
**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**


# GENEROVÁNÍ OBLIČEJŮ S POMOCÍ PODMÍNĚNÝCH GENERATIVNÍCH NEURONOVÝCH SÍTÍ

**GENERATING FACES WITH CONDITIONAL GENERATIVE ADVERSARIAL NETWORKS**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **TOMÁŠ VENKRBEC**
**AUTHOR**

**VEDOUCÍ PRÁCE** **MARTIN KOLÁŘ, M.Sc.**
**SUPERVISOR**



**BRNO 2020**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

# Zadání bakalářské práce


Student: **Venkrbec Tomáš**
Program: Informační technologie
Název: **Generování obličejů s pomocí podmíněných generativních neuronových sítí**
**Generating Faces with Conditional Generative Adversarial Networks**
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s problematikou cGANů
2. Získejte vhodnou datovou sadu
3. Proveďte implementaci
4. Natrénujte několik variant modelů
5. Proveďte evaluaci výsledků

Literatura:


- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).

- Karras, T., Aila, T., Laine, S. and Lehtinen, J., 2017. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

- Karras, Tero, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.
- Goodfellow, Ian. "NIPS 2016 tutorial: Generative adversarial networks." *arXiv preprint arXiv:1701.00160* (2016).


Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

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

Vedoucí práce: **Kolář Martin, M.Sc.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 21. května 2020

# Abstrakt

Cílem této práce je implementace a porovnání modelů různých architektur podmíněných generativních neuronových sítí. Jejich účelem je podmíněné generování realisticky vypadajících lidských obličejů s vybranými rysy. Byly porovnány výsledky z modelů architektur DCGAN, WGAN-GP a ProGAN, k jejichž implementaci bylo využito knihovny Tensorflow. Trénování modelů neuronových sítí probíhalo na datové sadě Flickr-Faces-HQ. Napříč všemi použitými architekturami se podařilo natrénovat modely generující realistické lidské obličeje, s možností výběru vzhledu podle pohlaví a věku.



# Abstract

The main goal of this thesis is to implement and compare models based on various architectures of conditional generative adversarial networks. Their main purpose is to conditionally generate realistic looking human faces with selected features. Results from models using DCGAN, WGAN-GP and ProGAN architectures were compared. Models were implemented using Tensorflow library and were trained on Flickr-Faces-HQ dataset. Across all used architectures, I managed to train models capable of generating realistic human faces, with an option to select age and gender.



# Klíčová slova

podmíněné generativní neuronové sítě, generování obličejů, hluboké učení, strojové učení, umělá inteligence

# Keywords

conditional generative adversarial networks, neural networks, generating faces, deep learning, machine learning, artificial intelligence


# Citace

VENKRBEC, Tomáš. *Generování obličejů s pomocí podmíněných generativních neuronových sítí*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Martin Kolář, M.Sc.

# Generování obličejů s pomocí podmíněných generativních neuronových sítí

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Martina Koláře, M.Sc. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Venkrbec
4. června 2020


## Poděkování

Rád bych poděkoval vedoucímu mojí bakalářské práce, panu Martinovi Kolářovi, M.Sc. za pomoc při jejím řešení a za mnoho cenných rad a připomínek. Děkuji také celé svojí rodině, za všechnu podporu a trpělivost. V neposlední řadě chci poděkovat také všem svým přátelům, kteří mi průběh prací zpříjemňovali.

# Obsah

# Kapitola 1

# Úvod

Umělá inteligence patří v dnešní době mezi nejrychleji rostoucí obory informačních technologií. Své uplatnění nachází v široké škále technologických i netechnologických odvětví. Mezi významné oblasti umělé inteligence se řadí strojové učení, jehož koncept fascinuje také veřejnost a stává se dokonce námětem mnohých vědeckofantastických uměleckých děl. Zároveň se jedná o téma, které ve společnosti budí emoce díky svým nejrůznějším uplatněním, z nichž některé jsou považovány za kontroverzní zásahy do soukromí nebo do bezpečnosti. Nejedná se přitom o novou disciplínu, samotný pojem *strojové učení* vznikl na konci 50. let minulého století. S pokračujícím vývojem strojového učení a objevením principu neuronových sítí bylo definováno ještě jedno odvětví - *hluboké učení*. To je specifickou podskupinou strojového učení, zaměřující se na učení pomocí řady napojených vrstev se stále smysluplnější reprezentací dat [4]. Především díky do nedávné doby exponenciálnímu růstu výkonu počítačů odpovídajícímu slavnému Mooreovu zákonu [22], ale také neustále pokračujícímu bádání mnoha výzkumníků je v dnešní době hluboké učení skvělým postupem pro řešení velkého množství jak klasifikačních, tak generativních úloh.

Jak již samotný název napovídá, *generativní neuronové sítě* (angl. *generative adversarial networks, GAN*) slouží především k řešení úloh generativních. Na rozdíl od klasifikačních neuronových sítí je v generativních neuronových sítích používána dvojice neuronových sítí, a to za prvé *diskriminátor*, plnící tradiční klasifikační funkci, a za druhé *generátor*, který má za úkol tvořit unikátní vzorky, v ideálním případě diskriminátorem nerozeznatelné od těch skutečných. Tyto sítě spolu navzájem soupeří, čímž se obě učí a zdokonalují. Ačkoliv se jedná o koncept představený teprve v roce 2014, tak jím bylo umožněno velmi rychle pokročit ve vývoji neuronových sítí. Z nepřeberného množství aplikací této technologie lze zmínit například generování realistických obrázků jak lidských osob, tak předmětů, kreslených postav, či dokonce uměleckých děl [16]. Kromě generování nových vzorků je také možné převádět již existující obrázky na jiné (angl. *image-to-image translation*), jako příklad může sloužit generování barevných obrázků z černobílých, fotografií z pouhých jednoduchých nákresů, tvoření map ze satelitních snímků [39, 36], ale také výměna osoby na videu za jinou osobu (tzv. *DeepFake* [34]). Je také možné provádět transformace, při kterých jsou chybějící informace predikovány. Tímto lze například několikanásobně zvýšit rozlišení snímku a zároveň s tím zvýšit i jeho kvalitu (tzv. *supersampling* [3, 37]).

Cílem této práce je implementovat modely různých architektur neuronových sítí a pomocí nich generovat lidské obličeje. Jak bylo již výše zmíněno, generování obličejů není nová disciplína, ovšem velká část již existujících řešení se zabývala pouze generováním nepodmíněným. Tato práce se zabývá podmíněným generováním. Podmíněné generování v praxi znamená to, že máme dostupné informace (angl. *label*) o jednotlivých vzorcích z datové

sady, což umožňuje klasifikaci vzorků sítěmi do tříd. Při generování je možné určit, do které třídy má vygenerovaný vzorek patřit. Pro představu, různé třídy mohou reprezentovat například věk, pohlaví, styl účesu, barvu vlasů generované osoby, či to, zdali jsou přítomny doplňky jako třeba brýle. U nepodmíněného generování taková možnost výběru neexistuje.

Shrnutí fundamentálních principů hlubokého učení a neuronových sítí, nutných pro pochopení pozdějších částí práce, se věnuje kapitola 2. V kapitole 3 je věnována pozornost fungování podmíněných generativních neuronových sítí, včetně ukázek výsledků různých generativních úloh, kde je tato technologie používána. Zároveň zde jsou porovnány nejmodernější postupy generování lidských obličejů. Kapitola 4 prozkoumává návrh řešení, jsou zde představeny knihovny používané k vývoji neuronových sítí a také jsou zde porovnány dostupné datové sady lidských obličejů. Následně jsou v kapitole 5 čtenáři přiblíženy konkrétní implementační detaily jednotlivých vybraných architektur generativních neuronových sítí. Výsledky trénování implementovaných modelů jsou vyhodnoceny v kapitole 6 a závěrečná kapitola 7 je ohlédnutím za dosaženými výsledky a jsou zde popsány možnosti pokračování vývoje projektu.

# Kapitola 2

# Úvod do neuronových sítí

Inspirací pro mnoho technologií z oblasti umělé inteligence a stejně tak i pro neuronové sítě se stala samotná příroda. V této kapitole se tedy v sekci 2.1 podíváme, jak je toto „spojení“ počítačů a přírody uskutečněno. Po této biologické motivaci je v sekci 2.2 představeno, jak samotné neuronové sítě fungují. V sekci 2.3 je řeč o principu konvolučních neuronových sítí, které jsou nejmodernějším řešením nejen pro účely zpracování obrazu.

## 2.1 Neuron a jeho počítačová reprezentace

Neuron je nervová buňka, která je základním prvkem nervové soustavy živočichů [38, str. 31]. Lidský mozek těchto buněk obsahuje zhruba 100 miliard. Každá taková buňka se skládá z těla (*soma)*, přibližně tisíce *dendritů*, které jsou pro tuto buňku vstupy a jednoho výstupu, nazvaným *axon*. Vzájemné spojení těchto buněk reprezentuje paměť člověka. Když se člověk učí, tyto spojení vznikají a přenastavují se a analogicky, když člověk zapomíná, spojení zanikají. Tyto spojení se nazývají *synapse* a jejich vlastností je váha, která ovlivňuje sílu tohoto spojení dvou neuronů.

Neuron pracuje tak, že vstupní signály z dendritů se v jeho těle sčítají a pokud jejich celková hodnota překročí prahovou hodnotu, neuron se aktivuje a svým axonem vyšle krátký elektrický výstup. Poté se neuron na chvíli deaktivuje a po aktivaci je opět schopen vysílat impulsy při překročení prahové hodnoty signálu.

### Perceptron

Jelikož s některými vlastnosti přírodního neuronu není snadné pracovat, byl vytvořen zjednodušený model umělého neuronu, nazvaný *perceptron* [23]. Tento model věrně reprezentuje systém vstupů a výstupů přirozeného neuronu tak, aby ho bylo snadné implementovat. Jeho vstupem je vektor $\vec{\boldsymbol{x}}$, přičemž každý vstup má také váhu, takže druhým vstupem je vektor vah $\vec{\boldsymbol{w}}$. Výpočet vnitřního potenciálu je tedy váženým součtem hodnot vstupů a jejich vah. Vlastností perceptronu je také hodnota prahu $\Theta$. Ta je implementována jako jeden ze vstupních prvků, nazývající se *bias*, se stálou hodnotou 1 a váhou $-\Theta$. Aktivace neuronu je daná skokovou aktivační funkcí $g$ podle velikosti vnitřního potenciálu $u$, viz rovnice 2.1.

$$g(u) = sign(u) = \begin{cases} 1 & \text{pro u} \geq 0 \\ 0 & \text{pro u} < 0 \end{cases} \tag{2.1}$$

## 2.2 Neuronové sítě

Samotný perceptron má pouze omezenou schopnost - dokáže být pouze *binárním klasifikátorem*, tedy rozdělovat vstupy do dvou tříd. Stejně jako neurony v lidském mozku, i umělé neurony své uplatnění nachází jako součásti sítě. Takto tedy vznikl *vícevrstvý perceptron*, který je základním typem neuronových sítí [23]. Jak je zřejmé, tato síť se skládá z perceptronů, dohromady tvořící vrstvy, které jsou mezi sebou spojené. Vstupem první vrstvy jsou trénovací data a jejich výstupy jsou vstupy následujících vrstev. Tímto získal tento typ neuronové sítě alternativní název *feedforward neural network* (česky *dopředná neuronová síť*).

### 2.2.1 Aktivační funkce

O aktivačních funkcích padla zmínka již v minulé sekci, kde se jednalo o skokovou aktivační funkci. Tato funkce se v praxi u neuronových sítí nevyužívá, jelikož její využití dává smysl pouze pro osamocené neurony. Neuronové sítě jsou totiž trénovány pomocí algoritmu *gradient descent* (česky *gradientní sestup*), a již zmíněná skoková funkce má skoro po celém svém definičním oboru gradient nulový. Tím tedy není vhodnou aktivační funkcí pro neuronové sítě, a proto jsou používány různé jiné funkce, které jsou pro trénování vhodnější [31].

#### Sigmoidní funkce

Sigmoidní funkce, viz rovnice 2.2, je nelineární funkce s oborem hodnot v intervalu $(0, 1)$, která je definovaná na celém oboru reálných čísel. Své využití kvůli příhodnému oboru hodnot nachází často v sítích, jejichž výstupem je pravděpodobnost určitého jevu. Jedním takovým jevem může být právě rozpoznávání, jestli je vygenerovaný vzorek skutečný.

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

#### Hyperbolický tangens

Hyperbolický tangens (angl. zkr. *tanh*) je funkcí velmi podobnou sigmoidní funkci. Je to také často používaná nelineární funkce, definovaná po celém oboru reálných čísel, ale s rozdílným oborem hodnot, který je v intervalu $(-1, 1)$. Tento obor hodnot jí činí vhodnější pro použití ve vrstvách neuronových sítí, kde není výstupem pravděpodobnost, protože se při použití na rozdíl od předchozí funkce zde nestává, že by se trénování těchto vrstev zaseklo. V této práci se s touto funkcí lze setkat ve výstupní vrstvě neuronové sítě generující falešné vzorky.

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.3}$$

#### ReLU

Tato aktivační funkce je v dnešní době pravděpodobně nejpoužívanější funkcí v oblasti konvolučních neuronových sítí. *ReLU*, neboli *rectified linear unit*, je lineární funkcí definovanou v rozsahu $<0, \infty)$. Její velkou výhodou je velká hodnota gradientu, jelikož derivace této funkce je po celém rozsahu rovna 1, což jí dělá efektivní během trénování sítě. Problémem je, že neumožňuje práci se zápornými čísly, které hned získají hodnotu 0, což může omezit či úplně zastavit trénování.

$$f(x) = max(0, x) \tag{2.4}$$

#### Leaky ReLU

K adresování problému funkce ReLU se zápornými čísly byla vytvořena funkce *Leaky ReLU*, viz rovnice 2.5, kde lze vidět, že tato funkce je již definována po celém oboru reálných čísel. Práce s zápornými čísly je tedy podporována, ale není velmi efektivní, vzhledem k tomu, že zde není příliš velká hodnota derivace, jelikož parametr $a$ je většinou malým číslem. Použití této aktivační funkce pro konvoluční vrstvy je doporučováno u mnoha moderních architektur [24, 10, 14, 16].

$$f(x) = \begin{cases} ax & \text{pro x} < 0 \\ x & \text{pro x} \geq 0 \end{cases} \tag{2.5}$$

### 2.2.2 Chybové funkce

Chybové funkce (angl. *loss* nebo *objective functions*), jsou během trénování důležitými metrikami. Udávají, jak moc se výstupy neuronových sítí liší od požadovaných výsledků. Smyslem trénování je tedy minimalizování hodnoty vybrané chybové funkce. Podívejme se na dva typy chybových funkcí, které byly v implementovaných modelech použity.

#### Binární křížová entropie

Křížová entropie [33] jako chybová funkce hodnotí klasifikační model, jehož výstupem je hodnota pravděpodobnosti z intervalu $<0, 1>$. Její hodnota se odvíjí od toho, jak moc se liší pravděpodobnost předpovězená modelem od skutečné hodnoty. Perfektní model klasifikační neuronové sítě by tedy měl hodnotu chybové funkce rovnou 0.

Binární křížová entropie [6] (angl. *binary crossentropy*) se specifická tím, že jsou klasifikovány pouze dvě různé třídy. Vzorec pro výpočet binární křížové entropie je:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y * log(\hat{y}_i) + (1 - y) * log(1 - \hat{y}_i)), \tag{2.6}$$

kde $y$ je požadovanou hodnotou a $\hat{y}$ je $N$ různých hodnot, které byly předpovězené modelem.

#### Wasserstein vzdálenost

Tato chybová funkce vychází z *Wasserstein vzdálenosti* [2], která je matematickou metrikou reprezentující minimální cenu převedení distribuce dat $p$ na distribuci dat $q$. V oblasti neuronových sítí se tedy jedná o distribuce reálných a syntetických dat. Jiným názvem pro tuto funkci, používaným zejména v oblasti informačních technologií, je *Earth mover's distance*, tedy do češtiny volně přeloženo jako *vzdálenost přenášení země*. Wasserstein vzdálenost bývá často zjednodušeně vysvětlována jako cena, respektive úsilí, nutné k přesunutí jedné hromady země na druhé místo, z čehož toto alternativní jméno vzniklo.

### 2.2.3 Optimalizační algoritmy

Doposud zde v této práci nebyla řeč o tom, jak se vlastně neuronová síť učí, pouze o tom, jaká je její struktura a jaké metriky k jejímu hodnocení máme. Minimalizování chybové

funkce je hledáním jejího globálního minima. Taková funkce nemá konvexní charakter, proto obsahuje mnoho lokálních minim a řešení tedy není přímočaré. Pokud by minimalizování funkce skončilo v takovém lokálním minimu, které není velmi blízké globálnímu minimu, odrazilo by se to na výsledcích, které by také nejspíše nebyly žádoucí. Dalším problémovým nalezeným výsledkem může být takzvaný *sedlový bod*, který je zároveň lokálním minimem i lokálním maximem [32]. K efektivnímu řešení tohoto problému bylo vytvořeny mnohé optimalizační algoritmy, a my se zde krátce na dva z nich podíváme.

### Gradientní sestup

Jedná se o nejzákladnější, ale pravděpodobně nejpoužívanější algoritmus v oblasti neuronových sítí [1]. Jeho výhodami je lehké pochopení, implementace a velká rychlost výpočtu. Průběh gradientního sestupu je následující:

1. Pomocí parciálních derivací jednotlivých parametrů vah je vypočítán gradient, reprezentující změny nutné k maximalizování hodnoty chybové funkce. Jelikož hodnotu minimalizujeme, pracujeme se zápornou hodnotou tohoto gradientu.
2. Upravit jednotlivé hodnoty vah na základě vypočtené hodnoty záporného gradientu.
3. Opakovat kroky 1 a 2, dokud nedosáhneme minima.

Významným parametrem při provádění těchto kroků, je *learning rate* (česky *míra učení*). Určuje, jak moc jsou změněny hodnoty vah při provádění kroků gradientního sestupu. Je důležité tuto hodnotu této proměnné správně zvolit. Při použití moc velké hodnoty je pravděpodobné, že prováděné kroky nebudou spolehlivě vést k určitému minimu. Ovšem použití hodnoty příliš malé by zase mohlo vést k tomu, že by se algoritmus snadněji zastavil v lokálním minimu.

Nevýhodami tohoto algoritmu je kromě zmíněného problému lokálních minim také skutečnost, že váhy jsou měněny až po iteraci celou datovou sadou. Toto může způsobit opravdu pomalé trénování na velkých datových sadách a také to přináší problém s paměťovou náročností tohoto procesu [5].

### Adam

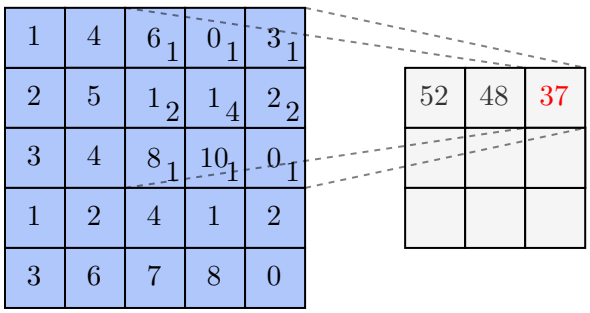
Optimalizační algoritmus *Adaptive moment estimation*, nejčastěji známý pod zkratkou *Adam*, byl vytvořen za účelem vyřešení těchto problémů [18]. V oblasti generativních neuronových sítí se jedná o nejpoužívanější optimalizační algoritmus, používaný v nejmodernějších řešeních [16]. Řadí se mezi algoritmy využívající stochastický gradientní sestup. Tyto algoritmy řeší paměťový problém gradientního sestupu tím, že kroky nejsou prováděny po průchodu celou datovou sadou, nýbrž po jednotlivých vzorcích, nebo ideálně po minidávkách vzorků. Další výhodou algoritmu Adam je používání *adaptivní míry učení*. To znamená, že velikost kroku není vždy stejná, ale dynamicky se mění. Docíleno je toho pomocí *hybnosti* (angl. *momentum*), kde je využito hodnot předchozích dvou gradientů k výpočtu hodnoty, která bude přidána ke stávajícímu gradientu. Toto kromě výrazného zrychlení také snižuje šanci, že se trénování kvůli velkému kroku vyhne minimu [5].

## 2.3 Konvoluční neuronové sítě

Princip neuronových sítí lze použít i na obrazová data. Problémem ale je, že obrázky by na vstupu neuronové sítě byly převedeny z matice, respektive tří matic, pokud se bavíme o barevných obrázcích, na pouhý vektor, čímž by ztratily své prostorové závislosti. Jednodušeji řečeno, konvoluční neuronové sítě se snaží přesněji napodobovat lidský mozek v tom, že v obrazových datech jsou hledány určité vzory a jejich souvislosti [27]. V této sekci je stručně představeno, jakým způsobem bylo tohoto docíleno.

### 2.3.1 Konvoluce

Tato operace je pro zpracování obrazu naprosto klíčová. Na nejnižší úrovni máme 2D pole vstupních dat a *konvoluční jádro*, které je také 2D polem a obsahuje různé hodnoty, které se v průběhu trénování mohou měnit. Toto konvoluční jádro je postupně „posunováno“ po poli vstupních dat a sousedící hodnoty obou struktur jsou vynásobeny a všechny tyto výsledky jsou sečteny, čímž dostaneme jedinou skalární hodnotu. Po zpracování celého vstupního vzorku je výsledkem nové 2D pole konvolucí vypočítaných hodnot. Tento proces je naznačen na obrázku 2.1. Dvě často používané techniky u konvoluce jsou *padding* a *stride*. Padding je proces, kdy jsou na okraje konvoluovaného vzorku přidány prázdné pixely s hodnotami 0. Toto je děláno, aby výsledné 2D pole mělo stejnou velikost, jako konvoluovaný vzorek. Stride neboli krok konvoluce značí, o kolik obrazových bodů se vždy posouvá konvoluční jádro. Konvoluce s kroky velikosti 2 se používají například při snižování rozlišení vzorku na polovinu.




Obrázek 2.1: Ukázka jednoho kroku konvoluce. Hodnoty vstupních dat v levé matici jsou násobeny hodnotami konvolučního jádra, zobrazenými vedle hodnot vstupních dat. Výsledek je uložen do pravého 2D pole.

### 2.3.2 Vrstvy sítě

V konvolučních neuronových sítích je k dispozici celá řada typů vrstev, určených pro různé operace s daty. Krátce zde budou popsány některé takové, které byly použity v implementační části této práce.

#### Konvoluční vrstva

Není žádným překvapením, že tato vrstva je hlavní součástí konvolučních neuronových sítí. Účelem této sítě je samozřejmě provádět konvoluce, tak jak byly popsané v sekci 2.3.1, ovšem důležité je také ještě zmínit, jaká je struktura této vrstvy.

Vstupní data nejsou vždy jen jednou vrstvou hodnot pixelů. Pokud pracujeme s barevnými obrázky, máme často vrstvy tři, pro každý barevný kanál jednu, a v hlubších vrstvách sítě mají výstupy často kanálů desítky až stovky. Při návrhu neuronových sítí je u konvolučních vrstev zvolen počet *filtrů*. Každý filtr obsahuje stejný počet konvolučních jader, jako je počet kanálů vstupních dat této vrstvy sítě. Každé konvoluční jádro tohoto filtru zpracovává právě jeden vstupní kanál a má nastavenou určitou váhu. Výstup všech konvolučních jader je poté sečten do jednoho výsledného kanálu, který je finálním výstupem filtru [30].

Za konvolučními vrstvami se nachází aktivační vrstvy, kde je výstup transformován podle zvolené aktivační funkce, viz sekce 2.2.1.

Dalším typem konvolučních vrstev jsou *transponované konvoluční vrstvy* (tzv. *dekonvoluční vrstvy*), které jsou používány ke zvýšení rozlišení dat. Vstupní data jsou zde prokládány prázdnými pixely, což zvětšuje prostor, kde se může pohybovat konvoluční jádro.

### Normalizační vrstvy

Distribuce výstupních hodnot konvoluční vrstvy se během trénování může prudce měnit, což výrazně zpomaluje trénování následujících konvolučních vrstev, které se musí adaptovat. Tímto problémem se zabývají normalizační vrstvy, které hodnoty transformují tak, aby byla jejich střední hodnota rovna 0 a rozptyl roven 1. Přidání normalizační vrstvy vede ke značnému zrychlení procesu trénování, a umožňuje zvýšit velikost learning rate.

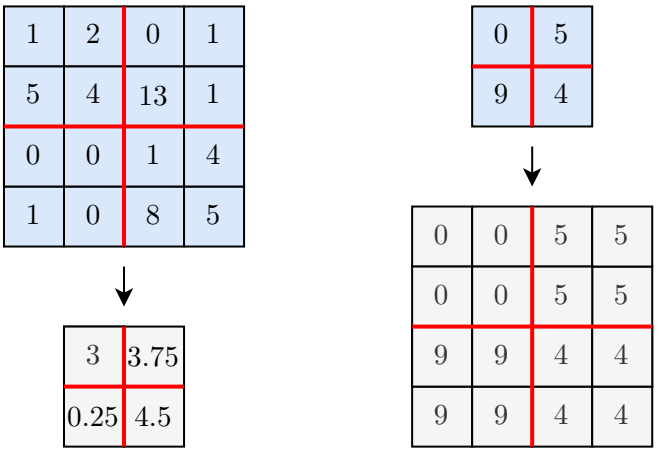
Nejpoužívanější je vrstva *batch normalization* (česky *dávková normalizace*), která hodnoty normalizuje napříč celým kanálem mezi všemi vzorky z dávky [13]. Dalšími možnými typy normalizací jsou *vrstvová normalizace*, kde jsou společně normalizovány hodnoty jednoho vzorku napříč všemi kanály a *instanční normalizace*, kde se normalizují zvlášť jednotlivé kanály vzorků.

### Sdružovací vrstvy

Tento typ vrstvy představuje alternativu pro konvoluce s krokem při snižování prostorového rozlišení vzorků [27]. Nejpoužívanějšími typy sdružovacích vrstev jsou *average pooling* (*sdružování podle průměru*) a *max pooling* (*sdružování podle maxima*). Jedná se o velmi jednoduchý typ vrstvy. Každý obrázek je procházen jádrem o zvolené velikosti, a ukládána je buď průměrná, v případě prvního jmenovaného typu, nebo maximální hodnota z jádrem pokryté oblasti.

### Upsampling vrstva

Jedná se vrstvu s opačnou funkcí oproti předchozím sdružovacím vrstvám, jelikož jejím cílem je prostorové rozlišení obrázku zvýšit. Je toho docíleno tak, že zvětšovaná data jsou pouze opakována. Tradičně se tato vrstva používá pro zdvojnásobení rozlišení, ovšem počet opakování řádků i sloupců lze zvolit libovolně. Porovnání upsampling vrstvy a average pooling vrstvy je vidět na obrázku 2.2.




Obrázek 2.2: Ukázka použití vrstev average pooling (vlevo) a upsampling (vpravo)

# Kapitola 3

# Generativní neuronové sítě

Od samotného počátku vývoje neuronových sítí byly aplikace nacházeny převážně v oblasti klasifikace. Ačkoliv probíhaly i výzkumy zabývající se myšlenkou generování syntetických dat, stále se nedařilo překonat překážky, které generování přináší, vzhledem k tomu, jak náročné je takové neuronové sítě trénovat. Toto vše se změnilo v roce 2014, kdy americký vědec Ian Goodfellow představil revoluční koncept generativních neuronových sítí [8], který otevřel cestu do světa generativních úloh.

Fungování této technologie je nastíněno už v názvu *generative adversarial networks*, který by se dal do češtiny přeložit jako *generativní soupeřící sítě*. Proti sobě jsou postaveny dva modely neuronových sítí, které mají odlišné cíle. Na jedné straně je generátor, který má za úkol tvořit vzorky nerozeznatelné od skutečných a na druhé straně je diskriminátor, který rozpoznává, které vzorky jsou skutečné a které jsou falešné. Ještě jednodušší vysvětlení, které je převzaté přímo z článku [8] představující tuto technologii je takové, že generátor je analogický týmu zločinců, vyrábějící padělky bankovek a snažící je se používat bez toho, aby byli odhaleni. Diskriminátor je analogický týmu policistů, kteří hledají padělky. Toto soupeření vede k tomu, že oba týmy zdokonalují své metody do té doby, než bankovky vytvářené zločinci jsou nerozpoznatelné od těch skutečných.

V oblasti hlubokého učení představení této technologie vyvolalo velký ohlas a jejího vývoje se chopilo mnoho výzkumníků. Na ukázky některých vzniklých technologií a také nejmodernějších aplikací generativních neuronových sítí se podíváme v sekci 3.1.

Jelikož cílem této práce je natrénovat více modelů podmíněných generativních neuronových sítí, byly proto vybrány architektury, které byly ve své době vždy něčím převratné a staly se state-of-the-art řešením generativních úloh. Každé z trojice implementovaných architektur je v sekci 3.3 věnován prostor, ve kterém je představeno, čím se daná architektura lišila od předchozích a jaké jsou její výsledky v oblasti generování lidských obličejů. Pozornost zde bude věnována také fungování architektury StyleGAN [15, 16], jejíž implementaci jsem se ve své práci nevěnoval, ale stojí za to ji zmínit, neboť je v dnešní době považována za nejmodernější řešení.

V neposlední řadě, protože analogie generativních neuronových sítí jakožto policistů hledající padělky kriminálníků je sice vhodná na pochopení principu technologie, je ale nicneříkající co se týče skutečného fungování této technologie, a proto je na tuto problematiku zaměřena sekce 3.2.

## 3.1 Generativní úlohy

Za relativně krátkou dobu, po kterou technologie generativních neuronových sítí existuje, se objevilo nepřeberné nejrůznějších aplikací primárně v různých podoborech počítačového vidění a zpracování obrazu, ale také v oblastech, kde se nepracuje přímo s obrazovými daty, jakými jsou například zpracování přirozeného jazyka a generování hudby [9]. Objevují se dokonce pokusy využívat generativní neuronové sítě v oblasti lékařství, například k rekonstrukci zubů [12], či dokonce k identifikaci osob nakažených nemocí COVID-19 z rentgenových snímků plic [17]. V této sekci se podíváme na některé zajímavé ukázky možností této technologie.

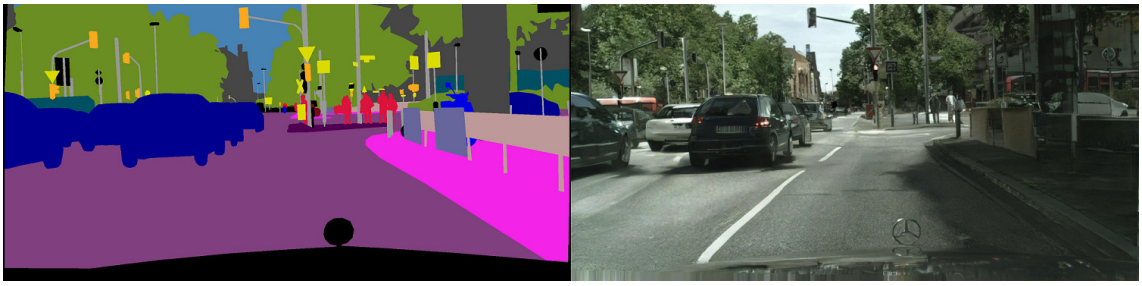
### Generování nových vzorků

Jedná se o úkol, který je pro generativní neuronové sítě typický. Již od počátků vývoje se používá pro porovnávání kvality vytvořených architektur neuronových sítí pomocí nejrůznějších metrik pro evaluaci vzorků a subjektivní kvality vzorků. Aktuálně nejmodernější implementací generativních neuronových sítí v tomto ohledu je technologie StyleGAN2 [16], která je podrobněji popsána v sekci 3.3. Během pouhých několika let se v tomto podoboru podařilo pokročit od generování rozmazaných vzorků v malém rozlišení k vzorkům prakticky nerozeznatelných od skutečných. Vygenerované vzorky z neuronových sítí s technologií StyleGAN2 natrénovaných na datových sadách aut, koní a kostelů lze vidět na obrázku 3.1.

Obrázek 3.1: Vygenerované vzorky technologií StyleGAN2 (převzato z [16])

### Převod obrázků

Významným rozdílem odlišující převádění obrázků (angl. *image-to-image translation*) od předešlého generování syntetických vzorků je použití *podmíněných generativních neuronových sítí* namísto nepodmíněných. Na rozdíl od této práce, kde jsou generované lidské obličeje podmíněné pouze diskrétními hodnotami označující věk a pohlaví, v těchto implementacích se na vstupu neuronových sítí nachází obrázek, jedná se tedy o takzvané *image-conditioned* modely. Mezi zajímavé úlohy se řadí například převod sémantických značení ulic na fotorealistické snímky s použitím architektury *pix2pixHD* [36] (viz obrázek 3.2), a nebo různé převody s použitím architektury *CycleGAN* [39] (viz obrázek 3.3).

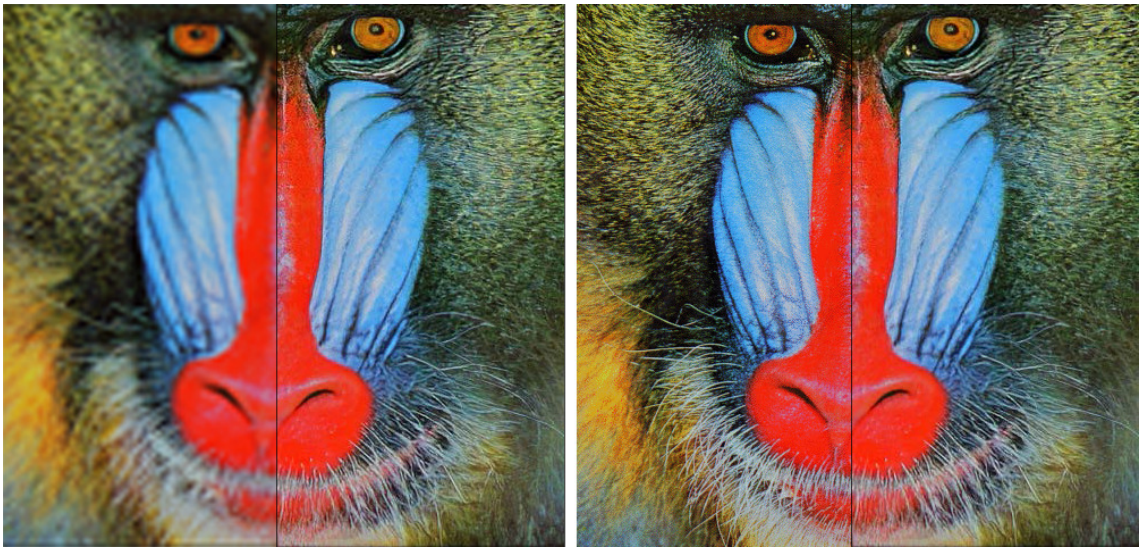


Obrázek 3.2: Převod sémantického značení ulice na fotorealistický snímek (převzato z [36])

Obrázek 3.3: Převod koní na zebry, jablek na pomeranče a náčrtů obuvi na realistický snímek (převzato z [39])

### Škálování

Dalším praktickým využitím je *superresolution* (česky *škálování*) obrazových dat. Je to technika umožňující rekonstrukci obrázků s vyšším rozlišením ze vstupních dat s nižším rozlišením. I s neustále zvyšující se kvalitou snímků, které lze zachytit i pomocí běžných zařízení se najdou oblasti, kde je stále problém s kvalitou dat z důvodů náročnosti procesu zachycování nebo kvůli omezené kapacitě úložiště. Takovými problémy tedy trpí například kontinuálně zaznamenávající bezpečnostní kamery či třeba lékařské snímky. Zde mohou přijít vhod technologie jako například *ESRGAN* (*Enhanced Super-Resolution Generative Adversarial Networks*) [37]. Na obrázku 3.4 je vidět, jak realisticky rekonstruuje neuronová síť původní snímek v rozlišení 496x480 pixelů ze snímku, který byl 4x zmenšen na rozlišení 124x120 pixelů, což je 16x méně pixelů, než bylo ve snímku původním.



Obrázek 3.4: Porovnání kvality čtyřnásobného zvětšení rozlišení technologií ESRGAN (pravá část obou obrázků) s původním zmenšením obrázkem (levá část obrázku vlevo) a originálním obrázkem ve velkém rozlišení (levá část obrázku vpravo) (převzato a upraveno z [37])

## 3.2 Podmíněné generativní neuronové sítě

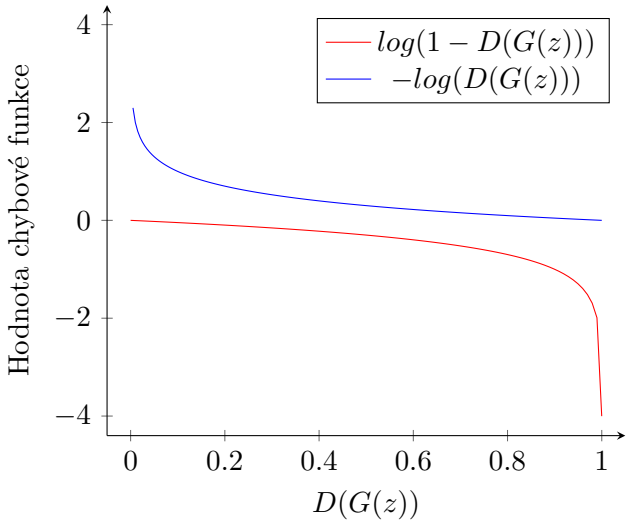
Jak bylo již zmíněno v úvodu této kapitoly, generativní neuronové sítě rozšířil v roce 2014 Ian Goodfellow. Ovšem jejich objev může sahat mnohem více do minulosti. Dokázat se to snaží jeden z předních výzkumníků v oblasti umělé inteligence a neuronových sítí, Jürgen Schmidhuber, přezdívaný dokonce jako *„otec AI“*. Ve svém článku již v roce 1990 vyslovil myšlenku použití dvou navzájem si konkurujících sítí, a celý proces nazval *artificial curiosity* (česky *umělá zvědavost*) [29]. Ať už je skutečným autorem této technologie kdokoliv, důležité je především to, že existuje a lze ji využít k mnoha zajímavým projektům. V této části bude přiblíženo, jak vlastně tyto generativní neuronové sítě fungují.

### 3.2.1 Základní princip

Aplikační rámec generativních neuronových sítí je nejjednodušší aplikovat, jsou-li oba modely neuronovými sítěmi [8, 9]. Ke zjištění distribuce generátoru $p_g$ nad daty $x$ je definována distribuce vstupních proměnných šumu $p_z(z)$, kde $z$ je proměnná šumu. Generativní neuronové sítě tedy reprezentují mapování prostoru šumu na datový prostor jakožto funkci $G(z, \Theta_g)$, kde $G$ je diferenční funkcí reprezentovanou neuronovou sítí s parametry $\Theta_g$. Druhá neuronová síť je diskriminátor, popsaný funkcí $D(x, \Theta_d)$, kde $x$ jsou vstupními daty a $\Theta_d$ jsou vstupními parametry. Výstupní hodnotou této funkce je jediná skalární hodnota $D(x)$. $D(x)$ reprezentuje pravděpodobnost, že $x$ pochází z datové sady, a tedy není výstupem generátoru. Obě neuronové sítě jsou trénovány, střídavě jedna po druhé, a každá se snaží docílit opačných cílů oproti druhé, co dělá z procesu trénování *minimax* hru. Diskriminátor je trénován k maximalizování pravděpodobnosti správného určení původu vstupních dat, tím pádem rozpoznává syntetická data od skutečných. Na druhou stranu, generátor je trénován ke snížení této pravděpodobnosti s funkcí $log(1 - D(G(z)))$. Minimax hra pro získání hodnoty chybové funkce je vidět v následujícím vzorci:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[log\, D(x)] + \mathbb{E}_{z \sim p_z(z)}[log\,(1 - D(G(z)))]. \tag{3.1}$$

V praxi se ale tento přístup neosvědčil, jelikož pokud jsou vzorky vytvořené generátorem velmi nekvalitní a diskriminátor je rozpoznává s jistotou, chceme aby diskriminátor dával generátoru větší zpětnou vazbu. Ovšem v této funkci je blízko hodnoty $D(x) = 0$ křivka chybové funkce plochá (neboli saturuje) a gradient je blízký nule. Místo trénování generátoru k minimalizování $log(1 - D(G(z)))$ tedy v praxi trénujeme k maximalizování $log(D(G(z)))$, což lze přeložit jako trénování generátoru k maximalizování pravděpodobnosti, že diskriminátor rozpozná vzorek špatně. Jelikož tato funkce řeší problém saturace chybové funkce, získala název *non-saturating loss* (česky *nesaturující chybová funkce*). Výsledkem je tedy stále stejný cíl, ovšem na počátku trénování s mnohem větším gradientem, jak lze vidět na obrázku 3.5, a tím tedy vede s větší zpětné vazbě diskriminátoru pro generátor, což urychluje proces trénování.




Obrázek 3.5: Vliv cíle generátoru na gradient v průběhu trénování

Úspěšné natrénování generativních neuronových sítí není jednoduchá záležitost. Konvergence nastává teprve když oba modely jsou v rovnováze a minimax hra je tím pádem u konce. Prakticky se ale stává, že modely navzájem negují svoje postupy a nikam se neposunují [7].

#### Mode collapse

Častým problémem objevujícím se u vygenerovaných vzorků a znemožňující konvergenci, je *mode collapse* (česky *kolaps režimu*). Tento fenomén nastává, když se generátor naučí mapovat více vstupních hodnot šumu na stejný výstup. Přesněji, tento kolaps se dělí na částečný a úplný. Jak se tyto druhy kolapsu rozpoznají, je z názvu evidentní. Aby nastal úplný kolaps a generátor produkoval pouze jediný vzorek, je spíše vzácné, většinou se tedy na některých vzorcích opakují stejné barvy, textury nebo v případě obličejů jejich některé rysy. Jádro tohoto problému tkví v používané chybové funkci. Pokud se dostane generátor do situace, kdy jeho vzorky diskriminátor neúspěšně klasifikuje jako reálné, co je cílem *non-saturating loss*, nemá potřebu se dále zlepšovat a pouze se učí charakteristiky těchto vzorků opakovat pro různé vstupy. Tento problém byl předmětem mnoha výzkumů a jeho řešení mělo velkou prioritu. Našly se ale i aplikace generativních neuronových sítí, kde existence tohoto problému neměla velký vliv, vzhledem k tomu, že jejím důsledkem bylo pouze snížení rozmanitosti výstupů, nikoliv jejich vizuální kvality.

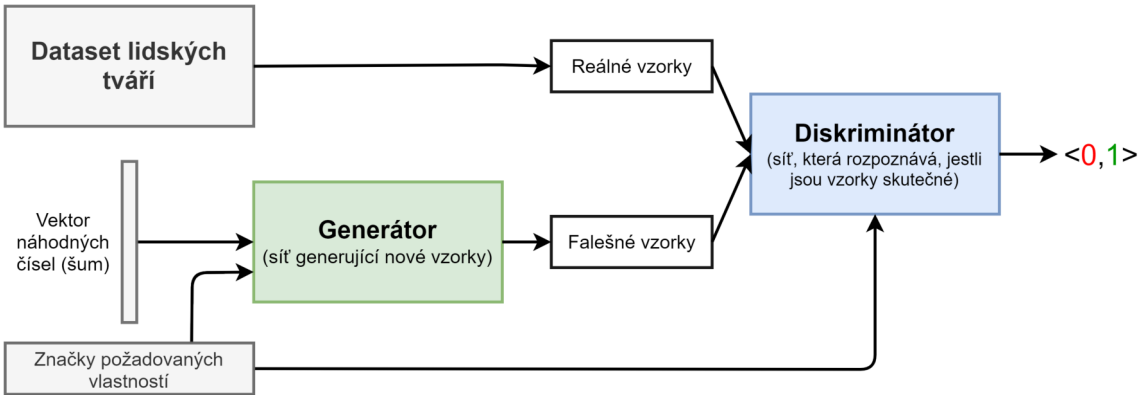
### 3.2.2 Podmíněnost sítí

Krátce po představení generativních neuronových sítí bylo představeno jejich rozšíření na podmíněné generativní neuronové sítě, díky použití značek (angl. *label*) [21]. Toto rozšíření spočívá v tom, že jsou generátoru kromě šumu a diskriminátoru kromě vzorku předloženy na vstupní vrstvě také další informace, viz obrázek 3.6. Těmito daty může být cokoliv, jak jednoduchá číselná hodnota značící třídu, do které vzorek patří, tak například celý další obrázek, který bude sítí zpracován, jako bylo vidět například na obrázku 3.2. V této práci jsou značky reprezentovány dvěma číslicemi, jednou pro určení třídy pohlaví a druhou pro určení třídy věkové skupiny vzorků. Značky tříd jsou na vstup neuronových sítí přidávány použitím *embedding* (česky *vkládacích*) vrstev. Tato vrstva převádí značky na vektor fixní velikosti, který je posléze spojen se šumem v případě generátoru nebo vzorkem v případě diskriminátoru a je s ním dále regulérně pracováno ve zbytku neuronové sítě.

Jak je zřejmé, tato podmíněnost přidává určitou možnost výběru finálního vzhledu vzorků, na rozdíl od nepodmíněných modelů, kde při generování taková možnost neexistuje a vše je dáno pouze hodnotami šumu. Další výhodou použití podmíněných modelů je, že podmíněné generování poskytuje subjektivně kvalitnější vzorky. Spekuluje se, že hlavním důvodem je to, že sítě se takto méně učí nepodstatné detaily a více se zaměřují na to, co lidé na generovaných vzorcích skutečně hledají [28].

## 3.3 Architektury generativních neuronových sítí

Výrazným aspektem návrhu generativních neuronových sítí, který se v průběhu let vyvíjí, je právě struktura sítí jednotlivých modelů. V této části se podíváme, čím se každá vybraná a v této práci implementovaná architektura lišila od předchozích a také na to, čím se od nich liší stávající nejmodernější řešení.




Obrázek 3.6: Schéma fungování podmíněných generativních neuronových sítí pro generování lidských tváří

### DCGAN

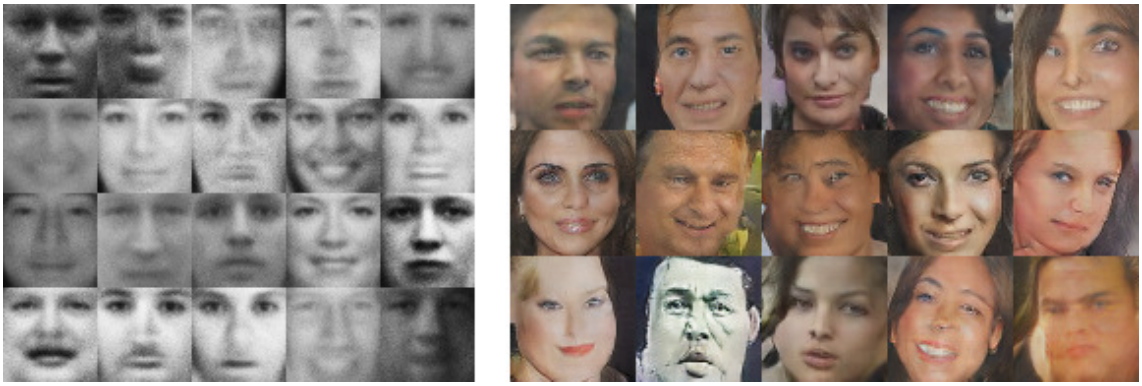
Originální představená architektura neuronových sítí měla neuronové sítě diskriminátoru i generátoru definované jako vícevrstvé perceptrony [8]. Vzhledem k tomu, že tyto plně propojené vrstvy nejsou nejlepším řešením pro zpracovávání obrázků, což je nejběžnější aplikací této technologie, netrvalo dlouho a vznikla architektura generativních neuronových sítí, která používá konvoluční neuronové sítě pro generátor i diskriminátor [24].

*Deep Convolutional Generative Adversarial Networks* (česky *hluboké konvoluční generativní neuronové sítě*), zkráceně *DCGAN*, je v dnešní době sice již zastaralou architekturou, ovšem všechny moderní architektury z ní alespoň částečně vychází, právě kvůli použití konvolučních vrstev [7]. Konvoluční vrstvy rovněž nahradily *pooling* a *unpooling* vrstvy, které zvyšovaly, resp. snižovaly rozlišení dat. Místo těchto vrstev jsou zde využívány *kroky* u konvolucí s velikostí alespoň 2. Generátor ke zvyšování rozlišení používá *transponovanou konvoluci*, tzv. *dekonvoluci*. Standardizováno bylo také použití *dávkové normalizace*. Normalizovány jsou výstupy všech vrstev obou neuronových sítí, kromě první vrstvy diskriminátoru a poslední vrstvy generátoru. Tyto vrstvy nemají své výstupy normalizované, aby se obě sítě mohly správně naučit střední hodnotu a rozptyl hodnot distribuce dat. Posledním výrazným rozdílem oproti původní architektuře je přechod k optimalizátoru *Adam*. Vliv všech těchto změn je vidět na obrázku 3.7, kde jsou porovnány lidské obličeje generované na původní architektuře a na architektuře DCGAN.

### WGAN-GP

Rozdílnou cestou se vydali výzkumníci stojící za architekturou *Wasserstein GAN* s *Gradient penalty* (česky *penalizace gradientu*) [2, 10]. Více než struktuře samotných sítí je totiž věnována pozornost chybové funkci.

Přestože výrazný nedostatek chybové funkce negativně ovlivňující průběh trénování generátoru byl odhalen a adresován již během vývoje základní architektury generativních neuronových sítí, alternativní řešení stále obnášelo různá úskalí. Používání představené nesaturující chybové funkce přináší problém s velkým rozptylem gradientu, který vede k nestabilitě celého modelu. Tento problém jde řešit přidáním šumu ke vzorkům na vstupu diskriminátoru. Dalším řešením je použití jiné chybové funkce, a tím se tedy dostáváme k Wasserstein vzdálenosti [10], o které byla řeč v sekci 2.2.2.



Obrázek 3.7: Porovnání obličejů generovaných sítěmi podle základní architektury generativních neuronových sítí (vlevo) a architektury DCGAN (vpravo) (převzato a upraveno z [8, 24])

Použití této chybové funkce řeší nejen tento problém u nesaturující funkce, ale také zjednodušuje proces trénování tím, že má rovnější křivku gradientu po celém svém průběhu, čímž je zajištěno, že bude trénování probíhat, ať se generátoru daří jakkoliv dobře či špatně. Rovněž se v této architektuře neobjevuje problém mode collapse. Další výhodou je, že nejsou vyžadovány žádné větší zásahy do konstrukce neuronových sítí při změně chybové funkce na Wasserstein vzdálenost. Jediným rozdílem je změna výstupu u diskriminátoru, který už na svém výstupu nemá pravděpodobnost danou sigmoidní funkcí, nýbrž výstupem je skalární hodnota. Jelikož výstup je nyní z oboru reálných čísel, jeho hodnota značí, jak reálné vzorky na vstupu diskriminátoru jsou. Vzhledem k tomu, že diskriminátor s touto chybovou funkcí nerozpoznává, jestli vzorky jsou nebo nejsou skutečné, ale hodnotí jejich kvalitu, byl v tomto článku diskriminátor přejmenován na *kritika*, aby to lépe reprezentovalo tuto novou úlohu.

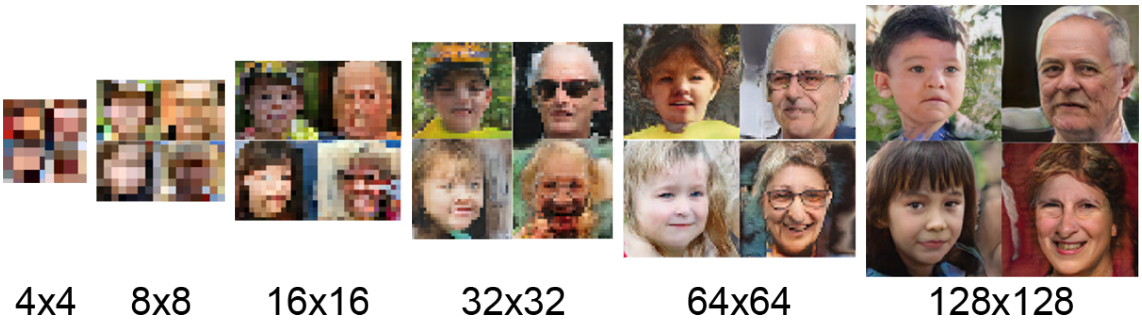
Poslední zbývající podmínkou pro fungování této architektury je splnění podmínky *Lipschitz kontinuity*, jak je důkladně popsáno v článku [2]. Prakticky toto omezení znamená, že váhy diskriminátoru musí být omezeny v určitém intervalu. V první architektuře [2] toto bylo provedeno ořezáním vyšších a nižších hodnot vah na maximální, respektive minimální hodnotu intervalu. Toto se projevilo jako možné řešení, ovšem obnášelo určité problémy, jelikož síť byla velmi citlivá na správné nastavení tohoto hyperparametru a jeho špatně zvolená hodnota způsobovala nekonvergenci trénování. Kvůli těmto problémům byla poměrně rychle představena alternativa [10], kde je model penalizován, pokud se normy gradientů příliš zvětší nebo zmenší od hodnoty 1, u které se musí držet pro splnění Lipschitz kontinuity.

Představení této technologie znamenalo další výrazný skok ve vývoji generativních neuronových sítí, trénování sítí bylo snadnější než předtím, vzhledem ke snadnější konvergenci modelů, a menší náchylnosti ke zkolabování procesu trénování i při mnohem větších rozlišeních vstupních i výstupních vzorků. Vygenerované lidské obličeje z neuronových sítí používajících tuto architekturu lze vidět na obrázku 3.8, kde jsou natrénovány sítě na obrázcích s rozlišením 1024x1024 pixelů.

Obrázek 3.8: Lidské tváře vygenerované neuronovými sítěmi s architekturou WGAN-GP (převzato a upraveno z [20])

### ProGAN

Jak již bylo řečeno, trénovat generativní neuronové sítě není lehký úkol, a s rostoucím rozlišením vstupních a výstupních vzorků také rychle stoupá komplexita trénování sítí k tomu určených. Proto se zjednodušením tohoto úkolu zabývali Tero Karras a spol., kteří přišli s novou metodologií trénování *Progressive Growing Generative Adversarial Networks*, zkráceně *ProGAN* nebo *PGGAN*. Zjistili, že tento proces jde výrazně zrychlit a stabilizovat, když jsou použity progresivně rostoucí modely neuronových sítí [14]. Tímto způsobem se nejdříve trénují vrstvy zpracovávající data s nízkým rozlišením a postupně se přidávají k již trénovaným modelům další vrstvy. Takto se sítě učí nejdříve výrazné charakteristiky a až poté stále jemnější detaily, jak je lze vidět na obrázku 3.9. Kromě kvality generovaných dat se zde byla věnována pozornost jejich variabilitě, a byla zde poprvé navržena a použita nová vrstva zachycující vlastnosti vzorků napříč minidávkou, která variabilitu významně zvětšuje.




Obrázek 3.9: Ukázka vygenerovaných vzorků v průběhu trénování

Přidávání nových vrstev do trénované sítě je rozděleno do dvou částí. Konvoluční bloky jsou do sítě přidávány pozvolna, při zdvojnásobování rozlišení je nový výstup sítě tvořen součtem výstupu předchozího bloku, zvětšeným na nové rozlišení a vynásobeným hodnotou $1 - alpha$, a výstupu nového přidávaného bloku, vynásobeným hodnotou $alpha$, kde $alpha$ je hodnota lineárně rostoucí od 0 do 1 v průběhu fáze přidávání nového bloku. Takto se nový blok natrénuje bez toho, aby bylo jeho přidání velkým šokem pro celou síť. Tuto fázi

střídá další, kdy je výstupem sítě pouze již nový blok a tato fáze trvá stejně dlouhou dobu. Během tohoto se síť dále *„vyrovnává“* s vyšším rozlišením. Tyto střídání probíhají do doby, než je dosaženo maximálního rozlišení, poté přestávají a síť se trénuje dál pouze na tomto rozlišení. Jednotlivé fáze se střídají vždy, když je sítěmi zpracován určitý zvolený počet vzorků.

### StyleGAN

Z výhod progresivního růstu těží i dvojice novějších architektur od stejného týmu výzkumníků, jako předchozí zmíněná ProGAN. Architektury *StyleGAN* [15] a následně *StyleGAN2* [16] jsou v době psaní této práce state-of-the-art řešeními problému generování vzorků.

Výraznými změnami zde prošla struktura sítě generátoru. Byl zahozen tradiční design se vstupní vrstvou zpracovávající šum, místo tohoto je prvním vstupem sítě pouze konstantní vrstva hodnot. Na výstup každé z konvolučních vrstev je přidáván šum podle Gaussova rozdělení. Speciální vrstvou, která se zde používá, je *Adaptive Instance Normalization*, zkráceně *AdaIN* [11]. Tato adaptivní normalizační vrstva pracuje se dvěma vstupy, za prvé s hlavním obsahem, kterým je výstup předešlé konvoluční vrstvy, a za druhé se vstupem obsahující požadovaný styl pro tento obsah. Střední hodnota a rozptyl tohoto obsahu jsou poté upravovány, aby odpovídaly hodnotám u přidávaného stylu. V praxi je tento styl podobný použití značek u podmíněných neuronových sítí.

Styl je podobně jako zmíněný šum bodem v latentním prostoru. O něm nebyla dosud v této práci zmínka, neboť v oblasti generativních neuronových sítí není tak důležitý, jako u jiných typů neuronových sítí, kde v závislosti na něm probíhá trénování, a většinou je v generativních neuronových sítích pouze Gaussovou distribucí, ze které jsou generovány body v prostoru. Hodnota zvoleného stylu může tedy být buď opět náhodná, nebo lze vybrat stejnou, která již byl použita jako šum pro vygenerování jiného vzorku, aby šlo sledovat vliv tohoto vzorku jako styl pro nový vzorek, jak lze vidět na obrázku 3.10. Tento vstup je


Obrázek 3.10: Ukázka použití vzorku jako stylu při generování jiného vzorku (převzato a upraveno z [15])

zpracován plně propojenou mapovací neuronovou sítí a následně je přidáván k již zmíněným adaptivním normalizačním vrstvám.

Ve článku představujícím novější architekturu StyleGAN2, navazující na původní architekturu, byly provedeny už pouze malé změny týkající se především vrstvy adaptivní normalizace [16]. Cílem práce bylo zanalyzování předchozí architektury a představení řešení několika specifických opakujících se artefaktů na vygenerovaných vzorcích. Ukázka vygenerovaných vzorků z této state-of-the-art architektury je vidět na obrázku 3.11.

Obrázek 3.11: Ukázka vygenerovaných vzorků v rozlišení 1024x1024 pixelů technologií StyleGAN2 (převzato z [35])

# Kapitola 4

# Návrh řešení

Tato kapitola se podrobně zabývá procesem návrhu řešení, přesněji jeho částmi, které jsou společné pro všechny vybrané architektury neuronových sítí. Konkrétními strukturami jednotlivých modelů neuronových sítí se posléze zabývá následující kapitola 5. Rostoucí popularita hlubokého učení zapříčinila zvýšený zájem o různé knihovny a nástroje pro usnadnění vývoje, proto je v sekci 4.1 zdůvodněn výběr použitého software k sestavování modelů a jejich trénování. S novými aplikacemi neuronových sítí často přibývají také nejrůznější datové sady, které jsou výzkumníky vytvářeny. Proto se v této kapitole v sekci 4.2 podíváme na několik zajímavých volně dostupných datových sad lidských obličejů. Jsou zde zdůvodněny jejich výhody a nevýhody a také faktory, které vedly k výběru a následnému použití konkrétní datové sady. Jelikož se pohybujeme v oblasti podmíněného generování, v sekci 4.3 je řeč také o informacích, které máme k dispozici o jednotlivých vzorcích z datové sady. Způsob, jakým probíhá práce se vstupy a výstupy všech modelů neuronových sítí, je rozebrán v sekci 4.4.

## 4.1 Použité knihovny

Jelikož vývoj konvolučních neuronových sítí bez knihoven pro hluboké učení je nesmírně náročná záležitost, máme k dispozici široký výběr nejrůznějších možností, především podle platformy, kde má být vyvíjený model zakomponován. Existují minimalistické knihovny určené například pro vestavěná nebo mobilní zařízení, ale také komplexní knihovny určené pro snadný vývoj modelů pro zařízení s velkým výpočetním výkonem.

### TensorFlow

TensorFlow[1] je nejpoužívanější knihovna pro hluboké učení. Vývoj této knihovny má na starosti společnost Google. Celý kód je open-source a je volně použitelný i modifikovatelný pod licencí Apache License 2.0[2]. Předností TensorFlow je využití vysokoúrovňového API umožňující snadnou práci s modely. Mezi další výhody této knihovny patří přenositelnost kódu díky širokému výběru podporovaných programovacích jazyků. TensorFlow umožňuje práci primárně v jazyce Python, který má plnou podporu, ale umožňuje také práci v jazycích C++, Java, Javascript, Go a Swift. Samozřejmostí je možnost trénovat modely jak na procesoru, tak jednoduše i na grafických kartách, což je v případě zpracování obrazu již

[1] https://www.tensorflow.org/
[2] https://www.apache.org/licenses/LICENSE-2.0

nutností. Knihovna získala své jméno po *tenzorech*, které jsou matematickými objekty v m-dimenzionálním prostoru s $n$ indexy. Jsou zobecněními vektorů a matic s potenciálně větším počtem dimenzí. V počítačovém světě se dají tenzory reprezentovat multidimenzionálními poli obsahující hodnoty stejného datového typu. Slovo *flow* ve svém názvu knihovna získala díky využití grafů datového toku (angl. *data flow graph*). Neuronové sítě jsou těmito grafy konstruovány, ve svých uzlech obsahují jednotlivé operace vrstev a hrany grafu značí tok dat, který je uskutečňován pomocí již zmíněných tenzorů.

### Keras

Keras[3] je vysokoúrovňové Python API pro Tensorflow, primárně vytvořené k zjednodušení práce s touto knihovnou, umožňující rychlejší experimentování s modely neuronových sítí. Úspěch tohoto API vedl k jeho oficiálnímu přidání do knihovny Tensorflow jako modul *tensorflow.keras*.

### NumPy

Pro přípravu multidimenzionálních polí obsahující vstupní data pro trénované modely je nejvhodnější použít knihovnu NumPy[4]. Je to užitečná knihovna programovacího jazyka Python přidávající objekt *ndarray*, který je n-dimenzionálním polem, a také mnoho matematických funkcí pro zjednodušení práce s touto datovou strukturou. Hlavní výhodou této knihovny je velmi rychlá rychlost výpočtů, jelikož velká část této knihovny je napsána v jazyce C, který je při výpočtech často až mnohonásobně rychlejší než Python. Samotný Tensorflow využívá NumPy pro své výpočty, proto také zajišťuje kompatibilitu objektů ndarray pro práci se svými modely.

## 4.2 Dostupné datové sady

Z nepřeberného množství různých datových sad lidských obličejů vytvořených výzkumníky z celého světa bylo potřeba vybrat jedinou, která bude udávat, jaká bude struktura finálních neuronových sítí. Datové sady se totiž liší technickými parametry, vzorky napříč různými datovými sadami nemají jednotné rozlišení a ani poměr stran, aby šly transformovat bez deformací na vstup neuronové sítě. Kvalitu jednotlivých datových sad lze soudit podle více faktorů, hlavním je samozřejmě dostatečné rozlišení a obrazová kvalita vzorků. Dalším důležitým ukazatelem, jelikož se pohybujeme v oblasti podmíněného generování, je také variabilita. V ideálním případě tedy hledáme datovou sadu, kde vzorky budou dostatečně pokrývat všechny věkové skupiny a obě pohlaví.

### CelebA

Datová sada *Large-scale CelebFaces Attributes*[5], zkráceně *CelebA*, obsahuje přes 200 tisíc obrázků lidských tváří, z nichž ke každému jsou k dispozici informace o 40 vybraných atributech tváře. Datová sada byla vytvořena v roce 2015 výzkumníky z Čínské univerzity v Hongkongu, za účelem vytvoření neuronové sítě k rozpoznávání rysů lidských tváří [19]. Vzorky jsou v rozlišení 178x218 pixelů a jejich obrazová kvalita je velmi vysoká. Ovšem

[3] https://keras.io/
[4] https://numpy.org/
[5] http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

pokud se podíváme na obrázek 4.1, můžeme spatřit jednu z nevýhod této datové sady, a to sice, že vzorky často nejsou ořezané tak, aby tvář zabírala maximální plochu a zbytečně se tím ztrácí detaily vzorků. Další problém je zřejmý až z analýzy celé sady, tím problémem je velká nevyváženost co se týče věku osob. Vysoce zde převažují lidé ze skupiny mladých dospělých, děti a senioři jsou v této datové sadě zastoupeni velmi zřídka.

Obrázek 4.1: Ukázka vzorků z datové sady CelebA

### IMDB-WIKI

Jak název této datové sady napovídá, její vzorky jsou shromážděné ze dvojice známých internetových stránek. Z internetové filmové databáze *IMDb* bylo získáno přes 460 tisíc obrázků obličejů a z Wikipedie dalších více než 62 tisíc. Celkově tedy datová sada *IMDB-WIKI*[6] obsahuje 523 051 vzorků, společně s informacemi o věku a pohlaví každé z osob, čímž se stala největší datovou sadou lidských obličejů. Za jejím vznikem v roce 2015 stál tým výzkumníků ze Spolkové vysoké technické školy ve švýcarském Curychu, kteří ji využili k trénování neuronových sítí odhadující stáří osob [25, 26]. Ovšem obrovská velikost této datové sady obnáší jedno velké úskalí, a tím je kvalita vzorků, viz obrázek 4.2. Jsou zde snímky vůbec neobsahující lidské obličeje a když tyto snímky pomineme, mnoho snímků je velmi nízké kvality, a často jsou prováděny transformace k dosažení čtvercového tvaru snímku. Dohromady se tedy jedná o nevhodnou datovou sadu pro generování, což bylo i krátkým experimentováním potvrzeno.

### FFHQ

Jako nejvhodnější byla nakonec zvolena datová sada *Flickr-Faces-HQ*[7], známá spíše pod zkratkou *FFHQ*. Tvoří ji 70 000 vzorků ve velmi vysoké kvalitě, dostupných v rozlišeních 128x128 a 1024x1024 pixelů. Jednotlivé vzorky jsou kvalitně ořezané tak, aby zde byly

[6] https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/
[7] https://github.com/NVlabs/ffhq-dataset

Obrázek 4.2: Ukázka vzorků z datové sady IMDB-WIKI

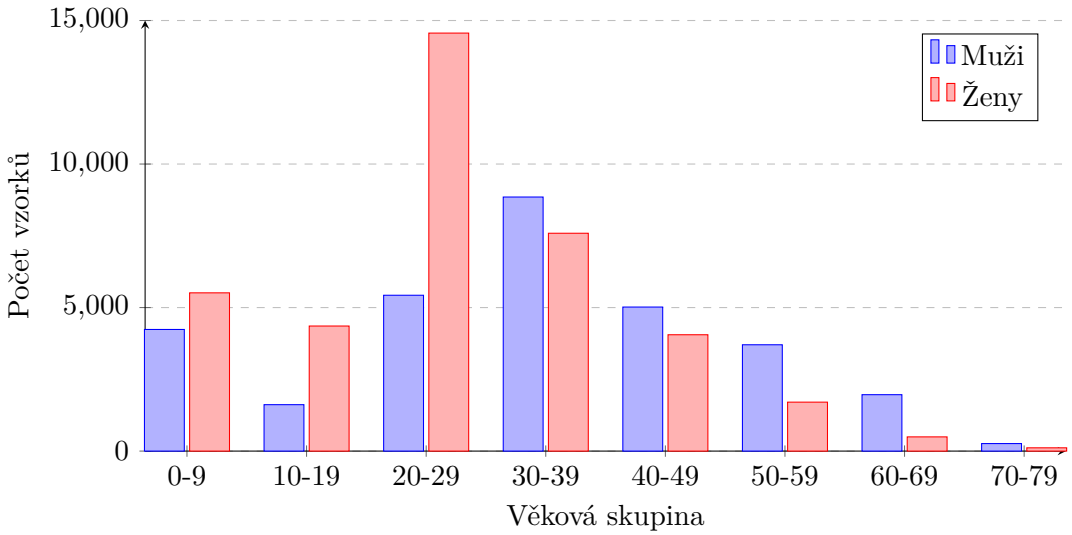
zachyceny především obličeje a vlasy osob na nich zobrazených. Stejně jako za ostatními popsanými datovými sadami, i za vznikem této stál tým výzkumníků. V tomto případě šlo o tým ze společnosti NVIDIA, která je předním celosvětovým výrobcem grafických procesorů. Účelem bylo vytvoření datové sady lidských obličejů dostatečně různorodé a vizuálně kvalitní, aby mohla sloužit jako měřítko kvality natrénovaných generativních neuronových sítí. Z tohoto důvodu nebyly použity fotky známých osobností, jako ve dříve zmíněných datových sadách, ale jednotlivé vzorky byly získány ze sítě pro sdílení fotografií Flickr. Jelikož jsou fotografie na této síti dostupné s různými licencemi, byly do ukázky vzorků v obrázku 4.3 proto vybrány pouze ty s licencí Public Domain, které lze svobodně užívat. Datová sada byla představena společně s architekturou StyleGAN [15]. Kromě nepopíratelně nejvyšší kvality vzorků je další předností také jejich velká variabilita týkající se věku, etnicity, pozadí a v neposlední řadě také doplňků, jako jsou například brýle a čepice. Jedinou nevýhodou je absence informací o věku a pohlaví jednotlivých osob ze vzorků, proto byla k jejich klasifikaci využita neuronová síť, viz sekce 4.3. Distribuce zjištěných dat je prezentována v obrázku 4.4.

## 4.3 Anotace vzorků datové sady

K překonání problému chybějících informací u jednotlivých vzorků u datové sady FFHQ bylo potřeba použít neuronovou síť, která se o automatickou anotaci postará. Zde bylo potřeba dbát na to, aby získané informace byly dostatečně přesné. Špatně anotované vzorky by mohly negativně poznamenat proces trénování, a hlavně kvalitu výsledků. Zde se jako řešení nabízí produkt Azure Cognitive Services společnosti Microsoft, který poskytuje zákazníkům jednoduchý přístup k široké škále služeb z různých odvětví umělé inteligence. Mezi nabízené služby zabývající se počítačovým viděním se řadí *Face*[8]. Z touto službou poskytovaných funkcí nás bude zajímat právě ta sloužící k detekci lidských obličejů. Hlavním

[8] https://azure.microsoft.com/en-gb/services/cognitive-services/face/

Obrázek 4.3: Ukázka vzorků z datové sady FFHQ




Obrázek 4.4: Distribuce vzorků datové sady FFHQ podle věku a pohlaví

účelem je zde určení pozic jednotlivých tváří, ovšem toto není potřeba, vzhledem k tomu, že vzorky pochází již z vytvořené datové sady, a víme tedy, že každý vzorek obsahuje jednu tvář zarovnanou na střed. Mimo jiné je ale možnost detekovat také atributy tváře, jako jsou například věk, pohlaví, barva vlasů, vousy, úsměv, emoce a také seznam nošených doplňků, jako jsou třeba brýle nebo čepice. Tyto informace jsou ze serverů Microsoftu získávány voláním API, které je zasílá ve formátu JSON. Při anotaci vzorků datové sady FFHQ se této

službě nepodařilo detekovat obličeje u všech vzorků, ale jelikož se jedná pouze o 529 vzorků z celkových 70 000, byla tato chyba zanedbána a tyto vzorky nebyly používány k trénování.

## 4.4 Práce s daty

V poslední sekci této kapitoly, než bude věnována pozornost konkrétním strukturám jednotlivých architektur, se podíváme na návrh vstupních a výstupních částí programu pro trénování neuronových sítí. Procesy zpracování vstupních dat i práce s výstupními daty jsou prakticky totožné pro všechny užité architektury.

### Vstupní data

Před začátkem trénování probíhá kontrola vstupních dat. Nalezené vzorky s validními anotacemi jsou spočítány a rovněž, k jejich snazšímu náhodnému výběru během trénování, jsou uloženy názvy jejich zdrojových souborů do seznamu. Do dalších seznamů jsou také ukládány získané informace z anotací, opět k umožnění náhodného výběru, který v případě anotací slouží pro zachování stejné distribuce rysů u generovaných vzorků, aby trénování bylo vyvážené. U informací ohledně věku osob ze vzorků je výrazně zmenšen finální počet vzniklých tříd rozdělením na věkové skupiny po 10 letech.

Každá iterace trénování začíná náhodným výběrem skutečných vzorků. Soubory PNG obsahující vzorky jsou načteny, v případě nutnosti jsou zmenšeny na požadovanou velikost a poté jsou převedeny do datového formátu ndarray. Taktéž je nutné jejich hodnoty přeškálovat z intervalu $<0; 255>$ reprezentující jednotlivé hodnoty barevných složek v RGB formátu na hodnoty z intervalu $<-1; 1>$, protože hodnot z tohoto intervalu dosahují také vygenerované vzorky, kvůli tomu, že je ve výstupní vrstvě generátoru používána aktivační funkce *tanh*. Na každou iteraci trénování jsou náhodně vybrány vzorky, jejich počet je daný zvolenou velikostí minidávky (angl. *minibatch*). Kvůli snížení paměťové náročnosti jsou v paměti uchovávány pouze aktuálně zpracovávané vzorky, nikoliv celá datová sada. Měřeními bylo zjištěno, že tento krok sice prodlužuje dobu trvání epochy zhruba o 3 minuty, ovšem u větších modelů se tato doba stává zanedbatelnou. Kromě získávání skutečných vzorků a jejich odpovídajícím anotacím pro diskriminátor je generován šum podle normálního (Gaussova) rozdělení a z již dříve vytvořených seznamů náhodně vybraných anotací je generátorem vytvořen falešný vzorek. Tyto falešné vzorky a šum jsou vstupy potřebné při trénování jak diskriminátoru, tak generátoru.

### Výstupní data

Hlavním výstupem generativní neuronové sítě je samozřejmě vygenerovaný vzorek, ovšem v zájmu pozdějšího možného zkvalitnění modelů a tím i výsledků je vhodné sledovat i další podrobnosti trénování.

Ke generování vzorků určených k uložení do souboru je jednorázově vygenerován šum, který zůstává statický po celou dobu trénování. Toto se dělá proto, aby bylo možné lépe vidět stav sítě na stále stejných bodech v latentním prostoru během celého procesu trénování. Stejně jako při generování vzorků během trénování, i zde je samozřejmě možné vybírat, do jakých tříd budou generované vzorky spadat. Na různé zajímavé experimenty, které jsou tímto podmíněným generováním umožněné se podíváme v kapitole 6.

K monitorování průběhu trénování jsou sledovány hodnoty chybových funkcí diskriminátoru i generátoru. U architektury DCGAN je možné navíc sledovat přesnost rozpoznávací

funkce diskriminátoru, u ostatních architektur o tuto možnost přicházíme, jelikož se zde mění cíl diskriminátoru z rozpoznávání falešných vzorků na posuzování kvality vzorků. Vývoj těchto hodnot v čase je ukládán ve formě přehledného grafu a tyto data jsou při ladění modelů neuronových sítí velmi užitečná.

Posledním typem výstupních dat jsou natrénované váhy neuronových sítí, které se pravidelně ukládají. Tyto data je vhodné mít zálohovaná, pokud by například trénování nečekaně skončilo. Takto lze pokračovat od tohoto uloženého bodu, místo toho, aby byl celý proces zbytečně ztracen. I pokud se síť dotrénovala úspěšně, je vhodné tyto váhy mít, protože lze sítě po ukončení trénování znovu jednoduše načíst a ihned vygenerovat další stejně kvalitní výsledky. Váhy natrénovaných neuronových sítí všech implementovaných modelů různých architektur jsou k dispozici na přiloženém paměťovém médiu.

# Kapitola 5

# Implementace

V této kapitole je věnována pozornost způsobu implementace konkrétních tří vybraných architektur neuronových sítí. Detailně popsány jsou zde struktury sítí včetně všech typů použitých vrstev a přidaných tenzorových operací, rovněž popsána jsou veškerá nastavení procesu trénování a způsob implementace v API Keras. Nejdříve zde bude v sekci 5.1 věnována pozornost nejjednodušší architektuře DCGAN. V sekci 5.2 je na řadě novější architektura WGAN-GP. Na konci kapitoly v sekci 5.3 následuje pravděpodobně nejzajímavější architektura z této trojice, ProGAN. Jednotlivé výsledky jsou vyhodnoceny v kapitole 6.

## 5.1 Hluboké konvoluční GAN

Jak bylo řečeno v sekci 3.2, natrénovat generativní neuronové sítě není jednoduchá záležitost. A potvrdilo se to i architektury DCGAN, jelikož se jedná o architekturu z dob, kdy byl výzkum teprve ve svých začátcích a nebyly ještě představeny mnohé techniky, které tento proces zjednodušují. V této sekci se podíváme, jakým způsobem jsem se dopracoval k podmíněnému generování barevných lidských tváří modelem této architektury od počátečních experimentů, kde byly generovány pouze černobílá ručně psaná čísla z datové sady *MNIST*[1].

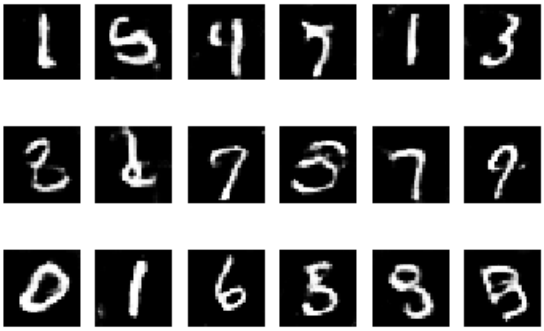
### 5.1.1 Škálování sítě

Vzhledem k tomu, že obtížnost trénování rychle stoupá s přibývající komplexitou vstupních dat, a také s přihlédnutím ke skutečnosti, že tato architektura byla první, kterou jsem implementoval, probíhala její implementace po částech.

Prvním milníkem bylo generování již zmíněných ručně psaných černobílých číslic z datové sady MNIST s rozlišením 28x28 pixelů. Zde nebylo velmi těžké implementovat takovou síť, která by tento problém úspěšně řešila. K řešení tohoto problému bylo využito konvolučních neuronových sítí, s výjimkou dvou vrstev, kde je zvykem používat vrstvy perceptronů [24]. Jedním takovým případem je plně propojená vstupní vrstva generátoru, na kterou navazují dvě dekonvoluční vrstvy s aktivačními funkcemi LeakyReLU a s využitím dávkové normalizace. Výstupní vrstvou generátoru je konvoluční vrstva s hyperbolickým tangentem jako aktivační funkcí, produkující jednokanálový obrázek, který je sledovaným výsledkem. Diskriminátor je tvořen čtyřmi konvolučními bloky se stejnou konfigurací jako u vrstev generátoru, a plně propojenou výstupní vrstvou se sigmoidní aktivační funkcí, aby

[1] http://yann.lecun.com/exdb/mnist/

byla výstupem diskriminátoru hodnota pravděpodobnosti. U obou sítí jsou využity kroky u konvolučních operací ke snižování nebo zvyšování rozlišení dat. Vygenerované čísla jsou vidět na obrázku 5.1.



Obrázek 5.1: Ukázka vygenerovaných vzorků z datové sady MNIST technologií DCGAN

Další pokrok byl podmíněn zvýšením rozlišení vstupních dat. Prvním krokem tedy byl přechod ze vstupního rozlišení 28x28 pixelů na rozlišení 128x128 pixelů, prozatím opět pouze s jedním kanálem dat. Z tohoto důvodu jsem již byl nucen přejít k datovým sadám FFHQ a IMDB-Wiki, které mají své vzorky v dostatečném rozlišení. Druhá zmíněná datová sada se neosvědčila z důvodů popsaných v sekci 4.2 a proto od této chvíle byla k veškerému experimentování využívána pouze datová sada FFHQ. Zvýšení rozlišení od původního k novému nepotřebovalo žádné další zásahy do celkové struktury, kromě přidání dalších konvolučních a dekonvolučních bloků a úměrnému zvětšení celkové kapacity sítě. Na následujícím obrázku 5.2 je vidět, jaké vzorky se podařilo vygenerovat. Trénování probíhalo na tomto rozlišení stabilně, objevují se ale příznaky mode collapse.

Obrázek 5.2: Ukázka vygenerovaných černobílých vzorků z datové sady FFHQ technologií DCGAN

Prvním výraznějším problémem byl přechod z černobílých na barevná data v rozlišení 128x128 pixelů. Zde se začalo ukazovat, že architektura DCGAN není vhodná na příliš velká rozlišení vstupních dat, a během pokusů trénovat neuronové sítě na barevných sním-

cích docházelo rychle k destabilizaci sítě a zkolabování trénování. Byl jsem nucen pečlivě ladit nastavení jednotlivých hyperparametrů neuronových sítí a vyvažovat kapacity generátoru a diskriminátoru tak, aby trénování probíhalo stabilně alespoň po dobu, než bude generátor schopen poskytnout dostatečně kvalitní výsledky. Kromě tohoto ladění se jako nápomocné ukázalo oslabení diskriminátoru použitím vrstvy přidávající šum na jeho vstup a také použití techniky zvané *label smoothing* [7], kde je snížena úroveň jistoty při klasifikaci reálných snímků z pravděpodobnosti 1.0 na 0.9. Toto vedlo k tomu, že diskriminátor se pomaleji učil rozpoznat distribuci reálných dat a podařilo se vygenerovat vzorky, které jsou k vidění na následujícím obrázku 5.3. Společně se stabilizací trénování se zde také podařilo zmírnit problém mode collapse.

Obrázek 5.3: Ukázka vygenerovaných barevných vzorků z datové sady FFHQ technologií DCGAN

Posledním mezikrokem byl přechod z nepodmíněného generování na podmíněné generování, dodáváním třídních informací o vstupních datech obou sítím. Jednalo se o první fázi přidávání tříd vstupních dat, konkrétně šlo prozatím pouze o informace týkající se pohlaví osob ze vzorků datové sady. Rychle se ale projevilo, že ne vždy platí to, že pouhým přidáním třídních informací se zvýší kvalita generovaných vzorků, jak bylo zmíněno v článku [28]. Ačkoliv mi toto poskytlo nástroj k ovlivnění vzhledu vygenerovaných vzorků, bylo to za cenu snížené stability trénování. Spekuluji, že toto bylo způsobené prohloubením rozdílů mezi výkonem generátoru a diskriminátoru, a tato výhoda pro diskriminátor opět způsobila potřebu pečlivě ladit kapacity sítí.

### 5.1.2 Struktury sítí

Finální struktury sítí se od ostatních implementovaných v průběhu prací velmi liší. Přidáním informací o věku (přesněji věkových skupinách) k již zpracovávaným informacím o pohlaví se celkový počet možných tříd vzorků zvýšil ze 2 na 14, a opět se zvýšila náročnost trénování takových neuronových sítí.

#### Diskriminátor

Částečné vyrovnání „sil“ generátoru a diskriminátoru se podařilo zajistit až kompletní změnou struktury generátoru, kterou si zde popíšeme a ukážeme. Začněme ale popisem

diskriminátoru, který zůstává pořád skoro stejný. Jádrem diskriminátoru jsou stále konvoluční bloky implementované vrstvami `Conv2D` s aktivační funkcí `LeakyReLU`. Počet filtrů konvolučních vrstev bloků se s každou vrstvou zdvojnásobuje, začíná na 32 a končí na 512. Rovněž je zde utilizována dávková normalizace přidáním vrstvy `BatchNormalization` do všech bloků kromě prvního. Snižování rozlišení dat je zajištěno nastavením parametru `stride` u konvolucí, který reprezentuje velikost kroku konvolučního jádra. Vstupem sítě jsou reálné a falešné vzorky, ke kterým je přidávaný šum vrstvou `GaussianNoise`. Tento vstup je vrstvou `Concatenate` spojený s třídními labely, které byly předem zpracovány vrstvou `Embedding` a pomocí vrstvy `Reshape` transformovány do požadovaného tvaru. Výstup konvolučních bloků v rozlišení 4x4 je zploštěn na vektor vrstvou `Flatten` a posléze zpracován plně propojenou vrstvou `Dense` s aktivační funkcí `sigmoid` k získání hodnoty pravděpodobnosti.

### Generátor

U generátoru se osvědčil přechod z dekonvolučních vrstev na konvoluční vrstvy. Bylo provedeno mnoho experimentů s různými nastaveními obou typů vrstev a trénování generátoru založených na konvolučních vrstvách se ukázalo jako celkově stabilnější. Generátor je tedy ve finální implementaci tvořen šesti konvolučními bloky, které jsou prakticky identické blokům použitým u diskriminátoru pouze s tím rozdílem, že místo kroků konvolucí ke snižování rozlišení jsou u prvních čtyř bloků přidány `UpSampling2D` vrstvy ke zdvojnásobování rozlišení. Počet filtrů začíná na 1024 v prvním bloku a v každém následujícím bloku je filtry produkovaný počet kanálů snížen na polovinu. Vstupní šum je nejdříve zpracován vrstvou `Dense` s 8096 neurony, která je aktivována funkcí `LeakyReLU`, poté je tento šum spojen ve vrstvě `Concatenate` se zpracovanými labely, výsledek je uspořádán vrstvou `Reshape` do rozlišení 4x4, následně pomocí vrstvy `UpSampling2D` zvětšen do rozlišení 8x8 a poté již prochází zmíněnými konvolučními bloky. Výstupní vrstva zůstává neměnná, jedná se o 1x1 konvoluční vrstvu `Conv2D` aktivovanou funkcí `tanh`, jejíž výstupem je barevný obrázek v rozlišení 128x128 pixelů.

### Nastavení hyperparametrů

Pro trénování byl vybrán optimalizační algoritmus `Adam`, již dříve popsaný v sekci 2.2.3. Hodnoty parametrů pro velikost learning rate a hybnosti prvního řádu (známější pod názvem $\beta_1$) byly převzaty z článku představující technologii DCGAN [24], ostatní hodnoty jsou ponechány na výchozích hodnotách nastavených v Keras. Volba těchto hodnot byla ověřena krátkými experimenty, kdy jsem vyzkoušel různé jiné hodnoty za stejných podmínek. Nastavené hodnoty jsou vidět v tabulce 5.1.

Vzhledem ke skutečnosti, že diskriminátor klasifikuje vzorky do dvou tříd (reálné/falešné), byla použita chybová funkce `BinaryCrossentropy`. Tato funkce v API Keras nativně podporuje label smoothing, proto byl zde nastaven na velikost 0.1.

Poslední nastavovanou hodnotou je velikost směrodatné odchylky přidávaného šumu u vrstvy `GaussianNoise` na vstupu diskriminátoru k regularizaci chybové funkce, jejíž hodnota začíná na počátku trénování na 0.5 a poté se její hodnota podle exponenciální funkce ve vzorci 5.1 snižuje. Trvání epochy je dané velikostí datové sady. Jedna epocha uplyne, když je zpracováno diskriminátorem množství vzorků odpovídající celkovému množství vzorků v datové sadě.

$$stddev = 2^{-epoch} * 0.5 \tag{5.1}$$

| Chybová funkce | Optimalizační algoritmus | Míra učení | Hybnost prvního řádu | Velikost minidávky | Velikost latentního prostoru |
|---|---|---|---|---|---|
| Binární křížová entropie | Adam | 0.0002 | 0.5 | 64 | 100 |

Tabulka 5.1: Nastavené hodnoty hyperparametrů u architektury DCGAN

### 5.1.3 Setrvávající problémy s implementovanou architekturou

Výsledný model sice úspěšně produkuje lidské obličeje podmíněně na základě vybraných tříd, ale nejedná se o ideální model. S pokračujícím škálováním sítě a zvyšováním komplexity vstupních dat se snižovala celková stabilita sítě a tento problém se nepodařilo úplně vyřešit. Trénování nemíří ke konvergenci a vždy skončí tím, že se obě sítě přestávají učit, protože diskriminátor dokonale rozpoznává falešné vzorky od skutečných a neposkytuje generátoru dostatečně efektivní zpětnou vazbu. Dalším problémem, který se zde hojně objevuje, je částečný mode collapse. Po nejrůznějších neúspěšných experimentech se strukturami obou sítí a s procesem trénování, například pokus generátor trénovat více než diskriminátor, pokud byla přesnost diskriminátoru příliš vysoká, byl výsledný model nechán v podobě, která se nejvíce osvědčila a tato část práce zůstává otevřenou. Naštěstí se jedná o problém, na který bylo naraženo pouze v případě této dnes již zastaralé architektury.

## 5.2 Wasserstein GAN s penalizací gradientu

Výběr této architektury byl dobrým krokem, protože se jedná o skutečně ideální přechod mezi architekturou DCGAN a nejmodernějšími architekturami, kterými jsou ProGAN a StyleGAN. V této sekci je vysvětleno, jakým způsobem lze přidat Wasserstein vzdálenost jako chybovou funkci, a jak je implementována penalizace gradientu. Co se týče struktury sítí, tak zde jsem vyzkoušel dvě různé - jedna je identická té z architektury DCGAN a druhá té z architektury ProGAN. Tyto sítě byly implementovány pro trénování na barevných obrázcích z datové sady FFHQ v rozlišení 128x128 a pro jejich podmíněné generování. Předtím, než budou popsány struktury jednotlivých sítí, bude vhodné se nejdříve podívat na dvojici nových chybových funkcí.

### 5.2.1 Wasserstein vzdálenost a penalizace gradientu

Přidání výpočtu Wasserstein vzdálenosti je velmi jednoduché. Celá implementace je pouze přidání nové funkce provádějící následující výpočet:

$$loss = avg(y\_true * y\_pred), \tag{5.2}$$

kde *y_true* je tenzorem s očekávanými hodnotami, *y_pred* je tenzorem s modelem předpovězenými hodnotami a funkce *avg* značí operaci aritmetického průměru.

Penalizace gradientu k zajištění podmínky Lipschitz kontinuity je implementována jako další chybová funkce. Začněme tedy popořádku - nejdříve je nutné si určit, co vlastně tato chybová funkce bude hodnotit. Cílem splnění podmínky je držet hodnoty gradientů pod určitou hranicí. Je ovšem nemožné toto zajistit pro všechny body ve vstupním prostoru, proto bylo jako řešení představeno získání náhodných bodů mezi reálnými a generovanými

vzorky a zjišťování hodnot gradientů zde [10]. Prakticky to znamená, že vezmeme minidávku falešných a reálných vzorků, vygenerujeme sadu náhodných hodnot $\alpha$ v rozsahu 0-1 a tvaru odpovídajícímu datům. Jednu sadu vzorků násobíme hodnotou $\alpha$ a druhou hodnotou $1-\alpha$. Následně je sečteme a na těchto nových vzorcích kontroluje chybová funkce gradienty. Aby bylo v API Keras umožněno tuto chybovou funkci použít, je vytvořena pomocí funkce `partial` z knihovny `functools` jazyka Python, která k tenzorům `y_pred` a `y_true` dovolí přidat jako další argument právě tyto nově vypočítané obrázky.

Nyní, když jsou všechny chybové funkce připraveny, stačí je přidat k modelu diskriminátoru a nastavit jim váhy. Zde je použita doporučená hodnota z článku [10]. Chybová funkce pro penalizaci gradientu má nastavenou desetinásobně vyšší váhu, než Wasserstein chybová funkce pro zpracovávání reálných a falešných vzorků.

### 5.2.2 Struktury sítí

Implementované neuronové sítě mají velmi velkou kapacitu a potenciálně by mohly stačit i ke generování obrázků v rozlišení 1024x1024 pixelů, ovšem z kapacitních důvodů trénovacího prostředí probíhalo trénování pouze na rozlišení 128x128 pixelů. Byl proveden pokus použít tuto dvojici nových chybových funkcí pro sítě architektury DCGAN představené v minulé podsekci, šlo ovšem pouze o částečný úspěch. Podrobnosti tohoto experimentu jsou popsány v části 5.2.3. Nakonec byla jako vhodnější alternativa použita dvojice vysokokapacitních neuronových sítí s konvolučními bloky obsahujícími dvě konvoluční vrstvy [14].

#### Diskriminátor

Práce se vstupními labely je identická napříč všemi implementovanými architekturami. Dvojice celočíselných hodnot reprezentující třídy vzorků je vrstvou `Embedding` zpracována do vektoru a pomocí operace `Reshape` je její tvar změněn na takový, aby ho bylo možné pomocí vrstvy `Concatenate` spojit dohromady se vstupním obrázkem. Tento vstup se 3 barevnými kanály obrázku a 2 kanály pro vstupní labely je poté pomocí `Conv2D` vrstvy s 1x1 konvolučním jádrem projektován do 128 kanálů.

Tato projekce poté prochází 6 konvolučními bloky, z nichž každý je tvořen sadou dvou `Conv2D` vrstev s 3x3 jádry, aktivovanými funkcí `LeakyReLU`. Následně je její rozlišení sníženo na polovinu sdružovací vrstvou `AveragePooling2D`. Poslední blok je rozdílný od ostatních, jelikož je výstupním blokem. Na jeho vstupu je přidána vrstva `MinibatchStdev`, která je implementací vrstvy zmíněné v sekci 5.3. Zároveň druhá z konvolucí výstupního bloku neobsahuje padding hodnoty a má zvýšenou velikost svého jádra na 4x4, aby jejím výstupem byl vektor. Tento výstupní vektor je zpracován `Dense` vrstvou s lineární aktivací. Výstupem je tedy reálná hodnota, nikoliv pravděpodobnost, jako v případě architektury DCGAN, jak bylo popsáno v sekci 3.3. Počet filtrů jednotlivých konvolucí je průběžně rostoucí a jejich množství je zmíněno v části zabývající se nastavenými hyperparametry.

#### Generátor

Vstupní náhodný šum zde není zpracováván plně propojenou vrstvou, nýbrž je pomocí operace `Reshape` nejdříve transformován na tenzor kompatibilní jako vstup `Conv2DTranspose` vrstvy s `LeakyReLU` aktivací, kterou je zvětšen na rozlišení 4x4, a posléze pomocí vrstvy `Concatenate` spojen s podmíněnými informacemi o vzorcích. Toto vše je zpracováno klasickou `Conv2D` vrstvou s `LeakyReLU` aktivací. Na tento vstupní blok dále navazují další konvoluční bloky, které zvyšují prostorové rozlišení. V této sítí je na všechny `Conv2D` vrstvy kromě

poslední aplikována speciální normalizace každého pixelu, nazvaná `PixelNormalization`, viz článek [14].

Konvoluční bloky jsou v generátoru rovněž tvořeny dvojicemi vrstev `Conv2D` s 3x3 konvolučními jádry a aktivací `LeakyReLU`. Na počátku každého bloku je vrstvou `UpSampling2D` nejdříve zdvojnásobeno rozlišení. Počet filtrů je u obou konvolucí v daném bloku stejný a snižuje se s rostoucí hloubkou sítě.

Výstupní vrstvou sítě je opět konvoluční vrstva s 1x1 jádrem, která projektuje všechny kanály do výsledných tří barevných kanálů. Zde není použita aktivace hyperbolickým tangentem, ale jedná se o lineární aktivaci, takže správnou distribuci hodnot dat se síť učí sama.

#### Nastavení hyperparametrů

Použit je pro konvoluční sítě nejpoužívanější optimalizační algoritmus, tedy `Adam`. Nastavovány jsou u něj kromě learning rate také hybnosti prvního i druhého řádu, kde jejich hodnoty odpovídají těm v článku [14] a jsou vidět přehledně v tabulce 5.2 společně s ostatními hyperparametry. Síť diskriminátoru je zde trénována vícekrát během jedné iterace trénování, jak bylo doporučeno v článku [10].

Počet filtrů konvolucí, který udává, kolik kanálů dat budeme konvoluovat, je pro obě sítě stejný a je daný polem $[512, 512, 512, 512, 256, 128]$. Každý index v tomto poli odpovídá dvěma konvolučním vrstvám. Generátor tímto polem postupuje od začátku a diskriminátor od konce.

Jak již bylo řečeno, kapacita sítí je velmi vysoká, a tomu odpovídají i celkové počty trénovatelných parametrů jednotlivých sítí. Jde o 23,194,497 parametrů pro diskriminátor a 22,935,939 pro generátor.

| **Optimalizační algoritmus** | **Míra učení** | **Hybnost prvního řádu** | **Hybnost druhého řádu** |
|---|---|---|---|
| Adam | 0.0001 | 0 | 0.99 |

| **Chybové funkce** | **Velikost minidávky** | **Velikost latentního prostoru** | **Počet aktualizací diskriminátoru** |
|---|---|---|---|
| Wasserstein vzdálenost, penalizace gradientu | 16 | 512 | 5 |

Tabulka 5.2: Nastavené hodnoty hyperparametrů u architektury WGAN-GP

### 5.2.3 Experiment se strukturou sítí z DCGAN

Než byla pro tuto architekturu implementována struktura představená v minulé podsekci, byla vyzkoušena struktura odpovídající sítím z předchozí architektury, aby bylo pozorováno, jaký má na ní vliv pouhá změna chybové funkce.

Výsledek byl ovšem částečným zklamáním. Při použití struktury sítí totožné té, která fungovala pro DCGAN se nepodařilo natrénovat sítě s Wasserstein chybovou funkcí, což je opačný výsledek, než jaký byl prezentován v článku představující tuto technologii [10]. Trénování bylo po určitou dobu stabilní, nebyly zjevné žádné problémy jako mode collapse,

kvalita vzorků se postupně zlepšovala, ale v určitém bodě vždy trénování nakonec zkolabovalo. Výsledky, které jsou k vidění na obrázku 5.4, nejsou tedy nakonec vizuálně kvalitní jako výsledky, kterých bylo dosaženo s DCGAN. Kvalitnějších výsledků se podařilo docílit až po změnách struktur sítí, které se spíše podobaly finální struktuře a navíc nefungovaly při použití zpět v architektuře DCGAN.

Obrázek 5.4: Ukázka vygenerovaných vzorků technologií WGAN-GP s použitím struktur sítí z architektury DCGAN

## 5.3 Progresivně rostoucí GAN

Nyní se dostáváme k pravděpodobně nejzajímavější architektuře z této trojice. Princip progresivně rostoucích sítí se stal klíčovou součástí moderních architektur a my se zde podíváme, jak lze takové sítě ve frameworku Keras implementovat.

### 5.3.1 Implementace růstu sítí

Jelikož struktura generátoru i diskriminátoru je totožná té, která byla již představena v minulé sekci o architektuře WGAN-GP, zaměříme se na to, jakým způsobem je umožněno v průběhu trénování přidávat nové vrstvy a jak je zajištěno jejich hladké přidání do sítě.

Vzhledem k tomu, že máme dva typy sítí - jeden, kde je v průběhu zvyšování rozlišení a druhý, kde jsou sítě stabilizovány, není možné mít jeden model, kde by byly všechny vrstvy a pouze by se v průběhu trénování povolovaly aktualizace jejich vah. Nejlepším řešením je tedy pro každou fázi trénování různých rozlišení vytvořit samostatné modely, které budou společně sdílet vrstvy, jejichž natrénování je cílem. Tyto modely jsou vytvořeny, zkompilovány a během tréninku se tedy v závislosti na aktuální fázi mění momentálně trénovaný model. Při trénování sítě na rozlišeních 4x4 pixelů až 128x128 pixelů tedy vytvoříme 12 různých modelů. Podívejme se, jakým způsobem se tyto modely tvoří.

### Diskriminátory

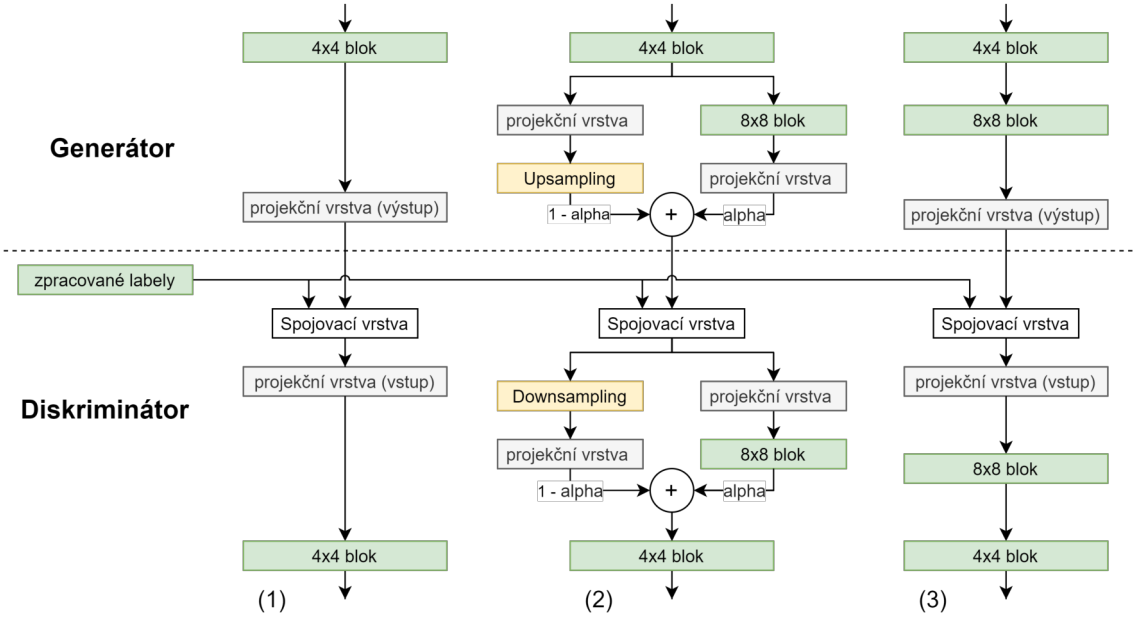
Při vytváření modelů diskriminátorů se začíná vytvořením modelu pro rozlišení 4x4 pixely. Nejsou zde zpočátku žádné konvoluční bloky, pouze výstupní vrstvy a vstupní vrstvy, které byly upraveny na práci s menším rozlišením. Při vytváření modelu pro stabilizaci rozlišení 8x8 je vytvořen nový vstup pro síť, ke kterému je přidán jeden konvoluční blok. Model pro zvyšování rozlišení tento nový vstup sítě zmenší na předchozí rozlišení, tedy 4x4, pomocí vrstvy `AveragePooling2D` a zpracuje ho pomocí projekční `Conv2D` vrstvy z modelu pro toto nižší rozlišení. Nyní, když jsou vstupy nových modelů připraveny, musí dojít k jejich napojení na model zpracovávající 4x4 rozlišení. Výstupem modelu pro zvyšování rozlišení je sečtený výstup modelu pro stabilizaci rozlišení vynásobený proměnnou $\alpha$ a vlastní výstup vynásobený hodnotou $1 - \alpha$, na který je napojen celý zbytek sítě z rozlišení 4x4. Model pro stabilizaci rozlišení 8x8 je rovněž napojen na předchozí model zpracovávající rozlišení 4x4. Tento proces se stejným způsobem opakuje pro každé zvýšení rozlišení. Tento princip je přehledněji vizualizován na obrázku 5.5.

### Generátory

Proces vytváření nových dvojic generátorů je jednodušší. Opět se začíná vytvořením modelu pro nejnižší zpracovávané rozlišení, tedy 4x4 pixely. Jedná se o stejný vstupní a výstupní blok generátoru, jaký byl použit u implementace architektury WGAN-GP. Když vytváříme modely pro stabilizaci vyššího rozlišení, stačí na předposlední vrstvu předchozího modelu (výstupní vrstvu tedy zahazujeme) napojit nový konvoluční blok a novou výstupní vrstvu. Model pro zvyšování rozlišení s vrstvami tohoto modelu také pracuje. Výstupem modelu pro hladké zvýšení rozlišení je vážený součet předchozího výstupu sítě, který má zdvojnásobené rozlišení vrstvou `UpSampling2D` s výstupem nového modelu pro stabilizaci tohoto vyššího rozlišení. Opět jsou využity hodnoty $\alpha$, kde zvětšený původní výstup má váhu $1-\alpha$ a nový výstup má váhu $\alpha$. Tímto je zajištěno, že nový konvoluční blok a výstupní vrstva se mohou trénovat, jejich výstup ale bude mít zpočátku nízký vliv na celkový výsledek a tím pádem na chybovou funkci, jelikož hodnota $\alpha$ se lineárně zvětšuje od 0 do 1 v průběhu fáze zvětšování rozlišení. K ujasnění tohoto procesu je i vytváření modelů generátoru ukázáno na obrázku 5.5.

## Nastavení hyperparametrů

Hodnoty hyperparametrů jsou prakticky totožné s hodnotami ze sekce 5.2.2, které byly použity pro architekturu WGAN-GP. Jediným rozdílem oproti referovaným nastavením je zde to, že obě sítě se trénují stejně často, neprobíhá zde tedy více aktualizací diskriminátoru během jedné iterace trénování.




Obrázek 5.5: Ukázka obecného průběhu zvyšování rozlišení na příkladu zvětšování sítě z rozlišení 4x4 pixelů na 8x8 pixelů. Část označená (1) reprezentuje model pro stabilizaci rozlišení 4x4, část (2) reprezentuje model pro hladké zvyšování rozlišení na 8x8 pixelů a část (3) reprezentuje model pro stabilizaci trénování na rozlišení 8x8 pixelů (inspirováno obrázkem z článku [14])

# Kapitola 6

# Vyhodnocení výsledků

V této kapitole se podíváme na vzorky vygenerované neuronovými sítěmi architektur implementovaných tak, jak bylo popsáno v kapitole 5. Nejdříve je sekci 6.1 popsáno, za jakých podmínek trénování modelů probíhalo a poté jsou v následující sekci 6.2 již ukázány konkrétní dosažené výsledky. Výsledky experimentů, kterými byla ověřena správná funkčnost modelu ProGAN, prezentuji v sekci 6.3. Závěrečná sekce 6.4 je poté ohlédnutím za dosaženými výsledky, kde prezentuji, jakým způsobem by bylo možné dále zlepšit jejich kvalitu.

## 6.1 Průběh trénování

Vzhledem k vysoké náročnosti na grafický výpočetní výkon nebylo možné trénování modelů neuronových sítí ani experimenty s nimi uskutečnit v domácích podmínkách. Na počáteční experimenty byla využita platforma *Google Colaboratory*[1], která je cloudovou službou společnosti Google. Každému je zde k dispozici grafická karta NVIDIA Tesla K80, kterou lze zdarma využívat na libovolné experimenty až 12 hodin v kuse. Ovšem při přílišném používání grafické karty je přístup k ní na den odebrán a původní limit je zkrácen na 6-8 hodin denně. Vzhledem k tomu, že trénování modelů generativních neuronových sítí je dlouhý proces, který na samostatné grafické kartě může trvat několik týdnů až měsíce, bylo třeba najít alternativní prostředí, kde mi bude umožněno modely trénovat libovolně dlouho.

Pozdější část práce a její dokončení bylo tedy možné díky fakultnímu Centru výpočetní techniky, kde mi byl poskytnut přístup k výpočetnímu clusteru[2]. Zde jsem měl k dispozici velký grafický výpočetní výkon, a zejména důležité pro mě bylo to, že jsem své modely mohl nechat trénovat libovolně dlouhou dobu bez omezení přístupu. Zpracování úloh v tomto clusteru řídí systém *Sun Grid Engine*, zkráceně *SGE*, který poté úlohu spouští na některém ze svých uzlů. Ve skriptu, který je tedy systému SGE předložen, bylo nutné správně zarezervovat potřebné zdroje pro běh programu, zajistit, aby byla na uzlu pro program správně vybrána volná grafická karta a rovněž vybrána správná verze platformy *CUDA*. CUDA je architekturou grafických karet NVIDIA, umožňující spouštět na grafických kartách programy napsané například v jazycích C/C++. Přesun výpočtů na grafické karty poskytuje značné zrychlení oproti počítání na procesoru. Běh všech experimentů probíhal v tomto prostředí, s využitím knihovny Tensorflow verze 1.15.0 a technologie CUDA verze 10.0.

[1] https://colab.research.google.com/
[2] https://www.fit.vut.cz/units/cvt/cluster/

### Datová sada

Všechny implementované modely byly trénovány na vzorcích z datové sady FFHQ společně se získanými anotacemi jednotlivých vzorků, tak jak bylo popsáno v kapitole 4. Bohužel verze této datové sady se vzorky ve vyšším rozlišení 1024x1024 pixelů byla příliš velká a její nahrání na školní servery nebylo možné. Z tohoto důvodu probíhalo trénování pouze na vzorcích ve velikosti 128x128 pixelů, kde byla celková velikost datové sady necelé 2GB. Nicméně tento problém se týká především architektury ProGAN, trénování modelů ostatních architektur by totiž bylo na vyšším rozlišení nejspíše náročné, zejména v případě modelu architektury DCGAN.

### Modely DCGAN

Trénování modelů neuronových sítí architektury DCGAN bylo nejjednodušší, jelikož se jedná o poměrně nízkokapacitní modely, díky čemuž jsou jejich nároky na paměť malé. Ke spuštění a trénování tohoto modelu je potřeba alespoň 4GB grafické paměti a 4GB paměti RAM. Doba potřebná k natrénování se pohybovala v rozmezí 7-14 dní, ačkoliv jak již bylo zmíněno v předchozí kapitole, trénování nekonverguje, toto časové rozmezí tedy pouze značí dobu, než prakticky vymizí gradienty a trénování se nikam dále neposunuje.

### Modely WGAN-GP a ProGAN

Paměťové nároky těchto modelů jsou velmi vysoké, jelikož celková kapacita jednotlivých modelů je až 4x vyšší, než u modelů DCGAN. Z tohoto důvodu se pro stabilitu trénování bez problémů s alokací paměti ukázalo jako potřebné zvýšit požadavky uvedené ve skriptu pro SGE na minimálně 12GB grafické paměti. Naštěstí školní výpočetní cluster disponuje více takovými grafickými kartami, které toto minimum splňují, a ani u těchto modelů nebyl problém s přidělením zdrojů na dobu potřebnou k natrénování modelů, která se v tomto případě pohybovala v rozmezí 3-4 týdnů.

## 6.2 Porovnání výsledků jednotlivých architektur

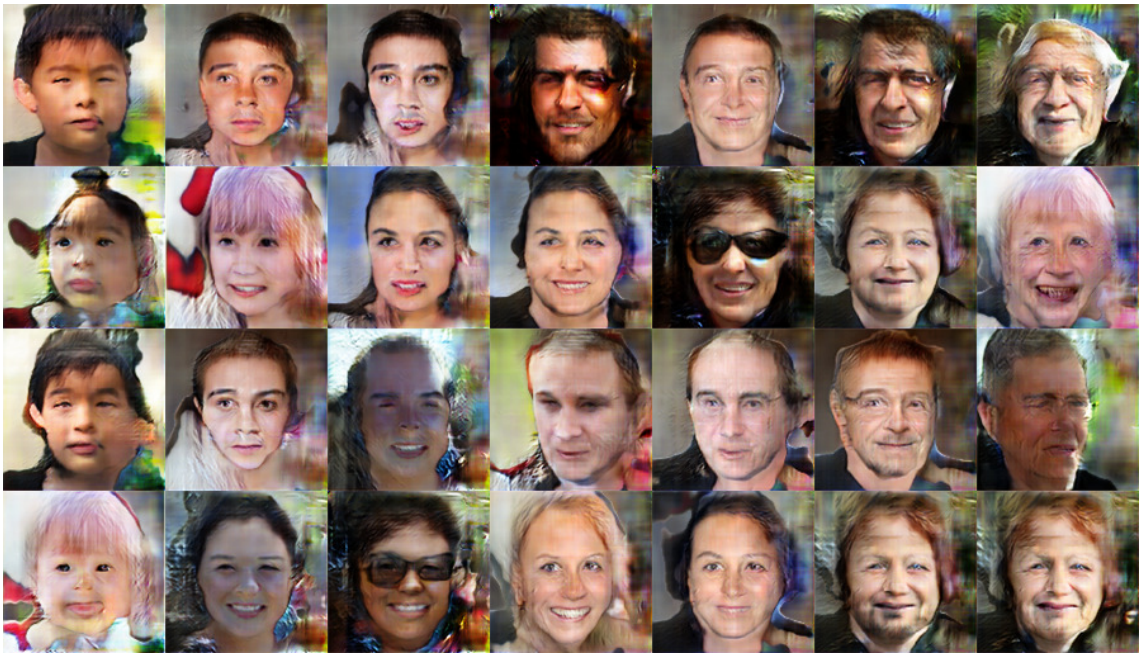
Poslední fází práce s generativními neuronovými sítěmi, je po fázích implementace a trénování fáze evaluace. Na základě těchto výsledků lze zhodnotit, jak dobře modely pracují a dá se odvodit, jakým směrem by se měl další vývoj za účelem jejich zkvalitnění dále ubírat. Podívejme se tedy na dosažené výsledky jednotlivých modelů sítí implementovaných podle popisů v kapitole 5, a na další zajímavé experimenty, které byly s natrénovanými modely provedeny, pomocí kterých lze také sítě ohodnotit.

### Kvalita generovaných vzorků

Hlavním hodnoceným faktorem u vygenerovaných vzorků lidských tváří, jejichž generování bylo cílem této práce, je samozřejmě vizuální kvalita. Ke správné evaluaci výkonu neuronových sítí bylo také ale vhodné sledovat další vlastnosti generovaných dat, kterými jsou například jejich variabilita, a také, což bylo dalším cílem této bakalářské práce, správné podmíněné generování. Splnit tyto základní cíle se podařilo napříč všemi třemi implementovanými architekturami.

### DCGAN

Jak již bylo zmíněno dříve, přidáním třídních informací na vstupy sítí zde došlo ke snížení variability generovaných vzorků, vzhledem k tomu, že výsledný natrénovaný model trpí příznaky problému mode collapse. Vzorky jsou správně generované na základě zvolených podmínek, a to jak pohlaví, tak věku. Projevy mode collapse lze sledovat především mezi vzorky stejného pohlaví různých věkových kategorií, kde se často opakují různé rysy obličeje, nebo se jedná přímo o úplně totožné vzorky. Kromě tohoto jsou některé vzorky naprosto nerealistické, a lze na nich rozpoznat pouze několik dominantních částí obličeje, zatímco zbytek vzorku je nerozeznatelný. Tato skutečnost je odůvodněná tím, že se nejedná o natrénovaný model sítě, který by úspěšně konvergoval k dostatečně kvalitnímu lokálnímu minimu, nýbrž o model, jehož trénování skončilo předčasně kvůli nerovnováze jednotlivých sítí. Vygenerované vzorky, které jsou k vidění na obrázku 6.1, jsou výstupem sítě po 14 dnech trénování.



Obrázek 6.1: Ukázka ručně vybraných vzorků podmíněně vygenerovaných modelem sítě architektury DCGAN, kde se na jednotlivých řádcích střídají pohlaví a ve sloupcích věkové skupiny. První sloupec reprezentuje věkovou skupinu 0-9 let, druhý 10-19 let, atd.

### WGAN-GP

Výrazně kvalitnějších výsledků bylo dosaženo použitím architektury využívající Wasserstein vzdálenost a penalizaci gradientu jako chybové funkce. U většiny vygenerovaných vzorků jsou k vidění realistické rysy lidských obličejů, společně se spolehlivým podmíněným generováním na základě kombinace věku a pohlaví. Trénování tohoto modelu do stavu, kdy byly generovány vzorky jako jsou k vidění na obrázku 6.2, trvalo 20 dní.

Obrázek 6.2: Ukázka ručně vybraných vzorků podmíněně vygenerovaných modelem sítě architektury WGAN-GP, kde se na jednotlivých řádcích střídají pohlaví a ve sloupcích věkové skupiny. První sloupec reprezentuje věkovou skupinu 0-9 let, druhý 10-19 let, atd.

#### ProGAN

Subjektivní kvalita vzorků vygenerovaných architekturou WGAN-GP byla ještě zvýšena použitím progresivního růstu, kterým bylo umožněno natrénovat neuronové sítě ke generování realistických tváří mnohem rychleji. Při podrobnějším pohledu je možné si všimnout, že vzorky architektury ProGAN jsou detailnější než předchozí vzorky. Důvodem je právě progresivní růst, který umožnil neuronovým sítím nejdříve se učit celkovou strukturu tváří a až poté stále drobnější detaily, což je efektivnější postup, než když se sítě učí od počátku na vzorcích s vysokým rozlišením, jak tomu bylo u předchozích architektur. Jelikož se tyto výsledky dají považovat za nejkvalitnější, byl stejný natrénovaný model použit také pro experimenty, které byly provedeny a kterými se zabývám v následující sekci. Trénování tohoto modelu bylo spuštěno na nejdelší dobu v porovnání s ostatními dvěma modely, probíhalo po 30 dní a vygenerované obličeje jsou k vidění na obrázku 6.3.

## 6.3 Experimenty s progresivním modelem

Vzhledem ke skutečnosti, že cílem generátoru je naučit se distribuci vstupních dat, na které jsou mapovány hodnoty šumu, a také k tomu, že tento prostor je rozdělen do tříd podle sledovaných rysů, je možné provést celou řadu experimentů. Každý experiment, který bude v této části předveden, se tedy týká některé z těchto vlastností, nebo jejich kombinací. Vzhledem k tomu, že se ale pohybujeme v kontextu generování lidských tváří, názvy těchto experimentů odpovídají výsledkům, které díky nim můžeme pozorovat.

Obrázek 6.3: Ukázka ručně vybraných vzorků podmíněně vygenerovaných modelem sítě architektury ProGAN, kde se na jednotlivých řádcích střídají pohlaví a ve sloupcích věkové skupiny. První sloupec reprezentuje věkovou skupinu 0-9 let, druhý 10-19 let, atd.

### Proces stárnutí

Prvním z řady experimentů je generování vzorků ze stejného vstupního šumu, ovšem s lišícími se vstupními labely. Manipulováno je zde s hodnotami labelů reprezentující konkrétní věkovou skupinu, do které má generovaný vzorek spadat. V praxi tedy můžeme pozorovat, jakým způsobem se generátor naučil vnímat závislost věku osoby na jejím vzhledu. Aby experiment měl vypovídací hodnotu, všechny generované vzorky na každém řádku, které jsou k vidění na obrázku 6.4, pochází ze stejné vstupní hodnoty šumu. Můžeme zde tedy sledovat, jak tatáž osoba může vypadat v mládí, v dospělosti a ve stáří.

### Transformace do opačného pohlaví

Principiálně stejný je i následující experiment, kde jsem rovněž měnil hodnoty vstupních labelů, ale tentokrát se jednalo druhý vstupní label, který odpovídá pohlaví. Na obrázku 6.5 tedy můžeme pozorovat, jakým způsobem se neuronová síť naučila odlišit vzorky na základě požadovaného pohlaví. Pro zajímavost tedy opět používám stejný vstupní šum pro dvojice generovaných vzorků. Výsledné vzorky se tedy liší v rysech, které jsou závislé na pohlaví, jako jsou například vousy. Naopak rysy obličejů, u kterých na pohlaví nezáleží, zůstávají na obou vzorcích stejné.

### Vektorová aritmetika s hodnotami šumu

Dvojice předchozích experimentů byla orientována na změny hodnot vstupních značek modelů. Jejich fungování bylo tedy umožněno tím, že se jedná o podmíněné sítě. Zbylé dva experimenty jsou ale realizovatelné i mimo oblast podmíněných neuronových sítí, kvůli

Obrázek 6.4: Experiment se vstupními labely modelu ProGAN - proces stárnutí různých osob, kde na každém řádku je vzorek generován ze stejného vstupního šumu, s postupně měnící se hodnotou labelu reprezentující třídu věkové skupiny

tomu, že jde pouze o práci s latentním prostorem, respektive vstupním šumem. Tyto experimenty byly představeny v článku [24], a jejich účelem bylo primárně ověření, zda se generátor opravdu naučil distribuci datové sady, či jen memoroval určité vzorky. Mimo jiné se ale autorům článku podařilo zjistit, že schopnost přesvědčivě modelovat vlastnosti vzorků v latentním prostoru není omezena pouze na podmíněné sítě.

Jednou takovou ukázkou této funkcionality je experiment s vektorovou aritmetikou v latentním prostoru. Myšlenka tohoto experimentu spočívá v tom, že pokud jsou data lineárně reprezentována v prostoru, lze s nimi provádět operace. Zde se podíváme, jak mezi sebou lze sčítat a odčítat hodnoty šumu určitých vzorků, a jejich výsledek využít ke generování nového vzorku, který bude sémanticky odpovídat očekávanému výsledku této aritmetiky.

Vzhledem k tomu, že tento experiment byl prováděn na podmíněné neuronové síti, která je již schopna rozpoznávat pohlaví, důležité je zde sledovat jiné charakteristiky, které odpovídají konkrétní aritmetice a nebyly naučeny již během trénování. Jednou takovou ukázkou může být úsměv, který se na finálním vzorku nenachází, pokud to je výsledkem konkrétní aritmetiky se vzorky. Ukázka výsledků čtyř výpočtů s hodnotami šumu je vidět na obrázku 6.6.

### Cesta latentním prostorem

Smyslem také tohoto experimentu je ověření, zda se neuronová síť generátoru skutečně naučila distribuci datové sady. Docíleno je toho průzkumem povrchu latentního prostoru. V tomto experimentu se nejedná o pohyb náhodným směrem, nýbrž je nejdříve ze dvou různých hodnot šumu vygenerována dvojice vzorků. Na úsečce spojující tyto body je poté rovnoměrně vybráno několik bodů, z jejichž hodnot jsou také vygenerovány vzorky, jak je

Obrázek 6.5: Experiment se vstupními labely modelu ProGAN - transformace do opačného pohlaví různých osob, kde každý vzorek z dvojic v jednotlivých sloupcích je generován za použití stejného vstupního šumu, s měnící se hodnotou labelu reprezentujícího pohlaví

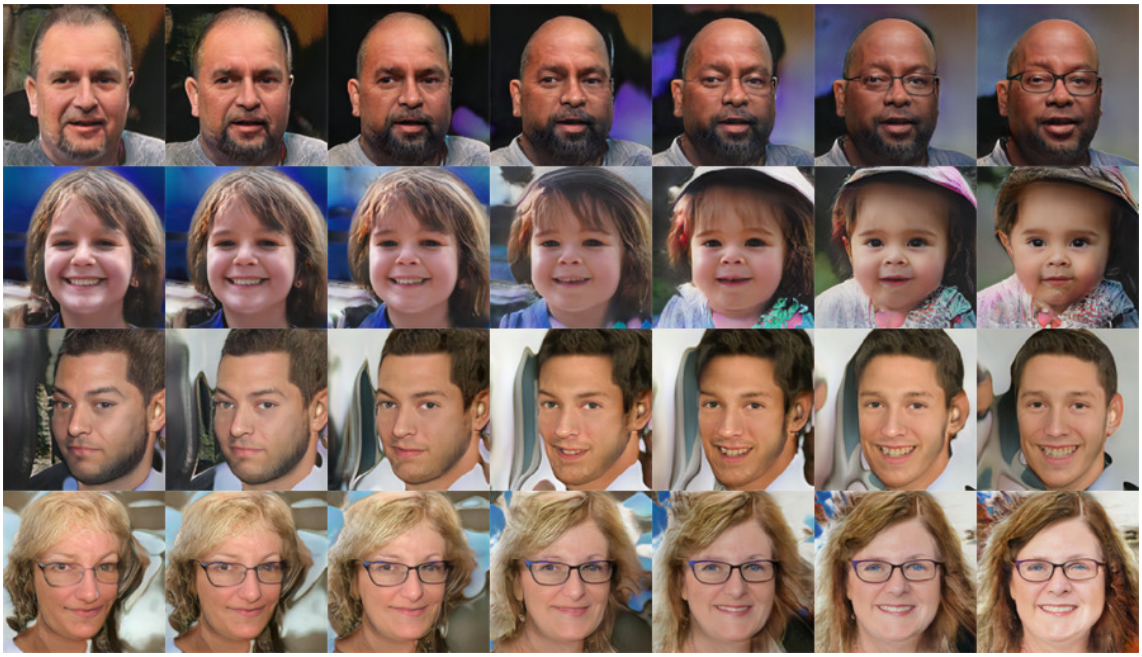
vidět na následujícím obrázku 6.7. To, že se jedná o správně natrénovanou sít schopnou správně generalizovat data poznáme z těchto vzorků tak, že přechod z jednoho do druhého vzorku je plynulý a nenachází se v něm žádné prudké přechody. Vliv průchodu latentním prostorem lze sledovat na jednotlivých rysech, doplňcích i dalších vlastnostech, jakými mohou být třeba barva pozadí a pozice hlavy.

## 6.4 Zhodnocení výsledků

Podařilo se mi úspěšně natrénovat modely podmíněných generativních neuronových sítí založených na třech různých architekturách - DCGAN, WGAN-GP a ProGAN. Všechny modely byly trénovány na vzorcích z datové sady FFHQ, které jsou v rozlišení 128x128 pixelů, a v tomto samém rozlišení lidské obličeje také modely generují. Model architektury DCGAN se potýká s nedostatky, které znemožňují úspěšnou konvergenci, a do značné míry omezují variabilitu vzorků a stejně tak i jejich kvalitu. Zbylé dva modely se podařilo natrénovat úspěšně, a jejich výstupem jsou do jisté míry přesvědčivé vzorky lidských obličejů, kde model architektury ProGAN vede, co se týče celkové detailnosti vzorků. Všechny tři modely ovšem zdárně produkují vzorky, které odpovídají požadavkům, které byly neuronové sítí předány na vstup prostřednictvím třídních značek.

Výkonnost modelu ProGAN byla ověřena dvojicí experimentů ověřující správné fungování podmíněností vzorků a poté další dvojicí experimentů zaměřenou na kontrolu správného natrénování sítě. Zde bylo zjištěno, že síti generátoru se podařilo naučit úspěšně reprezentovat distribuci dat, namísto toho, aby se snažila pouze napodobit některé vzorky z datové sady.

Obrázek 6.6: Experiment s latentním prostorem modelu ProGAN - vektorová aritmetika s hodnotami šumu, kde výsledek na pravé straně odpovídá aritmetickým operacím provedenými s hodnotami šumu vzorků zobrazených na levé straně



Obrázek 6.7: Experiment s latentním prostorem modelu ProGAN - průchod latentním prostorem, kde je vygenerováno několik vzorků s hodnotami šumu rovnoměrně získaných z úseček spojující body, které byly použity jako vstupní hodnoty šumu pro nejlevější a nejpravější vzorky na jednotlivých řádcích

# Kapitola 7

# Závěr

Cílem této práce bylo navržení a natrénování několika variant modelů podmíněných generativních neuronových sítí, které budou schopny generovat lidské obličeje s vybranými rysy. V počáteční části práce byly shrnuty základní informace z oblasti oblasti neuronových sítí, jejichž znalost je fundamentální pro pochopení celé práce. V následující kapitole se přešlo ke generativním neuronovým sítím, které jsou v dnešní době jedním z nejmodernějších typů neuronových sítí. Po představení principu, architektur a konkrétních využití této technologie se zbytek práce již zabývá návrhem a implementací trénovacího prostředí v jazyce Python s využitím knihovny Keras. V závěru práce je věnována pozornost porovnání vygenerovaných vzorků a evaluaci implementovaných modelů.

Podařilo se natrénovat trojici modelů, u kterých je při generování osob možnost výběru pohlaví a věkové skupiny. Trénování probíhalo na barevných vzorcích v rozlišení 128x128 pixelů z datové sady Flickr-Faces-HQ. Každý implementovaný model se liší použitou architekturou. První model, který je založený na architektuře DCGAN, je v dnešní době již zastaralým modelem, kde navzdory jeho složitému trénování byly získány výsledky splňující stanovené cíle práce. Větších úspěchů bylo dosaženo s modely založenými na architekturách WGAN-GP a ProGAN, kde finální modely jsou schopné vygenerovat tváře, které jsou v některých případech na první pohled až nerozpoznatelné od skutečných fotografií osob.

Nabízí se hned několik směrů, kterými by se mohl odvíjet budoucí vývoj této práce. Ačkoliv jde o zastaralý typ sítí, jednou možnou cestou by byla stabilizace trénování modelu architektury DCGAN, aby bylo dosaženo úspěšné konvergence. Mnohem zajímavějším krokem by bylo škálování modelů na práci s vyšším rozlišením vzorků, vzhledem k tomu, že nyní sítě pracují se vzorky v rozlišení 128x128 pixelů, přičemž dostupné jsou vzorky v rozlišení až 1024x1024 pixelů. Pochopitelně poslední možnou cestou by bylo úplné opuštění modelů stávajících architektur a s nově získanými znalosti z oblasti hlubokého učení a generativních neuronových sítí se věnovat implementaci modelů modernějších architektur.

# Literatura


[1] Algorithmia Inc.. *Introduction to Optimizers* [online]. Květen 2018 [cit. 2020-05-14]. Dostupné z: https://algorithmia.com/blog/introduction-to-optimizers.

[2] Arjovsky, M., Chintala, S. a Bottou, L. Wasserstein GAN. *ArXiv e-prints*. leden 2017, s. arXiv:1701.07875.

[3] Burnes, A. *NVIDIA DLSS 2.0: A Big Leap In AI Rendering* [online]. Březen 2020 [cit. 2020-04-23]. Dostupné z: https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/.

[4] Chollet, F. a Pecinovskỳ, R. *Deep learning v jazyku Python: knihovny Keras, Tensorflow*. Grada Publishing, 2019. Knihovna programátora. ISBN 9788024731001. Dostupné z: https://books.google.cz/books?id=rLauDwAAQBAJ.

[5] Doshi, S. *Various Optimization Algorithms For Training Neural Network* [online]. Leden 2019 [cit. 2020-05-14]. Dostupné z: https://medium.com/@sdoshi579/optimizers-for-training-neural-network-59450d71caf6.

[6] Godoy, D. *Understanding binary cross-entropy / log loss: a visual explanation* [online]. Listopad 2018 [cit. 2020-05-13]. Dostupné z: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[7] Goodfellow, I. NIPS 2016 Tutorial: Generative Adversarial Networks. *ArXiv e-prints*. prosinec 2016, s. arXiv:1701.00160.

[8] 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. Dostupné z: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[9] Gui, J., Sun, Z., Wen, Y., Tao, D. a Ye, J. A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications. *ArXiv e-prints*. leden 2020, s. arXiv:2001.06937.

[10] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. a Courville, A. Improved Training of Wasserstein GANs. *ArXiv e-prints*. březen 2017, s. arXiv:1704.00028.

[11] Huang, X. a Belongie, S. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization. *ArXiv e-prints*. březen 2017, s. arXiv:1703.06868.

[12] Hwang, J.-J., Azernikov, S., Efros, A. A. a Yu, S. X. Learning Beyond Human Expertise with Generative Models for Dental Restorations. *ArXiv e-prints*. březen 2018, s. arXiv:1804.00064.

[13] Ioffe, S. a Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv e-prints*. únor 2015, s. arXiv:1502.03167.

[14] Karras, T., Aila, T., Laine, S. a Lehtinen, J. Progressive Growing of GANs for Improved Quality, Stability, and Variation. *ArXiv e-prints*. říjen 2017, s. arXiv:1710.10196.

[15] Karras, T., Laine, S. a Aila, T. A Style-Based Generator Architecture for Generative Adversarial Networks. *ArXiv e-prints*. prosinec 2018, s. arXiv:1812.04948.

[16] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J. et al. Analyzing and Improving the Image Quality of StyleGAN. *ArXiv e-prints*. prosinec 2019, s. arXiv:1912.04958.

[17] Khalifa, N. E. M., Taha, M. H. N., Hassanien, A. E. a Elghamrawy, S. Detection of Coronavirus (COVID-19) Associated Pneumonia based on Generative Adversarial Networks and a Fine-Tuned Deep Transfer Learning Model using Chest X-ray Dataset. *ArXiv e-prints*. duben 2020, s. arXiv:2004.01184.

[18] Kingma, D. P. a Ba, J. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*. prosinec 2014, s. arXiv:1412.6980.

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

[20] Mescheder, L., Geiger, A. a Nowozin, S. Which Training Methods for GANs do actually Converge? *ArXiv e-prints*. leden 2018, s. arXiv:1801.04406.

[21] Mirza, M. a Osindero, S. Conditional Generative Adversarial Nets. *ArXiv e-prints*. listopad 2014, s. arXiv:1411.1784.

[22] Moore, G. E. *Cramming more components onto integrated circuits* [online]. 1965 [cit. 2020-04-20]. Dostupné z: https://newsroom.intel.com/wp-content/uploads/sites/11/2018/05/moores-law-electronics.pdf.

[23] Pilát, M. *Neuronové sítě - úvod* [online]. [cit. 2020-05-12]. Dostupné z: https://martinpilat.com/cs/prirodou-inspirovane-algoritmy/neuronove-site-uvod.

[24] Radford, A., Metz, L. a Chintala, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *ArXiv e-prints*. listopad 2015, s. arXiv:1511.06434.

[25] Rothe, R., Timofte, R. a Gool, L. V. DEX: Deep EXpectation of apparent age from a single image. In: *IEEE International Conference on Computer Vision Workshops (ICCVW)*. December 2015.

[26] Rothe, R., Timofte, R. a Gool, L. V. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision*. Springer. 2018, sv. 126, 2-4, s. 144–157.

[27] Saha, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* [online]. Prosinec 2018 [cit. 2020-05-14]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[28] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. et al. Improved Techniques for Training GANs. In: Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I. a Garnett, R., ed. *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016, s. 2234–2242. Dostupné z: http://papers.nips.cc/paper/6125-improved-techniques-for-training-gans.pdf.

[29] Schmidhuber, J. *Making the World Differentiable: On Using Self-Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments*. 1990.

[30] Shafkat, I. *Intuitively Understanding Convolutions for Deep Learning* [online]. červen 2018 [cit. 2020-05-15]. Dostupné z: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1.

[31] Sharma, S. *Activation Functions in Neural Networks* [online]. Září 2017 [cit. 2020-05-12]. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[32] Stewart, M. *Neural Network Optimization* [online]. červen 2019 [cit. 2020-05-13]. Dostupné z: https://towardsdatascience.com/neural-network-optimization-7ca72d4db3e0.

[33] Truong, P. *Loss functions: Why, what, where or when?* [online]. Březen 2019 [cit. 2020-05-13]. Dostupné z: https://medium.com/@phuctrt/loss-functions-why-what-where-or-when-189815343d3f.

[34] Verdoliva, L. Media Forensics and DeepFakes: an overview. *ArXiv e-prints*. leden 2020, s. arXiv:2001.06564.

[35] Wang, P. *This Person Does Not Exist* [online]. Dostupné z: https://thispersondoesnotexist.com/.

[36] Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Tao, A., Kautz, J. et al. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. *ArXiv e-prints*. listopad 2017, s. arXiv:1711.11585.

[37] Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y. et al. ESRGAN: Enhanced super-resolution generative adversarial networks. In: *The European Conference on Computer Vision Workshops (ECCVW)*. September 2018.

[38] Zbořil, F. a Zbořil, F. V. *Základy umělé inteligence*. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php?file=%2Fcourse%2FIZU-IT%2Flectures%2FIZU%2F1819izu_9.pdf.

[39] Zhu, J.-Y., Park, T., Isola, P. a Efros, A. A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *ArXiv e-prints*. březen 2017, s. arXiv:1703.10593.