velkou pamětí byla zapotřebí kvůli vyššímu počtu jednotlivých modelů neuronových sítí, které v práci využívám. Na prvním místě je to můj enkodér hlasových nahrávek a minimálně 14 malých mapovacích sítí pro převod příznaků na styly. Poté přichází na řadu *StyleGAN*, což je obrovský model pro generování obličejů. Tímto však využití neuronových sítí nekončí. Pro výpočet chybové funkce využívám další dva modely. Prvním z nich je *FaceNet* pro extrakci příznaků z fotek obličejů a druhým je konvoluční neuronová síť pro extrakci příznaků pro výpočet chybové funkce *LPIPS*.

V době dokončování experimentů k této práci jsem podle statistik Metacentra provedl přes 500 různých spuštění trénování mého systému nad různými daty, s různým nastavením hyperparametrů a s různými architekturami jednotlivých modelů neuronových sítí. Nejdůležitější z nich budou popsány v této kapitole.

5.2 Data

Pro experimentování a vyhodnocení systému jsem potřeboval příslušný dataset. Ještě před začátkem experimentování s modelem jsem si vytvořil velký dataset nahrávek řeči a korespondujících obrázků obličeje řečníka z dané nahrávky. Výsledný dataset má přes milion nahrávek řeči a obrázků obličejů. Je poskládán z datasetů VoxCeleb2 a části datasetu AVSpeech. Obrázky mají rozlišení 256×256 pixelů a nahrávky řeči jsou od cca od 3 do 10 sekund dlouhé. Pro trénování jsou nahrávky hlasu normalizovány na předem danou délku. V prvotních experimentech to bylo 6 vteřin, později jsem délku upravoval.

Jak již bylo v této práci zmíněno, všechna data jsou sbírána *in-the-wild*. Tato data nebyla pořizována za speciálních podmínek, aby byla co nejčistší, ale jsou sbírána z reálného prostředí, a obsahují nejrůznější šum. U hlasových nahrávek je to ruch z okolí nebo třeba nekvalitně zachycená řeč mluvčího. U obrázků to může být různé natočení a osvětlení obličejů, cizí předměty v obraze nebo různé pozadí za obličejem. U nahrávek řeči tento šum není velký problém, u obrázků obličejů jsem musel provádět tzv. normalizaci. Nenormalizovaná data můžou být výhodou při trénování modelu a jeho robustnosti, v mém případě by bylo ale nesmírně složité s takovými daty systém trénovat. Na obrázku 5.2 je ukázka původních obrázků a také k nim korespondujícím normalizovaným obrázkům.

[2] https://www.anaconda.com/

(a) (b) (a) (b)

Obrázek 5.1: Ukázka snímků z datasetu VGGFace. Ve sloupích (a) jsou originální obrázky a ve sloupcích (b) můžete vidět obrázky normalizované.

5.3 Prvotní experimenty na malém datasetu

Ještě před trénováním systému na velkém datasetu jsem se chtěl ujistit, že bude schopen naučit se daný úkol převodu hlasu na obličej na malém vzorku dat. Systém by se sice extrémně přetrénoval na tento malý vzorek dat, měl bych ale jistotu, že jsem se nevydal úplně slepou uličkou, a že systém dokáže generovat obličej na základě hlasu. Tato zmenšená datová sada obsahovala hlasové nahrávky a obrázky obličejů pro 20 identit, což dalo dohromady asi nějakých 2000 hlasových nahrávek.

5.3.1 Nastavení parametrů učení

Před spuštěním trénování jsem musel určit hodnoty několika parametrů, které mají vliv na učící proces – tzv. hyperparametrů. Prvotní nastavení těchto parametrů nebylo úplně uspokojivé. Po několika zkouškách trénování systému jsem se však dopracoval k následujícím hodnotám parametrů:

- počet epoch = 12,
- učící koeficient = 10^{-3},
- velikost dávky pro trénovací data = 16,
- velikost dávky pro testovací data = 16,
- a koeficienty částí chybové funkce 3.4:
  - koeficient MSE loss – $\lambda_1 = 1$,
  - koeficient LPIPS loss – $\lambda_2 = 1$,
  - koeficient ID loss – $\lambda_3 = 0,1$,
  - koeficient regularizační loss – $\lambda_4 = 0,0005$.

Jako optimalizační algoritmus byl použit Adam a pro učící koeficient jsem použil techniku postupného snižování – *Exponential learning rate decay* – kdy jsem každou epochu vynásobil učící koeficient hodnotou $0,95$.

5.3.2 Výsledky

Jak jsem předpokládal na začátku tohoto experimentu, systém se přetrénoval na malé datové sadě a generoval obrázky obličejů, které si zapamatoval při trénování. Účel tohoto experimentu byl tedy splněn. Zjistil jsem, že se systém dokáže naučit řešit daný úkol alespoň na malé datové sadě. Na obrázku 5.3.2 můžete vidět pár příkladů výsledků tohoto experimentu.

(a) (b) (a) (b)

Obrázek 5.2: Ukázka výsledků trénování na malém datasetu. Ve sloupcích (a) jsou referenční snímky a ve sloupci (b) jsou obličeje odhadnuté podle nahrávky řeči. Odhadnuté obrázky jsou generovány systémem, který se přetrénoval na malé trénovací sadě.

5.4 Experimenty s VoxCeleb2

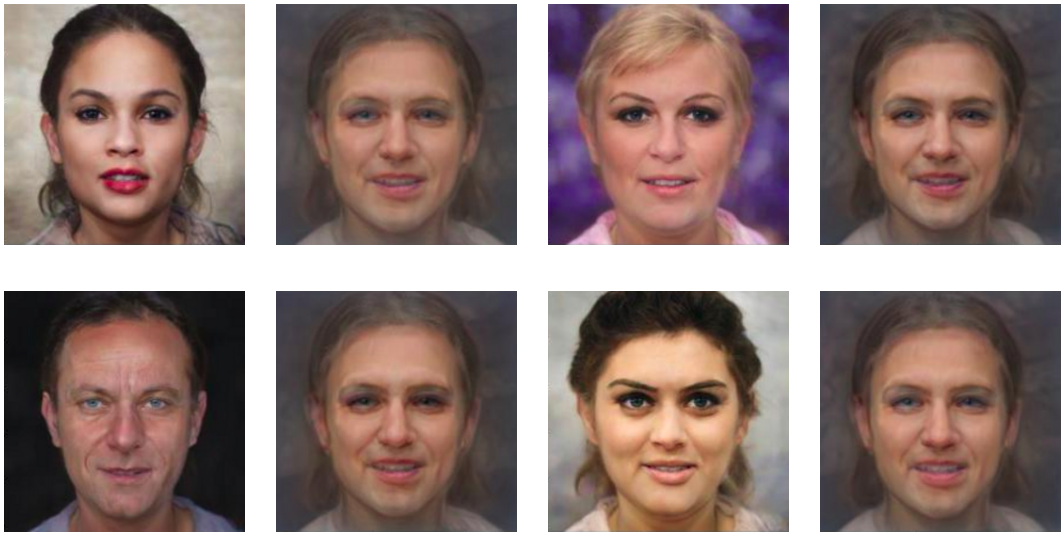
Na experimentu s minimálním datasetem jsem si ověřil, že můj systém by měl být schopen naučit se odhadovat obličej podle hlasové nahrávky mluvčího. Nejtěžší úkol ale byl pořád přede mnou, a to naučit systém generalizovat i mimo trénovací množinu dat. Nejdříve jsem chtěl tyto experimenty provádět nad celým svým datasetem, po několika spuštěních se však ukázalo, že pro trénování nad celým datasetem nebudu mít dostatečný výpočetní výkon. Jednotlivé experimenty by trvaly příliš dlouho. Rozhodl jsem se tedy dataset oříznout pouze na část z datové sady VoxCeleb2, i ta sama o sobě ale obsahovala něco málo přes půl milionu nahrávek řeči pro více 3 000 identit. Jedna epocha nad tímto datasetem trvala kolem 15 hodin. V rámci těchto experimentů jsem na vstup mého systému vkládal přímo audio nahrávky a ne vytvořené spektrogramy. Použití spektrogramů téměř vůbec neurychlilo trénování systému a přímo z nahrávky má enkodér šanci vytahovat příznaky, které se sám naučí během trénování.

Prvotní nastavení hyperparametrů jsem ponechal stejné, jako u experimentů s malým datasetem. Úplně první experiment nad velkým datasetem jsem nechal běžet 24 hodin, tzn. asi jeden a půl epochy. Už po této relativně krátké době trénování bylo jasné, že toto nastavení není nejšťastnější, protože se systém zasekl na jednom místě a pro všechny hlasové nahrávky generoval téměř totožný obličej, který se lišil pouze v minimálních detailech. Bylo jasné, že se systém zasekl v lokálním minimu chybové funkce a nedokázal se z něj dostat. Pro pár dalších experimentů jsem zkusil různě měnit nastavení koeficientů příslušných chybových funkci. Dokonce jsem přidal další upravené možnosti chybových funkcí *MSE* a *LPIPS* popsaných v kapitole o návrhu systému (3.3.1). Tyto modifikované verze počítají danou chybovou funkci pouze nad vyřezanou částí obrázku, kde se nachází obličej. Měly by tedy ignorovat pozadí obrázku, které je v našem případě nerelevantní pro výpočet chybové funkce. Po zavedení těchto nových částí chybové funkce jsem výrazně snížil koeficienty pro

staré verze *MSE* a *LPIPS* chybových funkcí a nastavil koeficienty vyšší pro nové verze. Výsledné nastavení jednotlivých koeficientů chybové funkce bylo následovné (upravené chybové funkce jsou označeny příponou $_{cropped}$):

- koeficient $MSE\ loss$ – $\lambda_1 = 0,01$;
- koeficient $LPIPS\ loss$ – $\lambda_2 = 0,01$;
- koeficient $ID\ loss$ – $\lambda_3 = 0,1$;
- koeficient $REGULARIZATION\ loss$ – $\lambda_4 = 0,0005$;
- koeficient $MSE_{cropped}\ loss$ – $\lambda_5 = 1$;
- koeficient $LPIPS_{cropped}\ loss$ – $\lambda_6 = 1$.

Ani tato změna chybové funkce však nepomohla k uniknutí z lokálního minima. Systém po určitém počtu kroků začal generovat stejné obličeje pro nahrávky různých identit a ani po delší době trénování (několik epoch), nebyl schopný z tohoto lokálního minima vyskočit. Dalším postupem tedy bylo zvýšení učícího koeficientu, aby optimalizační algoritmus dělal větší kroky pomocí vypočítaných gradientů a dokázal se dostat z míst, kde jsou lokální minima.



(a) (b) (a) (b)

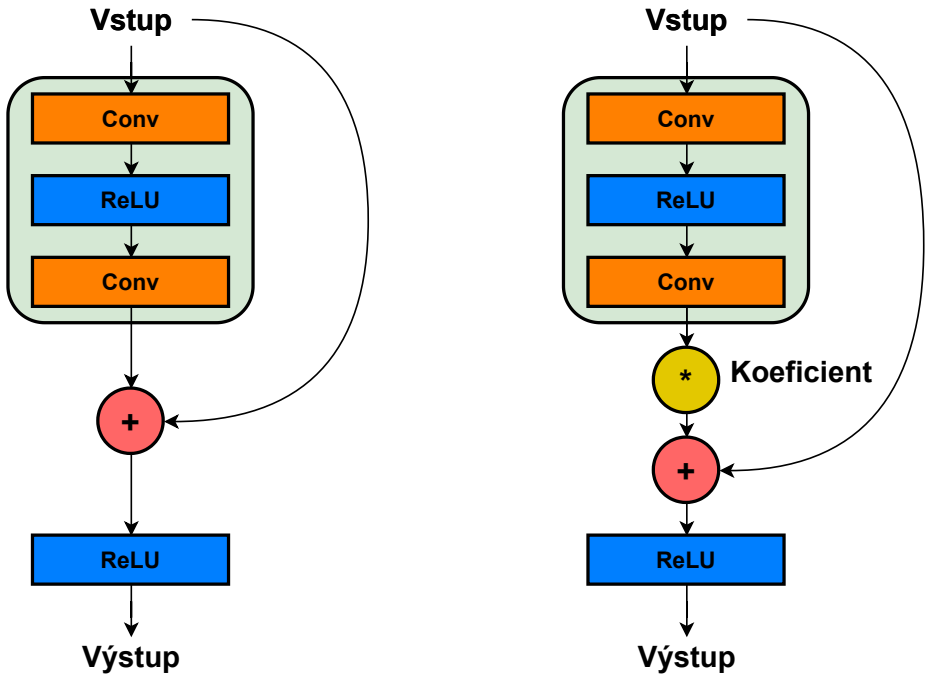
Obrázek 5.3: Ukázka výsledků trénování na datasetu VoxCeleb2. Ve sloupcích (a) jsou referenční snímky a ve sloupci (b) jsou snímky generované ze systému, který se během učení zaseknul v lokálním minimu chybové funkce, a generuje tak pro všechny vstupy stejný obličej.

Postupným zkoušením a zvětšováním učícího koeficientu jsem pomalu přišel do bodu, kdy systém začal být schopný vyskočit z lokálního minima chybové funkce. Systém začal konečně generovat pro nahrávky odlišných identit rozdílné obrázky. Po několika desítkách až stovkách kroků optimalizačního algoritmu bylo již na první pohled jasné, že obličeje z generovaných obrázků nepatří korespondujícím identitám. Tento fakt však byl pozorovatelný po krátké době trénování, takže jsem měl důvěru, že po několika epochách se začne odhad

obličejů zlepšovat. Na jednu epochu s takto velkým datasetem připadalo asi 65 000 kroků optimalizačního algoritmu. Systém se však díky velkému učícímu koeficientu dostal po několika tisících kroků do bodu, kdy začal generovat nesmyslné obrázky (připomínající pouze šum), pro které byla chybová funkce rovna hodnotě *NaN – Not a Number*. Z tohoto bodu se systém zase nedokázal dostat do místa chybové funkce, ve kterém by vracel rozumné výsledky. Tento problém je známý pod názvem *problém explodujících gradientů*. Vyskytuje se hlavně u hodně hlubokých neuronových sítí, kdy se pomocí zpětného šíření chyby akumulují gradienty až do velmi vysokých hodnot, které mají za následek obrovské kroky optimalizačního algoritmu a nestabilní trénování sítě. V extrémních případech, jako byl ten můj, můžou hodnoty vah neuronové sítě přetéct až do hodnoty *NaN*. Snížení učícího koeficientu však nepřipadalo v úvahu, protože by se systém zase začal zasekávat v lokálním minimu.

5.4.1 Úpravy systému

Jako první opatření, které může pomáhat proti problému explodujících gradientů (nebo i problému *mizejících gradientů*, což je opak explodujících gradientů), jsem implementoval tzv. *reziduální škálování* [58]. Jedná se o škálování výstupu jednotlivých reziduálních bloků pomocí koeficientu z intervalu $(0; 1)$. Škálování je umístěno před sčítání výstupu poslední vrstvy reziduálního bloku se vstupní hodnotou do reziduálního bloku přivedené na výstup pomocí reziduálního (nebo také dopředného) spojení. Na obrázku 5.4 můžete vidět ResNet blok bez reziduálního škálování a s ním.



Obrázek 5.4: Reziduální škálování.

Implementace reziduálního škálování sama o sobě však moc nepomohla. Po několika tisících iterací se problém začal stejně opakovat. Musel jsem se tedy uchýlit k implementaci tzv. *L2 regularizace*. Jedná se o praktiku, kdy se k hodnotě chybové funkce přičítá suma kvadrátů hodnot vah našeho modelu:

$$L = L' + wd * \sum_i w_i^2, \tag{5.1}$$

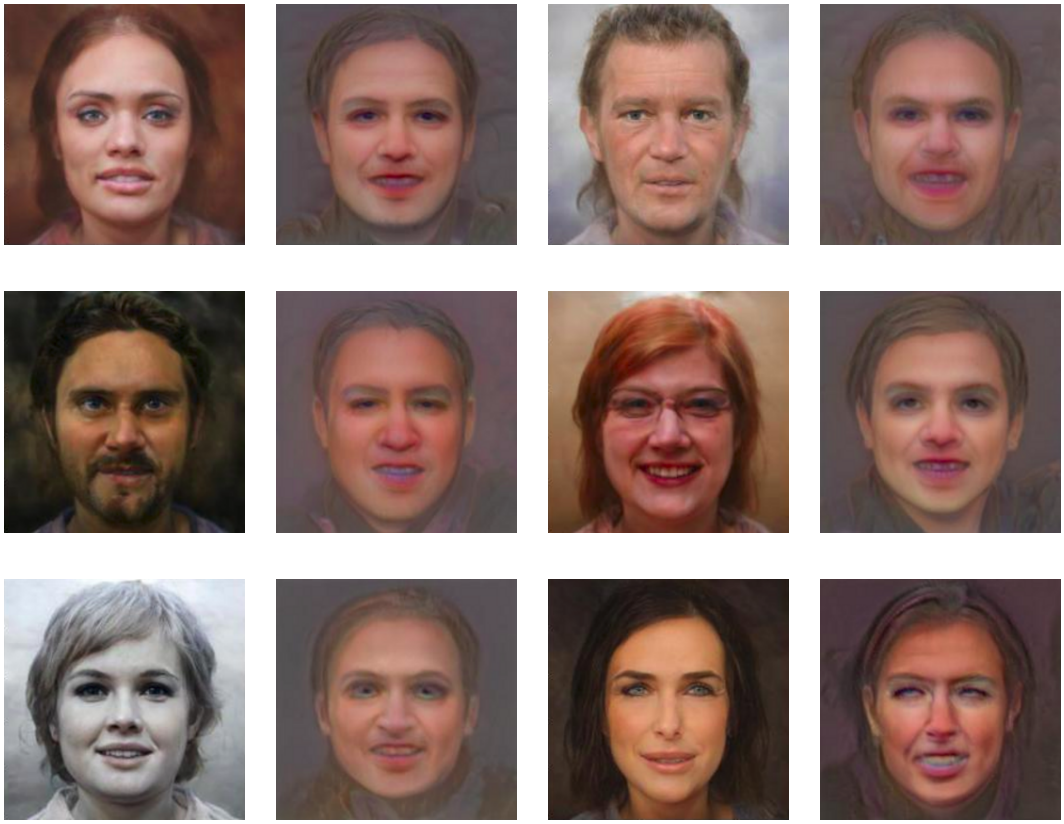
kde L je celková chybová funkce, L' je původní chybová funkce bez regularizace, wd je konstanta naší regularizace (tzv. *weight decay*) a w_i je i-tý parametr našeho modelu. Zavedení této metody zvyšuje chybovou funkci a tím automaticky snižuje hodnoty vah, pomocí kterých se potom počítají menší kroky optimalizačního algoritmu.

Samotná regularizace chybové funkce nebyla dostatečná pro stabilizaci učení systému. Poslední možností, která mi zbývala bylo použití techniky zvané *gradient clipping*. Jedná se o metodu, kdy se výsledná hodnota gradientu pro výpočet kroku chybové funkce ořízne na určitou prahovou hodnotu, pokud ji překročí. To nám umožní zmenšit velikost kroku optimalizačního algoritmu, který bude ale pořád moci postupovat správným směrem.

Aplikace všech výše zmíněných metod mi konečně umožnila trénovat systém, který se dokázal učit generovat odlišné obrázky obličejů pro nahrávky řeči odlišných identit. Identity z generovaných obrázků však neodpovídají identitám mluvčích z nahrávek. Dle obrázku 5.4.1, který ukazuje výsledky z tohoto modelu, můžete usoudit, že ani tento model nebyl pořád úspěšný v naučení se odhadovat obličej podle hlasové nahrávky. Jedním z problémů bylo, že byl můj model příliš velký, a k jeho natrénování by bylo potřeba opravdu velkého množství epoch. S mým omezeným výpočetním výkonem jsem se chtěl vydat jinou cestu. Rozhodl jsem se pro zmenšení modelu enkodéru nahrávek řeči a úpravu samotného trénování mého systému. Prvním krokem bylo použití dvou optimalizátorů – první pro učení enkodéru a druhý pro učení mapovacích sítí, které převádějí vektory příznaků na styly. Dva optimalizátory jsem se rozhodl použít, protože enkodér nahrávek řeči je velká konvoluční neuronová síť, a mapovací sítě příznaků na styly jsou o poznání menší. Přiřadil jsem tedy optimalizátorům pro tyto sítě odlišené učící koeficienty, aby se tyto modely trénovaly s odlišnou rychlostí a důrazem na chybovou funkci. Po zmenšení enkodéru se tento model skládal z pouze 4 reziduálních bloků a jedné vstupní vrstvy. Díky tomuto zmenšení se snížil i počet trénovatelných parametrů, takže by mělo být jednodušší tento model natrénovat. Další změnou v trénování bylo zavedení tzv. *warm-up fáze*, kdy se systém nejdříve trénuje s malým učícím koeficientem po dobu několika desítek tisíc kroků optimalizačního algoritmu. Po této warm-up fázi se učící koeficient začne pomalu zvyšovat až po určitou hodnotu, kterou jsem volil empiricky. Po dosažení této hodnoty se učící koeficient opět pomalu snižuje.

5.4.2 Předtrénování enkodéru

Samotné zmenšení modelu však moc nepomohlo. Učení trvalo pořád příliš dlouho a výsledky stále zůstávaly neuspokojivé. Napadlo mě tedy zkusit můj model předtrénovat na jednodušším úkolu, pomocí kterého by se enkodér naučil z nahrávek řeči extrahovat důležité informace specifické pro řečníka. Jako tento úkol jsem zvolil rozpoznávání řečníka (angl. *speaker recognition*). Pokud by se model naučit extrahovat informace o řečníkovi pro jeho rozpoznání, mělo by to pomoci i s extrahováním informací potřebných pro odhad obličeje. Implementovaný princip učení rozpoznávání řečníka není příliš složitý. Enkodér extrahuje z nahrávky řeči příznaky, které jsou poté předány plně propojeným vrstvám, které mají na výstupu stejný počet neuronů, jaký je počet identit všech řečníků v datasetu. Pro každou identitu je spočítána pravděpodobnost, že se jedná zrovna o ni, a poté je pomocí chybové funkce *Cross Entropy* spočítána aktuální hodnota chyby. *Cross Entropy* je popsána v kapitole o trénování neuronových sítí 1.2.2. Část pro extrakci příznaků zůstává samozřejmě stejná,



(a) (b) (a) (b)

Obrázek 5.5: Ukázka výsledků trénování na polovině datasetu VoxCeleb2. Ve sloupcích (a) jsou referenční snímky a ve sloupci (b) jsou snímky generované systémem, který se dokázal naučit odhadnout obličej pro identity obsažené v trénovací datové sadě řečových nahrávek. Systém dokázal dobře odhadovat tvar tváře, velikost nosu a úst, ale moc si neuměl poradit s barvou pleti nebo stářím mluvčího.

jediná změna v enkodéru pro rozpoznání řečníka je v tom, že za extraktor příznaků je napojena část s dvěma plně propojenými vrstvami pro samotné rozpoznání.

Model pro rozpoznání řečníků jsem trénoval po dobu 16 epoch na celém datasetu VoxCeleb2. Úspěšnost na konci trénování byla 89 %. Po konci tohoto předtrénování jsem natrénované parametry enkodéru načetl do modelu, který měl být trénován na odhad obličejů. Předtrénování na rozpoznání řečníka však nepřineslo kýžený úspěch. S největší pravděpodobností je to tím, že u rozpoznávání řečníka jde z enkodéru pouze jeden vektor příznaků z poslední vrstvy, který se poté zpracovává pomocí plně propojených sítí. U odhadu obličejů jsou z enkodéru extrahovány příznaky na třech úrovních v průběhu inference, ne až na konci. Příznaky z tří úrovní enkodéru, který je natrénován vracet relevantní příznaky až na výstupu sítě, tedy úplně nefungovaly pro generování správných stylů pro *StyleGAN*.

5.4.3 Finální systém

Všechny předešlé změny neměly na výsledky mého systému příliš velký vliv. Rozhodl jsem se proto pro razantnější změny v celém systému, ne jenom v rámci enkodéru. Jako první jsem zvýšil počet stylů, které vkládám na vstup *StyleGANu*. Do teď jsem využíval 14 stylů, podle kterých generátor upravuje výsledný obrázek. Tento počet stylů jsem využíval z důvodu

Přehled parametrů	
Enkodér	37 511 616
Mapovací sítě	33 057 792
Dekodér	30 370 060

Tabulka 5.1: Přehled počtu trénovatelných parametrů jednotlivých modelů poslední verze systému.

rozlišení, ve kterém jsem chtěl dostávat výsledné obrázky z generátoru. *StyleGAN* však nabízí možnost generovat snímky až v rozlišení 1024×1024, pro které potřebuje celkových 18 stylů. Generovat obrázky v tak vysokém rozlišení je pro můj úkol zbytečné. Získal jsem tak ale možnost předávat generátoru o 4 styly navíc, pomocí kterých může upravovat výsledný obrázek, a tím využít jeho plnou sílu. Cenou za toto zvýšení rozlišení výsledných obrázků bylo malé zvýšení výpočetního výkonu potřebného pro učení systému. Pro výpočet chybové funkce snižuji výsledným obrázkům rozlišení pomocí adaptivního average-poolingu na předpokládáných 256×256 pixelů.

Další změnou v pořadí bylo zavedení mé vlastní *Style Loss*. Při výpočtu této chybové funkce využívám práce *pSp* [52]. Jak již bylo popsáno dříve v této textové zprávě, *pSp* dokáže ze vstupního obrázku vytvořit takové styly pro *StyleGAN*, že *StyleGAN* velmi kvalitně duplikuje daný obrázek. *Style Loss* tedy nejdříve propaguje referenční obrázek obličeje systémem *pSp* pro získání referenčních stylů. Jako druhý krok jsou extrahovány styly pomocí mého enkodéru hlasových nahrávek. Hodnotou chybové funkce je pak rozdíl mezi referenčními styly z *pSp* a mými styly z enkodéru hlasových nahrávek. *Style Loss* zkusím použít jako jednu z částí chybové funkce, která by měla přiblížit extrahované styly z hlasových nahrávek blíže referenčním stylům, jejichž hodnoty správně reprezentují referenční obrázek obličeje.

Po pár experimentech se *Style Loss* bylo vidět, že se systém dokázal v odhadování obličejů posunout zase o kousek dál. Rozhodl jsem se znovu vyzkoušet razantně větší model enkodéru. Nakonec jsem skončil s modelem obsahujícím 32‘ResNet bloků – to tedy dává 1 vstupní konvoluční vrstvu a 64 konvolučních vrstev pro extrakci příznaků. Pro trénování tohoto modelu jsem využíval chybovou funkci jako v předchozích experimentech, ke které jsem navíc přidal *Style Loss*. Tato verze systému měla dohromady obrovský počet trénovatelných parametrů – konkrétně jich dohromady bylo 101 138 892 (počty jednotlivých modelů jsou uvedeny v tabulce 5.1) – a na grafické kartě při zpětném propagování chyby zabírá asi 34,6 GB při zpětném propagování chyby pro dávku dat o velikosti 16. Výsledky tohoto enkodéru byly ze všech dosavadních modelů nejlepší, ani tak nebyly dokonalé. Model se naučil odhadovat obličej pro identity řečníků, jejichž nahrávky obsahovala i trénovací sada. Pro identity, které systém neviděl při trénovaní, generoval hodně podobné obličeje. Pro nahrávky, které jako takové součástí trénovací sady nebyly, ale nahrávky od stejné identity ano, dokázal systém odhadovat obličeje celkem slušně. Pro natrénování však bylo potřeba využití *warm-up* fáze trénování. Ukázka úspěšně odhadnutých obličejů můžete vidět na obrázku 5.4.3. Dále na obrázku 5.4.3 je ukázáno několik špatných odhadů obličeje. Další výsledky na obrázcích v příloze A.

5.4.4 Využití cizího enkodéru

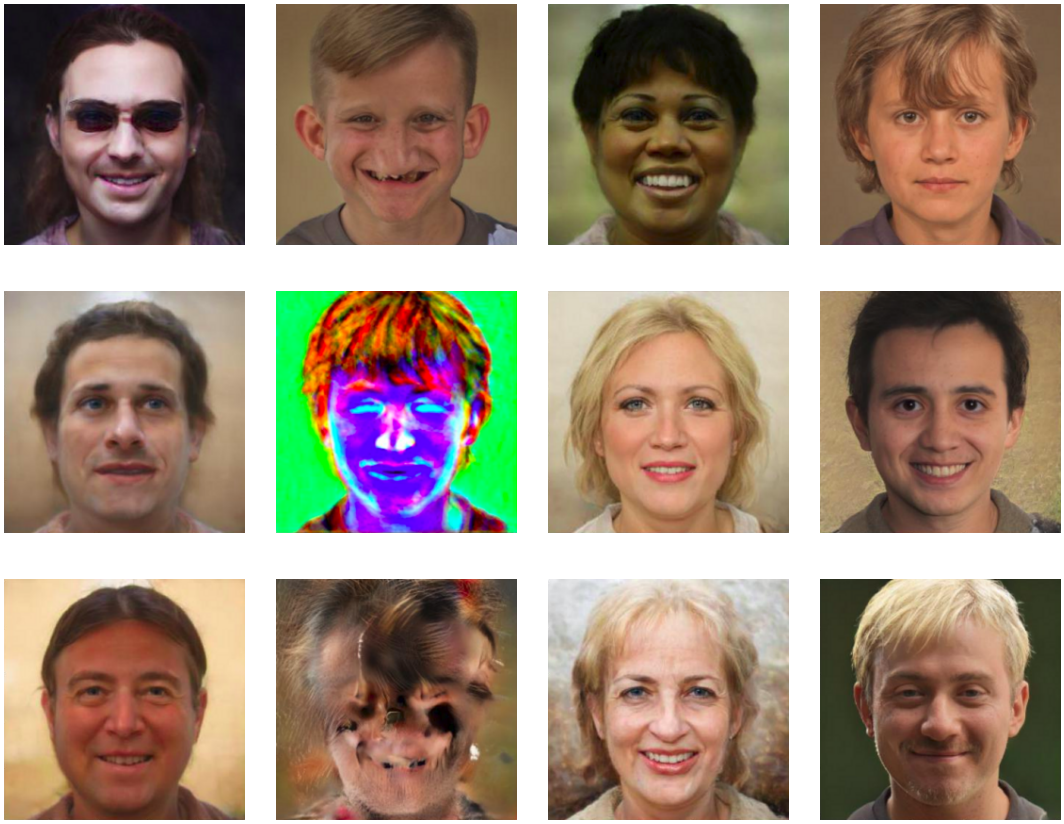
Úplně posledním experimentem bylo využití již hotového a předtrénovaného enkodéru hlasových nahrávek pro získání embeddingů nesoucí informace o řečníkovi. Tímto enkodérem je model z toolkitu *SpeechBrain* [51], který je natrénován pro úkol rozpoznávání řečníka a diari-

(a) (b) (a) (b)

Obrázek 5.6: Ukázka výsledků dosažených pomocí finálního systému trénovaného na polovině datasetu VoxCeleb2. Ve sloupcích (a) jsou referenční snímky a ve sloupci (b) jsou snímky generované tímto systémem. Tato verze systému dosáhla v odhadu obličejů nejlepších výsledků pro identity obsažené v trénovací datové sadě.

zaci řeči – tedy rozpoznání, který mluvčí v daný okamžik na nahrávce mluví. Tento model je založen na tzv. *ECAPA-TDNN* modelu, který dokáže zpracovávat různě dlouhé nahrávky postupně po jednotlivých rámcích, a generovat pro ně embedding vektory o konstantní velikosti. Více o tomto modelu se můžete dozvědět v článku z roku 2020 [21]. Implementaci tohoto modelu i natrénované parametry jsem získal z Python balíčku *speechbrain*[3]. Nebylo tedy složité jej zakomponovat do mého systému. Jedinou slabou stránkou tohoto předtrénovaného modelu je fakt, že vrací pro nahrávku pouze 1 embedding vektor, takže všech 18 stylů pro *StyleGAN* jsou vytvářeny z jediného vektoru. Generátor tedy nemá tolik informací o řečníkovi jako při vytváření stylů ze tří vektorů příznaků. Na druhou stranu, jeden vektor z tohoto předtrénovaného modelu by měl obsahovat kvalitnější a relevantnější informace o řečníkovi, než původní enkodér, který byl trénován pouze v rámci úlohy odhadu obličejů z řečové nahrávky. Při trénování mého systému bude při tomto experimentu enkodér zmrazen a jeho parametry se už v rámci učení nebudou měnit. Učit se budou pouze parametry mapovacích sítí, které převádí daný vektor příznaků na styly. Pro každý z 18 stylů se trénuje jedna malá konvoluční neuronová síť. Tato neuronová síť se skládá z kaskády konvolučních vrstev, které postupně snižují dimenzionalitu vstupního vektoru příznaků o délce 192 pouze

[3] https://pypi.org/project/speechbrain/



(a) (b) (a) (b)

Obrázek 5.7: Ukázka výsledků dosažených pomocí finálního systému trénovaného na polovině datasetu VoxCeleb2. Ve sloupcích (a) jsou referenční snímky a ve sloupci (b) jsou snímky generované tímto systémem. Systém si moc nedokázal poradit s některými identitami, které nebyly součástí trénovací sady. Problém mu dělalo odhadnutí jak stáří mluvčího tak některých základních rysů obličeje.

s jedním kanálem na vektor o délce 1 s 512 kanály. Tento vektor po pár následných úpravách reprezentuje daný styl.

Při trénování tohoto systému jsem dal největší důraz na chybovou funkci *ID Loss*, která by měla způsobit, že bude systém generovat obličeje, které budou mít základní rysy stejné, jako obličej řečníka. *ID Loss* však nezajistí odhad například správné barvy pleti nebo stáří mluvčího. Výstupem tedy určitě nebudou velmi podobné obličeje, jako jsou ty referenční, měla by mezi nimi však být jistá korelace, co se základních rysů týče – velikost nosu, tvar úst, šířka obličeje nebo tvar lícních kostí.

Kvůli omezenému výpočetnímu času na konci fáze experimentování jsem omezil dataset VoxCeleb2 pouze na 2000 identit, aby experimenty probíhaly rychleji.

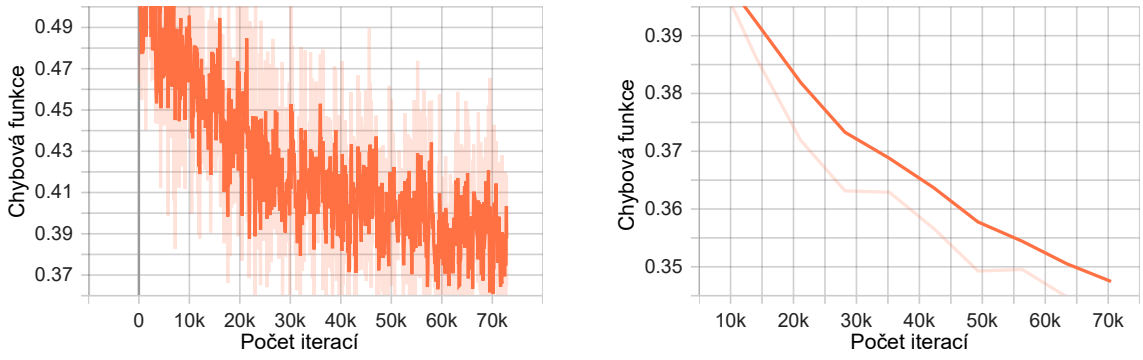
Bohužel, pouze z jednoho extrahovaného $1D$ embeddingu o délce 192 se mapovací sítě nedokázaly naučit vytvořit potřebné styly pro *StyleGAN*, v průběhu trénování se systém vždycky dostal velmi rychle do místa, kdy začal generovat pro všechny nahrávky stejný obličej. Jelikož neměly mapovací sítě dostatečně detailní informace, bylo pro systém nejjednodušší generovat pro všechny vstupy stejný výstup, pro který byla chybová funkce v nízkém bodě. Nepomohly ani změny učícího koeficientu nebo koeficientů jednotlivých částí chybové funkce.

5.5 Zhodnocení a návrh na pokračování

Ve fázi experimentů se mi podařilo natrénovat systém složený z několika modelů neuronových sítí, který byl schopen naučit se odhadovat obličej pro identity, které viděl během trénování, avšak i na nahrávkách, které součástí trénovací sady nebyly. Pro natrénování modelu tak, aby se naučil generalizovat i na identity mimo trénovací sadu, bych potřeboval daleko větší výpočetní výkon, protože i na zmenšené datové sadě se systém trénoval několik dní. Výše v této kapitole jsou popsány vybrané experimenty, které jsem prováděl s cílem co nejlepšího natrénování modelu. Od prvotních, ne příliš dobrých, výsledků se mi postupnými úpravami jednotlivých modelů a procesu učení podařilo posunout do finální fáze, kdy je systém schopen odhadovat obličeje identit obsažených v trénovací sadě velice hezky. U identit mimo trénovací sadu mu dělají problémy například děti nebo starší lidé, kteří nebyli v trénovací sadě tolik zastoupeni. Na obrázku 5.4.3 můžete vidět příklady těchto nepřesných odhadů. Na druhou stranu, na obrázku 5.4.3 jsou k vidění obrázky s celkem přesnými odhady obličejů, které obsahují stejné rysy, jako obličeje daných mluvčích z nahrávek.

Pokud bych měl pro experimentování s mým systémem k dispozici větší výpočetní kapacitu, určitě bych vyzkoušel trénování s celou datovou sadou, kterou jsem si sehnal, protože s omezenými daty, nad kterými jsem trénoval, se systém moc dobře nenaučil generalizovat mimo trénovací sadu. Trénování se všemi daty by však zabral o dost větší čas potřebný pro trénování. Zkusil bych tedy provést ablační studii, kdy bych zkoušel snižovat velikost a komplexnost jednotlivých částí systémů, abych zjistil, jak lze redukovat výpočetní náročnost bez zhoršení úspěšnosti systému.

Pro zajímavost jsem na konci fáze experimentování vyzkoušel spustit trénování mého systému na celé dostupné datové sadě. Systém se od prvních iterací dokázal učit a postupně malými kroky snižoval chybovou funkci. Její průběh je zobrazen na obrázku 5.8. Pro trénovací i testovací část datové sady je vidět postupný pokles hodnoty chybové funkce. Bohužel, jak již bylo zmíněno dříve, pro dotrénováni systému do podoby, kdy by byl schopen efektivně odhadovat obličej na základě řeči, jsem neměl pro takové množství dat dostatečný výpočetní výkon.



Průběh chybové funkce na trénovací sadě.

Průběh chybové funkce na testovací sadě.

Obrázek 5.8: Průběh chybové funkce při trénování na všech dostupných datech. Horizontální osa udává počet iterací trénovacího algoritmu a osa vertikální hodnotu chybové funkce.

V další fázi práce bych určitě vyzkoušel použít již předtrénovaný model pro extrakci embeddingů z řeči, který dokáže extrahovat relevantní informace o řečníkovi. Sám jsem

jeden z těchto modelů vyzkoušel. U mého experimentu byl problém, že daný model extrahoval pouze jeden vektor příznaků, ze kterého byly vytvářeny styly. V mnou navrženém systému jsou z enkodéru nahrávek extrahovány příznaky z více úrovní, které obsahují různě detailní informace o daném řečníkovi. Ke stejnému řešení by se dal použít model *UniSpeech-SAT* ze článku z roku 2021 [11], který na svém vstupu přijímá audio nahrávky, které zpracovává rámec po rámci. Jednotlivé rámce nejdříve vstupují do konvolučního enkodéru, jehož výstup je propagován do *Transformer* enkodérů, které pro každý rámec dávají na výstupu informaci o vstupní nahrávce. Podobně jako v práci [12], by šel tento model upravit tak, aby extrahoval informace z nahrávky řeči na různých úrovních inference. Z těchto informací by poté šly tvořit styly pro generátor *StyleGAN* pomocí mapovacích sítí. V tomto případě by se nejednalo o konvoluční neuronové sítě ale o sítě s architekturou *ECAPA-TDNN*, které by z výstupních hodnot jednotlivých rámců vytvořily za pomoci statistického poolingu jeden embedding vektor reprezentující informace o nahrávce. *UniSpeech-SAT* model je předtrénován na obrovském množství dat, takže bych měl jistotu, že extraktor informací z nahrávky pracuje správně a mohl bych se soustředit pouze na jeho ladění a na další části systému pro splnění mého cíle.

Důvoded proč jsem tento přístup s modelem *UniSpeech-SAT* nevyzkoušel, je jeho výpočetní náročnost. Tento model má kolem sto milionů trénovatelných parametrů a ladění jeho napojení na můj systém by s mými HW prostředky zabralo opravdu velké množství času.

Závěr

Hlavním cílem této diplomové práce byl návrh a natrénování systému, který bude schopen generovat obličej na základě vstupní nahrávky řeči. Mým cílem však nebylo odhadovat naprostou kopii obličeje, ale spíše zjistit, jaké by mluvčí z nahrávky měl mít rysy obličeje, které mají vliv na to, jak člověk mluví.

V rámci práce jsou tedy navrženy dva modely konvolučních neuronových sítí. Prvním z nich je enkodér hlasových nahrávek, ze kterého jsou během inference extrahovány příznaky s různě detailními informacemi o vstupní nahrávce. Tyto příznaky jsou poté převedeny na takzvané styly pomocí druhého modelu konvoluční neuronové sítě. Vytvořené styly jsou předávány na vstup generátoru *StyleGAN*, který jsem převzal již předtrénovaný, protože natrénovat vlastní dekodér obličejů v takové kvalitě by zabralo obrovské množství času a výpočetního výkonu. Finální podoba jednotlivých modelů se od návrhu mírně liší, protože je bylo potřeba upravovat pro co nejlepší odladění systému.

Navržený systém a jeho trénování bylo implementováno v jazyce Python za pomoci frameworku PyTorch, který poskytuje funkcionalitu pro implementaci vlastních modelů různých typů neuronových sítí. V rámci práce byla dále implementována funkcionalita pro normalizaci fotek obličejů a skript pro stahování dat z datasetu AVSpeech.

Pro trénování systému jsem vytvořil dataset složený z datových sad VoxCeleb, VGGFace a AVSpeech obsahující dvojice fotek obličejů s korespondujícími nahrávkami řeči. Kvůli nedostatku výpočetní kapacity jsem však nebyl schopný všechna data využít a systém jsem trénoval na zredukované datové sadě.

Na začátku fáze experimentování nebyly výsledky původního modelu moc kvalitní. Systém se sice dokázal naučit odhadovat obličej na malé datové sadě, na velkém vzorku dat měl však systém problém a pro jakákoliv vstupní data generoval stejný obličej. Po změnách popsaných v poslední kapitole této práce se mi s problémy však podařilo vypořádat a model se naučil odhadovat obličej pro identity, které viděl v rámci trénování. Pro nahrávky identit mimo testovací sadu měl problém například pro děti nebo starší lidi. Dále se moc nedokázal naučit odhadovat barvu pleti a etnickou příslušnost mluvčího z nahrávky. V jedné z posledních částí experimentů jsem zkusil využít již předtrénovaný enkodér nahrávek z toolkitu *SpeechBrain*, pomocí kterého se mi však nepodařilo dosáhnout dobrých výsledků, jelikož možnosti jeho napojení na zbytek mého systému nebyly ideální. Po dalších experimentech a dalších úpravách mých modelů jsem se dostal k finální podobě systému, který se v kvalitě generovaných obrázků opět posunul dále. Systém se naučil odhadovat základní rysy obličeje identit, které byly součástí trénovací sady. Zároveň se zlepšila jeho generalizace mimo trénovací sadu. Pro další zlepšení v generalizaci bych musel využít větší datovou sadu, pro kterou jsem však neměl dostatečnou výpočetní kapacitu.

Návrh na pokračování v této práci je popsán na konci kapitoly Experimenty. Zaměřuje se na využití již předtrénovaného enkodéru na obrovském množství dat a způsob jeho napojení na zbytek mnou navrženého systému.

Literatura

[1] *Feature Extraction Linear Filters*. 2005. [online].[cit. 16. 1. 2022]. Dostupné z: http://users.umiacs.umd.edu/~ramani/cmsc426/Lecture5.pdf.

[2] *Linear classification: Support Vector Machine, Softmax*. 2019. [online].[cit. 18. 3. 2019]. Dostupné z: http://cs231n.github.io/linear-classify/.

[3] *Hyperbolický tangens*. 2021. [online].[cit. 12. 1. 2022]. Dostupné z: https://cs.wikipedia.org/wiki/Hyperbolick%C3%BD_tangens.

[4] *LibriVox*. 2021. [online].[cit. 6. 1. 2021]. Dostupné z: https://librivox.org/.

[5] *Logistic function*. 2021. [online].[cit. 12. 1. 2022]. Dostupné z: https://en.wikipedia.org/wiki/Logistic_function.

[6] *Using display.specshow*. 2021. [online].[cit. 16. 1. 2022]. Dostupné z: https://librosa.org/doc/main/auto_examples/plot_display.html.

[7] Abdal, R., Qin, Y. a Wonka, P. Image2StyleGAN++: How to Edit the Embedded Images? *CoRR*. 2019, abs/1911.11544. Dostupné z: http://arxiv.org/abs/1911.11544.

[8] Abdal, R., Qin, Y. a Wonka, P. Image2StyleGAN: How to Embed Images Into the StyleGAN Latent Space? *CoRR*. 2019, abs/1904.03189. Dostupné z: http://arxiv.org/abs/1904.03189.

[9] Cao, Q., Shen, L., Xie, W., Parkhi, O. M. a Zisserman, A. *VGGFace2: A dataset for recognising faces across pose and age*. 2018.

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

[11] Chen, S., Wu, Y., Wang, C., Chen, Z., Chen, Z. et al. UniSpeech-SAT: Universal Speech Representation Learning with Speaker Aware Pre-Training. *CoRR*. 2021, abs/2110.05752. Dostupné z: https://arxiv.org/abs/2110.05752.

[12] Chen, Z., Chen, S., Wu, Y., Qian, Y., Wang, C. et al. Large-scale Self-Supervised Speech Representation Learning for Automatic Speaker Verification. *CoRR*. 2021, abs/2110.05777. Dostupné z: https://arxiv.org/abs/2110.05777.

[13] Chung, J. S., Nagrani, A. a Zisserman, A. VoxCeleb2: Deep Speaker Recognition. In: *INTERSPEECH*. 2018.

[14] Chung, J. S. a Zisserman, A. Lip Reading in the Wild. In: Lai, S.-H., Lepetit, V., Nishino, K. a Sato, Y., ed. *Computer Vision – ACCV 2016*. Cham: Springer International Publishing, 2017, s. 87–103. ISBN 978-3-319-54184-6.

[15] Chung, J. S. a Zisserman, A. Out of Time: Automated Lip Sync in the Wild. In: Chen, C.-S., Lu, J. a Ma, K.-K., ed. *Computer Vision – ACCV 2016 Workshops*. Cham: Springer International Publishing, 2017, s. 251–263. ISBN 978-3-319-54427-4.

[16] Chung, J. S. a Zisserman, A. Learning to lip read words by watching videos. *Computer Vision and Image Understanding*. 2018, sv. 173, s. 76–85. DOI: https://doi.org/10.1016/j.cviu.2018.02.001. ISSN 1077-3142. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1077314218300134.

[17] Cole, F., Belanger, D., Krishnan, D., Sarna, A., Mosseri, I. et al. Face Synthesis from Facial Identity Features. *CoRR*. 2017, abs/1701.04851. Dostupné z: http://arxiv.org/abs/1701.04851.

[18] Davis, S. a Mermelstein, P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. 1980, sv. 28, č. 4, s. 357–366. DOI: 10.1109/TASSP.1980.1163420.

[19] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. et al. ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*. 2009. ISSN 1063-6919.

[20] 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.

[21] Desplanques, B., Thienpondt, J. a Demuynck, K. ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification. In: *Interspeech 2020*. ISCA, Oct 2020. DOI: 10.21437/interspeech.2020-2650. Dostupné z: https://doi.org/10.21437%2Finterspeech.2020-2650.

[22] Duarte, A. C., Roldan, F., Tubau, M., Escur, J., Pascual, S. et al. Wav2Pix: Speech-conditioned Face Generation using Generative Adversarial Networks. *CoRR*. 2019, abs/1903.10195. Dostupné z: http://arxiv.org/abs/1903.10195.

[23] Ephrat, A., Mosseri, I., Lang, O., Dekel, T., Wilson, K. et al. Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation. *ArXiv preprint arXiv:1804.03619*. 2018.

[24] Everingham, M., Sivic, J. a Zisserman, A. Taking the bite out of automated naming of characters in TV video. *Image Vis. Comput.* 2009, sv. 27, s. 545–559.

[25] G., A. a Ramakrishnan, A. *Music and Speech Analysis Using the 'Bach' Scale Filter-Bank*. Disertační práce.

[26] Gimel'farb, G. *Image Filtering and Segmentation*. 2013. [online].[cit. 16. 1. 2022]. Dostupné z: https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/2013/CS373-IP-03.pdf.

[27] Goodfellow, I., Bengio, Y. a Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[28] Goodfellow, I., Pouget Abadie, J., Mirza, M., Xu, B., Warde Farley, D. et al. Generative Adversarial Nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. a Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, s. 2672–2680.

[29] Härkönen, E., Hertzmann, A., Lehtinen, J. a Paris, S. GANSpace: Discovering Interpretable GAN Controls. *CoRR*. 2020, abs/2004.02546. Dostupné z: https://arxiv.org/abs/2004.02546.

[30] Haykin, S. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[31] He, K., Zhang, X., Ren, S. a Sun, J. Deep Residual Learning for Image Recognition. *CoRR*. 2015, abs/1512.03385. Dostupné z: http://arxiv.org/abs/1512.03385.

[32] Hoover, K., Chaudhuri, S., Pantofaru, C., Slaney, M. a Sturdy, I. Putting a Face to the Voice: Fusing Audio and Visual Signals Across a Video to Determine Speakers. *CoRR*. 2017, abs/1706.00079. Dostupné z: http://arxiv.org/abs/1706.00079.

[33] 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.

[34] Ioffe, S. a Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*. 2015, abs/1502.03167.

[35] Jahanian, A., Chai, L. a Isola, P. On the "steerability"of generative adversarial networks. *CoRR*. 2019, abs/1907.07171. Dostupné z: http://arxiv.org/abs/1907.07171.

[36] Karras, T., Laine, S. a Aila, T. A Style-Based Generator Architecture for Generative Adversarial Networks. *CoRR*. 2018, abs/1812.04948. Dostupné z: http://arxiv.org/abs/1812.04948.

[37] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J. et al. Analyzing and Improving the Image Quality of StyleGAN. *CoRR*. 2019, abs/1912.04958. Dostupné z: http://arxiv.org/abs/1912.04958.

[38] King, D. E. Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*. 2009, sv. 10, č. 60, s. 1755–1758. Dostupné z: http://jmlr.org/papers/v10/king09a.html.

[39] Kingma, D. P. a Ba, J. Adam: A method for stochastic optimization. In: *International Conference on Learning Representations (ICLR)*. 2015.

[40] Krizhevsky, A., Sutskever, I. a Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. a Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, s. 1097–1105.

[41] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E. et al. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.* Cambridge, MA, USA: MIT Press. prosinec 1989, sv. 1, č. 4, s. 541–551. DOI: 10.1162/neco.1989.1.4.541. ISSN 0899-7667.

[42] Liu, D. *A Practical Guide to ReLU.* 2017. [online].[cit. 12. 1. 2022]. Dostupné z: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7.

[43] McLaren, M., Ferrer, L., Castán, D. a Lawson, A. D. The Speakers in the Wild (SITW) Speaker Recognition Database. In: *INTERSPEECH.* 2016.

[44] Mehrotra, K., Mohan, C. K. a Ranka, S. *Elements of Artificial Neural Networks.* Cambridge, MA, USA: MIT Press, 1996. ISBN 0262133288.

[45] Munakata, T. *Fundamentals of the New Artificial Intelligence: Neural, Evolutionary, Fuzzy and More (Texts in Computer Science).* 2nd ed. Springer Publishing Company, Incorporated, 2008. ISBN 184628838X.

[46] Nagrani, A., Chung, J. S. a Zisserman, A. VoxCeleb: a large-scale speaker identification dataset. In: *INTERSPEECH.* 2017.

[47] Nair, V. a Hinton, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning.* USA: Omnipress, 2010, s. 807–814. ICML'10. ISBN 978-1-60558-907-7.

[48] Oh, T., Dekel, T., Kim, C., Mosseri, I., Freeman, W. T. et al. Speech2Face: Learning the Face Behind a Voice. *CoRR.* 2019, abs/1905.09773. Dostupné z: http://arxiv.org/abs/1905.09773.

[49] Oord, A. van den, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O. et al. WaveNet: A Generative Model for Raw Audio. *CoRR.* 2016, abs/1609.03499. Dostupné z: http://arxiv.org/abs/1609.03499.

[50] Parkhi, O. M., Vedaldi, A. a Zisserman, A. Deep Face Recognition. In: *British Machine Vision Conference.* BMVA Press, 2015, s. 41.1–41.12. ISBN 1-901725-53-7.

[51] Ravanelli, M., Parcollet, T., Plantinga, P., Rouhe, A., Cornell, S. et al. *SpeechBrain: A General-Purpose Speech Toolkit.* 2021. ArXiv:2106.04624.

[52] Richardson, E., Alaluf, Y., Patashnik, O., Nitzan, Y., Azar, Y. et al. Encoding in Style: a StyleGAN Encoder for Image-to-Image Translation. *CoRR.* 2020, abs/2008.00951. Dostupné z: https://arxiv.org/abs/2008.00951.

[53] Rosenblatt, F. *The perceptron - A perceiving and recognizing automaton.* 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory, January 1957.

[54] Sainath, T. N., Weiss, R. J., Senior, A. W., Wilson, K. W. a Vinyals, O. Learning the speech front-end with raw waveform CLDNNs. In: *INTERSPEECH.* 2015.

[55] 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.

[56] Shen, Y., Gu, J., Tang, X. a Zhou, B. Interpreting the Latent Space of GANs for Semantic Face Editing. *CoRR*. 2019, abs/1907.10786. Dostupné z: http://arxiv.org/abs/1907.10786.

[57] Shrivakshan, G. a Chandrasekar, C. A Comparison of various Edge Detection Techniques used in Image Processing. *International Journal of Computer Science Issues*. Září 2012, sv. 9, s. 269–276.

[58] Szegedy, C., Ioffe, S., Vanhoucke, V. a Alemi, A. A. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In: *CoRR*. 2016.

[59] Tewari, A., Elgharib, M., Bharaj, G., Bernard, F., Seidel, H. et al. StyleRig: Rigging StyleGAN for 3D Control over Portrait Images. *CoRR*. 2020, abs/2004.00121. Dostupné z: https://arxiv.org/abs/2004.00121.

[60] 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.

[61] Zhang, K., Zhang, Z., Li, Z. a Qiao, Y. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks. *CoRR*. 2016, abs/1604.02878. Dostupné z: http://arxiv.org/abs/1604.02878.

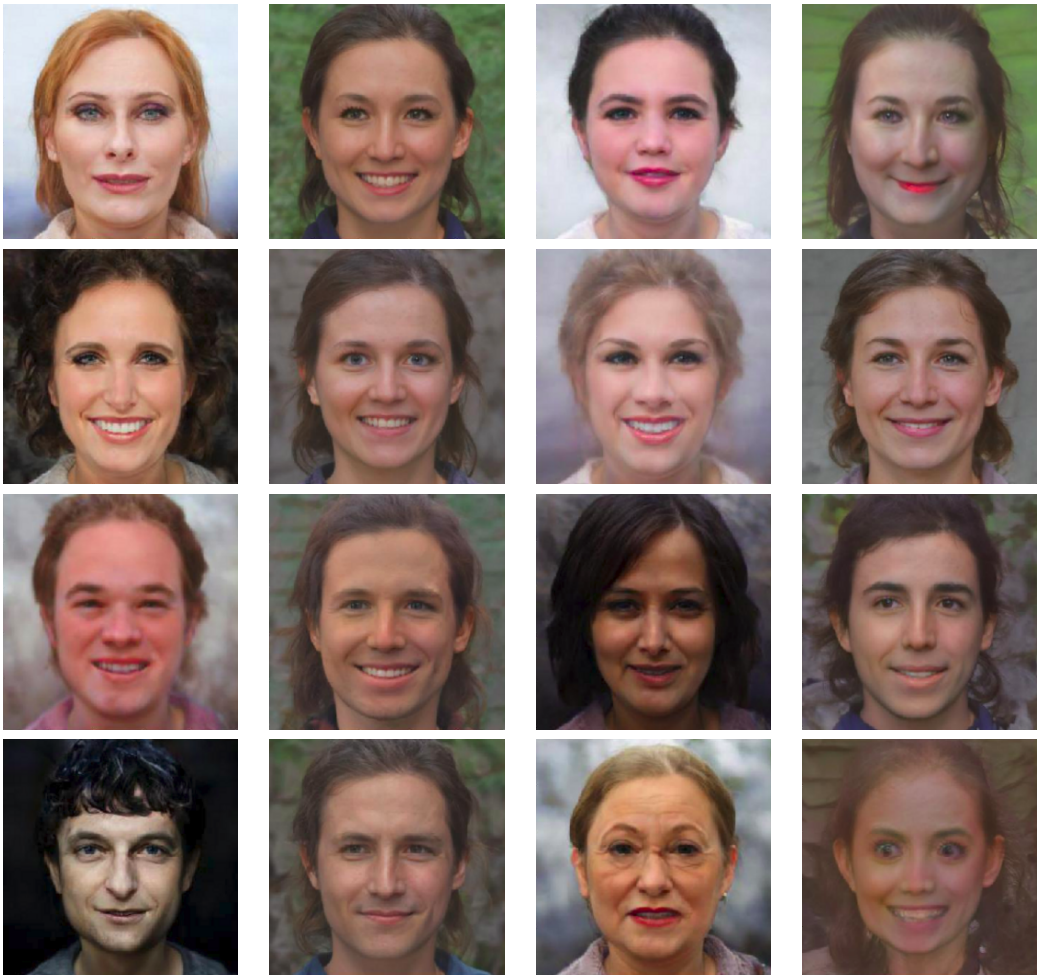
[62] Zhang, R., Isola, P., Efros, A. A., Shechtman, E. a Wang, O. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. *CoRR*. 2018, abs/1801.03924. Dostupné z: http://arxiv.org/abs/1801.03924.

[63] Zhu, J., Shen, Y., Zhao, D. a Zhou, B. In-Domain GAN Inversion for Real Image Editing. *CoRR*. 2020, abs/2004.00049. Dostupné z: https://arxiv.org/abs/2004.00049.

[64] Černocký, H. *Zpracování řečových signálů – studijní opora*. 2006. [online].[cit. 16. 1. 2022]. Dostupné z: http://www.fit.vutbr.cz/study/courses/ZRE/public/opora/zre_opora.pdf.

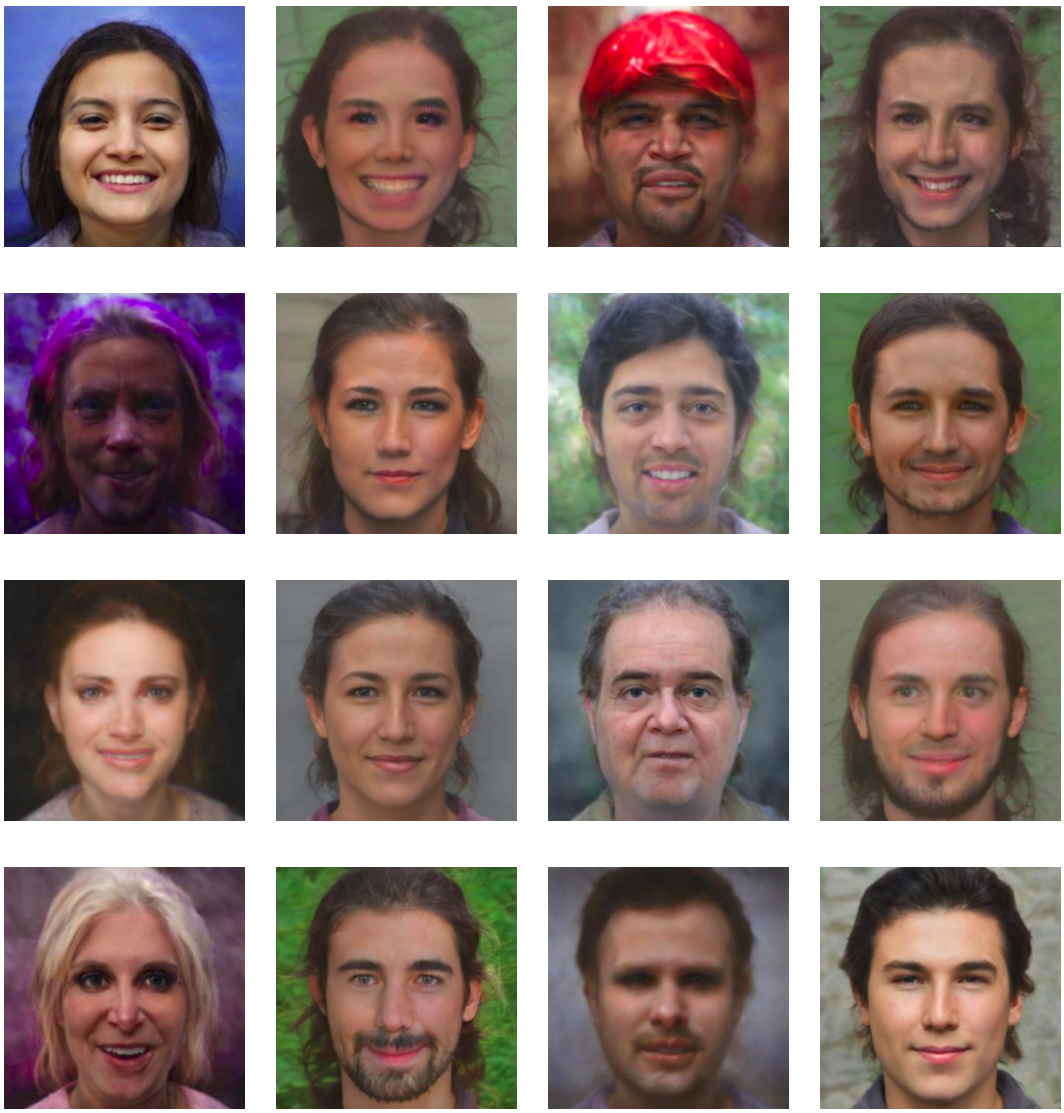
Příloha A

Výsledky systému



(a) (b) (a) (b)

Obrázek A.1: Ukázka výsledků dosažených pomocí finálního systému trénovaného na polovině datasetu VoxCeleb2. Ve sloupcích (a) jsou referenční snímky a ve sloupci (b) jsou snímky generované mým systémem.



(a) (b) (a) (b)

Obrázek A.2: Ukázka výsledků dosažených pomocí finálního systému trénovaného na polovině datasetu VoxCeleb2. Ve sloupcích (a) jsou referenční snímky a ve sloupci (b) jsou snímky generované mým systémem.

Příloha B

Obsah datového média

Přiložené datové médium obsahuje následující adresářovou strukturu:

- **src/** – zde jsou obsaženy veškere zdrojové kódy implementovaných modelů a trénovacího algoritmu.
  - **pretrained_models/** – v tomto adresáři jsou uloženy předtrénované modely neuronových sítí, které jsem převzal již hotové,
  - **checkpoints/** – v tomto adresáři jsou uloženy natrénované modely mých neuronových sítí,
  - **data_preprocessing/** – zde jsou k nalezení skripty, které jsem implementoval pro předzpracování dat,
  - **datasets/** – v tomto modulu je implementována třída pro načítání datasetu,
  - **models/** – v tomto modulu je k nalezení implementace všech modelů neuronových sítí, které v práci využívám:
    - **encoders/** – implementace všech použitých enkodérů,
    - **stylegan2/** – implementace generátoru *StyleGAN*,
    - **voice_styled_fac**e – implementace mého systému nazvaného *Voice Styled Face*.
  - **train/** – v tomtu modulu je k nalezení implementace trénování systému:
    - **loss/** – implementace veškeré funkcionality pro počítání chybové funkce.
  - **utils/** – implementace funkcionality, která byla potřeba napříč celým projektem – logování, tvorba grafů, ukládání obrázků.
- **src_doc/** – tento adresář obsahuje zdrojové kódy textové části práce.
- **doc/** – zde je uložena textová část práce ve formátu pdf.
- README.md – popis všech částí a návod na spuštění systému.