VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

REKONSTRUKCE SNÍMKŮ OBLIČEJE S VYUŽITÍM NEURONOVÝCH SÍTÍ

RECONSTRUCTION OF FACIAL IMAGES USING NEURAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR ZUBALÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ GOLDMANN
SUPERVISOR

BRNO 2019

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

21391

Student: **Zubalík Petr**
Program: Informační technologie
Název: **Rekonstrukce snímku obličeje s využitím neuronových sítí**
Reconstruction of Facial Images Using Neural Networks
Kategorie: Umělá inteligence
Zadání:

1. Seznamte se s problematikou městských kamerových systému a sumarizujte jejich vlastnosti. Zjistěte, jaké jsou z hlediska rozlišení rozdíly mezi analogovými a digitálními systémy.

2. Seznamte se základy neuronových sítí. Vyhledejte řešení, která se používají pro rekonstrukci obrazu.

3. Navrhněte s využitím frameworku Tensorflow aplikaci, která dokáže rekonstruovat obličej osoby ve snímků s nízkým rozlišením. Nízkým rozlišením rozumíme obraz tváře, kde jsou významně rozmazané základní části obličeje (tvář, nos, ústa).

4. Navržený algoritmus implementujte v libovolném programovacím jazyce. Pro ovládání vytvořte jednoduché uživatelské rozhraní.

5. Vytvořte dataset se snímky obličeje ve vysokém rozlišení. Dále dataset rozšiřte o snímky se sníženým rozlišením, které budou sloužit pro otestování algoritmu. Porovnejte rekonstruované snímky s originály. Při porovnání určete odchylku vybraných zrekonstruovaných parciálních části obličeje od originálu.

Literatura:

- GURNEY, Kevin. An introduction to neural networks. London: UCL Press, 1997. ISBN 1857286731.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Goldmann Tomáš, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Hlavním cílem této bakalářské práce je navrhnout a implementovat model neuronové sítě, který bude schopen rekonstruovat snímky obličeje v tak nízkém rozlišení, že na nich budou rozmazány základní části obličeje. Zadaný problém rekonstrukce obličejů je vyřešen pomocí modelů založených na konvolučních neuronových sítích. První model je postaven na architektuře ResNet, kdežto druhý staví na principech generativních kompetitivních sítí. Navržené modely jsou implementovány v programovacím jazyce Python za pomoci aplikačního programového rozhraní frameworku TensorFlow. Součástí práce je i aplikace s velmi jednoduchým grafickým uživatelským rozhraním pro snadné používání modelu. V poslední části práce je rozebráno několik experimentů provedených pro otestování úspěšnosti navržených modelů.

Abstract

The main purpose of this bachelor's thesis is to propose and implement a model, using neural networks, that will be able to reconstruct low-resolution facial images with blurry parts of the face. The task of super-resolution of facial images is solved by two models based on convolutional neural networks. The first model is built upon the architecture of ResNet whereas the other model makes use of the principles of generative adversarial networks. The proposed models are implemented in the Python programming language with the use of application programming interface of the TensorFlow framework. Moreover, as a part of this work, an application with a simple grafical user interface was created. This application makes it easy to use the implemented models. Several experiments are analyzed in the last chapter of this thesis to evaluate the performance of the models.

Klíčová slova

kamerový systém, kamera, rekonstrukce obrazu, umělá inteligence, konvoluční neuronové sítě, generativní kompetitivní sítě, ResNet, TensorFlow

Keywords

CCTV, camera, image reconstruction, artificial intelligence, convolutional neural networks, generative adversarial networks, ResNet, TensorFlow

Citace

ZUBALÍK, Petr. *Rekonstrukce snímků obličeje s využitím neuronových sítí.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Goldmann

Rekonstrukce snímků obličeje s využitím neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Goldmanna. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Petr Zubalík
14. května 2019

Poděkování

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

Obsah

Úvod **2**

1 Městské kamerové systémy **3**
1.1 Kamery 3
1.2 Kamerové systémy 6

2 Umělé neuronové sítě pro rekonstrukci obrazu **9**
2.1 Umělé neuronové sítě 10
2.2 Učení neuronových sítí 17
2.3 Modely neuronových sítí pro rekonstrukci obrazu 22
2.4 Generativní kompetitivní neuronové sítě 23

3 Návrh modelů pro rekonstrukci obrazu **26**
3.1 Model založený na architektuře ResNet 26
3.2 Model založený na architektuře GAN 29

4 Implementace **32**
4.1 Tensorflow 32
4.2 Předzpracování dat 33
4.3 Implementace ResNet modelu 33
4.4 Implementace GAN modelu 35
4.5 Implementace grafického uživatelského rozhraní 36

5 Experimenty **38**
5.1 Dataset 38
5.2 Výpočet odchylky originálních a rekonstruovaných snímků 39
5.3 Prostředí experimentů 40
5.4 Experimenty s ResNet modelem 40
5.5 Experimenty s GAN modelem 46
5.6 Zhodnocení výsledků 47

Závěr **50**

Literatura **51**

A Výsledky modelů **55**

B Obsah datového média **63**

Úvod

Kamerové systémy se v dnešní době hojně využívají pro zabezpečení objektů, veřejných prostranství a v boji s kriminalitou obecně. Přestože jsou na trhu dostupné velmi kvalitní kamery snímající obraz s rozlišením v řádech jednotek až desítek megapixelů, velké množství kamerových systémů nepořizuje záznam této kvality. Jedním z důvodů může být vyšší cena kvalitních kamer při pořizování systému. Dalším důvodem je i fakt, že hodně kamerových systémů bylo instalováno v době, kdy nebyly tyto technologie vůbec dostupné. V praxi se tedy může stát, že kamera sice zachytí pachatele při činu, ale záznam z ní je tak nekvalitní – v tak nízkém rozlišení – že z něj nejde rozpoznat obličej kriminálníka. V tomto okamžiku přichází čas pro aplikaci rekonstrukce obrazu, přesněji řečeno zvýšení rozlišení daného snímku rozmazaného obličeje.

Rekonstrukce obrazu je oblast spadající do oboru zpracování obrazu, která má mnoholetou historii. První výsledky pokusů o rekonstrukci postrádaly ostré přechody barev a kvalitní rekonstrukci textur. To se ale změnilo s příchodem hlubokého učení a konvolučních neuronových sítí a jejich aplikace na rekonstrukci obrazu. Většina těchto aplikací se však soustředí na rekonstrukci snímků, které už obsahují relativně dostatečné množství informací pro věrohodnou rekonstrukci. Stejně je to i u aplikací, které se zaměřují konkrétně na rekonstrukci obličejů. Velká většina z nich rekonstruuje snímky, které už obsahují dostatek informace pro rozpoznání identity člověka na obrázku. Tato bakalářská práce má za cíl rekonstruovat snímky obličejů, které jsou v tak nízkém rozlišení, že na nich nejde identifikovat osoba na snímku.

V úvodu práce se v první kapitole věnuji problematice městských kamerových systémů – hlavně jejich technologickému vývoji. Dále jsem se v první kapitole soustředil na kvalitu obrazu z hlediska rozlišení, který je snímán kamerami takových systémů. Pro návrh a implementaci výkonného modelu pro rekonstrukci obrazu bylo potřeba nastudovat si principy neuronových sítí a hlubokého učení. Na toto téma se zaměřuje druhá kapitola, ve které jsou navíc rozebrány modely používající se pro rekonstrukci obrazu, ať už využívají hluboké učení nebo ne. Kapitoly 1 a 2 jsou čistě teoretické.

Ve třetí kapitole popisuji návrh dvou modelů. Oba modely jsou založeny na konvolučních neuronových sítích. Rozdílem mezi nimi je ale jejich architektura a způsob učení. První model je založen na architektuře ResNet a druhý využívá principů generativních kompetitivních sítí. V následující kapitole Implementace je popsána implementace obou modelů a dalších programů, které byly potřebné pro úspěšný trénink rekonstrukce. Pro implementaci všech částí jsem použil programovací jazyk Python a pro implementaci modelů framework TensorFlow. V páté kapitole Experimenty rozebírám výsledky experimentů, pomocí kterých jsem modely testoval. Výsledky jsou reprezentovány formou obrázků s rekonstruovanými snímky. V rámci jednoho z experimentů je popsána úprava architektury jednoho z navržených modelů. Dále je v této kapitole obsaženo zhodnocení implementovaných modelů a porovnání jejich výsledků.

Kapitola 1

Městské kamerové systémy

Městské kamerové systémy jsou speciálním typem tzv. *CCTV* systémů používaných pro sledování veřejných prostranství – například náměstí, památek a dalších. Název CCTV vychází z anglického pojmu *closed-circuit television*, což znamená uzavřený přenos televizního signálu pro vybranou skupinu zařízení.

V poslední době se využití CCTV kamerových systémů významně rozšířilo. Velké uplatnění nalezly v průmyslu, kde slouží pro dohled nad technologickými procesy a pro zabezpečení technologických budov. Další využití je v dopravě, kde se používají pro monitorování dopravní situace v dané části komunikace. Z hlediska mé práce je nejdůležitější využití CCTV systémů pro sledování veřejných prostor, jakými jsou například nákupní centra, veřejnoprávní budovy, nemocnice, centra měst a obecně místa s vysokým počtem přítomných lidí.

CCTV systémy se v praxi vyskytují asi od poloviny minulého století. V té době se však používaly hlavně pro přenos boxerských a jiných sportovních zápasů [1]. Tyto přenosy byly poté přehrávány veřejnosti například v divadlech a kinech. Toto využití přetrvávalo asi až do 90. let, kdy bylo nahrazeno kabelovými televizemi [1].

První použití pro boj s kriminalitou je datováno do roku 1968 v New Yorku [49]. Od té doby se použití tzv. *CCTV surveillance*, neboli sledování pomocí CCTV rapidně rozrostlo, ale také prošlo velkou technologickou obměnou.

V této kapitole postupně uvedu jaké typy kamer se v kamerových systémech používají a jakou kvalitu má jimi pořizovaný záznam, jelikož se tato práce zaměřuje na zvýšení rozlišení snímků obličeje, které jsou často také zachyceny takovými kamerami. Dále se zaměřím na existující typy kamerových systému a jejich technologický vývoj.

1.1 Kamery

Na kvalitu snímaného obrazu mají největší vliv použité kamery. V této sekci bude rozebráno, jaké druhy kamer se používají, ať už z hlediska zpracování obrazu, nebo z hlediska signálu, který z nich vychází, a jaké kamery jsou nejlepší co se kvality obrazu a rozlišení týče. Dále pak jaké jsou nejdůležitější součástky v těchto kamerách a jak vůbec je přijímaný obraz zpracován.

Zachycení obrazu

Obraz, který zachytí kamera, není nic jiného než určité množství světla dopadajícího na senzor, který převádí světelnou energii na elektrickou energii. Princip tohoto senzoru spočívá

v tom, že zachycuje fotony obsažené ve světelném záření na tzv. *pixely*. Jednotlivé pixely shromažďují elektrický náboj úměrný počtu dopadajících fotonů na daný pixel. Celý převod světelné energie na elektrickou je součástí fotoelektrického jevu, kdy narážející fotony uvolňují elektrony v pixelu, které se stávají nosiči elektrického proudu. Akumulovaný náboj je poté převeden na elektrické napětí a dále zpracováván. V praxi se používají téměř výhradně 2 typy senzorů, které se liší konstrukcí pixelů a zpracováním elektrického náboje [34].

Senzor CCD

Tento typ senzoru, nazývaný *Charge-coupled device* (CCD), používá jako pixely kondenzátory technologie MOS (*Metal Oxide Semiconductor*) [34], které při ozáření světlem začnou akumulovat elektrický náboj. Jakmile doba ozáření (expozice) skončí, CCD senzor začne přenášet elektrický náboj jednotlivých pixelů ven ze senzoru sekvenčně přes součástky, které převedou náboj na elektrické napětí, se kterým se posléze pracuje. Například u digitálních kamer je v analogově-digitálním převodníku převedeno do digitální podoby. Sekvenční přenášení náboje způsobuje pomalejší zpracování a nutnost delší doby expozice.

Senzor CMOS

Senzory typu CMOS (*Complementary Metal Oxide Semiconductor*) mají velkou výhodu v rychlosti zpracování a převodu světelné energie na elektrickou. Převod akumulovaného elektrického náboje na elektrické napětí se koná paralelně přímo v konstrukci každého z pixelů, což výrazně urychluje zpracování a umožňuje kratší dobu expozice. Implementace převodníku elektrického náboje na napětí přímo na senzoru má za následek menší flexibilitu a kvalitu pořizovaných snímku, hlavně za slabého světla.

Díky malé velikosti a nízké spotřebě energie našly senzory CMOS využití ve velkém množství praktických aplikaci, jako například webové kamery nebo chytré telefony. Kvalita jimi snímaného obrazu není tak dobrá, jako u senzorů CCD, ale na druhou stranu je toto vyváženo již zmíněnou kompaktností a nízkou spotřebou. Další nevýhodou této technologie je to, že velkou část plochy senzoru zabírají elektronické obvody, a tím snižují využitelnost plochy senzoru pro snímání světelného signálu. Další využití CMOS senzorů je aplikováno v oblastech, kde není vyžadována špičková kvalita snímaného obrazu – například videokonference, čtečky čárových kódů nebo třeba biometrie [34].

Na druhou stranu, CCD senzory našly využití tam, kde je kvalita obrazu požadavkem číslo jedna. Do této kategorie patří i kamerové systémy, kam kamery s CCD senzory vstoupily už v 80. letech minulého století. Jako další příklady využití uveďme fotografování, broadcastové televizní přenosy nebo aplikace v medicíně [34]. Výběr správného senzoru má největší vliv na kvalitu pořizovaných snímků ze všech součástek kamery.

Další důležitou součástí kamery pro kvalitní zachycení obrazu je objektiv. Jedná se o optickou část kamery, přes kterou do ní vchází světlo. Objektiv je poskládán ze série čoček, které mají na starosti ostření obrazu a správné usměrnění světelných paprsků na obrazový senzor kamery. Pro zaostření obrazu se používá automatický systém. Manuální zaostřování je nepraktické, protože by bylo zapotřebí operátora, který by musel nastavovat každou kameru v kamerovém systému. V kamerových systémech se většinou používají objektivy s pevným zoomem, což znamená, že nemají nastavitelnou ohniskovou vzdálenost a možnost optického přiblížení obrazu. Další součástí objektivu je tzv. clona, která udává množství světla, pokračujícího dále do objektivu.

Analogové kamery

Snímání obrazu analogovými a digitálními kamerami funguje na stejném principu pomocí obrazových senzorů popsaných výše. Odlišnost těchto typů tkví ve formátu nasnímaného obrazu posílaného do kamerového systému. I v nejnovějších systémech s analogovými kamerami musí být kamera připojena koaxiálním kabelem k nahrávacímu zařízení – například DVR nebo video-enkodéru. Toto zařízení následně signál digitalizuje a ukládá na pevný disk nebo jiný typ úložiště. Fakt, že kamera předává analogový signál, má za následek horší výslednou kvalitu obrazu, protože není schopna přenést dostatek informací, kvůli nedostatečné kapacitě analogového rozhraní [35].

Pro zachování kompatibility s prodávanými analogovými nástroji pro zobrazení a uložení tohoto signálu, se pro přenos v Evropě používá oblíbený analogový standard PAL (například v Severní Americe se používá standard NTSC) [35]. Nejdůležitějším údajem, který tento standard předepisuje je počet horizontálních linek, které tvoří obraz – vertikální rozlišení. PAL udává, že tento počet by měl být 625. Ne všechny linky nesou obrazovou informaci, takže viditelných linek je nakonec jenom 575. Tento standard říká, že standardní rozlišení analogově přenášeného obrazu bude maximálně 720×575 pixelů. Toto ale není jediné omezení kvality přenášeného obrazu. Další překážkou je, že signál může být redukován i šířkou pásma, po kterém se přenáší rádiový signál. Stejná technologie je použita u kamerových systémů s analogovými kamerami. Jeden MHz přenosového pásma radiového signálu může přenést 80 linek analogového video signálu. Pro přenesení 540 linek signálu horizontálního rozlišení, což odpovídá 720 pixelům, bychom potřebovali 6,75 MHz přenosového pásma. PAL ve střední Evropě používá šířku pásma 5 MHz a z toho plyne, že nezvládne přenést celých 540 linek [7]. Pásma s šířkou 5 MHz nám umožní přenášet pouze 400 linek. Pokud máme poměr stran 4:3, můžeme převést 400 linek na 533 pixelů. To ale ještě neznamená, že naše výsledné rozlišení bude 533×575 pixelů. PAL, aby ušetřil co nejvíce pásma při přenosu dat, používá kompresní metodu zvanou prokládání (angl. *interlace*), která využívá toho, že PAL aktualizuje obrazová data každou sekundu padesátkrát, neposílá ale ve všech případech celých 533×575 pixelů. S každým datovým „balíčkem" přijde půlka pixelů, které se zrovna nezobrazí, počká se, až přijde druhá půlka a následně se obraz na obrazovce aktualizuje [35]. Na starých analogových televizích se zobrazily obě půlky obrazu zvlášť, protože lidské oko stejně nestihlo zaregistrovat, že se vykreslují odděleně. Na nových digitálních televizích a monitorech se obě půlky obrazu spojí v paměti a vykreslují se jako jeden celek. V každém případě je nutno uvědomovat si, že se obraz posílá rozpůlený na dvě vertikální půlky, a to způsobuje problémy, pokud kamera snímá rychle se pohybující objekty, protože bude na každé půlce obrazu na jiné pozici relativně ke kameře. V tomto případě tedy nemůžeme zobrazit spojení 2 obrazů a zobrazuje se jenom jedna půlka – rozlišení tedy je pouze poloviční.

Maximální rozlišení obrazu z analogových kamer při použití standardu PAL je 533×575 pixelů, což je asi 0,3 megapixelu [35]. Ve srovnání s digitálními kamerami je to mizerný poměr výkonnosti.

Dalším rozhraním, které v minulosti omezovalo rozlišení obrazu z analogové kamery, je *VGA (Video Graphics Array)*. Tento grafický standard dokázal přenášet video o rozlišení 640×480 pixelů v černobílém nebo barevném (16 barev) formátu. V dnešní době však standard VGA zvládá i rozlišení 1920×1080 pixelů, a tak se dokáže rovnat obrazu z *Full HD* kamer [9].

Digitální kamery

Po zachycení obrazu digitální kamerou dojde přímo v kameře k převodu analogového signálu na digitální. Každá digitální kamera je připojena pomocí datového kabelu do sítě a je ji přidělena IP adresa, pomocí níž ji můžeme adresovat. To přidává možnost připojit se ke kameře z jakéhokoliv jiného zařízení v sítí. Obraz zachycený kamerou je přenášen na připojené zařízení, kde se přehrává v reálném čase anebo ukládá na pevný disk. Datový tok mezi kamerou a jiným zařízením je realizován pomocí dvou napěťových hodnot odpovídajícím logické 0 a logické 1, pomocí kterých je obraz zakódován a posílán většinou sekvenčně.

Fakt, že je kamera připojena pomocí síťového kabelu, a že posílá už zdigitalizovaná data, dovoluje, aby byl posílán obraz o daleko větším rozlišení, než u analogových kamer, protože zde nemáme tak velká omezení. Po sítí je možné posílat video o jakékoliv velikosti, ať už co se samotného rozlišení týče, tak co se týče náročnosti na paměť [35].

IP kamery, jak se říká digitálním kamerám připojeným k internetu, jsou schopny snímat s rozlišením v řádech jednotek megapixelů. To je například oproti analogovým kamerám, využívajícím standard PAL, které dokázaly získat rozlišení jednotek desetin megapixelu, velký skok. Profesionální kamery jsou schopny dávat video o rozlišení až 60 megapixelů, ale tyto kamery nemají praktické využití v kamerových systémech, protože jsou příliš drahé, a tak vysoké rozlišení není pro většinu kamerových systémů nutné.

Velkou výhodou připojení datového ethernetového kabelu do kamery je jeho multifunkčnost. S jeho pomocí můžeme přijímat data z kamery, dávat kameře příkazy a napájet kameru elektrickou energií. Napájení pomocí ethernetového kabelu se říká *Power over ethernet* [35].

Jak již bylo zmíněno dříve, digitální kamery snímají obraz s rozlišením v řádech jednotek megapixelů. Na trhu s digitálními kamerami převládá v současné době několik typů nejpoužívanějších rozlišení obrazu. Na nejnižší hranici se pohybují kamery s rozlišením 1280×720 pixelů (*HD, 720p, 1 MPx*), které ale nedosahují ani kvalit snímání obrazu současných analogových kamer připojeným na rozhraní VGA [19]. HD rozlišení však poskytuje flexibilitu v možnosti zachycení obrazu s různými poměry stran (angl. *aspect ratio*). Příkladem může být poměr 4:3 nebo širokoúhlý poměr 16:9. Větší kvalitu obrazu přinášejí kamery s rozlišením 1920×1080 pixelů (*Full HD, 1080p, 2MPx*), které se rovnají možnostem nejlepších analogových kamer. Použití kamer s rozlišením Full HD výrazně stouplo na úkor HD kamer od roku 2014 [2]. V poslední době se začínají hojně používat i kamery s vyšším rozlišením než je Full HD. Jedná se o rozlišení 3 – 6 MPx [2]. Díky klesající ceně se začínají používat i kamery s rozlišením 3840×2160 pixelů (*4K, 8MPx*), které dokáží snímat vysoce kvalitní obraz, za nějž ale pořád zaplatíme daleko víc, než za kamery s nižším rozlišením [2]. Rozlišení však není jediným ukazatelem kvality obrazu, dále se musíme brát v potaz například na typ použitých čoček nebo počet snímaných rámců za sekundu. V dnešních kamerách hraje důležitou roli i software, který zpracovává zachycený obraz.

1.2 Kamerové systémy

Kamerové systémy umožňují vzdálené sledování prostor, které vyžadují určitý stupeň zabezpečení. Toto sledování je prováděno pomocí kamer, z nichž získaný obrazový záznam je posílán na vzdálené zařízení, kde je buď uložen nebo zobrazen v reálném čase. Kamerové systémy prošly v průběhu doby několika technologickými obměnami, kdy se vyvinuly z čistě analogového systému na systém plně digitální. Digitální systém je připojený k lokální síti a záznam ze svých kamer šíří pomocí technologií TCP/IP přímo na zařízení, kde se může

sledovat v reálném čase nebo uložit. Tento vývoj bude rozebrán v následujících částech této sekce.

Analogové kamerové systémy

První kamerové systémy se skládaly čistě z analogových součástek. Jejich nejdůležitějším prvkem jsou analogové kamery, které pomocí koaxiálních kabelů přenášejí snímaný obraz na druh rekordéru, který pro uložení záznamu využívá video kazet – tzv. *videocassette recorder* (*VCR*). Tento druh kamerových systémů se začal používat jako první. Jeho použití v rozsáhlých městských kamerových systémech nebylo možné kvůli nutnosti přímého spojení kamer s VCR pomocí kabelů, jež měly omezenou délku. Další nevýhodou byla nemožnost komprese záznamu. Pokud byl záznam pořizován s maximálním počtem rámců za sekundu, vydržela jedna video kazeta nejvýše asi 8 hodin [41], a poté musela být manuálně vyměněna za novou.

Později byly vyvinuty multiplexory, které byly umístěny mezi kamery a VCR a spojovaly tyto součástky. Multiplexor dokázal zpracovat záznam z více kamer a zobrazit je na monitoru vedle sebe. To vše ale za cenu nízkého rozlišení [41].

Částečně digitální kamerové systémy

Prvním velkým technologickým skokem v oblasti kamerových systémů bylo vyvinutí součástky zvané *digital video recorder* (*DVR*), využívající pevné disky pro ukládání dat. Nahrazením VCR došlo k použití první digitální součástky v jinak stále zcela analogovém systému.

DVR umožňoval uložit signál z analogových kamer v digitalizované podobě, který přicházel až z 32 snímacích zařízení [41]. Nahrazoval tedy VCR i multiplexor, umožňující přijímat signál z více kamer. Digitalizovaný záznam s sebou přináší velké množství výhod, jako například nepotřebnost manuálně vyměňovat kazety, konstantní kvalita nahrávání a nejdůležitější z nich, možnost posílat digitalizované video na dlouhé vzdálenosti pomocí sítě. To bylo nejdříve umožněno pomocí telefonního modemu připojeného na sériový port DVR. Toto řešení se ale ukázalo jako velmi nepraktické, kvůli velmi malé šířce přenosového pásma telefonního modemu. V dalším kroku vývoje byl do DVR přidán ethernetový port. Pomocí tohoto portu bylo možné připojení do lokální internetové sítě, z čehož plynou hlavní výhody:

- pozorování a přehrávání záznamů na vzdáleném PC,
- možnost spravovat systém vzdáleně.

I když bylo DVR daleko účinnější než VCR, nebylo ideální komponentou do velkých kamerových systémů. Tyto rekordéry byly extrémně zatěžovány, protože měly na starosti všechny úkoly daného systému. Musely digitalizovat video ze všech kamer, kterých mohlo být až 32 na jeden rekordér [41], dále toto video uložit nebo odeslat na vzdálené úložiště.

Digitální kamerové systémy

I přesto, že analogové kamery napojené na DVR a síťový směrovač dokáží přenášet snímaný obraz přes ethernet na velké vzdálenosti, oproti systémům s IP kamerami výrazně zaostávají, a to jak z hlediska kvality záznamu, tak z hlediska škálovatelnosti systému.

Tyto systémy využívají kamery s IP připojením, které zachycený záznam okamžitě digitalizují a odesílají na zařízení určené k uložení nebo monitorování záznamu. Jejich hlavní výhodou je ethernetové připojení, které umožňuje připojení pouze jednoho kabelu do kamery, přes který se odesílají data z kamery, přijímají data z řídicích zařízení, například příkazy pro otočení kamery nebo zoom obrazu. Další využití ethernetového kabelu je přívod elektrické energie do kamer – tzv. *Power over Ethernet* [41].

Kapitola 2

Umělé neuronové sítě pro rekonstrukci obrazu

Úvod této kapitoly vychází z [23]. Umělé neuronové sítě jsou výpočetním modelem, který je analogií k biologickým neuronovým sítím, ze kterých se skládá mozek zvířat a lidí, a jejichž funkcionalitu se snaží napodobit. Lidský mozek se skládá z řádově bilionů výpočetních jednotek nazvaných *neurony*. Každý neuron dokáže zpracovávat vstupy v podobě elektrických signálů, které jsou generovány smyslovými receptory, a které přenáší informace do mozku z okolí člověka. Výsledek, vypočtený z těchto vstupních signálů, posílá do dalších neuronů v síti, na které je přímo napojen pomocí tzv. *synapsí*. Právě díky vytváření nových a přizpůsobením existujících synaptických spojení je umožněno učení lidského mozku.

Na stejném principu pracují i umělé neuronové sítě, které se skládají z umělých neuronů, jenž jsou zjednodušenou analogií biologického neuronu. Tyto umělé neurony jsou mezi sebou také propojeny, aby si mohli předávat informace.

Dopředné neuronové sítě, též často nazývané jako *vícevrstvé perceptrony*, jsou základem pro celé pole modelů hlubokého učení. Jsou nazývány dopředné, protože informace jimi protéká postupně od vstupní vrstvy, přes vrstvy skryté až po vrstvu výstupní. V těchto modelech neexistují žádná zpětná spojení, kterými by byl výstup skrytých vrstev přenášen zpátky na jejich vstup. Takové modely se nazývají *rekurentní neuronové sítě*.

Modely dopředných neuronových sítí nalezly obrovské množství uplatnění v praxi, v nepřeberném množství ať už komerčních či nekomerčních aplikací. Například *konvoluční neuronové sítě*, kterým se v této práci budeme věnovat podrobněji v dalších kapitolách, se používají od algoritmů pro zpracování snímků v lékařství až po rozpoznávání lidí a jiných objektů na obrázcích nahraných na sociální sítě.

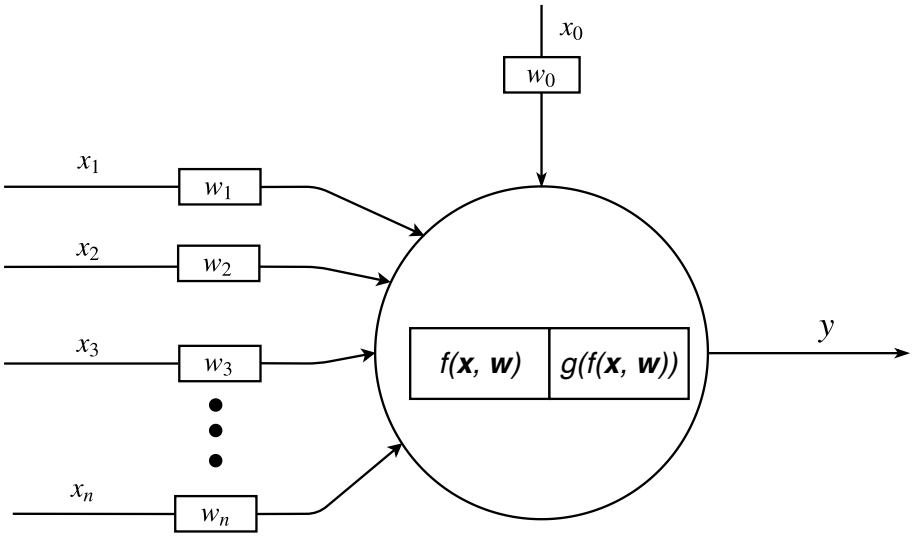
Cílem neuronových sítí je aproximace matematické funkce *f'*. Například binární klasifikátor $y = f'(\mathbf{x})$ mapuje vstupní vektor $\mathbf{x}$ na výslednou třídu y. Neuronová síť oproti tomu definuje mapování vstupního vektoru $\mathbf{x}$ a parametrů neuronové sítě $\boldsymbol{\theta}$ na výstupní vektor tříd $\mathbf{y}$ ($\mathbf{y} = f'(\mathbf{x}, \boldsymbol{\theta})$), přičemž se učí hodnoty parametrů $\boldsymbol{\theta}$, které dávají nejlepší aproximaci mapující funkce.

Každý model dopředné neuronové sítě je spojen s acyklickým grafem, který popisuje, jak jsou jednotlivé funkce spojeny do jedné. Například pokud jsou tři funkce f_1, f_2, f_3, spojeny do jedné sítě, dostaneme výslednou funkci $f(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x})))$. Takové zřetězené struktury funkcí jsou nejčastěji používané modely neuronových sítí. Jednotlivým funkcím f_1, f_2, f_3 se říká *vrstvy neuronové sítě*, kde f_1 je první vrstva, f_2 je druhá vrstva a f_3 je třetí vrstva. První vrstva se nazývá *vstupní vrstva* a poslední vrstva se nazývá *výstupní*

vrstva. Druhá vrstva se nazývá *skrytá vrstva*, protože není vidět z vnějšku neuronové sítě. Počet skrytých vrstev se v průběhu času neustále zvyšuje. Na počátku se používalo pouze pár skrytých vrstev, protože nebyly známé techniky pro učení sítí s větším počtem vrstev. V dnešní době už jsme schopni trénovat sítě i s tisícem skrytých vrstev. Výsledný počet vrstev udává tzv. *hloubku* modelu neuronové sítě. Odtud pochází pojem hluboké učení [20].

2.1 Umělé neuronové sítě

Původní předchůdci dnešních neuronových sítí a hlubokého učení byly jednoduché lineární modely abstrahující fungování lidského mozku na velmi zjednodušené úrovni. Tyto modely měly odpovídat funkci jednoho neuronu. Jejich koncept nebyl složitý, na vstupu braly n vstupních hodnot $x_1, x_2, ..., x_n$, pro všechny tyto hodnoty určily váhy $w_1, w_2, ..., w_n$, zpracovaly vstupní hodnoty a vypočítaly z nich jedinou výstupní hodnotu y [23].



Obrázek 2.1: Model umělého neuronu.

McCulloch-Pittsův neuron

Prvním z těchto modelů byl model, který v roce 1943 představili vědci W. S. McCullochem a W. Pittsem [36]. Jejich výpočetní model obsahoval pouze jeden neuron a dokázal počítat s binárními vstupy [0, 1] a bipolárními váhami [-1, 1] těchto vstupů. Váha o hodnotě 1 znamenala, že vstup je excitační, a hodnota -1 znamenala, že je daný vstup inhibiční. Výstup modelu je také binární, to znamená, že dokáže klasifikovat do dvou tříd. Výstup y dokážeme spočítat jako vážený součet všech vstupů násobených příslušnými vahami.

McCulloch s Pittsem ve své práci dokázali, že pomocí sítě těchto neuronů lze vypočítat jakoukoliv matematickou či logickou funkci, kromě logické funkce XOR. Tento model má stále mnoho nedostatků. Například pokud budeme chtít pracovat se vstupem, který není binární, nebo když potřebujeme určit pro jednotlivé vstupní hodnoty různé váhy mimo in-

terval [-1, 1]. Dalším problémem je, že tento model dokáže řešit pouze lineárně separovatelné úlohy.

Perceptron

Část problémů McCulloch-Pittsova neuronu vyřešil F. Rosenblatt v roce 1958, který přišel s modelem nazvaným *perceptron* [43]. Perceptron se také sestává z pouze jedné výpočetní jednotky (tzv. *jednovrstvý perceptron*), a proto může řešit jedině lineárně separovatelné úlohy. Pomocí jeho výstupu můžeme klasifikovat do dvou tříd. Velkým pokrokem oproti předchozímu modelu je, že dokáže zpracovávat vstupy v jiné formě než binární, váhy jednotlivých vstupů můžou také nabývat různých hodnot a hlavně existuje trénovací pravidlo, pomocí kterého můžeme upravovat hodnoty vah pro dané vstupy přímo v průběhu výpočtu.

Stejně jako předchozí model pracuje perceptron s vektorem vstupů $\mathbf{x}$, kterému přiřazuje vektor vah $\mathbf{w}$. Dalším vstupem do perceptronu je tzv. *bias*, neboli *práh*, jehož hodnota byla u předchozího modelu zvolena napevno bez možnosti změny [23]. Nyní se může měnit při učení společně s ostatními vahami z vektoru $\mathbf{w}$. Výstupní hodnotu počítá perceptron pomocí dvou funkcí – bázové funkce $f(\mathbf{x})$ a přenosové funkce $g(f(\mathbf{x}))$:

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i w_i + \theta \tag{2.1}$$

$$g(f(\mathbf{x})) = \begin{cases} 0 & \text{pro } f(\mathbf{x}) < \theta \\ 1 & \text{pro } f(\mathbf{x}) \geq \theta \end{cases}, \tag{2.2}$$

kde $\mathbf{x}$ je vektor vstupních hodnot, x_i je konkrétní vstupní hodnota, w_i je váha pro daný vstup, n je počet vstupů a θ je práh perceptronu [23].

Často se počítá práh jako váha w_0 nultého vstupu $x_0 = 1$, který má vždy hodnotu 1. Potom, jestliže je navíc hodnota prahu záporná, dostáváme funkce:

$$f(\mathbf{x}) = \sum_{i=0}^{n} x_i w_i \tag{2.3}$$

$$g(f(\mathbf{x})) = \begin{cases} 0 & \text{pro } f(\mathbf{x}) < 0 \\ 1 & \text{pro } f(\mathbf{x}) \geq 0 \end{cases}, \tag{2.4}$$

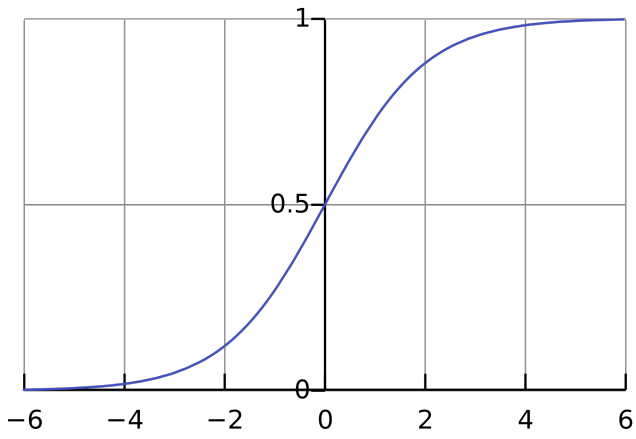
Výše zmíněná přenosová (někdy nazývaná také *aktivační*) funkce $g(f(\mathbf{x})$ je tzv. *skoková přenosová funkce* [20]. Různé typy přenosových funkcí budou popsány v následující sekci.

Přenosové funkce

Každý model neuronu řídí své chování podle vybrané přenosové funkce. Volba této funkce má velký vliv na chování modelu, a proto je potřeba zmínit několik nejpoužívanějších funkcí.

Skoková funkce

Velmi často používaná, především u osamocených neuronů, je skoková funkce, jejíž funkční předpis je naznačen v předchozí kapitole. Tato funkce umožňuje skokový přechod neuronu z neaktivního stavu do aktivního při překročení dané hodnoty prahu. Její definiční obor je definován na množině reálných čísel *(D(f) = R)* a obor hodnot je množina *H(f) = {-1, 1}*.



Obrázek 2.2: Ukázka aktivační funkce logistická sigmoida. Obrázek převzat z [6].

Logistická sigmoida

Je jednou z nejpoužívanějších funkcí pro vyjádření pravděpodobnosti určitého jevu jako výstupu neuronové sítě, protože její obor hodnot se nachází v intervalu (0, 1). Funkční předpis nejčastěji používané logistické funkce je:

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{2.5}$$

kde e je eulerovo číslo a x je hodnota vstupu. Obor hodnot této funkci patří do intervalu *H(f) = (0, 1)* a definiční obor je, stejně jako u předchozí funkce, definován na celé množině reálných čísel [38][37].

Hyperbolický tangens

Funkce velmi podobná funkci sigmodiální je hyperbolický tangens. Jediný rozdíl mezi těmito dvěma funkcemi je ten, že hyperbolický tangens má definován obor hodnot na intervalu (-1, 1). Použití hyperbolického tangentu spěje k rychlejší konvergenci učících algoritmů, než při použití sigmodiální funkce [38][37].

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{2x}} \tag{2.6}$$

ReLU

Rectified linear unit je skvělou výchozí volbou pro přenosovou funkci umělého neuronu, když si nejsme úplně jistí, kterou z přenosových funkcí vybrat. ReLU je velmi dobrá pro optimalizaci kvůli její podobnosti s lineárními funkcemi. Jediné v čem se liší je její část pro záporné f(x) z rovnice:

$$f(x) = max(0, x). \tag{2.7}$$

Z grafu funkce můžeme vidět, že v její aktivní části je její derivace rovna 1. Z toho plyne, že při optimalizaci za použití směru gradientů je daleko efektivnější, než předešlé funkce [40].

Konvoluční neuronové sítě

Konvoluční neuronové sítě, často označované pouze jako *CNN* z anglického názvu *Convolutional Neural Networks*, jsou speciálním typem neuronových sítí, které dokáží zpracovávat vstupní data ve tvaru n-rozměrných vektorů. Příkladem využití konvolučních neuronových sítí může být zpracování textu, který přijímají jako 1D vektor, nebo obrazu, který je reprezentován 3D maticí pixelů buď s jedním nebo se třemi kanály.

Tento koncept zpracování dat byl představen již v roce 1989, kdy Yann LeCun a spol. použili algoritmus *backpropagation* pro učení koeficientů konvolučního jádra na rozpoznávání ručně psaných číslic [31].

Velkým milníkem pro konvoluční neuronové sítě, ale i pro hluboké učení obecně, byla první implementace modelu konvoluční sítě pro grafickou kartu v roce 2006 [13]. Tato implementace byla asi čtyřikrát rychlejší než implementace pro CPU. Možnost trénování všech možných typů neuronových sítí na grafických kartách umožnilo zvládat potřebné složité výpočty v relativně krátkém čase. Od té doby udělal výzkum v oblasti konvolučních sítí velký pokrok. V roce 2012 Alex Krizhevsky a spol. představili hlubokou konvoluční neuronovou síť [30] pro klasifikaci obrazů z datasetu ImageNet [15], což je dataset 15 milionů označených obrázků z bezmála 22 000 tříd. Architektura této sítě obsahovala 8 trénovaných vrstev – 5 konvolučních a 3 plně propojené. O tři roky později byla představena architektura sítě ResNet s tzv. reziduálními bloky, které umožnily prohloubit síť až na 100, dokonce i 1000 vrstev [24].

V posledních letech velmi stoupla oblíbenost konvolučních neuronových sítí v praxi. Používají se nejrůznější modely pro klasifikaci, ať už v chytrých fotoaparátech a kamerách, nebo na zpracování fotek přidávaných na sociální sítě. Dále modely pro zvětšení rozlišení, tzv. *super-resolution*, fotek i videa, ale také například snímků v lékařství [33]. Dalším moderním využitím konvolučních sítí může být například řízení autonomních aut. Tyto modely dokáží segmentovat snímaný obraz na třídy nebo rozpoznávat barvu na semaforech a dopravní značení [11]. Tyto modely však zatím mají bezpečnostní problémy. Vědci z Oxfordské univerzity dokázali, že stačí změna několika málo pixelů v obraze a může dojít k záměně barvy na semaforech nebo špatnému rozpoznání dopravních značek [26].

V této kapitole popíšu samotnou operaci konvoluce, dále uvedu, jaké vrstvy obsahují typické konvoluční neuronové sítě a rozeberu principy modelů, používaných pro rekonstrukci obrazu.

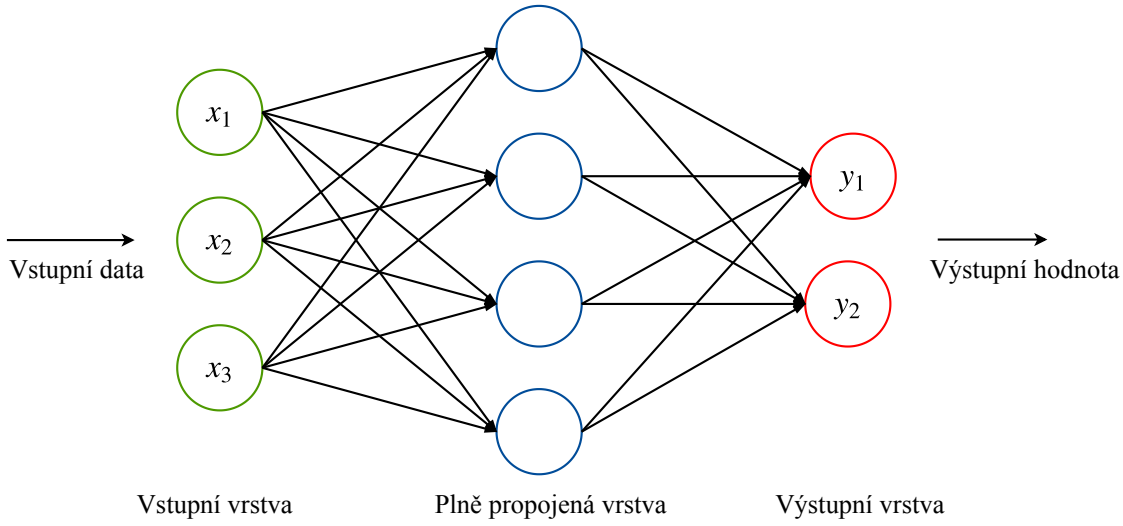
Plně propojená vrstva

Plně propojená vrstva je základní vrstvou neuronových sítí. Její název vyplývá z toho, že všechny neurony v ní jsou napojeny na všechny aktivace z předchozí vrstvy. Počet jejích trénovatelných parametrů roste úměrně s počtem neuronů, které obsahuje. Velký počet trénovatelných parametrů a z toho plynoucí velká výpočetní náročnost jsou jednou z hlavních nevýhod plně propojené vrstvy, které se snaží řešit vrstva konvoluční [20].

Konvoluce a konvoluční vrstva

Konvoluce je matematická operace, na které jsou konvoluční sítě založeny. Obecně je to operace nad dvěma funkcemi reálných čísel x a w:

$$s(t) = \int x(a)w(t-a)da, \tag{2.8}$$



Obrázek 2.3: Příklad plně propojené vrstvy.

kde $x(a)$ a $w(t-a)$ jsou spojité funkce označované jako vstup a konvoluční jádro a $s(t)$ je výstupem konvoluční operace, někdy nazývaným jako mapa příznaků (angl. *feature map*).

Klasicky se operace konvoluce uvadí operátorem $*$:

$$s(t) = (x * w)(t) \tag{2.9}$$

Rovnice 2.8 značí konvoluci nad nekonečně velkým spojitým 1D prostorem. Pro účel konvoluce v neuronových sítích však potřebujeme pracovat s konečnou množinou diskrétních hodnot. Nejčastěji je zpracováván obraz. Pro 2D černobílý obrázek I s jedním kanálem jako vstup použijeme i 2D konvoluční jádro K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{2.10}$$

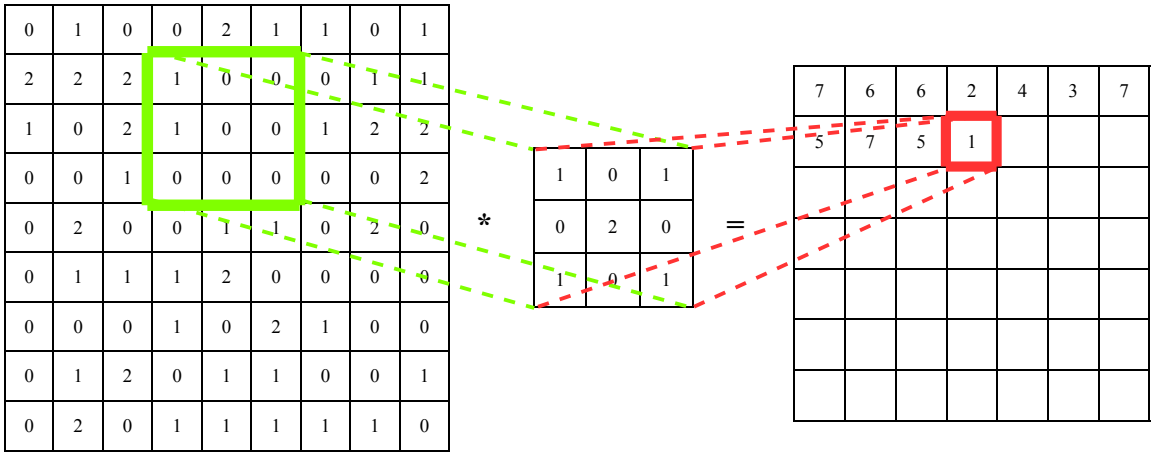
kde I je vstupní obrázek, K je konvoluční jádro, i a j jsou indexem bodu v obrázku a m a n jsou rozměry konvolučního jádra [20].

Hlavní motivací pro vznik konvoluční vrstvy bylo to, že reálná data trpí různými druhy nepřesností a deformací, například posunutí objektu v obraze oproti ostatním vzorkům v datasetu. Při zpracovávání takto deformovaných dat plně propojenými vrstvami neuspějeme, protože jednotlivé neurony těchto vrstev se učí absolutní pozice pixelů v obrázku, které jsou však u našeho vzorku deformovány a neodpovídají naučeným vzorům. Konvoluční vrstva však vidí i okolí právě zpracovávaných pixelů tak velké, jaké rozměry má její konvoluční jádro, a dokáže se učit hodnoty pixelů relativně v závislosti na jejich okolí.

Dalším důvodem je počet parametrů v plně propojené vrstvě. Jelikož je každý neuron napojen na všechny aktivace předchozí vrstvy, je mezi těmito dvěma vrstvami obrovské množství spojení a k nim patřících trénovaných parametrů. To zvyšuje výpočetní náročnost modelu a snížuje efektivitu.

Na druhé straně, konvoluční vrstvy umožňují tzv. *řídké interakce* [20] (angl. *sparse interactions*), které jsou realizovány pomocí konvolučních jader menších než je samotný vstupní obrázek, a pomocí kterých můžeme zkoumat pouze malé oblasti kolem daných pixelů, které jsou pro ně nejrelevantnější. Toto má za následek možnost extrakce významných

příznaků jako jsou například hrany objektů. Jako bonus dochází k významnému snížení počtu parametrů aniž by došlo ke snížení kvality modelu. Pokud chceme, aby konvoluční vrstva viděla oblast o větší ploše, jednoduše zvětšíme rozměry konvolučního jádra.



Obrázek 2.4: Princip konvoluční vrstvy pro výpočet nových hodnot za pomoci konvolučního jádra.

Aktivační vrstva

Aktivační vrstva provádí transformaci svého vstupu na odpovídající výstup podle zvolené aktivační funkce. Mezi nejčastěji používané aktivační funkce v konvolučních neuronových sítích patří tzv. ReLU popsaná v předchozí sekci rovnicí 2.7. V poslední době se i na poli rekonstrukce obrazu začaly hojně používat upravené verze této aktivační funkce – *Leaky ReLU* (LReLU) a *Parametric ReLU* (PReLU) [20].

Hlavní nevýhodou aktivační funkce ReLU je, že se může dostat do stavu, kdy bude pro téměř jakýkoliv vstup neaktivní, tj. bude vracet hodnotu 0. Pokud nastane tato situace, hodnoty gradientů propagovaných zpětně do sítě budou pro daný neuron také vždy 0. Toto je forma problému mizejícího gradientu popsaného v této kapitole v sekci 2.2. Řešením může být právě aktivační funkce Leaky ReLU, která má i pro záporné hodnoty vstupu malé kladné hodnoty gradientu, a tím pádem se stále může učit pomocí upravování hodnot vah. Leaky ReLU je popsaná pomocí následujícího funkčního předpisu:

$$f(x) = \begin{cases} x & \text{pro } x > 0 \\ 0,01x & \text{jinak} \end{cases}, \tag{2.11}$$

kde x je vstupem do aktivační funkce.

Jediným rozdílem mezi Leaky ReLU a Parametric ReLU je ten, že u Parametric ReLU si můžeme volit libovolnou hodnotu konstanty, kterou se násobí záporné hodnoty vstupu x do aktivační funkce.

$$f(x) = \begin{cases} x & \text{pro } x > 0 \\ ax & \text{jinak} \end{cases}. \tag{2.12}$$

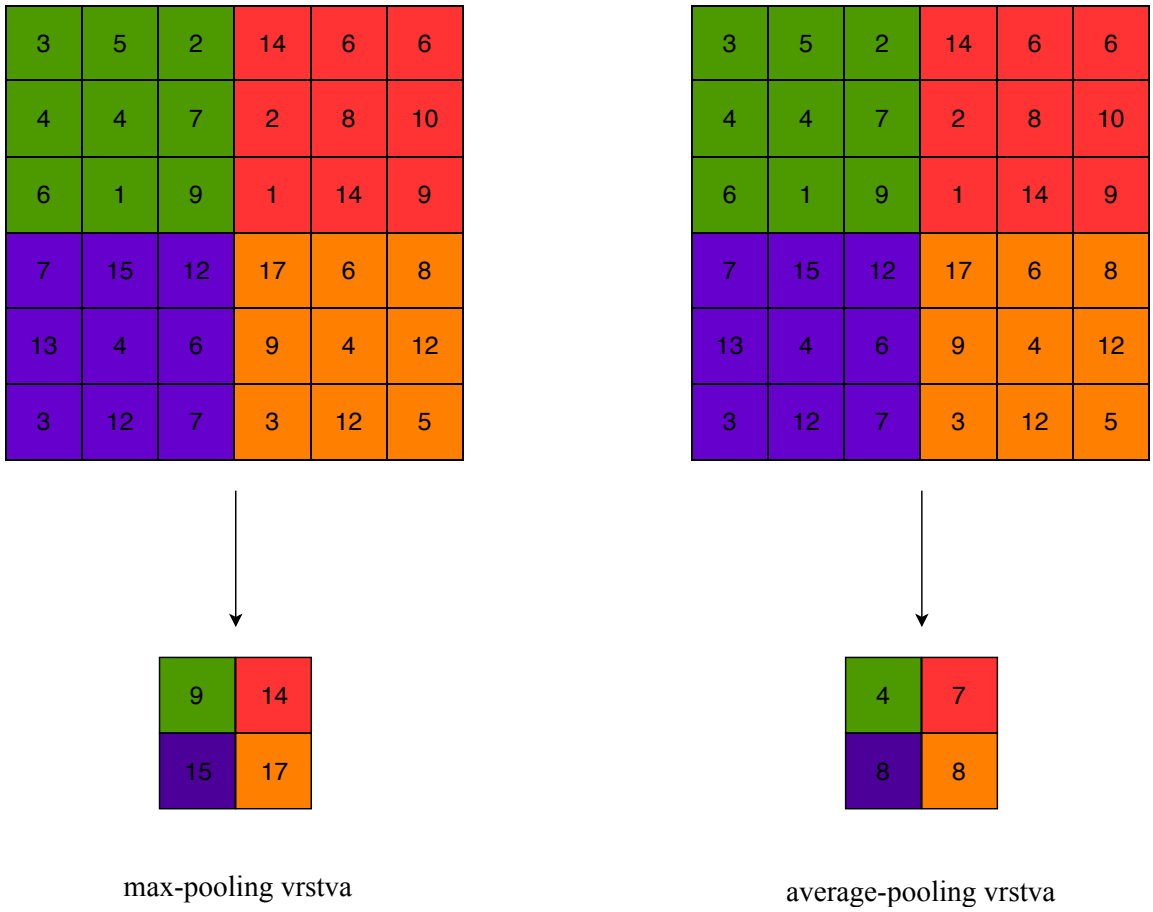
Normalizace dávek

Normalizace dávek (angl. *batch normalization*) [27] je vrstvou, která normalizuje vstupní dávku vzorků tak, aby průměr jejich hodnot byl co nejblíže 0 a směrodatná odchylka se blížila k hodnotě 1. Tato normalizační vrstva je velmi významná z hlediska urychlení a stabilizace trénování. Velkou nevýhodou této vrstvy ale je její paměťová náročnost, kdy potřebuje stejné množství paměti, jako její předcházející konvoluční vrstva, za kterou se umisťuje.

Sdružovací vrstva

Důležitou vrstvou je sdružovací vrstva (angl. *pooling layer*). Princip této vrstvy je velmi jednoduchý. Postupně prochází data a z okolí o předem dané velikosti vybere nebo vypočítá 1 hodnotu. Pokud se po datech pohybuje s posunem (angl. *stride*) větším než 1, dochází ke snížení počtu dimenzí výstupních dat oproti datům vstupním. Tato vrstva tedy může provádět podvzorkování (angl. *downsampling*), což umožňuje méně náročný výpočet v dalších vrstvách. Další výhodou této vrstvy je, že zabezpečuje invarianci proti malým posunutím vstupních dat.

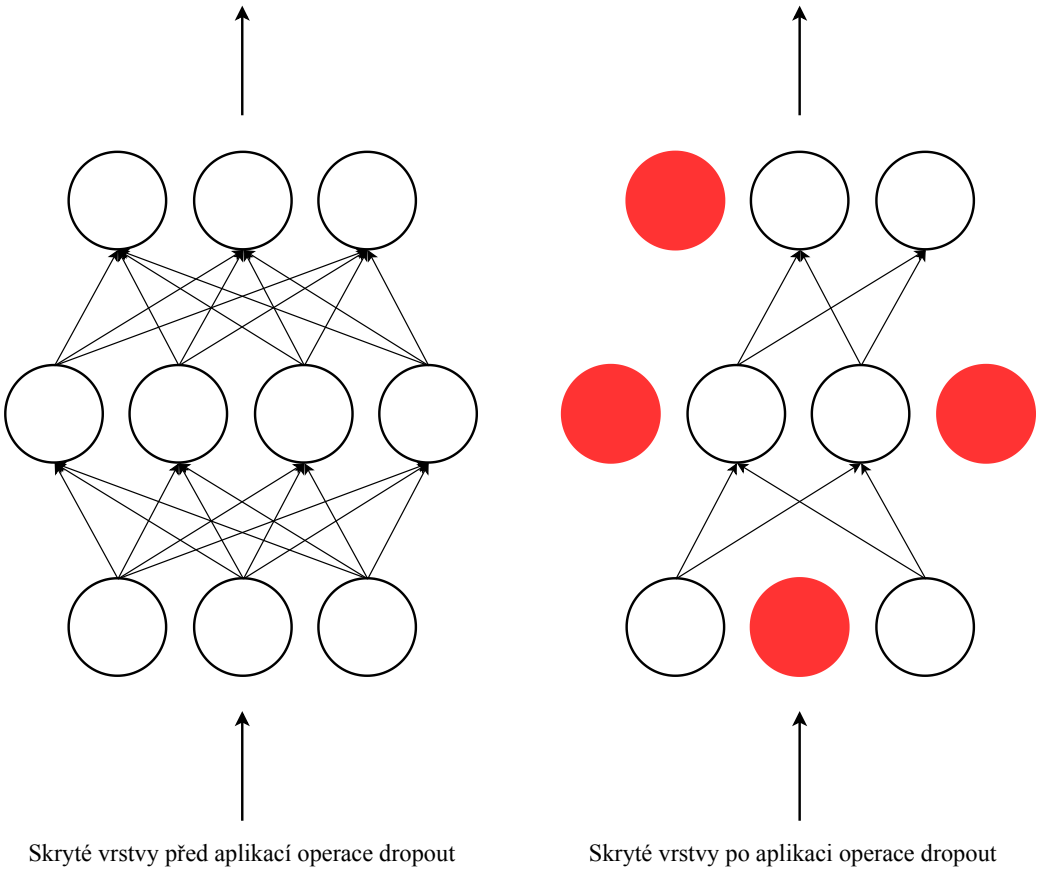
Sdružovací vrstva má několik verzí, které se používají v praxi, pro příklad uvedu nejznámější, kterými jsou *max-pooling* a *average-pooling*. První jmenovaná vybere z okolí prvek s největší hodnotou a ten propaguje jako jediný dál. Druhá v pořadí spočítá průměr všech hodnot prvků v daném okolí a tuto hodnotu posune na svůj výstup.



Obrázek 2.5: Ukázka použití operací max-pooling a average-pooling.

Dropout vrstva

Funkcí této vrstvy je náhodně vybrat předem daný počet neuronů a zahodit jejich výstup. Tento proces pomáha významným způsobem u problému přetrénování, který je popsán v sekci 2.2, kdy se model sítě natrénuje až příliš přesně podle trénovací množiny dat, ale nedokáže řešení problému zobecnit na vstupy mimo trénovací množinu.



Obrázek 2.6: Příklad použití operace dropout na plně propojené vrstvě.

2.2 Učení neuronových sítí

Schopnost učit se je jednou z nejdůležitějších vlastností neuronových sítí. Během procesu učení dochází ke změnám hodnot vah jednotlivých spojení mezi neurony v neuronové síti, a tím se zlepšuje odezva sítě na trénovací data. Cílem učení je minimalizovat výsledek chybové funkce, který určuje jaký je rozdíl mezi výstupem neuronové sítě a očekávaného výstupu [20]. Během dlouhého vývoje bylo představeno mnoho učících algoritmů, z nichž nejznámějším je určitě *back-propagation*, neboli zpětné šíření chyby [20].

Chybová funkce

Výběr chybové funkce, anglicky nazývané *cost function* nebo také *loss function*, je velmi důležitým krokem při návrhu modelu neuronové sítě. Udává, jak moc je hodnota získaná

jako výstup neuronové sítě odlišná od očekávaného výsledku, který je součástí trénovacího datasetu.

Mean squared error

Jedna ze základních používaných chybových funkcí je *mean squared error (MSE)*. Její definice vychází z rozdílu mezi tzv. *ground truth*, což je očekávaná výsledná hodnota z trénovací sady, a hodnotou, která vypadla jako výsledek neuronové sítě. Definice vypadá následovně:

$$MSE = \frac{1}{N}\sum_{p=1}^{N}\sum_{j=1}^{M}(y^*_{p,j} - y_{p,j})^2, \tag{2.13}$$

kde N je počet vstupních trénovacích dat a M je počet výstupních neuronů. $y^*_{p,j}$ je *ground truth* (originální data) a $y_{p,j}$ je výstup neuronové sítě. Pokud platí $y^*_{p,j} = y_{p,j}$, je $MSE = 0$ [20].

Cross entropy

Chybová funkce *cross entropy* se nejvíce osvědčila jako součást *Softmax klasifikátorů* klasifikujících 1 z K tříd, který dává výstupy z intervalu (0, 1), jenž se dají považovat za normalizovanou pravděpodobnost s jakou patří daný vstup do konkrétní třídy. Funkční cross-entropy pro N vzorků z trénovací sady je počítána jako průměr všech hodnot cross-entropy pro jednotlivé vzorky:

$$E = \frac{1}{N}\sum_{n=1}^{N} -log\left(\frac{e^{\mathbf{f}_{y_n}}}{\sum\limits_{j=0}^{J} e^{\mathbf{f}_j}}\right), \tag{2.14}$$

kde N je počet vzorků z trénovací sady, y_n je správně predikovaná třída pro daný vstup neuronové sítě uložená ve výstupním vektoru $\mathbf{f}$ s pravděpodobnostmi pro jednotlivé třídy j jsou indexy všech pravděpodobností uložených ve vektoru $\mathbf{f}$ [42][5].

Least absolute deviation error

Jednou z používaných chybových funkcí u rekonstrukce obrazu je funkce *least absolute deviations error (L1)*. Tato funkce se snaží minimalizovat absolutní odchylky uměle generovaných dat od dat originálních. Výsledná hodnota funkce je počítána podle vzorce:

$$L1 = \sum_{i=0}^{n} |y^*_i - y_i|, \tag{2.15}$$

kde n je počet vzorků, ze kterých se chyba počítá, y^*_i je ground-truth a y_i jsou generovaná data.

Optimalizační algoritmy

Výběr optimalizačního algoritmu má velký vliv na průběh učení neuronové sítě, a proto je jeho volba velmi důležitým rozhodnutím. Mezi nejznámější a nejvíce používané algoritmy patří bezesporu algoritmus gradientního sestupu (angl. *gradient descent*), od kterého jsou odvozeny další známé algoritmy – například *Adam*. Oba tyto algoritmy budou rozebrány v této sekci.

Gradientní sestup

Iterativní algoritmus gradientního sestupu byl představen už v roce 1847 [12]. Cílem tohoto algoritmu je minimalizovat funkci $y = f(\mathbf{x})$. Minimalizace se dosáhne pomocí parciálních derivací funkce y podle x_i $\left(\frac{\delta y}{\delta x_i}\right)$, kde x_i je vstup do neuronové sítě, pomocí které získáme směr růstu funkce y v bodě x_i. Jednotlivé parciální derivace v bodech z vektoru $\mathbf{x}$ tvoří tzv. *gradient*, který udává, jakým směrem funkce y nejrychleji roste. Pro minimalizaci chybové funkce tedy použijeme zápornou hodnotu gradientu. Nové hodnoty vah v neuronové síti budeme upravovat podle vzorce:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \epsilon \nabla_{\mathbf{w}_i} f(\mathbf{w}_i), \tag{2.16}$$

kde $\mathbf{w}_{i+1}$ jsou nově vypočtené hodnoty vah pro další iteraci $i + 1$, w_i jsou hodnoty vah z aktuální iterace i, ϵ je učící konstanta, tzv. *learning rate*, která se většinou volí jako malé kladné číslo. Dále pak $\nabla_{\mathbf{w_i}} f(\mathbf{w}_i)$ je gradient vypočtený v iteraci i [20]. Algoritmus opakujeme, dokud se hodnota chybové funkce nepřestane snižovat, nebo neklesne pod předem určenou hodnotu.

Algoritmů gradientního sestupu máme několik typů, které se liší velikostí dávky dat z datasetu, se kterou se pracuje v jedné iteraci pro počítání nových hodnot vah. Základní verze, která počítá novou hodnotu gradientu z dávky o velikosti celého datasetu (angl. *batch gradient descent*), se neukázala jako ideální. Pro výpočet nových hodnot chybové funkce a poté gradientů se musí projít celý dataset, což může být velmi pomalé u velkého počtu dat. Hodnota chybové funkce se počítá jako průměr hodnot chybové funkce pro jednotlivé vzorky z datasetu. Navíc se může stát, že tento algoritmus bude konvergovat do lokálního minima chybové funkce, které má daleko horší vlastnosti než globální minimum chybové funkce [20].

Místo průchodu přes celý dataset, počítání průměrné hodnoty chybové funkce a až následné úpravy parametrů můžeme počítat nové hodnoty parametrů pro každý jednotlivý vzorek v datasetu. Tomuto přístup se říká stochastický gradientní sestup (angl. *stochastic gradient descent*). Mezi jeho hlavní výhody patří rychlost, která vychází z toho, že pro výpočet nových hodnot parametrů nemusí procházet všechny vzorky dat. Dále tento algoritmus pomáhá zlepšit generalizaci tím, že přidává malou dávku šumu do učícího procesu.

Třetím typem je výpočet hodnot gradientu z dávky vzorků o předem určené velikosti. Je rychlejší než klasický gradientní sestup z celé dávky a navíc, stejně jako stochastický gradientní sestup, přidává do učícího procesu malé množství šumu, čímž pomáhá lepší generalizaci. Jeho nevýhodou je, že kvůli šumu může v každém kroku skákat tam a zpět a nepohnout se z místa – nekonvergovat do minima. Tomu se dá ale předejít postupným snižováním učicího koeficientu.

Adam

Optimalizační algoritmus *Adam* [29], jehož jméno je zkráceninou z anglického *Adaptive moment estimation*, pracuje na velmi podobném principu jako stochastický gradientní sestup zmíněný dříve. Oba algoritmy používají koeficient učení, který říká, jak moc se mají změnit váhy v závislosti na gradientu chybové funkce. Stochastický gradientní sestup používá pro všechny parametry modelu stejnou hodnotu tohoto koeficientu, která se v průběhu učení nemění. Adam, nadruhou stranu, používá koeficient učení pro každý parametr jiný a zároveň upravuje jejich hodnoty během učení. Pro výpočet gradientů je použit algoritmus zpětného šíření chyby, který bude popsán v další sekci.

Backpropagation

Algoritmus zpětného šíření chyby, nazývaný *backpropagation*, je často zaměňován za kompletní učicí algoritmus dopředných neuronových sítí. Jeho úkolem je však umožnit informacím získaným pomocí výpočtu chybové funkce šířit se zpětně neuronovou sítí, aby bylo možno počítat gradient pro jednotlivé parametry sítě. Samotné učení má pak na starosti například algoritmus stochastického gradientního sestupu za pomoci těchto gradientů.

Učení s algoritmem Backpropagation sestává ze tří etap:

1. dopředné šíření vstupních hodnot,
2. výpočet gradientů pomocí zpětného šíření chyby a řetězového pravidla,
3. aktualizace hodnot váhových koeficientů na spojeních mezi neurony.

Řetězové pravidlo

Řetězové pravidlo (angl. *chain rule*) je princip, pomocí kterého se počítají parciální derivace funkce, která je složena z jiných funkcí, jejichž parciální derivace v daném bodě už známe. V neuronové síti to znamená, že můžeme pomocí zpětného šíření počítat derivace chybové funkce v závislosti na daném parametru sítě pomocí derivací chybové funkce v závislosti na parametrech, které se v architektuře sítě nacházejí blíže výstupního bodu.

Obecně můžeme řetězové pravidlo pro dvě funkce f a g, které reprezentují zobrazení realného čísla na reálné číslo, zapsat jako:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}, \tag{2.17}$$

kde platí, že $y = g(x)$ a $z = f(g(x)) = f(y)$, a x a y jsou reálná čísla. Předchozí rovnice může být zobecněna i pro vektory:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}, \tag{2.18}$$

kde x_i je prvek z vektoru $\mathbf{x} \in \mathbb{R}^m$, y_j je prvek z vektoru $\mathbf{y} \in \mathbb{R}^n$, g je funkce definující zobrazení $\mathbb{R}^m$ do $\mathbb{R}^n$ a f je funkce definující zobrazení z $\mathbb{R}^n$ do $\mathbb{R}$. Dále platí, že $\mathbf{y} = g(\mathbf{x})$ a $z = f(g(\mathbf{x})) = f(\mathbf{y})$. Ekvivalentní vektorový zápis předešlé rovnice pomocí gradientu je:

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_{\mathbf{y}} z, \tag{2.19}$$

kde $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ je Jacobiho matice funkce g o rozměrech n x m, $\nabla_{\mathbf{x}} z$ je gradient funkce z s ohledem na vektor $\mathbf{x}$ a $\nabla_{\mathbf{y}} z$ je gradient funkce z s ohledem na vektor $\mathbf{y}$ [20].

Problémy při trénování neuronových sítí

Problém mizejícího gradientu

Tento problém vzniká například při použití aktivační funkce, která mapuje velká čísla na čísla z relativně malého intervalu. Příkladem může být tangens hyperbolický nebo sigmoida. Toto mapování způsobuje, že velká změna vstupu zapříčiní pouze malou změnu výstupu, takže derivace funkce jsou malé hodnoty. Při počítání gradientů pomocí řetězového pravidla

dochází k násobení jednotlivých parciálních derivací od poslední vrstvy směrem k počáteční. Jelikož jsou tyto parciální derivace nízká čísla (kladné hodnoty nižší než 1), jejich násobením získáme gradient, který je ještě daleko nižší hodnota, než samotné parciální derivace z těchto vrstev. Po aplikaci gradientu na váhy přechodu sítě se jejich hodnota téměř nezmění, takže nedochází k učení a trénování sítě je zaseknuto na stejném místě.

Problému mizejícího gradientu se dá předejít několika způsoby. Prvním z nich je použití aktivační funkce, která nemapuje velká čísla do intervalu o malém rozptylu, a jejíž derivace pak nejsou tak nízké hodnoty. Příkladem může být již zmíněná funkce *Parametric ReLU*. Dalším řešením může být použití vrstvy normalizace dávek před aktivační vrstvou, nebo použití reziduálních bloků s reziduálními spojeními, které jsou napojeny přímo na vstup do reziduálního bloku a neprochází aktivací. Tato spojení pak přivádí hodnotu vstupu do součtu s výstupem celého bloku, a tím zvyšují celkovou hodnotu derivace celého bloku. Princip reziduálních spojení bude více popsán v dalších kapitolách. Použití normalizace dávek před aktivační vrstvou zajistí, že vstup do aktivační vrstvy už je normalizován do potřebného intervalu, a tím pádem nedochází ke snížení hodnot derivací.

Špatná generalizace

Při trénování neuronových sítí se snažíme minimalizovat chybovou funkci na základě trénovací sady dat. Další důležitou vlastností modelu neuronové sítě ale je, aby byla schopná fungovat s nízkou chybou i při zpracování dat nových, při trénování neviděných. Tuto chybu počítáme během procesu testování na základě testovací sady, která obsahuje odlišné vzorky než sada trénovací. Schopnost podávat dobré výsledky i na datech nových se nazývá generalizace.

Pro zjištění, jak dobře si model vede slouží 2 faktory:

1. schopnost modelu minimalizovat chybu na trénovací množině a
2. schopnost modelu minimalizovat rozdíl chyb na trénovací množině a testovací množině.

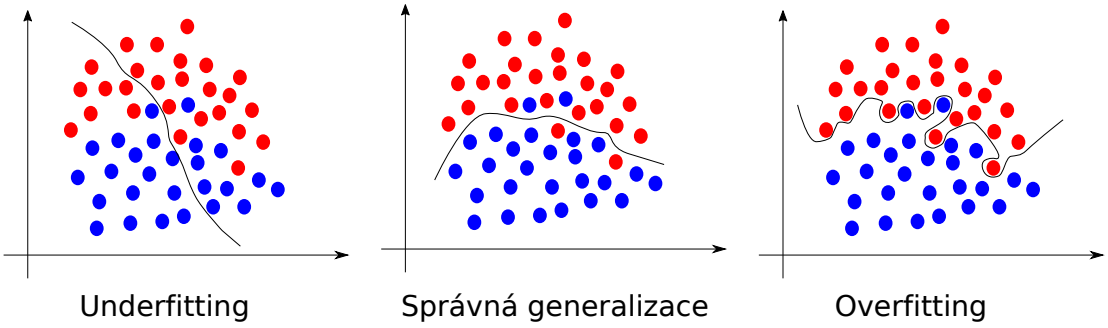
Tyto 2 faktory úspěšnosti odpovídají dvěma problémům strojového a hlubokého učení obecně, a těmi jsou *underfitting*, neboli nedostatečné natrénování, a *overfitting*, neboli přetrénování.

Underfitting

Vzniká pokud model není schopen minimalizovat chybovou funkci na trénovací sadě. Znamená to, že není schopný naučit se vztahy mezi trénovacími daty a upravit podle toho svoje parametry. Možné řešení tohoto problému může být zvětšení modelu a přidání více parametrů.

Overfitting

Dochází k němu, když je velký rozdíl mezi chybami na trénovací a testovací sadě. Model si zapamatuje vlastnosti a vztahy mezi trénovacími daty, ale nedokáže pracovat se vzorky mimo trénovací množinu. Proti overfittingu často pomáhá zvětšení trénovací sady a trénování na více vzorcích.



Obrázek 2.7: Porovnání případů kdy nastává underfitting a overfitting oproti téměř úplně správné generalizaci.

2.3 Modely neuronových sítí pro rekonstrukci obrazu

Rekonstrukce obrazu z hlediska zvětšení rozlišení (angl. *super-resolution*), tedy zvýšení počtu pixelů pro danou oblast obrazu, bývá často zaměňováno s podobnými tématy z pole zpracování obrazu, jako například odstranění šumu z obrazu (angl. *image deblurring*) nebo interpolace. Během tzv. odstranění šumu dochází k obohacení kvality obrazu, ale nedochází ke zvětšení rozlišení. U interpolace dochází ke zvětšení rozlišení, ale nejsou obnoveny detaily obrázku s ostrými přechody barev a obraz je rozmazaný, např. hrany objektů nebo rysy na obličeji. Dále v textu se budu rekonstrukcí obrazu odkazovat na zvětšení rozlišení (*super-resolution*).

Metody rekonstrukce obrazu se dělí na 2 kategorie:

1. Metody specifické pro danou doménu – soustředí se pouze na rekonstrukce obrazů z jedné třídy (obličeje, panoramata, atd.),

2. Obecné metody – snaží se rekonstruovat obraz z jakékoliv třídy a přistupovat k problémům obecně.

Dále můžeme metody rozdělit na ty, které pracují v prostoru vysokého rozlišení (HR metody), a ty, které pracují v prostoru s nízkým rozlišením (LR metody). Jestli pracují v prostoru nízkého nebo vysokého rozlišení znamená, že berou jako vstup obraz s rozlišením původním, které bude zvětšeno až na konci zpracování obrázku, nebo obrázek s rozlišením už zvětšeným například bikubickou interpolací.

První pokusy o rekonstrukci obrazu používali interpolační metody založené na vzorkovací teorii. Tyto metody však nedávaly kvalitní výsledky. Výsledné obrázky obsahovaly velmi nekvalitní rekonstrukci textur a hran, které byly rozmazané. Poté byly představeny metody, které se zaměřovaly na predikci hran v obraze [10][48], ale ani výsledky těchto metod nebyly uspokojivé.

Slibný přístup představili Freeman a spol. [18]. Jednalo se o metodu, která byla hodně závislá na trénovacím datasetu. Základní princip byl najít správné mapování mezi dvojicemi obrázků, kde první ze dvojice je obrázek s nízkým rozlišením a druhý je jemu odpovídající obrázek s vysokým rozlišením. Na těchto principech se zakládají i metody využívající hluboké učení.

Od představení těchto prvních metod pro rekonstrukci obrazu uběhla už dlouhá doba a mezitím vznikaly další pokusy s nejrůznějšími metodami. Za zmínku stojí přístup, který porovnával vstupní obrázek s nízkým rozlišením s dalšími nízko-dimenzionálními obrázky,

vybral nejvíce podobné a výsledný obrázek se zvětšeným rozlišením byl seskládán z odpovídajících vysoko-dimenzionálních obrázků ve vysokém rozlišení [45][46].

Poslední a zároveň nejúspěšnější pokusy o rekonstrukci obrazu se zakládaly na metodách hlubokého učení. Jedním z prvních přístupů využívající metod hlubokého učení a konvolučních neuronových sítí byl prezentován v roce 2015 [16]. Jeho autoři Dong a spol. implementovali konvoluční síť se třemi konvolučními vrstvami, která si v porovnání s tehdejšími *state-of-the-art* metodami vedla velmi dobře. Nedlouho poté, co byla představena architektura ResNet [24], došlo k jejímu využití i v rekonstrukci obrazu. V roce 2017 byla zveřejněna síť SRResNet [32], která aplikovala ResNet architekturu na rekonstrukci. SRResNet pouze převzala stavební reziduální bloky tak, jak byly představeny v ResNet modelu pro klasifikaci, přesto ale dokázala podávat velmi dobré výsledky a stala se *state-of-the-art* metodou pro čtyřnásobné zvýšení rozlišení. Pozdější práce, které aplikovaly architekturu ResNet na rekonstrukci a jiné nízko-úrovňové úlohy z oblasti zpracování obrazu, použily upravené reziduální bloky, které neobsahovaly vrstvy normalizace dávek, protože ty měly velkou paměťovou náročnost (téměř stejnou jako jim předcházející konvoluční vrstvy) a navíc kvůli normalizaci příznaků snižovaly flexibilitu modelu [39][33]. Příkladem takového modelu může být EDSR [33], který dokázal podávat skvělé výsledky a stal se nový *state-of-the-art* modelem. Společně s EDSR byl ve stejné práci navrhnut i model VDSR [33]. Rozdíl mezi nimi je ten, že EDSR dělá samotné zvýšení rozlišení pouze v jednom kroku, kdežto VDSR ve více krocích. Tato úprava nepřinesla významné vizuální rozdíly ve výsledných rekonstruovaných snímcích, měla ale za následek výrazné snížení paměťové náročnosti modelu a použití daleko menšího počtu trénovatelných parametrů. Spolu s modelem SRResNet byl ve stejné práci představen další model, a to SRGAN [32]. Jedná se o model, který staví na principech generativních kompetitivních sítí. Také používá stavební reziduální bloky jako SRResNet, ale princip kompetitivního trénování umožnil generování ostřejších hran a věrohodnější zpracování textur. Další z modelů postavených na tomto principu trénování je například ESRGAN [47]. Tento model také použil reziduální bloky bez normalizace dávek, ale ty jsou ještě upraveny tak, že každá konvoluční vrstva v něm bere jako vstup vstupní tenzor do reziduálního bloku konkatenovaný s tenzory, které jdou jako výstup z každé dvojice konvoluce-aktivace, která je v daném reziduálním bloku před ní. Tento model patří mezi jeden z nejlepších, co se rekonstrukce obrazu týče. Generativní kompetitivní sítě jsou zajímavým přístupem pro řešení rekonstrukce obrazu a jejich princip bude ve zkratce rozebrán v následující sekci této kapitoly.

Stejný problém jako tato bakalářská práce řeší Dahl a spol. v práci nazvané *Pixel Recursive Super Resolution* [14]. Rovněž se snaží rekonstruovat snímky obličejů, které jsou významně rozmazány a mají velmi nízké rozlišení. Jejich přístup je založen na tzv. PixelRNN a PixelCNN modelech, což jsou rekurzivní neuronové sítě, které generují jednotlivé pixely postupně v závislosti na již vygenerovaných pixelech. PixelCNN navíc přidává větší váhu závislosti hodnoty právě generovaného pixelu na pixelech v jeho okolí pomocí konvolučního jádra. Tento přístup generování pixelů je velmi pomalý a časově náročný, protože se v každém kroku generuje pouze jeden pixel. Ve své práci jsem si tedy zvolil cestu rekonstrukce pomocí konvolučních neuronových sítí založených na architektuře ResNet.

2.4 Generativní kompetitivní neuronové sítě

Generativní kompetitivní sítě (angl. *Generative Adversarial Nets, GANs*) představili Goodfellow a spol. v roce 2014 [21]. Jednalo se o zcela nový princip generativních modelů neuronových sítí, který bude ve zkratce rozebrán v této sekci, která vychází z [21] .

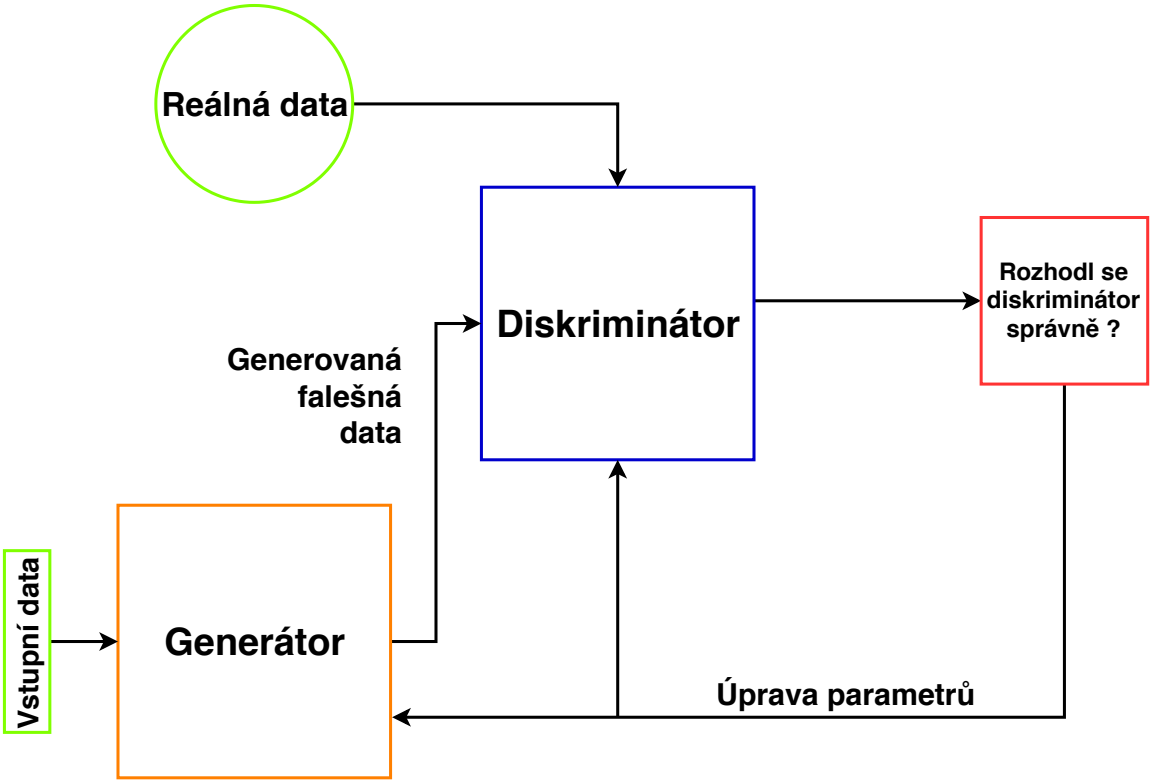
V hlubokém učení měly úspěch hlavně diskriminativní modely, které mapují vysoce dimenzionální vstupy na třídy objektů. Generativní modely nebyly v tomto směru tak úspěšné, hlavně kvůli extrémním pravděpodobnostním výpočtům, ze kterých je počítán odhad maximální pravděpodobnosti určitého rozložení. Dalším problémem byl fakt, že výhody, které výpočetní jednotky (neurony) v síti poskytovaly během klasifikace nebo rozpoznávání, nemohly být plně využity při generování dat.

Při učení generativních kompetitivních sítí netrénujeme pouze jeden model, ale přímo dva. První model je generativní, nazývaný *generátor*, který generuje data, a druhý je diskriminativní model, zvaný *diskriminátor*, který se snaží rozpoznat, zda je zpracovávaný vzorek falešný, vygenerovaný generativním modelem, nebo jestli se jedná o originální vzorek z trénovacího datasetu. Autoři tohoto principu to přirovnávají k padělatelům bankovek, kteří se snaží bankovky falšovat tak kvalitně, že je nikdo nerozezná od originálu, a policii, která se snaží falešné bankovky rozeznat od ostatních. Pokud je však padělatel dostatečně dobrý, naučí se bankovky falšovat tak dobře, že po nějaké době už nebude policie schopna rozpoznat originální bankovku od falešné.

Modely generativních kompetitivních sítí se učí pravděpodobnostní rozložení p_g, ze kterého generuje vzorky generátor. Pro správné naučení tohoto rozložení musíme definovat rozložení $p_z(\boldsymbol{z})$, ze kterého bude generován vstupní šum $\boldsymbol{z}$ do generátoru. Daný generátor reprezentuje mapování vstupního šumu $\boldsymbol{z}$ na vzorky z rozložení p_g. Toto mapování je zapsáno jako $G(\boldsymbol{z}, \boldsymbol{\theta}_g)$, kde G je neuronová síť s parametry $\boldsymbol{\theta}_g$ představující derivovatelnou funkci. Poté musíme definovat i druhý model, který je také reprezentován neuronovou sítí, a tím je diskriminátor $D(\boldsymbol{x}, \boldsymbol{\theta}_d)$. Jeho výstupem je pravděpodobnost $D(\boldsymbol{x})$, která říká, že $\boldsymbol{x}$ patří mezi originální data z datasetu a ne mezi vzorky z p_g. Diskriminátor trénujeme tak, aby maximalizoval pravděpodobnost správného rozpoznání obou vzorků, jak generovaného, tak originálního. Generátor je pak trenován tak, aby minimalizoval pravděpodobnost správného rozpoznání generovaného vzorku diskriminátorem – $log(1 - D(G(\boldsymbol{z})))$. Diskriminátor a generátor vlastně hrají hru zvanou minimax s hodnotovou funkcí $V(G, D)$:

$$\min_G \max_D V(G, D) = E_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}(logD(\boldsymbol{x})) + E_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}(log(1 - D(G(\boldsymbol{z})))). \tag{2.20}$$

U předchozí rovnice se může stát, že diskriminátor bude s velkým „sebevědomím" rozpoznávat generované vzorky, protože generátor se ještě dostatečně nenaučil, a generuje vzorky úplně odlišné od originálních dat. Pak dochází ke ztrátě gradientů pro generátor a $log(1 - D(G(\boldsymbol{z}))$ saturuje. Abychom zamezili takové situaci, budeme raději trénovat generátor, aby maximalizoval $log(D(G(\boldsymbol{z})))$, což je pravděpodobnost, že dokáže oklamat diskriminátor. Toto zajistí mnohem silnější gradienty na začátku trénování, kdy dává generátor ještě velmi špatné výsledky. Této funkci, kterou maximalizuje generátor se říká *non-saturating loss*.



Obrázek 2.8: Princip fungování generativních kompetitivních sítí. Generátor přijme vstupní data a vygeneruje z nich falešný vzorek. Tento vzorek je poté předán diskriminátoru, který se snaží správně rozlišit falešný a pravý vzorek dat. Posledním krokem je úprava parametrů obou modelů – diskriminátoru i generátoru – podle toho, jak byl diskriminátor úspěšný.

Kapitola 3

Návrh modelů pro rekonstrukci obrazu

Cílem této práce bylo navrhnout model, který vezme snímek obličeje v tak nízkém rozlišení, že na něm jsou významně rozmazané jeho základní části – nos, ústa, oči. Realizaci této úlohy je potřeba seskládat z několika základních kroků. Prvním z nich je návrh modelů, které dokážou správně extrahovat příznaky ze vstupního snímku v nízkém rozlišení, a namapovat je do vyššího rozlišení, do kterého bude vstupní snímek rekonstruován. Pro rekonstrukci jsem si vybral modely založené na hlubokém učení, jehož metody dokázaly v poslední době posunout rekonstrukci obrazu na novou úroveň. Konkrétně jsem zvolil model postavený na konvolučních neuronových sítích a architektuře ResNet, jehož návrh popíšu v první části této kapitoly. Tímto modelem bych chtěl dosáhnout čtyřnásobného zvýšení rozlišení vstupního snímku. Druhý model je postaven na principu generativních kompetitivních sítí, které měli v posledních letech v rekonstrukci obrazu velký úspěch. I tento model bude pracovat s čtyřnásobným zvýšením rozlišení vstupního snímku.

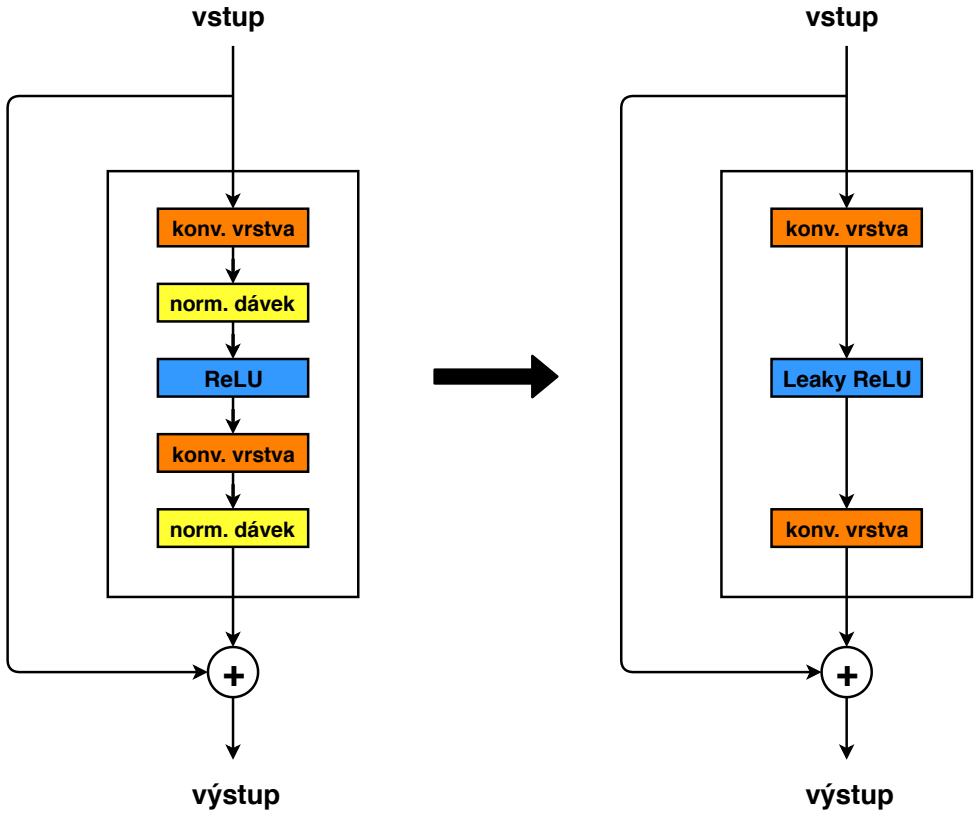
3.1 Model založený na architektuře ResNet

Prvním rozhodnutím bylo, zda můj model bude realizovat rekonstrukci snímků ve vysokém nebo nízkém rozlišení, tzn. zda se zpracuje snímek v nízkém rozlišení, který se zvětší až na úplném konci zpracování, nebo zda se snímek zvětší hned na začátku (např. pomocí bikubické interpolace), a následně dojde k jeho zpracování. Jasnou volbou byla rekonstrukce v nízkém rozlišení, protože má daleko nižší paměťové nároky a bylo prokázáno, že rekonstrukce v nízkém rozlišení nedává horší výsledky, než rekonstrukce ve vysokém rozlišení [33][32][47]. Jak již bylo zmíněno dříve, zvolil jsem architekturu ResNet, a u ní jsou nejdůležitější komponentou reziduální bloky a vrstvy, ze kterých jsou složeny.

V rámci experimentování jsem vytvořil návrh modelu, který se od modelu popsaného v této kapitole liší jiným uspořádáním reziduálních bloků. Změněný model má větší hloubku, s čímž přichází i větší paměťová a výpočetní náročnost. Nově navržený model je popsán v kapitole Experimenty v sekci 5.4.

Reziduální bloky

Pro svůj model jsem nejdříve zvolil reziduální bloky velmi podobné těm, které představili Zhang a spol. ve své práci o reziduálním učení [24], a které byly také použity v rekonstrukci obrazu [32]. Tyto bloky obsahují dvě konvoluční vrstvy. Za první z nich následuje vrstva



Obrázek 3.1: Tento obrázek představuje změnu návrhu reziduálního bloku. Vlevo je původní návrh a na pravé straně lze vidět návrh po změně, která spočívala v odstranění vrstev normalizace dávek.

normalizace dávek a aktivační vrstva ReLU. Druhá konvoluční vrstva je následována vrstvou normalizace dávek, jejíž výstup je sečten se vstupem do reziduálního bloku, který je dopředně propagován pomocí reziduálních spojení.

Po dalším studiu problematiky rekonstrukce obrazu došlo ale ke změně v návrhu reziduálních bloků, protože, jak bylo zmíněno dříve v přehledu modelů pro rekonstrukci, vrstva normalizace dávek výrazně zvyšuje paměťovou náročnost modelu, a dále také normalizace hodnot získaných příznaků snižuje flexibilitu modelu, co se generovaných hodnot pixelů týče. Rozhodl jsem se tedy použít podobné bloky, které představili Nah a spol. ve své práci o odstranění dynamického šumu [39]. Tyto bloky se skládají také ze dvou konvolučních vrstev, postrádají ale vrstvy normalizace dávek. Za první konvoluční vrstvou následuje aktivační vrstva ReLU, kterou jsem ve svém modelu vyměnil za aktivační vrstvu aplikující aktivační funkci *leaky ReLU*, která má i pro záporné hodnoty vstupu malé kladné hodnoty gradientů, a tak umožňuje učení i když model pracuje se zápornými hodnotami příznaků. Nevýhodou odstranění normalizace dávek je to, že tato vrstva pomáhá stabilizovat trénování modelu, které bude nyní složitější.

Zvýšení rozlišení

Pro zvýšení rozlišení snímků existuje několik způsobů. Jedním ze základních přístupů je například dopočítání nových pixelů pomocí bikubické interpolace. Tento přístup však nevrací dostatečně kvalitní výsledky. Ve svém modelu pro zvýšení rozlišení použiji vrstvu obrácené konvoluce (angl. *transposed convolution*), někdy také nazývanou *dekonvoluční vrstva* [17]. Její hlavní výhodou je to, že stejně jako konvoluční vrstva obsahuje konvoluční jádro s trénovatelnými parametry, pomocí kterého se může učit, jaké hodnoty dosazovat na základě vstupních hodnot. Pokud u klasické konvoluční vrstvy zvolíme skok (angl. *stride*) s hodnotou 2, na výstupu dostaneme data, která mají poloviční velikost dimenzí, po kterých se pohybuje konvoluční jádro. U obrácené konvoluce je to přesně naopak, ze vstupních dat dostaneme data s dimenzí o dvojnásobné velikosti. Tento důsledek umožňuje použití obrácené konvoluce pro zvýšení rozlišení obrázku.

Celkový návrh modelu

Jako vstup bude model brát barevné obrázky se třemi barevnými kanály RGB, jejichž hodnoty budou normalizovány do intervalu $\langle 0; 1\rangle$. Než bude tento vstup zpracován skupinou na sebe navazujících reziduálních bloků, projde dvěma konvolučními vrstvami, které budou sloužit pro extrakci nízko-úrovňových příznaků, jako jsou například hrany objektů v obraze. Po první z nich bude následovat vrstva aplikující aktivační funkci *leaky ReLU*. Další využití těchto dvou vrstev bude v postupném zvýšení počtu kanálů vstupních dat na počet, se kterým budou pracovat reziduální bloky. Po těchto dvou konvolučních vrstvách přijde na řadu již zmíněné zpracování pomocí reziduálních bloků, které budou mít za úkol ze vstupních dat extrahovat vysoko-úrovňové detaily. V tomto případě jde o co nejlepší predikci detailů rekonstruovaného obličeje. Po průchodu všemi bloky bude k výstupu posledního bloku přičtena hodnota vstupu do prvního bloku (výstup druhé konvoluční vrstvy v pořadí) přivedená pomocí dopředného reziduálního spojení. Tady končí fáze extrakce příznaků a získávání dat ze vstupního snímku.

Po skupině reziduálních bloků bude následovat jedna konvoluční vrstva, která převede počet kanálů výstupu posledního reziduálního bloku na stejný počet, jaký bude použit u vrstev pro zvětšení rozlišení. Na zvětšení rozlišení budou použity dvě vrstvy obrácené konvoluce – každá pro dvojnásobné zvětšení.

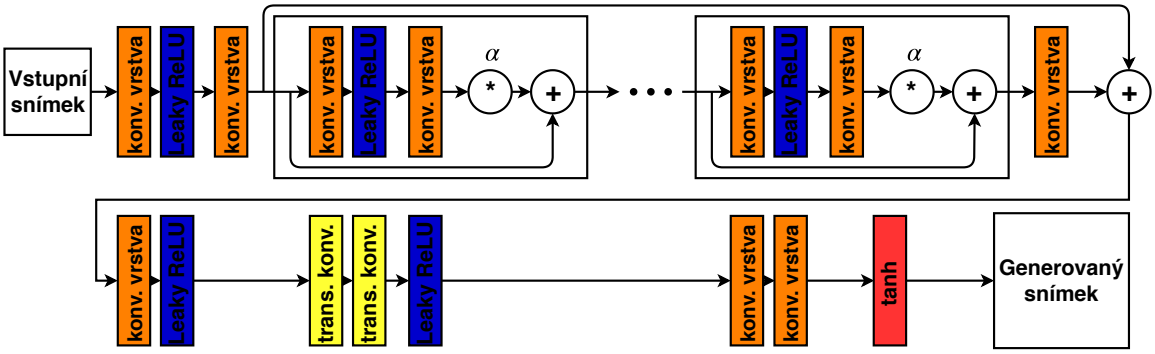
Jakmile dojde ke zvýšení rozlišení, musí se data převést zpátky na tři barevné kanály RGB. K tomuto úkolu budou v modelu sloužit poslední dvě konvoluční vrstvy, které postupně převedou počet kanálu právě na hodnotu tři. Za první z těchto dvou vrstev bude následovat aktivační vrstva *leaky ReLU*.

Jako úplně poslední vrstvu tohoto modelu jsem zvolil vrstvu aplikující aktivační funkci tangens hyperbolický, který převádí hodnoty jednotlivých pixelů do intervalu $(-1; 1)$, což je přesně rozsah hodnot, který umožňuje uložit rekonstruovaný obrázek.

Celá architektura modelu je zobrazena na obrázku.

Trénování modelu

Pro trénování modelu budou použity dvojice snímků. Jeden bude ve vysokém rozlišení a druhý bude mít rozlišení čtyřikrát nižší. Více o použitém datasetu bude uvedeno v následující kapitole o implementaci. Pro počítání chyby mezi originálními snímky a snímky generovanými navrženým modelem jsem zvolil chybovou funkci *L1*, díky které by měl model podávat lepší vizuální výsledky než při použití chybové funkce *L2*. Jako optimalizační



Obrázek 3.2: Architektura ResNet modelu tak jak je popsána v této kapitole. Jádrem modelu je skupina reziduálních bloků pro extrakci příznaků, díky kterým je model rekonstruovat snímky obličeje. Zvýšení rozlišení se provádí pomocí dvou vrstev obrácené konvoluce. Kromě reziduálních bloků jsou v modelu ještě konvoluční vrstvy hlavně pro převod počtu kanálů aktuálně zpracovávaných dat. První dvě konvoluční vrstvy slouží také pro extrakci nízko-úrovňových příznaků.

algoritmus jsem zvolil Adam, který bude minimalizovat již zmíněnou chybovou funkci *L1* pomocí dávek dat – dvojic obrázků. Velikost jedné dávky bude určena empiricky podle kapacit grafické karty a stroje, na kterém bude model trénován.

3.2 Model založený na architektuře GAN

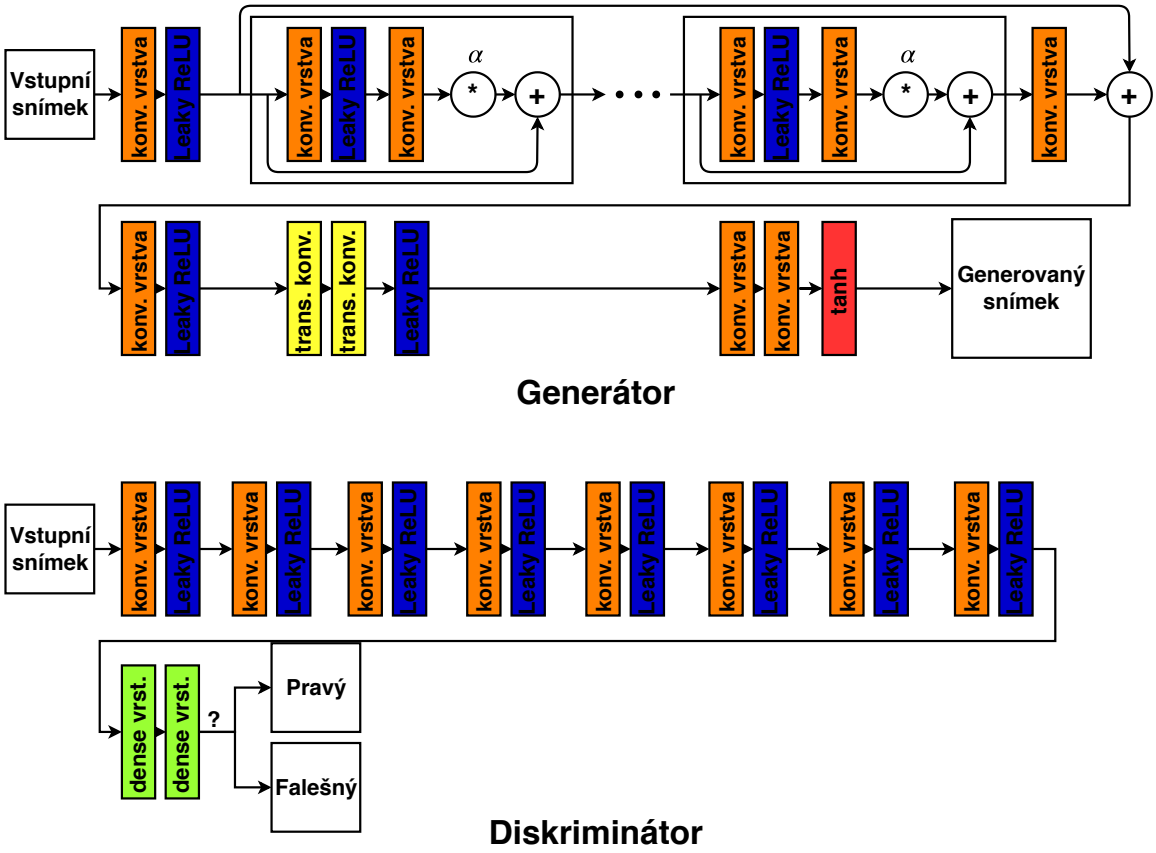
Obrovským trendem poslední doby v rekonstrukci obrazu se staly generativní kompetitivní sítě, které dokázaly velmi kvalitně rekonstruovat obrázky [47][32]. Jejich hlavní výhodou je to, že dokáží velmi přesně rekonstruovat ostré hrany a jiné detaily – například textury. Kvůli těmto úspěchům generativních kompetitivních sítí na poli rekonstrukce obrazu jsem se rozhodl zkusit navrhnout další model, který bude rekonstruovat snímky obličeje ve velmi nízkém rozlišení (řádově jednotky až desítky pixelů) na snímky o čtyřnásobné velikosti.

První část tohoto modelu, tzv. generátor, se velmi podobá modelu založeném na architektuře ResNet, který byl popsán dříve v této kapitole. Také využívá reziduálních bloků pro extrakci vysoko-úrovňových příznaků a vrstvy obrácené konvoluce pro zvýšení rozlišení. Hlavním rozdílem tohoto modelu od předchozího je způsob učení, kterým se snaží dosáhnout optimálních hodnot parametrů. Tento model využívá tzv. kompetitivního učení (angl. *adversarial training*), které spočívá v soutěžení dvou modelů neuronových sítí mezi sebou. První z nich je generátor, který generuje falešné obrázky, a snaží se je úspěšně podstrčit diskriminátoru jako originální snímky. Úkolem diskriminátoru je odhalit všechny falešné snímky a odlišit je od pravých. V této části popíšu architekturu diskriminátoru, rozdíly mezi generátorem tohoto modelu a modelem předchozím a způsob kompetitivního učení modelu.

Diskriminátor

Pro rozlišení originálních snímků od falešných budu trénovat *VGG* model diskriminátoru, který se skládá z kaskády konvolučních vrstev následovaných normalizací dávek a aktivační vrstvou Leaky ReLU. Tento model bude obsahovat sedm trojic těchto vrstev, kterým bude předcházet jedna konvoluční vrstva s aktivační vrstvou Leaky ReLU. Následovat je budou

dvě plně propojené vrstvy, které budou mít za úkol určit, zda se jedná o falzifikát nebo originální snímek, a to z příznaků, které jsou extrahovány pomocí předchozích konvolučních vrstev. Architektura diskriminátoru bude vyobrazena na obrázku 3.3



Obrázek 3.3: Architektura generátoru tohoto modelu je velmi podobná jako architektura předešlého modelu založeného na architektuře ResNet. Základem diskriminátoru je architektura VGG. Jedná se o kaskádu konvolučních a aktivačních vrstev pro extrakci příznaku z klasifikovaného snímku. Na konec řetězce vrstev jsou umístěny dvě plně propojené vrstvy, ze kterých vyjde hodnota reprezentující rozhodnutí diskriminátoru, zda je vstupní snímek klasifikován jako pravý nebo falešný.

Generátor

Jak již bylo zmíněno výše, generátor tohoto kompetitivního modelu je velmi podobný modelu založenému na architektuře ResNet. Na vstupu bere obrázek v nízkém rozlišení, který projde jednou konvoluční vrstvou pro extrakci nízko-úrovňových příznaků. Tato vrstva také převede počet kanálů na stejnou hodnotu, se kterou budou pracovat reziduální bloky tohoto generátoru. Reziduální bloky generátoru extrahují vysoko-úrovňové detaily pro dobrou rekonstrukci detailů. Po kaskádě reziduálních bloků následuje jedna konvoluční vrstva. K jejímu výstupu je pomocí dopředného spojení přičten výstup z první konvoluční vrstvy modelu. Poté následuje ještě jedna konvoluční vrstva s aktivační vrstvou Leaky ReLU. Po těchto dvou vrstvách už dochází k zvýšení rozlišení pomocí dvou vrstev obrácené kon-

voluce následovaných aktivační vrstvou Leaky ReLU. Nakonec jsou data pomocí dvou konvolučních vrstev převedena zpátky na tři kanály, které reprezentují barvy RGB. Po tomto převodu už projdou pouze vrstvou aplikující aktivační funkci tangens hyperbolický převádějící hodnoty pixelů do intervalu potřebného pro správné reprezentování jednotlivých barev.

Trénování modelu

Na první pohled je princip trénování kompetitivního modelu jednoduchý. Generátor generuje obrázky, které diskriminátor označí buď za pravé nebo falešné, a podle toho, jak a s jakou pravděpodobností se rozhodne, upravíme hodnoty vah obou částí modelu. Pro oba modely musíme definovat chybovou funkci, podle které se bude tento trénink řídit. Jako chybovou funkci pro generátor použiji tzv. *non-saturating loss* zmíněnou v sekci 2.4. Tato funkce maximalizuje $logD(G(\boldsymbol{z}))$, kde $\boldsymbol{z}$ jsou vstupní data. Předchozí výraz znamená maximalizaci faktu, že diskriminátor bude špatně klasifikovat generovaná data. Pro diskriminátor jsem vybral chybovou funkci *WGAN-GP loss* (*Wasserstein Generative Adversarial Networks with Gradient Penalty loss*) [22]. Tato chybová funkce penalizuje diskriminátor pokud se hodnota normy gradientu odchýli od hodnoty 1. Více o ní si může čtenář přečíst v práci [22]. Tato penalizace gradientu je poté přičtena ke klasické kritické chybové funkci diskriminátoru:

$$L = [E_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}(D(\boldsymbol{x})) - E_{\boldsymbol{x}^*\sim p_r(\boldsymbol{x}^*)}(D(\boldsymbol{x}))] + \lambda E_{\boldsymbol{x}} \sim p_{\boldsymbol{x}}[(||\nabla_{\boldsymbol{x}} D(\boldsymbol{x})||_2 - 1)^2], \tag{3.1}$$

kde D značí diskriminátor, $\boldsymbol{x}$ je vzorek vygenerovaný generátorem, $\boldsymbol{x}^*$ je ground-truth snímek a $\nabla_{\boldsymbol{x}}$ značí gradient pro vzorek $\boldsymbol{x}$. Tato chybová funkce se snaží minimalizovat chybu při detekci falešných vzorků, přičemž je ještě penalizována pomocí penalizace gradientu.

Kapitola 4

Implementace

Společně s návrhem modelu je jeho implementace nejdůležitější částí této práce pro dosažení cílených výsledků. Ještě před začátkem této implementace samotného modelu ale bylo potřeba implementovat skript, který vezme obrázky ve vysokém rozlišení, a vytvoří k nim odpovídající obrázky v nízkém rozlišení, pomocí kterých se bude model učit. Jako další jsem implementoval třídu, která bude načítat data a předávat je modelu v takové podobě, v jaké budou zpracovávány konvolučními a dalšími vrstvami modelu. Po dokončení těchto pomocných prvků jsem mohl začít s implementací samotného modelu. Ze zadání práce vyplynulo použití frameworku TensorFlow[1], a jako programovací jazyk jsem zvolil pro všechny části Python[2]. Veškeré výsledky, kterých bylo dosaženo pomocí implementace popsané v této kapitole, budou uvedeny v kapitole Experimenty 5.

4.1 Tensorflow

TensorFlow je kompletní open-source framework vytvořený výzkumnou skupinou Google Brain ze společnosti Google. Jeho využití je v hlubokém a strojovém učení a numerickém počítání. Jádro tohoto frameworku je napsáno v jazyce C++ s podporou speciální platformy NVIDIA CUDA[3], která umožňuje efektivní optimalizaci výpočtů na grafických kartách značky NVIDIA, což má za následek velké urychlení výpočtů [4]. Další zefektivnění výpočtů umožňuje TensorFlow díky podpoře tzv. *TPU (Tensor Processing Unit)*, které přináší hardwarovou podporu tenzorových počtů. V TensorFlow jsou všechny entity (proměnné, konstanty, i tzv. *placeholder*) reprezentovány jako tenzor, což je zobecnění vektorů a matic do vyšších dimenzí. TensorFlow pak ukládá tenzory jako n-dimenzionální pole daného datového typu [8]. Nad těmito tenzory je pak možno provádět výpočetní operace. Celkový výpočet je poskládán do grafu, jehož uzly reprezentují jednotlivé výpočetní operace. Tento graf definuje tok dat modelem.

Každý, kdo využívá TensorFlow, má na výběr z rozhraní o třech úrovních, která může použít. Na nejvyšší úrovni je aplikační programové rozhraní (dále už jen API) *Estimators API* [3], které výrazně zjednodušuje programování a poskytuje obálky na různé akce – například evaluaci nebo trénování modelu. Na nižší úrovni se nachází API modulu zvaného *layers*, které poskytuje implementaci nejčastěji používaných vrstev a funkcí. Při použití tohoto rozhraní nemusíme myslet na inicializaci parametrů jednotlivých vrstev, TensorFlow

[1] https://www.tensorflow.org/
[2] https://www.python.org/
[3] https://developer.nvidia.com/cuda-zone

to zařídí za nás. Poslední možností je použití nízkoúrovňového API. V tom případě musí uživatel myslet na všechny úkoly, které by za něj vyřešila rozhraní vyšší úrovně. Na druhou stranu, uživatel pak má úplnou svobodu a kontrolu nad tvorbou všech částí svého modelu. Dále mu toto API umožňuje tvorbu vlastních vrstev a jiných výpočetních operací, kterých by nedocílil pomocí vysokoúrovňového API.

Ve své práci budu používat převážně funkce z modulu *layers* pro implementaci potřebných funkcí reprezentujících vrstvy modelu pro rekonstrukci obrazu.

4.2 Předzpracování dat

Pro snížení rozlišení originálních snímků jsem implementoval skript `downscale.py` v jazyce Python. Pro práci s obrázky – načtení, změnu rozlišení a uložení – používám knihovnu OpenCV. Pro načtení je použita funkce `imread`, pro změnu rozlišení funkce `resize` a pro uložení nových snímků se sníženým rozlišením funkce `imwrite`.

Pro načítání datasetu, tedy dvojic snímků s nízkým a vysokým rozlišením, jsem implementoval třídu `DatasetLoader`, kterou si model bude instanciovat, a pomocí ní načítat snímky potřebné pro trénování. Tato třída bere jako parametry při instanciaci cesty k adresářům se snímky. Dále obsahuje metodu `get_all_files_path` pro získání cest ke všem souborům se snímky a nakonec metodu `load_images` pro načtení snímků pomocí funkce `imread` z knihovny OpenCV, která vrací pole knihovny NumPy ve tvaru *[velikost dávky, šířka snímku, výška snímku, počet barevných kanálů]*. Přesně s tímto tvarem pole pracují funkce reprezentující vrstvy v TensorFlow.

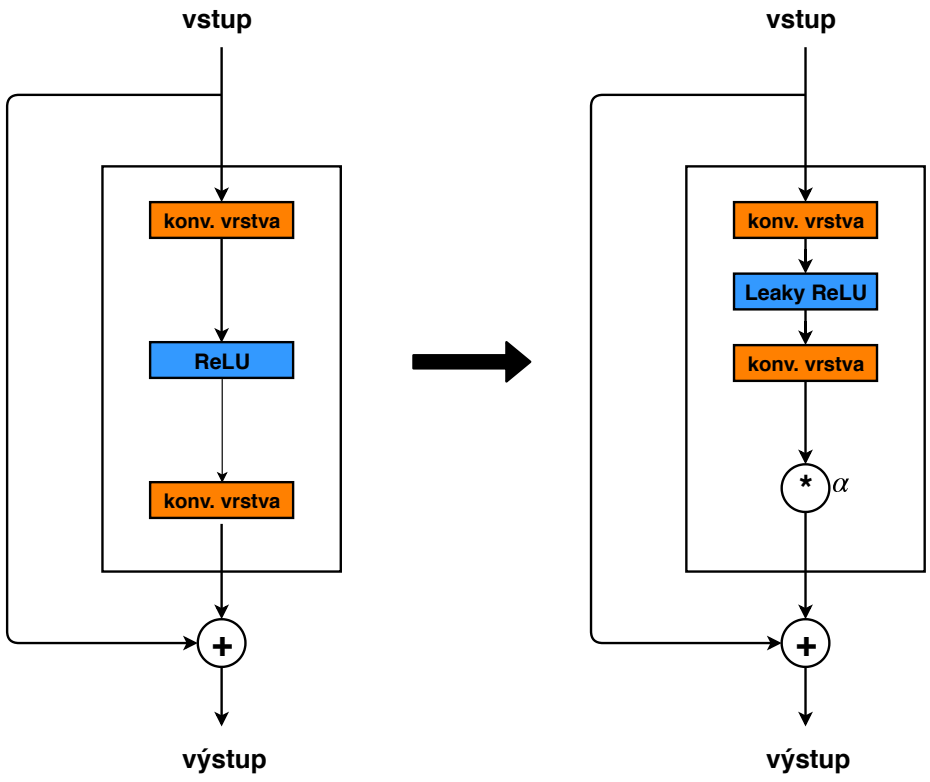
4.3 Implementace ResNet modelu

Základem implementace modelu založeného na architektuře Resnet je sada funkcí pro vytvoření vrstev a dalších struktur daného modelu. Jedná se hlavně o funkce `conv2d`, `leaky_relu` a `upscale2x_conv2d`. První v pořadí vytvoří konvoluční vrstvu s danými parametry, jako například velikost konvolučního jádra nebo počet filtrů. Vytvoření konvoluční vrstvy je vykonáno zavoláním funkce `conv2d` z TensorFlow modulu *layers*. Tato funkce přijímá parametry pro nastavení vstupu, který bude zpracovávat, počet filtrů konvoluční vrstvy, velikost konvolučního jádra, krok, se kterým se bude pohybovat dané konvoluční jádro po zvolených dimenzích snímku, nebo třeba přepínač s boolean hodnotou, zda je volání funkce součástí tréninku nebo evaluace. Funkce `upscale2x_conv2d` je z hlediska rozhraní velmi podobná funkci `conv2d`, protože implementuje vrstvu obrácené konvoluce pro zvětšení rozlišení snímku. Tato funkce se sice dívá na problém konvoluce z druhého konce, ale pracuje se stejnými parametry jako vrstva konvoluční. Tato funkce je také implementována pomocí funkce z modulu *layers*. Konkrétně se jedná o funkci `conv2d_transpose`. Další z elementárních vrstev je použitých v modelu vrstva aplikující aktivační funkci *leaky ReLU*, která je implementována funkcí `leaky_relu`. Tato funkce také využívá funkci z TensorFlow, nyní se ale jedná o funkci z nízko-úrovňového rozhraní. Daná funkce se nazývá `nn.leaky_relu`.

Z těchto elementárních funkcí jsou dále skládány reziduální bloky. Jeden takový blok je implementován funkcí `resnet_block`. Jako parametry tato funkce přijímá vstupní data, poté počet filtrů, se kterými budou pracovat konvoluční vrstvy, rozměry konvolučního jádra a boolean hodnotu, zda se jedná o trénink nebo ne.

Při implementaci došlo ke změně reziduálních bloků oproti návrhu. V každém reziduálním bloku je zapotřebí použít tzv. reziduální škálování (angl. *residual scaling*), což je vlastně

konstanta, kterou se násobí výstup z poslední konvoluční vrstvy v daném bloku [44]. Pokud je použito velké množství filtrů napříč modelem, může se stát, že trénování začne být nestabilní a po pár tisících iterací tréninku začne model generovat pouze samé nuly. V mém případě černé snímky. Tomuto jevu právě předchází zmíněné reziduální škálování, které sníží hodnoty výstupu poslední vrstvy předtím, než jsou přičteny k výstupu předchozího reziduálního bloku.



Obrázek 4.1: Na tomto obrázku je zobrazena změna reziduálního bloku pro model založený na architektuře ResNet. Když je použito mnoho filtrů napříč modelem, může se stát, že se trénování stane nestabilním. V takovém případě začne model generovat pouze černé snímky. Přidáním tzv. *reziduálního škálování* lze této situaci zabránit a stabilizovat trénování modelu. Jedná se o vynásobení výstupu poslední konvoluční vrstvy předem určenou konstantou α většinou z intervalu $(0; 1)$.

Z hlediska samotné rekonstrukce je nejdůležitější funkce `double_scale_4x`, která vezme na vstupu snímek se třemi barevnými kanály a na výstupu vrátí také RGB snímek s čtyřikrát zvětšeným rozlišením. V této funkci je implementována struktura navrženého modelu. Na začátku se nachází dvě konvoluční vrstvy, implementované funkcemi `conv2d`, mezi kterými je aktivační vrstva `leaky_relu`. Tyto 2 konvoluční vrstvy extrahují nízko-úrovňové příznaky a zvýšení počtu kanálů dat na hodnotu, se kterou budou pracovat reziduální bloky. První z vrstev bude pracovat s 64 filtry a druhá, stejně jako reziduální bloky, s 128. Po těchto úvodních vrstvách bude následovat 32 reziduálních bloků, které jsou reprezentovány funkcí `resnet_block`, která se opakovaně volá uvnitř těla cyklu `for`. Jakmile dojde k extrakci vysoko-úrovňových příznaků pomocí reziduálních bloků, následuje jedna konvoluční vrstva, která převede počet kanálů zpracovávaných dat na počet, se kterým budou pracovat vrstvy

pro zvýšení rozlišení. Konkrétně se jedná o převod ze 128 kanálů na 256. Předposlední částí funkce pro rekonstrukci jsou dvě vrstvy obrácené konvoluce pro zvýšení rozlišení. Ty jsou implementovány funkcí `upscale2x_conv2d`, která tedy pracuje s 256 filtry, a s krokem 2, se kterým prochází obrázek jak co se výšky, tak šířky týče. Díky kroku s hodnotou 2 získáme jako výstup každé vrstvy obrácené konvoluce snímek s dvakrát zvětšeným rozlišením. Díky zřetězení právě dvou těchto vrstev dostáváme snímek čtyřikrát zvětšený. V poslední části se nachází dvakrát volání funkce `conv2d`, které reprezentují poslední dvě konvoluční vrstvy. Tyto vrstvy mají za úkol snížit počet kanálů zpátky na tři barevné kanály RGB. První z této dvojice sníží počet kanálů z 256 na 128 a druhá z 128 na 3. Výstupní vrstvou modelu je aktivační vrstva aplikující funkci tangens hyperbolický, která normalizuje hodnoty do intervalu $(-1; 1)$ a je implementována voláním funkce z nízkoúrovňového API frameworku TensorFlow se jménem `nn.tanh`.

Implementace trénování tohoto modelu je obsažena ve funkci `train`. Na jejím začátku jsou vytvořeny entity zvané *placeholders*, které v TensorFlow slouží jako zástupné prvky, které budou později, při spuštění programu, nahrazeny reálnými daty. Poté je inicializována proměnná reprezentující koeficient učení, jejíž počáteční hodnota je nastavena na 10^{-4}, a v průběhu učení je ještě několikrát snížena vždy o jeden řád dolů. Další důležitou částí implementace trénování je definice optimalizačního algoritmu a chybové funkce. Chybová funkce *L1* použitá v modelu je implementována ve funkci `L1_loss`. Tato funkce vezme vstupní obrázky, které nechá rekonstruovat pomocí funkce `double_scale_4x`, a rekonstruované obrázky poté porovná s originálními snímky pomocí chybové funkce *L1*. Jako optimalizační algoritmus byl zvolen Adam, který je v implementaci reprezentován instancí třídy `AdamOptimizer` z modulu `train` frameworku TensorFlow. Při instanciaci je mu předán koeficient učení a je zrovna zavolána jeho metoda `minimize` s parametrem odkazujícím na chybovou funkci *L1*. Dále je ve funkci `train` implementováno načítání obrázků z datasetu pomocí třídy `DatasetLoader`. V rámci úspory paměti se načítají jednotlivé dávky dat zvlášť, právě když s nimi bude potřebovat model pracovat. V poslední části je vytvořena relace v TensorFlow pomocí instance třídy `Session`, díky které můžeme následně spustit optimalizační algoritmus pro minimalizaci chybové funkce, které místo dříve zmíněných placeholderů předáme reálna data z datasetu. Tento algoritmus se spouští zevnitř těla dvou zanořených cyklů, kde první cyklus představuje počet epoch, určujících délku tréninku, a druhý, který zajišťuje průchod přes všechna data z datasetu v rámci dané epochy. Poče epoch není stanovený napevno, protože trénování tohoto modelu zabírá dlouhou dobu – v řádech dní – a velikost jedné dávky se pohybuje od 15 do 25 snímků, podle možností stroje, na kterém je model trénován.

4.4 Implementace GAN modelu

Základem implementace tohoto modelu jsou elementární funkce, které vytváří jednotlivé vrstvy modelu. Tyto funkce jsou stejné jako pro ResNet model a jsou popsány v sekci 4.3.

Generátor tohoto modelu byl implementován velmi podobně jako model založený na architektuře ResNet. První konvoluční vrstva převede vstup na 64 kanálů a extrahuje nízkoúrovňové příznaky. Jednotlivé reziduální bloky jsou reprezentovány funkcí `resnet_block`, která se volá v cyklu typu `for` třicet-dvakrát. Samotné zvýšení rozlišení je implemetováno jako v předchozím modelu pomocí funkce `uspcale2x_conv2d`, která reprezentuje vrstvu obrácené konvoluce.

Diskriminátor, který je postaven na architektuře VGG, je implementován kaskádou funkcí `conv2d` a `leaky_relu` a na konec jsou umístěny dvě funkce `dense`, které reprezentují

plně propojené vrstvy. Počet kanálů vrstev roste od 64 až po 1024, kdy vždycky dvojice bloků [`conv2d`, `leaky_relu`] má stejný počet filtrů a po nich následující dvojice bloků má filtrů dvakrát více. Poslední dvojice `conv2d` a `leaky_relu` mají 512 filtrů. První z funkcí `dense` pracuje s 1024 filtry a druhá pouze s jedním, kdy vrací hodnotu, zda se jedná o reálný nebo falešný snímek.

Chybové funkce jsou implementování ve funkcích `G_nonsaturating_loss` a `D_loss_gp`. První jmenovaná se snaží maximalizovat pravděpodobnost, že generátor oklame diskriminátor, a druhá se snaží minimalizovat chybu klasifikace diskriminátoru.

Trénování modelu je ale poněkud odlišné než u předchozího modelu. Zaprvé jsem musel nadefinovat dva optimalizační algoritmy Adam, které jsou reprezentovány instancí třídy `AdamOptimizer`. Pomocí těchto instancí a jejich metody `minimize` pak definovat operace minimalizování chybových funkcí pro generátor a diskriminátor `g_loss` a `d_loss`. Hlavní částí implementace trénování je vytvoření relace v TensorFlow pomocí třídy `tf.Session`, pomocí které budeme moci spustit trénování modelu. uvnitř této relace jsou definovány dva cykly typu `for`. První reprezentuje počet epoch tréninku a druhý průchod přes všechna data po jednotlivých dávkách. Uvnitř těla druhého cyklu se pomocí metody `run` instance třídy `tf.Session` spouští trénink pro diskriminátor a generátor zvlášť. Počet epoch a velikost jedné dávky bude určen na základě možností stroje, na kterém bude probíhat trénink modelu.

Změna v návrhu trénování

Během implementace modelu musela být provedena změna návrhu trénování – konkrétně chybové funkce generátoru – protože proces učení vždycky skončil tzv. *kolapsem modelu* (angl. *mode collapse*). Tento jev má původ v samotné *non-saturating loss*, která se snaží maximalizovat pravděpodobnost, že diskriminátor špatně klasifikuje falešný vzorek jako pravý. Fakt, že tento proces maximalizuje chybu diskriminátoru, má za následek situaci, kdy se generátor dostane do bodu, ve kterém dostává velkou pravděpodobnost chyby diskriminátoru. To je pro něj příznivý stav, a tak nemá důvod pohnout se z tohoto bodu při trénování dále. Nakonec to skončí tak, že generátor přestane úplně brát v potaz vstupní snímky, které má rekonstruovat, a začne vracet pro odlišné vstupy vždycky stejný výstup, pro který je vysoká pravděpodobnost chyby diskriminátoru.

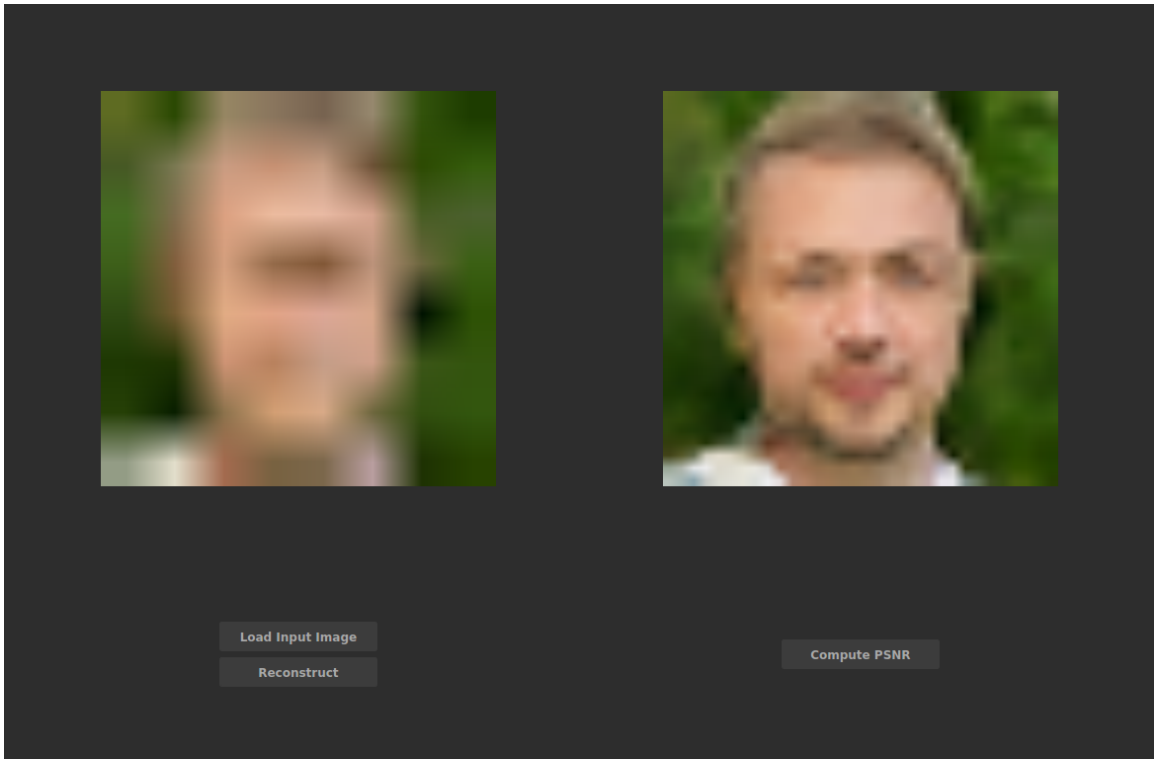
Protože chybová funkce generátoru obsahovala pouze maximalizaci chyby diskriminátoru, nastával výše zmíněný kolaps. Řešením této situace bylo přidání dalšího prvku do chybové funkce, který se bude soustředit i na to, jak moc se výsledný falešný snímek podobá originálnímu ground-truth snímku. Výsledná chybová funkce je tedy složena z funkce maximalizace chyby diskriminátoru a chybové funkce *L1*, která je použita i v ResNet modelu, a která bude mít za úkol zajistit, že finální rekonstruované snímky jsou generovány v závislosti na jejich vstupu. Chybová funkce *L1* je implemetována funkcí `L1_loss`.

4.5 Implementace grafického uživatelského rozhraní

Pro implementaci jednoduchého grafického uživatelského rozhraní jsem si vybral framework PyQt[4]. Tato knihovna poskytuje programové aplikační rozhraní frameworku Qt, který je primárně pro jazyk C++, v Pythonu. Pro návrh uživatelského rozhraní jsem použil program *Qt Designer*, který poskytuje jednoduché prostředí pro rychlý návrh a tvorbu GUI.

[4] https://www.qt.io/qt-for-python

Základem GUI je prvek `QMainWindow`, který reprezentuje okno aplikace. Toto okno se otevře okamžitě po spuštění programu. Dalším prvkem, na kterém je celé uživatelské rozhraní postaveno, je `QStackedWidget`, který umožňuje jednoduché přepínání obrazovek. Tato aplikace obsahuje pouze dvě obrazovky. Na první probíhá samotná rekonstrukce obrazu. Jsou na ní dva prvky `QWidget`, obsahující další prvek `QLabel`, ve kterém budou vykresleny snímky, se kterýma bude program pracovat. První snímek bude vstupní a druhý bude zrekonstruovaný. První snímek bude muset uživatel načíst. Po načtení snímku a stisknutí tlačítka *Reconstruct* dojde ke spuštění procesu rekonstrukce, který když skončí, dojde k vykreslení rekonstruovaného snímku v druhém prvku `QLabel`. Po rekonstrukci snímku bude možné spočítat odchylku originálního snímku od rekonstruovaného. Tento výpočet bude probíhat na druhé stránce aplikace, na kterou se přejde pomocí tlačítka *Compute PSNR*. Po přechodu na tuto stránku bude zobrazen rekonstruovaný snímek. Druhý snímek, originální, bude muset uživatel načíst do aplikace sám. Po načtení tohoto snímku může uživatel vypočítat odchylku pomocí tlačítka *Compute PSNR*. Výsledná hodnota bude zobrazena v text boxu pod tímto tlačítkem. Princip výpočtu této odchylky bude popsán v následující kapitole Experimenty 5 v sekci 5.2.



Obrázek 4.2: Snímek jedné z obrazovek grafického uživatelského rozhraní. Na první obrazovce uživatel načte vstupní snímek s nízkým rozlišením (vlevo) a pomocí tlačítka „Reconstruct” ho může rekonstruovat na snímek s čtyřikrát větším rozlišením.

Kapitola 5

Experimenty

V této kapitole budou popsány nejzajímavější experimenty, které jsem s mými modely umělých neuronových sítí prováděl, a pomocí kterých jsem testoval jejich úspěšnost. Při experimentování jsem pracoval se snímky o rozlišení 8×8 pixelů a 16×16 pixelů. Těmto snímkům bylo pomocí modelů zvýšeno rozlišení čtyřikrát. Pro určení odchylky originálních a rekonstruovaných snímků byla použita metrika PSNR, která je popsána v této kapitole 5.2. Dále v této kapitole popíšu porovnání kvality rekonstrukce jednotlivých modelů. Veškeré vstupní snímky uvedené jako součást obrázků v této kapitole byly náhodně vybrány z validační části datasetu. V příloze A je uvedeno pár dalších rekonstruovaných snímků ke každému modelu.

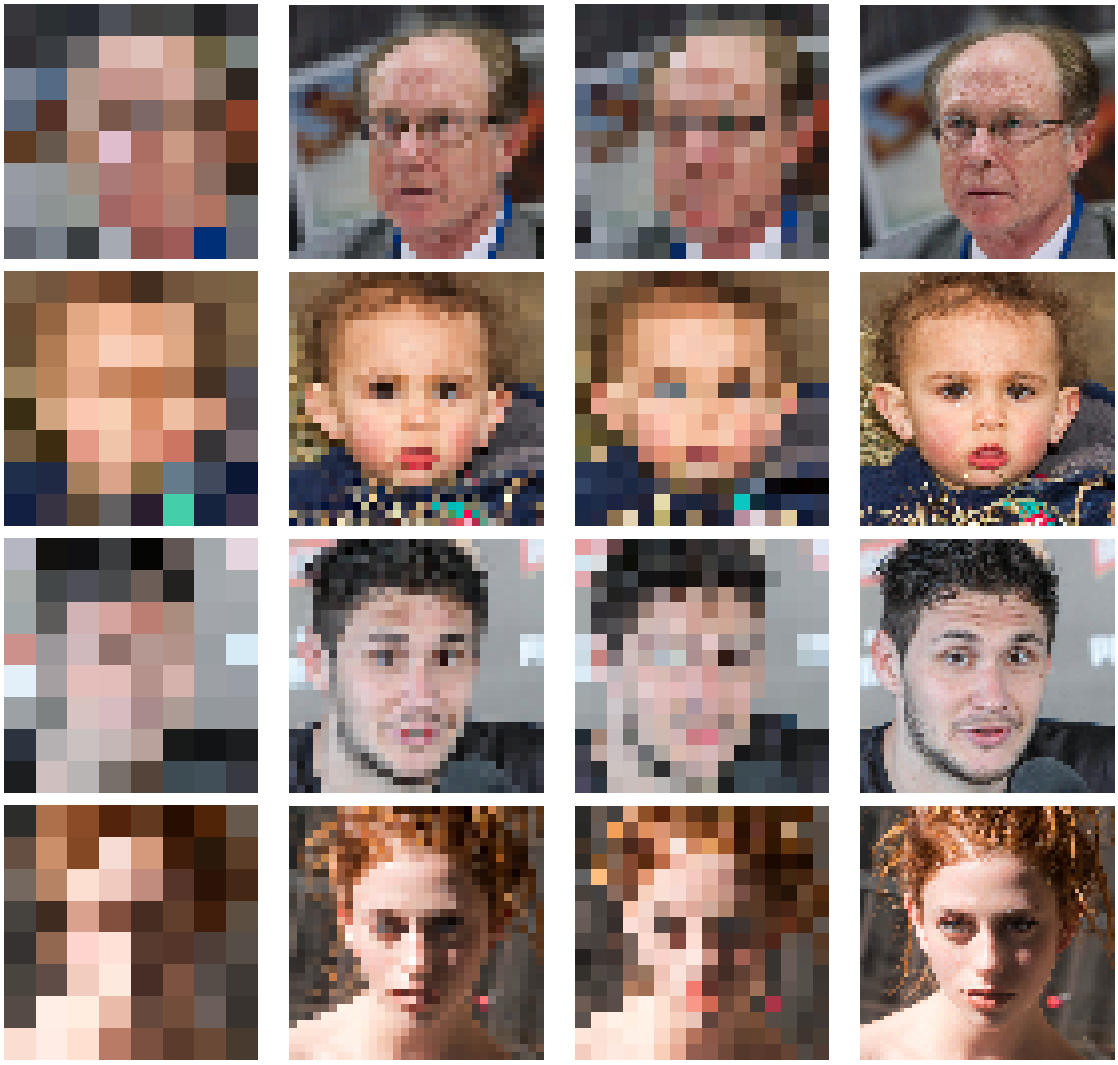
5.1 Dataset

Pro trénování modelů jsem použil již existující dataset se snímky obličejů nazvaný *Flicker-Faces-HQ (FFHQ)*, který představili jeho autoři jako součást práce zaměřené na generování snímků obličeje pomocí architektury generativních kompetitivních sítí [28]. Jelikož jsou snímky v tomto datasetu v rozlišení 1024×1024 pixelů, které je pro tuto bakalářskou práci zbytečně vysoké, uměle snížím mnou používané snímky na menší rozlišení, a teprve ty použiji jako tzv. *ground-truth* pro svůj model.

Jako ground-truth budu používat snímky s rozlišením 32×32 a 64×64 pixelů a k nim vytvořím odpovídající snímky do dvojice s rozlišením 8×8 a 16×16 pixelů, které se budu snažit rekonstruovat na původní, čtyřikrát větší rozlišení.

Jedná se o *in-the-wild* snímky, což znamená, že nejsou pořizovány za speciálních okolností určených pro pořizování snímků pro řešení konkrétního problému. Každý snímek obsahuje obličej jinak natočený, osvětlený a s jiným pozadím. Tento dataset jsem vybral, protože jsem chtěl pracovat se snímky z reálného prostředí, jelikož snímek z bezpečnostní kamery bude také pokaždé jinak natočený a jinak osvětlený.

Příklady snímků z datasetu jsou uvedeny na obrázku 5.1.



(a) 8×8 (b) 32×32 (c) 16×16 (d) 64×64

Obrázek 5.1: Ukázka snímků z datasetu. Snímky jsou rozděleny po dvojicích do dvou skupin. První skupinu reprezentují snímky (a) a (b), které slouží pro trénování rekonstrukce snímku s rozlišením 8×8 pixelů na snímek s rozlišením 32×32 pixelů. Do druhé skupiny patří snímky (c) a (d), na kterých se model trénuje pro zvýšení rozlišení z 16×16 pixelů na 64×64 pixelů.

5.2 Výpočet odchylky originálních a rekonstruovaných snímků

Pro výpočet odchylky mezi snímky originálními a rekonstruovanými jsem si vybral metriku zvanou *PSNR* (*Peak signal-to-niose ratio*) [25], což je standardně používaná metrika na poli rekonstrukce obrazu. PSNR je definováno pomocí střední kvadratické odchylky MSE takto:

$$PSNR = 10 * log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 * log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right), \tag{5.1}$$

kde MAX je maximální hodnota pixelu (v tomto případě 255 pro 8 bitů na kanál) a MSE je střední kvadratická odchylka originálního snímku od rekonstruovaného [25].

Z rovnice 5.1 vyplývá, že čím víc jsou snímky odlišné, tím je $PSNR$ nižší hodnota. Pokud jsou snímky totožné, je MSE rovno 0 a $PSNR$ není definované.

Přestože je metrika *PSNR* běžně používaná v oblasni rekonstrukce obrazu, její maximalizace ještě nemusí znamenat vizuálně kvalitní a věrohodné snímky. Tuto skutečnost můžeme pozorovat u jednoho z níže uvedených experimentů na obrázku 5.8. Snímky z modelu GAN jsou vizuálně značně nekvalitní, přesto mají vyšší hodnotu *PSNR* než snímky rekonstruované modelem ResNet.

Výše uvedenou rovnici 5.1 jsem implementoval pomocí funkce `compare_psnr` z modulu `image` patřící pod balíček algoritmů SciKit-Image[1].

5.3 Prostředí experimentů

Tato práce vznikla za podpory projektů CERIT Scientific Cloud (LM2015085) a CESNET (LM2015042) financovaných z programu MŠMT Projekty velkých infrastruktur pro VaVaI. Veškeré experimenty byly uskutečněny na výpočetních zařízeních, které jsou součástí české Národní Gridové Infrastruktury Metacentrum[2]. Díky těmto zařízením jsem mohl provádět experimenty se svými modely, které mají vysoké výpočetní i paměťové požadavky.

Jednotlivé experimenty byly v prostředí Metacentra spouštěny pomocí tzv. *dávkových skriptů*. V tomto skriptu je definováno jaké programy chce uživatel spustit, jaká k tomu bude využívat data, a jaké technologie k tomu bude používat. V této bakalářské práci je využit hlavně framework TensorFlow v programovacím jazyce Python. Bohužel, na zařízeních Metacentra nešla rozběhnout verze TensorFlow 1.13.1. Tato verze nemohla být použita kvůli konfliktům s verzemi jiných modulů, na kterých je závislá (například modul NumPy). Kvůli tomuto problému musela být použita verze TensorFlow 1.7.1. I navzdory starší verzi frameworku TensorFlow, Metacentrum poskytlo velmi dobré zázemí pro vykonávání experimentů, a to hlavně díky možnosti akcelerace trénování pomocí grafický karet a technologie CUDA, bez které by samotný trénink nebyl možný. Výhodou Metacentra je dále možnost spuštění tréninku modelů až na 24 hodin v kuse, což jsem při několikadenním tréninku modelů ocenil.

5.4 Experimenty s ResNet modelem

Nejdříve jsem prováděl experimenty s modelem založeným na architektuře ResNet, jehož návrh (3.1) a implementace (4.3) byly popsány dříve v této práci. Mezi základní experimenty jsem zařadil rekonstrukci snímků s rozlišením 8×8 pixelů na snímky 32×32 pixelů a z 16×16 pixelů na 64×64 pixelů. Výsledkem těchto experimentů by měly být snímky obličejů s věrohodně rekonstruovanými detaily a parciálními částmi obličeje.

Nastavení parametrů

Pro trénování modelu bylo potřeba nastavit několik parametrů, které na tento proces mají největší vliv. Pro optimalizaci parametrů byl již ve fázi návrhu vybrán optimalizátor Adam.

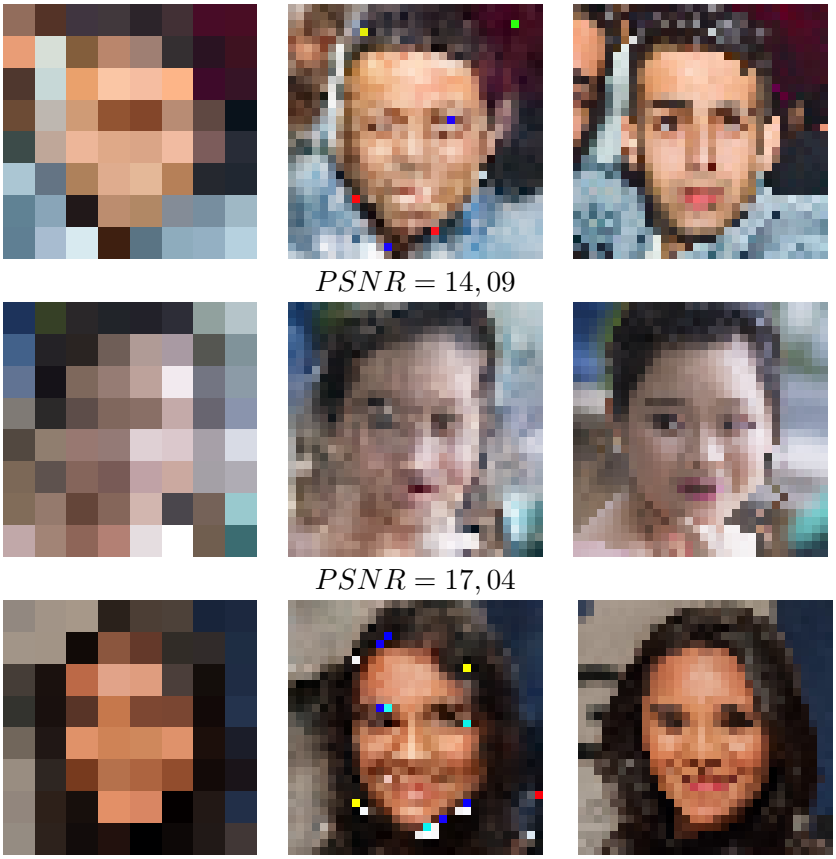
[1] https://scikit-image.org/
[2] https://metavo.metacentrum.cz/

Velikost jedné dávky, se kterou bude tento optimalizátor pracovat, byla stanovena na 20 snímků. Dalším parametrem je počet epoch určující délku tréninku. Tento parametr nebyl stanoven na pevnou hodnotu, protože trénink bude zabírat dobu v řádech dní. Velikost datasetu je 15 000 obrázků v nízkém rozlišení a k nim korespondující obrázky ve vysokém rozlišení. Učící koeficient je inicializován na hodnotu 10^{-4}.

Rekonstrukce snímků s rozlišením 8×8 pixelů

Úplně první experiment se týkal čtyřnásobného zvětšení snímků s rozlišením 8×8 pixelů na snímky o velikosti 32×32 pixelů. V rané fázi tréninku – po jednom dni učení – nedosahoval model uspokojivých výsledků. Pár příkladů je možno vidět na obrázku 5.2. Na tomto obrázku je vidět, že model už umí ve většině případů rekonstruovat tvar obličeje a jeho obrysy. Po jednom dni tréninku ale model ještě nedokáže dobře rekonstruovat detaily v obličeji.

V další fázi tréninku se model už značným způsobem v rekonstrukci nezlepšil. Na obrázku 5.2 je vidět, že snímky, které jsou rekonstruovány až po dotrénování modelu, nejsou o moc lépe zrekonstruované. Důležité části obličeje jsou stále rozmazané. Model se navíc nedokáže vypořádat se špatným osvětlením, neobvyklým natočením hlavy nebo cizími předměty v obličeji (např. sluneční brýle). Další obrázky jsou umístěny v příloze A na obrázku A.1.

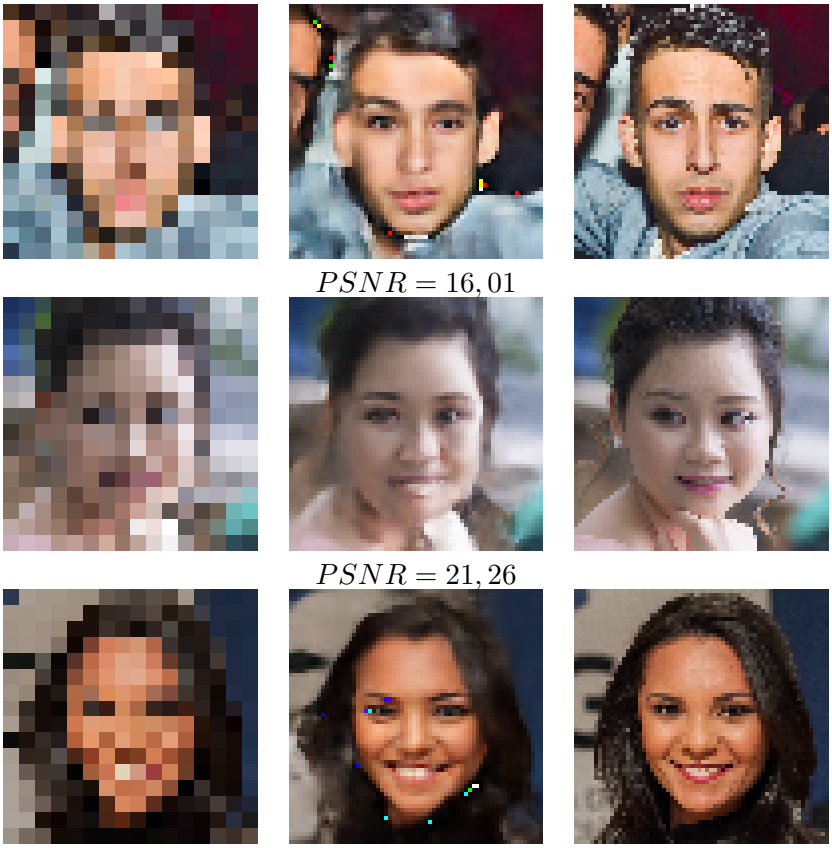


(a) (b) $PSNR = 15,82$ (c)

Obrázek 5.2: Příklady rekonstruovaných snímků po dotrénování modelu. (a) jsou vstupní snímky se sníženým rozlišením, (b) jsou rekonstruované snímky po dotrénování modelu a (c) jsou originální snímky. Hodnoty $PSNR$ jsou uvedeny v jednotkách dB (decibel).

Rekonstrukce snímků s rozlišením 16×16 pixelů

Další zkouškou bylo použití stejného modelu jako v předchozím experimentu na rekonstrukci snímků s vyšším vstupním rozlišením. Jako toto rozlišení jsem zvolil 16×16 pixelů, které bude modelem zvýšeno na 64×64 pixelů. Jelikož vstupní snímky obsahují více informace než u předchozího experimentu, dokáže model už po jednom dni učení částečně rekonstruovat i detaily v obličeji jednotlivých osob. I přes to ale není kvalita rekonstruovaných snímků dostačující, protože obličeje vypadají umělé a nepřirozené. Na obrázku 5.3 uvádím opět pár příkladů. Jedná se o stejné obličeje jako na obrázku 5.2. Po dotrénování modelu nejsou mezi výsledky významné rozdíly, stejně jako u předchozího experimentu. Pět dalších snímků rekonstruovaných tímto modelem je umístěno v příloze A na obrázku A.2.



(a) (b) $PSNR = 20,12$ (c)

Obrázek 5.3: Příklady rekonstruovaných snímků po dotrénování modelu. (a) jsou vstupní snímky se sníženým rozlišením, (b) jsou rekonstruované snímky po dotrénování modelu a (c) jsou originální snímky. Hodnoty $PSNR$ jsou uvedeny v jednotkách dB (decibel).

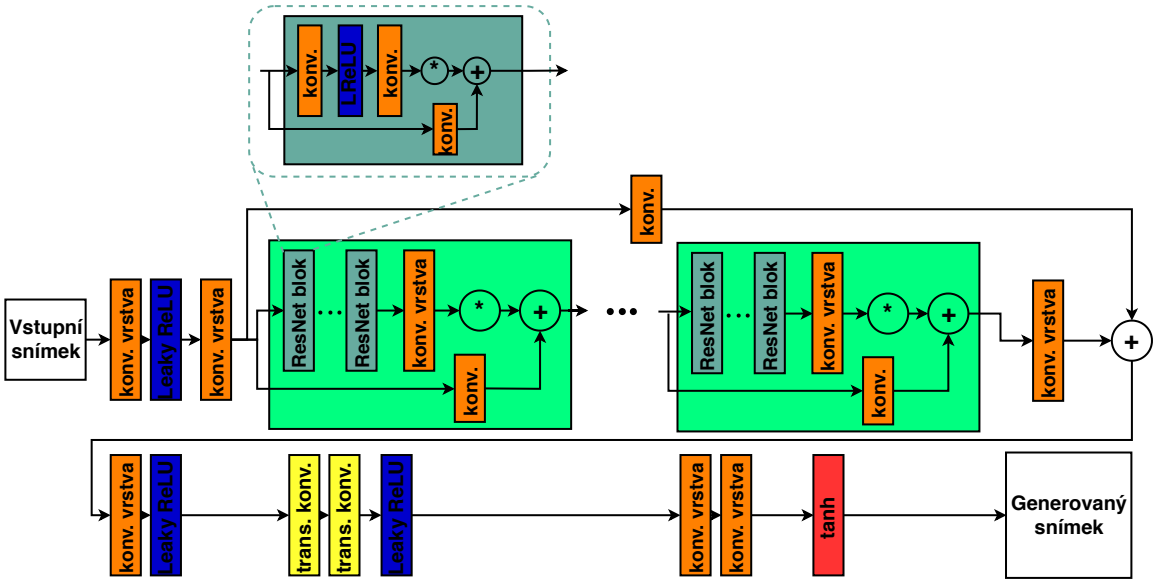
Změna v návrhu modelu

V předchozích experimentech jsem zjistil, že model založený na architektuře ResNet, který má 32 reziduálních bloků, dokáže spolehlivě rekonstruovat obrys obličeje. Tento model ale nezvládá věrohodnou rekonstrukci detailů v obličeji u vstupních obrázků s rozlišením 8×8. V tomto experimentu jsem chtěl zjistit, zda zvýšením počtu reziduálních bloků a prohlou-

bením modelu se docílí také větší schopnosti rekonstrukce detailů v obličeji, a zároveň jestli se toto prohloubení vyplatí vzhledem k velké paměťové náročnosti modelu.

K prohloubení modelu jsem vytvořil reziduální skupiny obsahující na sebe navazující reziduální bloky. V tomto modelu jsou využity tři typy dopředných spojení. První slouží pro propagaci vstupních hodnot do první reziduální skupiny až za výstup poslední reziduální skupiny. Druhý typ, který se nachází uvnitř každé reziduální skupiny, propaguje vstup do dané skupiny za výstup posledního reziduální bloku ve skupině. Poslední typem je reziduální spojení, které se nachází uvnitř každého reziduálního bloku. Toto spojení propaguje vstupní hodnoty první konvoluční vrstvy až za výstup poslední konvoluční vrstvy v daném bloku. Hodnoty, propagované pomocí těchto typů reziduálních spojení, jsou před přičtením k odpovídajícím výstupním hodnotám zpracovány jednou konvoluční vrstvou, která má konvoluční jádro o velikosti 1×1 a odpovídající počet filtrů, který musí být stejný jako počet kanálu výstupních dat, ke kterým bude propagovaná hodnota přičtena. Tyto tři typy reziduálních spojení mi umožnily výrazné prohloubení modelu na více než dvojnásobný počet vrstev.

Vstupní a výstupní vrstvy zůstávají stejné jako u předchozího modelu. Stejné jsou i vrstvy pro zvýšení rozlišení. Dokonce i struktura reziduálních bloků zůstavá stejná. Jediná změna je ve struktuře a spojení reziduálních bloků. Nový návrh modelu je zobrazen na obrázku 5.4.



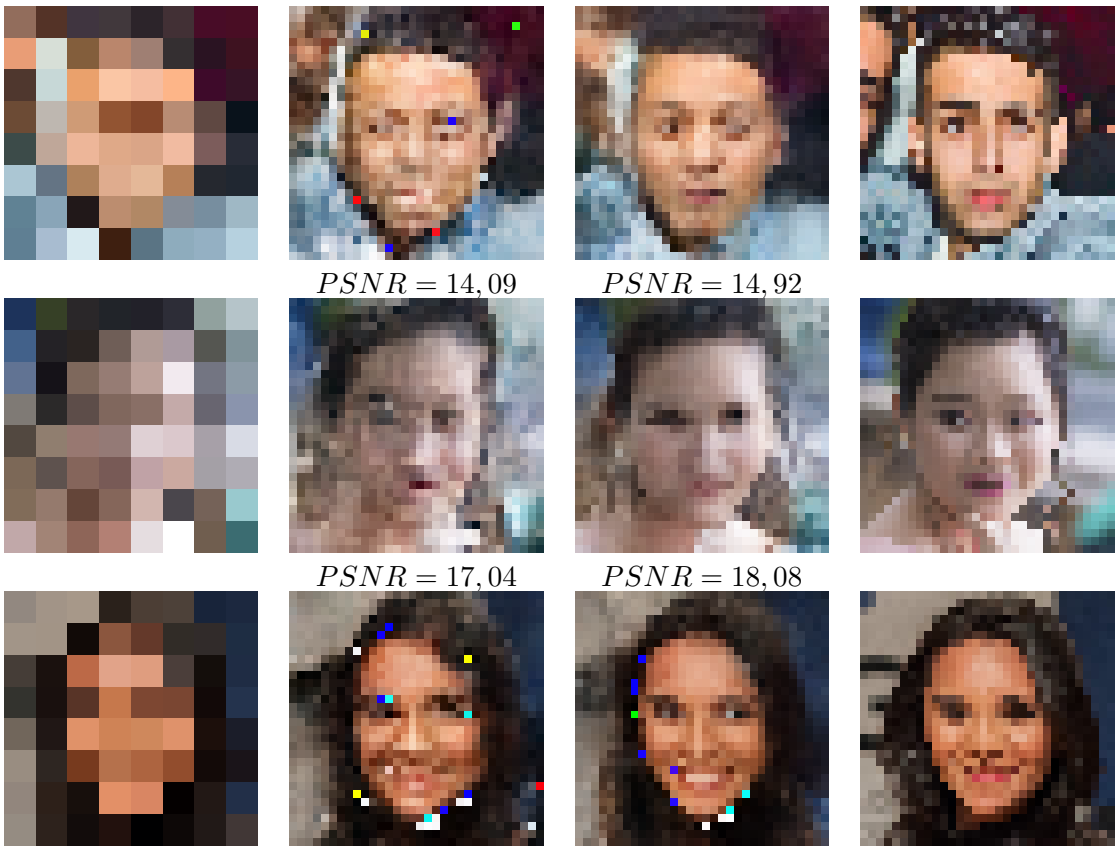
Obrázek 5.4: Nový návrh modelu s architekturou ResNet. Od původního návrhu se liší použitím reziduálních skupin, které obsahují několik reziduálních bloků. Využití těchto reziduálních skupin umožňuje výrazné prohloubení modelu.

Parametry učení zůstali stejné jako u předchozích experimentů se starším modelem. V experimentech jsem použil model s 20 reziduálními skupinami, kde každá z nich obsahuje 4 reziduální bloky. Celkový počet reziduálních bloků je tedy 80, což je více než dvojnásobný nárůst oproti 32 blokům v předchozím modelu.

Rekonstrukce snímků s rozlišením 8×8 pixelů pomocí nového modelu

Hlavním cílem experimentů s novým modelem bylo zjistit, zda prohloubení modelu zajistí kvalitnější rekonstrukci detailů v obličeji. Důležitým faktem bude, zda bude rekonstrukce natolik lepší, že se vyplatí tak velké zvýšení paměťové náročnosti modelu.

Na obrázku 5.5 můžeme vidět, že snímky, které byly pořízeny evaluací nového modelu během jeho tréninku, jsou rekonstruovány kvalitněji. Oproti původnímu modelu jsou snímky více vyhlazené, ale pořád nejsou dostatečně kvalitní po vizuální stránce. Na některých snímcích jsou oproti originálům rozmazány základní části obličeje – oči, nos, ústa. Na obrázku 5.5 jsou porovnány výstupní snímky z původního modelu a z modelu s pozměněným uspořádáním reziduálních bloků.V příloze A jsou na obrázků A.3 další výsledky získané pomocí tohoto upraveného modelu.

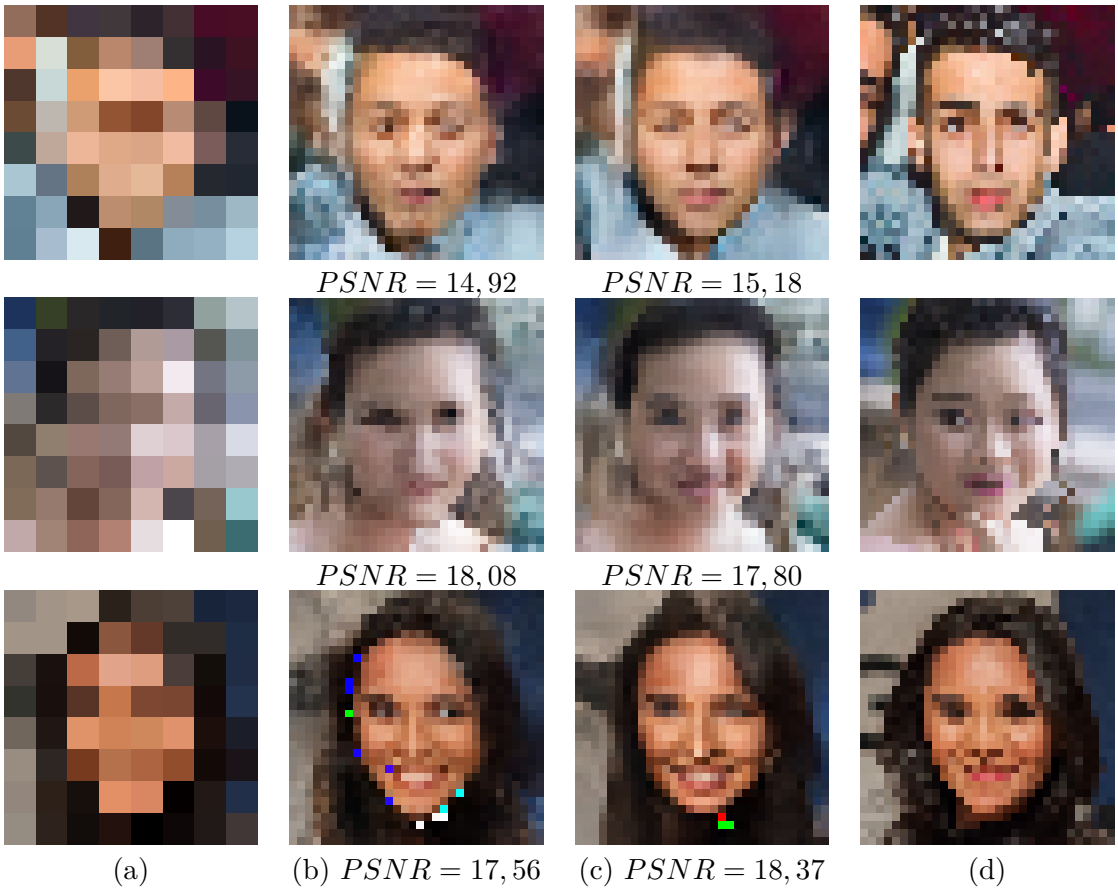


(a) (b) $PSNR = 15,82$ (c) $PSNR = 17,56$ (d)

Obrázek 5.5: Porovnání snímků ze starého modelu se snímky z modelu s pozměněným propojením reziduálních bloků. (a) jsou vstupní snímky se sníženým rozlišením, (b) jsou snímky z původního modelu, (c) jsou z snímky z nového modelu s pozměněným uspořádáním reziduálních bloků získané během tréninku a (d) jsou originální snímky. Hodnoty $PSNR$ jsou uvedeny v jednotkách dB (decibel).

Po dotrénování modelu však nastal problém. Jelikož má model velké množství parametrů, začal se přetrénovávat na trénovací data a došlo k tzv. *overfittingu*. Kvůli tomuto problému jsem se pokusil model natrénovat s větším počtem dat. Zvýšil jsem tedy počet snímků v trénovacím datasetu na 30 000. Výsledky jsou uvedeny na obrázku 5.6 v porovnání s obrázky z průběhu trénování modelu trénovaného na 15 000 obrázcích. Zvětšení tréno-

vacího datasetu pomohlo ke kvalitnější rekonstrukci obličejů. Na takto rekonstruovaných snímcích jsou lépe odlišitelné jednotlivé části obličeje a výraznější hrany. Model sice ještě nedokáže rekonstruovat snímky úplně přesně a snímky zobecňuje podle trénovacích dat, ale oproti snímkům z prvního experimentu s původním modelem se jedná o velký posun kupředu. Výsledky z pozměněného ResNet modelu trénovaného na 30 000 snímcích jsou k vidění také v příloze A na obrázku A.5.

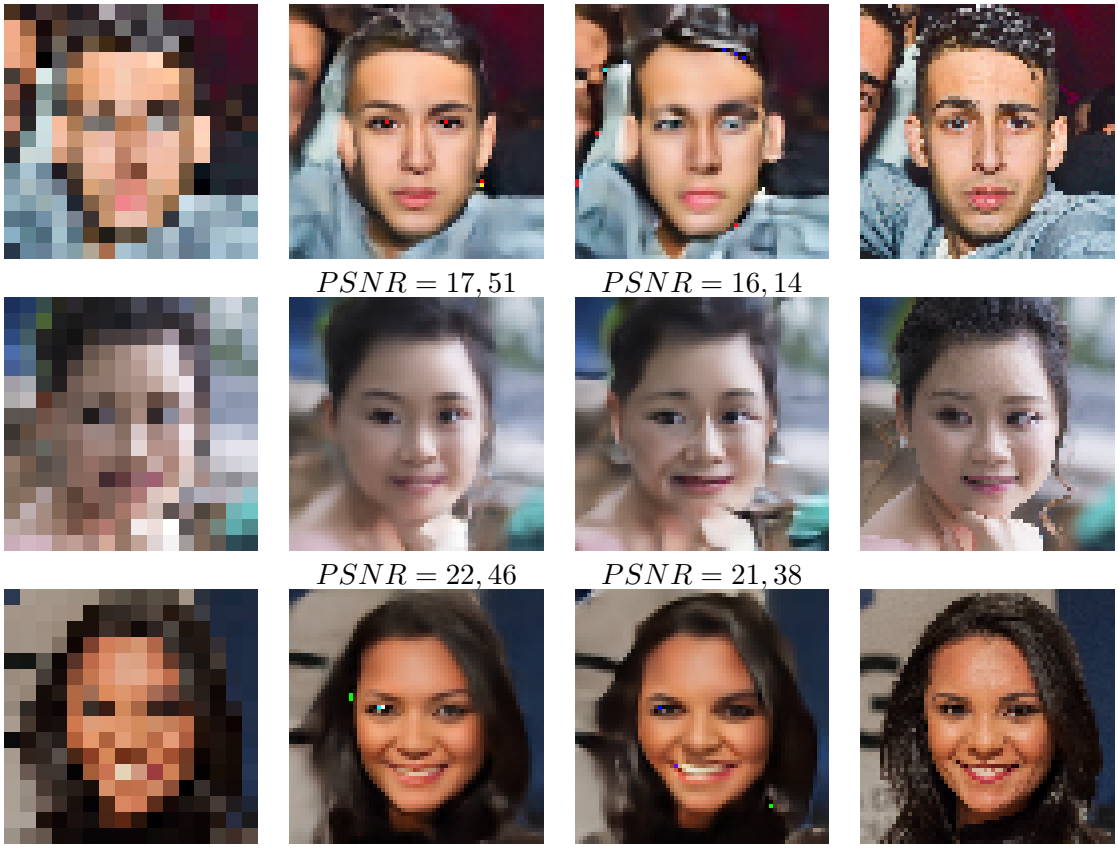


Obrázek 5.6: Porovnání snímků získaných během trénování a po dotrénování nového modelu s pozměněným propojením reziduálních bloků. (a) jsou vstupní snímky se sníženým rozlišením, (b) jsou snímky získané během učení modelu na 15 000 snímcích, (c) jsou snímky získané po dotrénování modelu s 30 000 snímky a (d) jsou originální snímky. Hodnoty $PSNR$ jsou uvedeny v jednotkách dB (decibel).

Rekonstrukce snímků s rozlišením 16×16 pixelů pomocí nového modelu

Model se změněným návrhem reziduálních bloků jsem trénoval i pro rekonstrukci snímků s rozlišením 16×16 pixelů. Opět jsem použil dataset s 30 000 snímky, aby se zmenšila pravděpodobnost přeučení na trénovací data. Na obrázku 5.7 jsou obrázky rekonstruované modelem trénovaným na 30 000 obrázcích porovnány se snímky rekonstruovanými modelem, který byl trénován na 15 000 snímcích. Kupodivu model trénovaný na 30 000 snímcích dosahuje horší vizuální kvality rekonstrukce než model natrénovaný na 15 000 snímcích. Pár dalších snímků rekonstruovaných tímto pozměněným modelem natrénovaným na 30

000 snímcích je v příloze A na obrázku A.6. Výsledku modelu, který byl trénován na 15 000 snímcích, je možné vidět také v příloze A na obrázku A.4.



(a) (b) $PSNR = 22,19$ (c) $PSNR = 21,42$ (d)

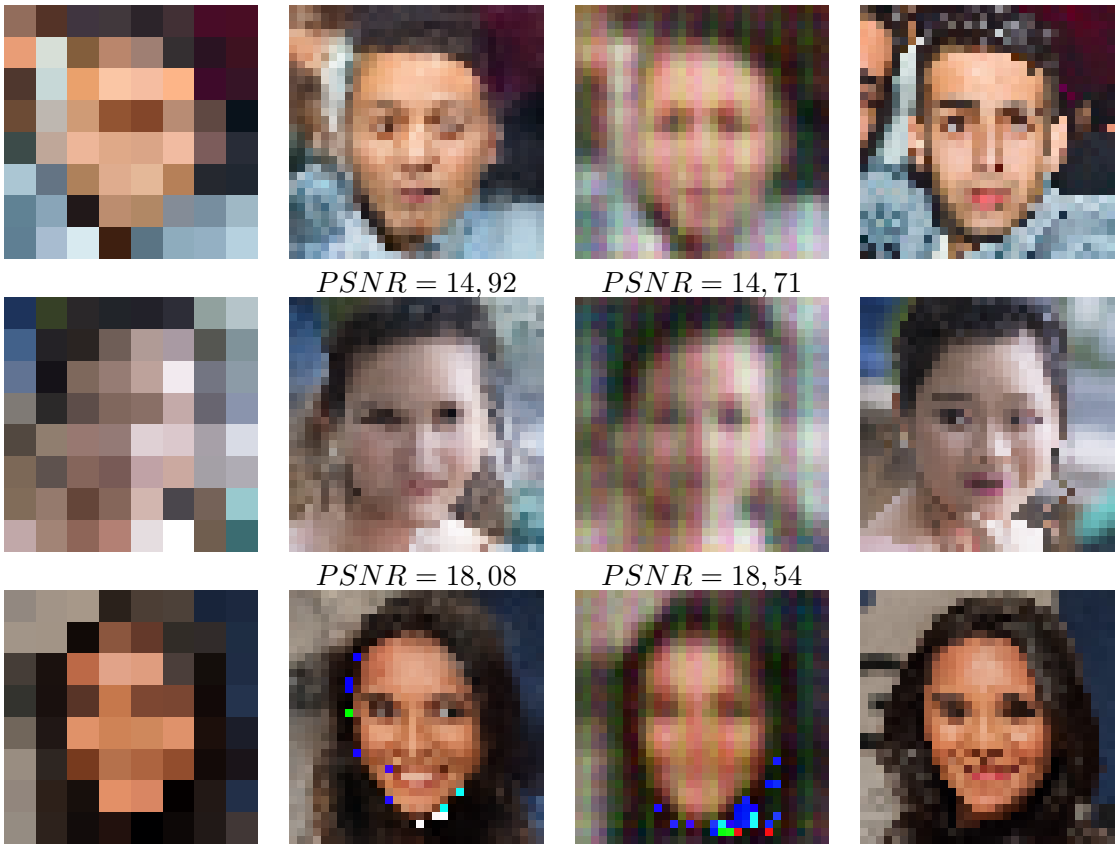
Obrázek 5.7: Porovnání snímků získaných během trénování a po dotrénování nového modelu s pozměněným propojením reziduálních bloků. (a) jsou vstupní snímky se sníženým rozlišením, (b) jsou snímky získané během učení modelu na 15 000 snímcích, (c) jsou snímky získané po dotrénování modelu s 30 000 snímky a (d) jsou originální snímky. Hodnoty $PSNR$ jsou uvedeny v jednotkách dB (decibel).

5.5 Experimenty s GAN modelem

Poslední sadu experimentů jsem prováděl s modelem založeným na generativních kompetitivních sítích. Hlavně z počátku experimentování jsem se setkal s řadou problémů. Největším z nich byl tzv. *mode collaps*, který je stručně popsán v kapitole Implementace v sekci o implementaci toho modelu (4.4). Po vyřešení problémů s učením jsem mohl provést pár experimentů s tímto modelem. Dva z nich byly, stejně jako u modelu založeného na architektuře ResNet, rekonstrukce obrázků s rozlišením 8×8 a 16×16 pixelů. Rovnou musím uznat, že tento model nedosahuje ani takových kvalit, jako modely založené na ResNetu. Právě tyto dva experimenty uvedu v této sekci pro porovnání výkonnosti modelů založených na architektuře generativních kompetitivních sítích a ResNet.

Jako první je na obrázku 5.8 uvedeno porovnání rekonstruovaných snímků se vstupním rozlišením 8×8 pixelů. Na obrázku je jasně vidět, že model založený na architektuře

GAN dokáže rekonstruovat obrys obličeje. Ostré přechody barev a zpracování detailů nezvládá skoro vůbec. Pro vyšší kvalitu generovaných snímků by bylo potřeba lépe vyvážit sílu generátoru a diskriminátoru, a poměr odhadu maximální pravděpodobnosti a funkce *L1* v chybové funkci generátoru.



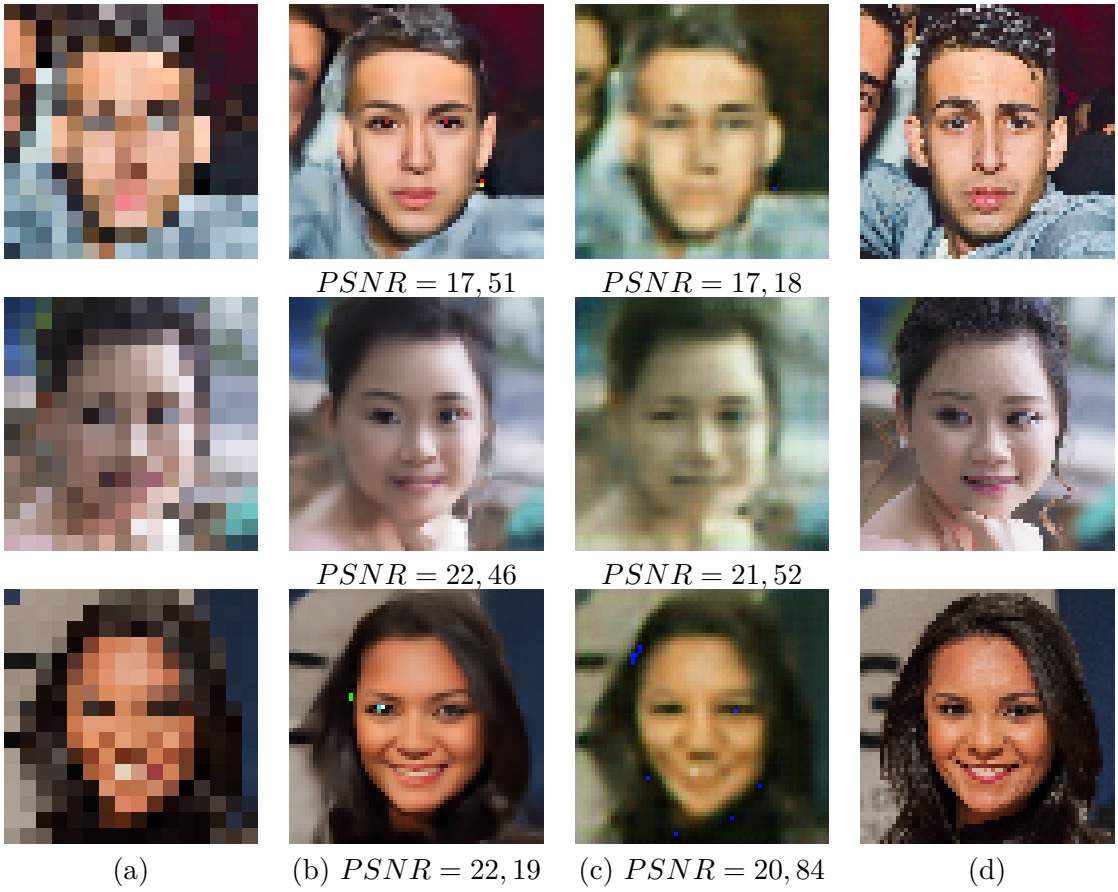
(a) (b) $PSNR = 17,56$ (c) $PSNR = 16,53$ (d)

Obrázek 5.8: Porovnání snímků s rozlišením 8×8 pixelů rekonstruovaných na rozlišení 32×32 pixelů modely GAN a ResNet. (a) jsou vstupní snímky se sníženým rozlišením, (b) jsou snímky získané z pozdějšího modelu ResNet s pozměněnou architekturou reziduálních bloků, (c) jsou snímky získané z modelu založeného na architektuře generativních kompetitivních sítí a (d) jsou originální snímky. Hodnoty $PSNR$ jsou uvedeny v jednotkách dB (decibel).

Další experiment pro porovnání výkonnosti modelů byla opět rekonstrukce vstupních snímků s rozlišením 16×16. V tomto testu už obstál model založený na architektuře GAN lépe. Ná obrázku 5.9 můžeme vidět, že pro snímky s větším vstupním rozlišením už dokáže trochu lépe rekonstruovat i detaily obličejů. V čem ale zaostává za ResNet modelem jsou ostré hrany a pořádné zpracování textur. Výsledky modelu GAN jsou daleko více rozmazané a všechny barvy mají podobný odstín. To je kvůli tomu, že generátor se „rozhodl", že je to nejlepší způsob jak obelstít diskriminátor.

5.6 Zhodnocení výsledků

Řadou experimentů jsem otestoval, jak dobře moje modely dokáží provádět rekonstrukci obrazu ve smyslu zvýšení rozlišení. Všechny modely jsou natrénovány na čtyřnásobné zvýšení



Obrázek 5.9: Porovnání snímků s rozlišením 16×16 pixelů rekonstruovaných na rozlišení 64× 64 pixelů modely GAN a ResNet natrénovanými na 15 000 snímcích. (a) jsou vstupní snímky se sníženým rozlišením, (b) jsou snímky získané z modelu ResNet, (c) jsou snímky získané z modelu GAN a (d) jsou originální snímky. Hodnoty $PSNR$ jsou uvedeny v jednotkách dB (decibel).

rozlišení snímků, na kterých je významně rozmazán lidský obličej. Pro trénování jsem zvolil vstupní snímky s rozlišením 8×8 a 16×16. S rekonstrukcí vstupních snímků s rozlišením 16×16 pixelů si dokázaly poradit všechny modely až na model založený na architektuře GAN, který po vizuální stránce nedosahoval tak dobré rekonstrukce a nárůst kvality u rekonstruovaných snímků oproti původním není moc velký. V těchto rekonstrukcích chyběly ostré přechody barev a velmi špatně byly zpracovány detaily. Zbylé modely si s touto rekonstrukcí poradily dobře. Rekonstruované obličeje vesměs odpovídají obličejům na originálních snímcích. Rekonstrukce samozřejmě není dokonalá. Největším mínusem těchto rekonstruovaných snímků je jejich ne úplně přirozená vizuální podoba. Dalším nedostatkem je to, že modely trošku zobecňují rekonstruované obličeji na ty, které viděly během tréninku, a rekonstrukce tedy nemůže být stoprocentní. Velkým problémem pro tyto modely jsou také cizí předměty přítomné na snímcích. Pro příklad můžu jmenovat pokrývky hlavy, jejichž barva ještě ke všemu téměř splývá s pozadím. Na druhou stranu si modely vcelku dokázaly poradit s natočením hlavy ve snímku i s mimickými výrazy lidí na obrázcích.

Horší výkonnost předvedly modely natrénované na rekonstrukci snímků se vstupním rozlišením 8×8, které neobsahují tolik informací. Tyto modely si dokáží celkem dobře

poradit s rekonstrukcí obličejů, které nejsou natočené, a které mají na snímku „konvenční" tvar. S čím si tyto modely nedokázali téměř vůbec poradit jsou cizí předměty nebo jiné části těla (většinou dlaň) překrývající část obličeje. Další věci způsobující problémy jsou úšklebky lidí na snímcích a různé pootočení hlavy.

Jelikož se mnoho prací nezaměřuje na rekonstrukci snímků v tak nízkém rozlišení, porovnával jsem modely pouze mezi sebou a ne s prací nějaké druhé strany. Pro lepší přehled o úspěšnosti jednotlivých modelů jsou do přílohy A umístěny výsledné rekonstruované snímky vzniklé pomocí těchto modelů.

Závěr

Hlavním cílem této bakalářské práce bylo navržení modelu neuronové sítě, který bude schopen rekonstruovat snímky obličeje v tak nízkém rozlišení, že v něm budou významně rozmazány jeho základní části. V rámci práce je tedy představen návrh a implementace dvou modelů. Každý z těchto modelů je založen na jiné architektuře konvolučních neuronových sítí. První z nich je postaven na architektuře ResNet, kdežto druhý využívá principů generativních kompetitivních sítí. Během experimentování s prvním modelem došlo k jeho úpravě hlavně ve struktuře propojení reziduálních bloků. Touto úpravou jsem sice dosáhl zlepšení kvality rekonstrukce, ale bylo to za cenu větší paměťové a výpočetní náročnosti modelu.

Navržené modely byly implementovány v programovacím jazyce Python za pomoci frameworku TensorFlow, který poskytuje aplikační programové rozhraní pro implementaci neuronových sítí. Dále byla implementována aplikace s grafickým uživatelským rozhraním pro jednoduché ovládání jednoho z modelů pro rekonstrukci. Úpravou již existujícího datasetu Flicker-Faces-HQ, který obsahuje snímky obličejů v rozlišení 1024×1024 pixelů, byl vytvořen dataset pro trénování těchto modelů a jejich evaluaci. Dataset se skládá ze snímků s vyšším rozlišením (32×32 a 64×64) a z k nim korespondujících snímků se sníženým rozlišením (8×8 a 16×16).

Oba dva modely – s úpravou prvního z nich respektive tři – dokáží do určité míry rekonstruovat snímky obličejů v nízkém rozlišení, přičemž nejlépe z nich si vede třetí model s pozměněnou strukturou reziduálních bloků oproti modelu původnímu. Tento model má ale největší počet trénovatelných parametrů a největší výpočetní a paměťovou náročnost. Dokonce musel být i trénován na více datech než předešlé modely. Největším problémem pro všechny tři modely jsou cizí předměty ve snímku, které se nedaří věrohodně rekonstruovat. Dále například snímky s nezvyklým natočením hlavy, které se moc nevyskytovaly v trénovacích datech. Na druhou stranu, rekonstrukce snímků s obličeji natočenými čelem ke kameře a nepřekrývanými cizím objektem je, troufám si tvrdit, kvalitní. A to hlavně s ohledem na to, jak málo informací obsahují rekonstruované vstupní snímky.

Možnosti a prostor pro pokračování v této práci rozhodně jsou. Další činnost by mohla být zaměřena na snížení paměťové a hlavně výpočetní náročnosti modelů a na vývoj aplikace, která bude schopna rekonstruovat obličeje v záznamu kamery v reálném čase pomocí těchto zjednodušených modelů.

Literatura

[1] History of Prizefighting's Biggest Money Fights. 2017, [online].[cit. 20. 3. 2019].
URL https://www.bloodyelbow.com/2017/8/24/16170894/history-of-prizefightings-biggest-money-fights-boxing-mma-ufc

[2] 2019 Video Surveillance Cameras Overview. January 2019, [online].[cit. 20. 5. 2019].
URL https://ipvm.com/reports/cameras-2019

[3] Estimators. 2019, [online].[cit. 24. 4. 2019].
URL https://www.tensorflow.org/guide/estimators

[4] GPU support. 2019, [online].[cit. 24. 4. 2019].
URL https://www.tensorflow.org/install/gpu

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

[6] Logistic function. 2019, [online].[cit. 20. 3. 2019].
URL https://en.wikipedia.org/wiki/Logistic_function

[7] PAL. 2019, [online].[cit. 23. 1. 2019].
URL https://en.wikipedia.org/wiki/PAL

[8] Tensors. 2019, [online].[cit. 24. 4. 2019].
URL https://www.tensorflow.org/guide/tensors

[9] Video Graphics Array. 2019, [online].[cit. 23. 1. 2019].
URL https://en.wikipedia.org/wiki/Video_Graphics_Array

[10] and, J. A.: Edge-directed interpolation. In *Proceedings of 3rd IEEE International Conference on Image Processing*, ročník 3, Sep. 1996, s. 707–710 vol.3, doi:10.1109/ICIP.1996.560768.

[11] Bojarski, M.; Yeres, P.; Choromanska, A.; aj.: Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car. *CoRR*, ročník abs/1704.07911, 2017.

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

[13] Chellapilla, K.; Puri, S.; Simard, P.: High Performance Convolutional Neural Networks for Document Processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, Université de Rennes 1, La Baule (France): Suvisoft, Říjen 2006.

[14] Dahl, R.; Norouzi, M.; Shlens, J.: Pixel Recursive Super Resolution. *CoRR*, ročník abs/1702.00783, 2017.

[15] Deng, J.; Dong, W.; Socher, R.; aj.: ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009, ISSN 1063-6919.

[16] Dong, C.; Loy, C. C.; He, K.; aj.: Image Super-Resolution Using Deep Convolutional Networks. *CoRR*, ročník abs/1501.00092, 2015.

[17] Dumoulin, V.; Visin, F.: A guide to convolution arithmetic for deep learning. 2016.

[18] Freeman, W. T.; Pasztor, E. C.; Jones, T. R.: Example-Based Super-Resolution. *IEEE Computer Graphics and Applications*, ročník 22, č. 02, mar 2002: s. 56–65, ISSN 0272-1716, doi:10.1109/38.988747.

[19] Goldsmith, D.: Understanding network IP camera resolutions. November 2018, [online].[cit. 20. 5. 2019].
URL https://www.broadbandbuyer.com/advice/2297-understanding-network-ip-camera-resolutions/

[20] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press, 2016, ISBN 9780262035613, http://www.deeplearningbook.org.

[21] Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; aj.: Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, editace Z. Ghahramani; M. Welling; C. Cortes; N. D. Lawrence; K. Q. Weinberger, Curran Associates, Inc., 2014, s. 2672–2680.

[22] Gulrajani, I.; Ahmed, F.; Arjovsky, M.; aj.: Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30*, editace I. Guyon; U. V. Luxburg; S. Bengio; H. Wallach; R. Fergus; S. Vishwanathan; R. Garnett, Curran Associates, Inc., 2017, s. 5767–5777.

[23] Haykin, S.: *Neural Networks: A Comprehensive Foundation (2nd Edition)*. New York: Prentice Hall, 1998, ISBN 0132733501.

[24] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. *CoRR*, ročník abs/1512.03385, 2015.

[25] Hore, A.; Ziou, D.: Image Quality Metrics: PSNR vs. SSIM. In *2010 20th International Conference on Pattern Recognition*, Aug 2010, ISSN 1051-4651, s. 2366–2369, doi:10.1109/ICPR.2010.579.

[26] Huang, X.; Kwiatkowska, M.; Wang, S.; aj.: Safety Verification of Deep Neural Networks. *CoRR*, ročník abs/1610.06940, 2016.

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

[28] Karras, T.; Laine, S.; Aila, T.: A Style-Based Generator Architecture for Generative Adversarial Networks. *CoRR*, ročník abs/1812.04948, 2018.

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

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

[31] LeCun, Y.; Boser, B.; Denker, J. S.; aj.: Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.*, ročník 1, č. 4, Prosinec 1989: s. 541–551, ISSN 0899-7667, doi:10.1162/neco.1989.1.4.541.

[32] Ledig, C.; Theis, L.; Huszar, F.; aj.: Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *CoRR*, ročník abs/1609.04802, 2016.

[33] Lim, B.; Son, S.; Kim, H.; aj.: Enhanced Deep Residual Networks for Single Image Super-Resolution. *CoRR*, ročník abs/1707.02921, 2017.

[34] Litwiller, D.: CCD vs. CMOS: facts and fiction. In *Photonics Spectra*, ročník 35, 2001, s. 154–158.

[35] Majidimehr, A.: UNDERSTANDING ANALOG AND IP TV CCTV CAMERAS. 2019, [online].[cit. 23. 1. 2019].
URL https://www.monitoryourassets.com/ip-vs-analog/

[36] McCulloch, W. S.; Pitts, W.: Neurocomputing: Foundations of Research. kapitola A Logical Calculus of the Ideas Immanent in Nervous Activity, Cambridge, MA, USA: MIT Press, 1988, ISBN 0-262-01097-6, s. 15–27.

[37] Mehrotra, K.; Mohan, C. K.; Ranka, S.: *Elements of Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1997, ISBN 0-262-13328-8.

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

[39] Nah, S.; Kim, T. H.; Lee, K. M.: Deep Multi-scale Convolutional Neural Network for Dynamic Scene Deblurring. *CoRR*, ročník abs/1612.02177, 2016.

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

[41] Nilsson, F.; Axis, C.: *Intelligent Network Video: Understanding Modern Video Surveillance Systems, Second Edition*. Boca Raton, FL, USA: CRC Press, Inc., druhé vydání, 2013, ISBN 1466555211, 9781466555211.

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

[43] Rosenblatt, F.: The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, 1958: s. 65–386.

[44] Szegedy, C.; Ioffe, S.; Vanhoucke, V.; aj.: Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *CoRR*, 2016.

[45] Timofte, R.; De, V.; Gool, L. V.: Anchored Neighborhood Regression for Fast Example-Based Super-Resolution. In *2013 IEEE International Conference on Computer Vision*, Dec 2013, ISSN 1550-5499, s. 1920–1927, doi:10.1109/ICCV.2013.241.

[46] Timofte, R.; De Smet, V.; Van Gool, L.: A+: Adjusted Anchored Neighborhood Regression for Fast Super-Resolution. 04 2015, s. 111–126, doi:10.1007/978-3-319-16817-3_8.

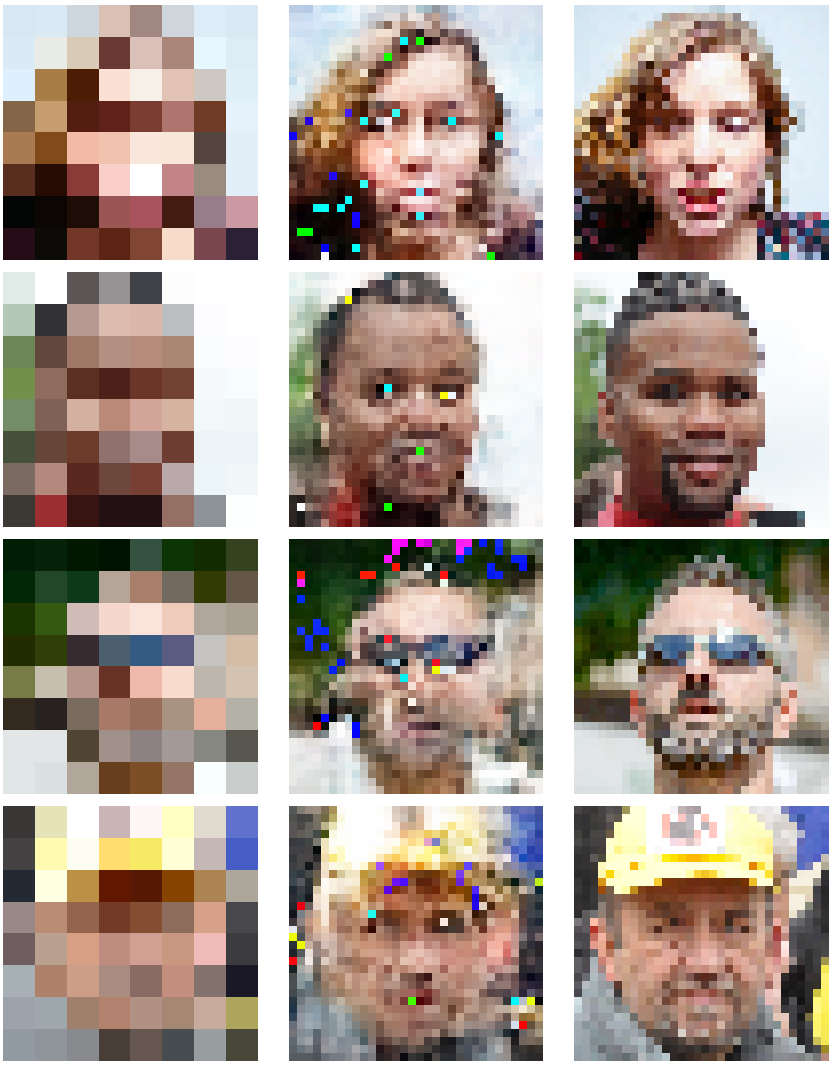
[47] Wang, X.; Yu, K.; Wu, S.; aj.: ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. *CoRR*, ročník abs/1809.00219, 2018.

[48] Xin Li; Orchard, M. T.: New edge-directed interpolation. *IEEE Transactions on Image Processing*, ročník 10, č. 10, Oct 2001: s. 1521–1527, ISSN 1057-7149, doi:10.1109/83.951537.

[49] Yesil, B.: WATCHING OURSELVES. *Cultural Studies*, ročník 20, č. 4-5, 2006: s. 400–416, doi:10.1080/09502380600708770.

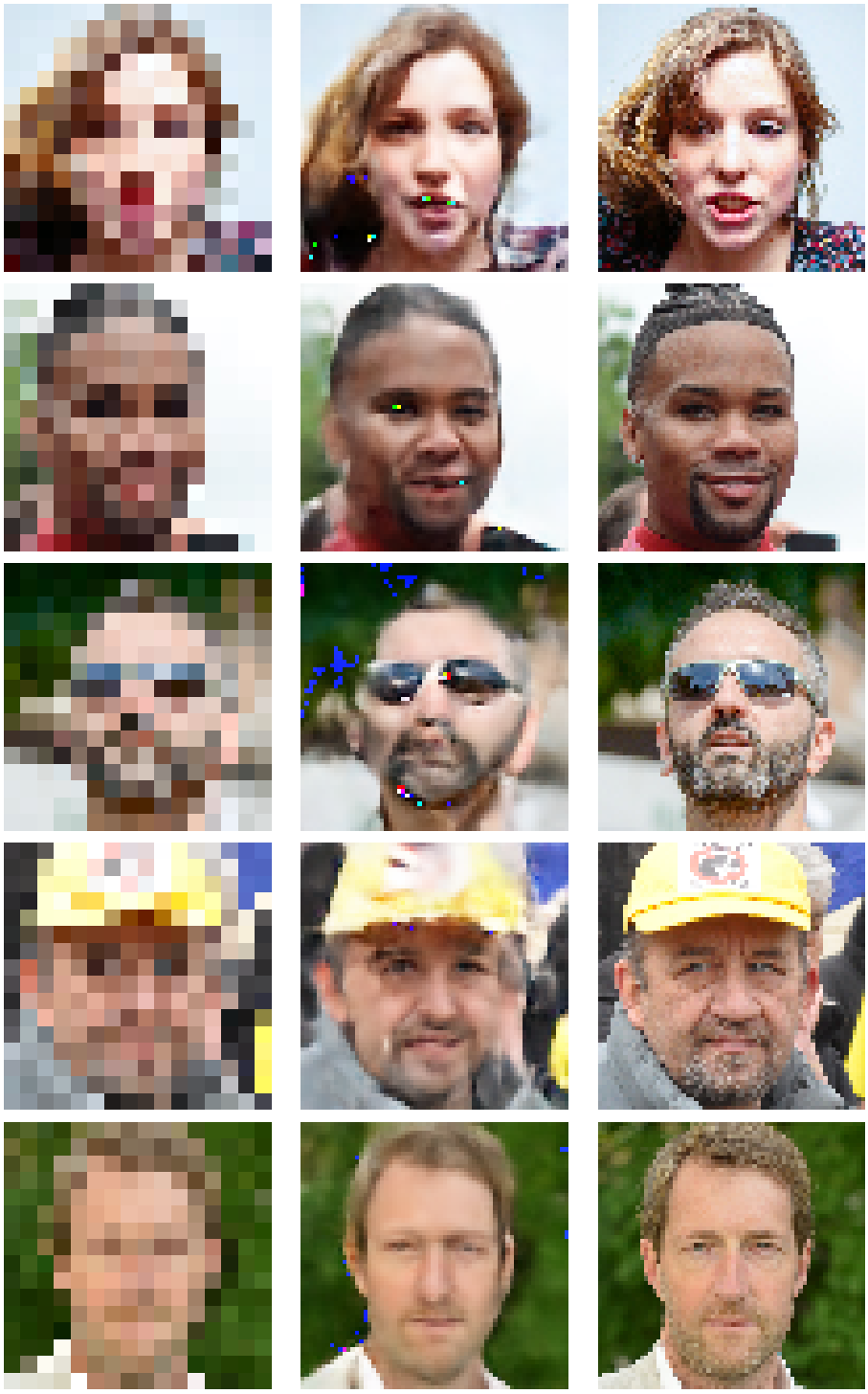
Příloha A

Výsledky modelů



(a) (b) (c)

Obrázek A.1: Snímky s rozlišením 8×8 pixelů rekonstruované pomocí původního ResNet modelu. (a) jsou vstupní snímky, (b) jsou rekonstruované snímky a (c) je *ground-truth*.

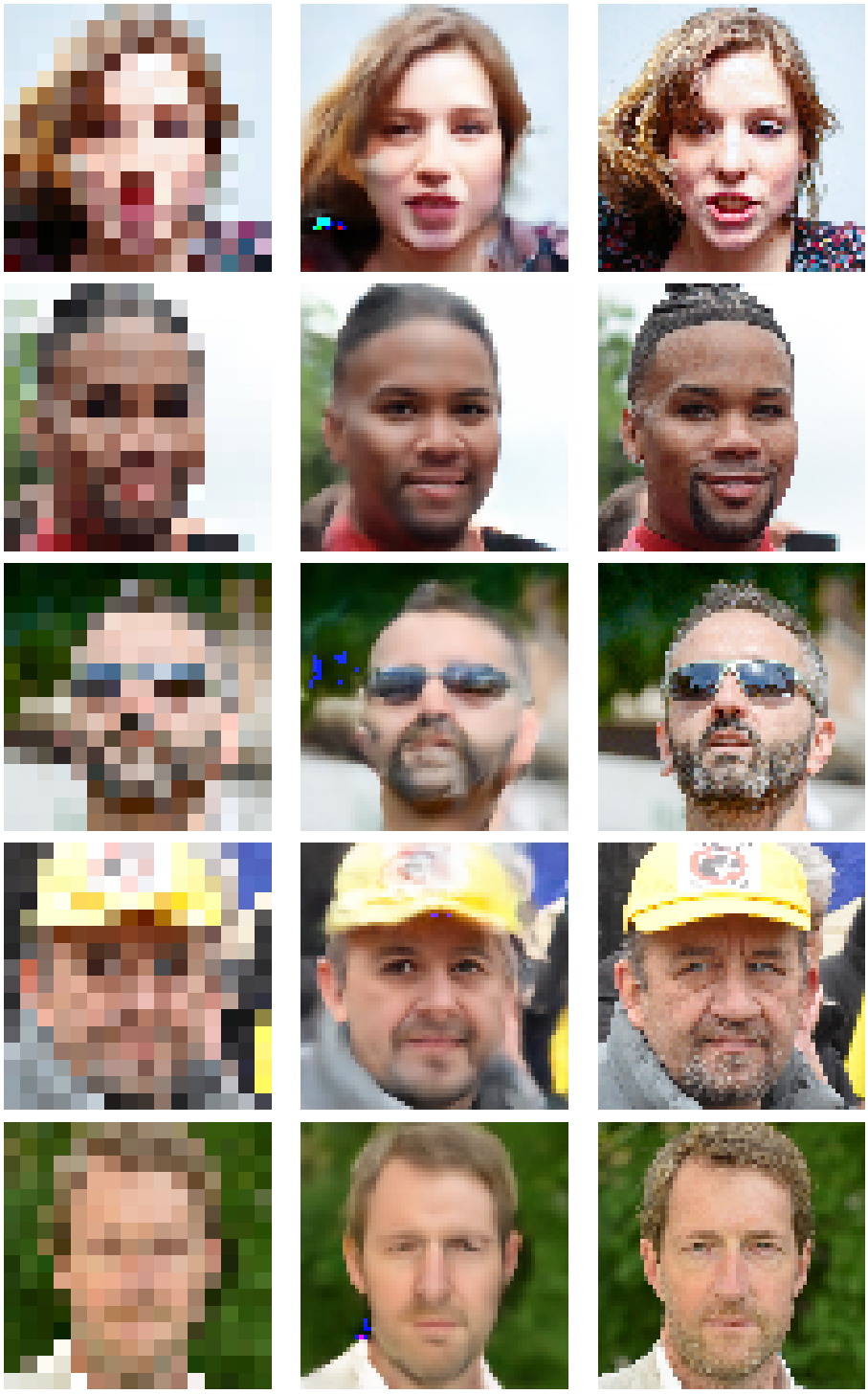


(a) (b) (c)

Obrázek A.2: Snímky s rozlišením 16×16 pixelů rekonstruované pomocí původního ResNet modelu natrénovaného na 15 000 snímcích. (a) jsou vstupní snímky, (b) jsou rekonstruované snímky a (c) je *ground-truth*.

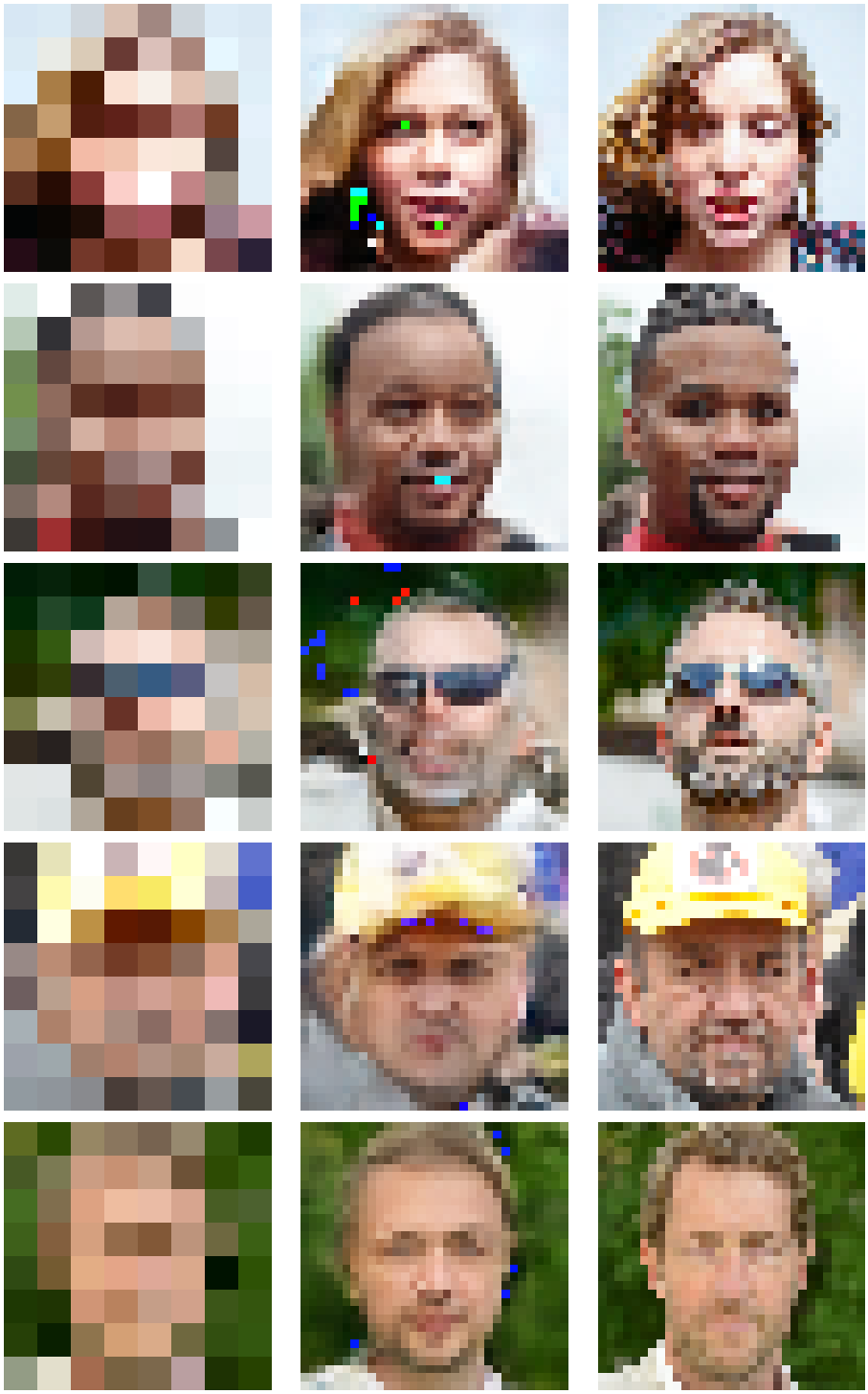
(a) (b) (c)

Obrázek A.3: Snímky s rozlišením 8×8 pixelů rekonstruované pomocí ResNet modelu s pozměněnou strukturou propojení reziduálních bloků natrénovaného na 15 000 snímcích. (a) jsou vstupní snímky, (b) jsou rekonstruované snímky a (c) je *ground-truth*.



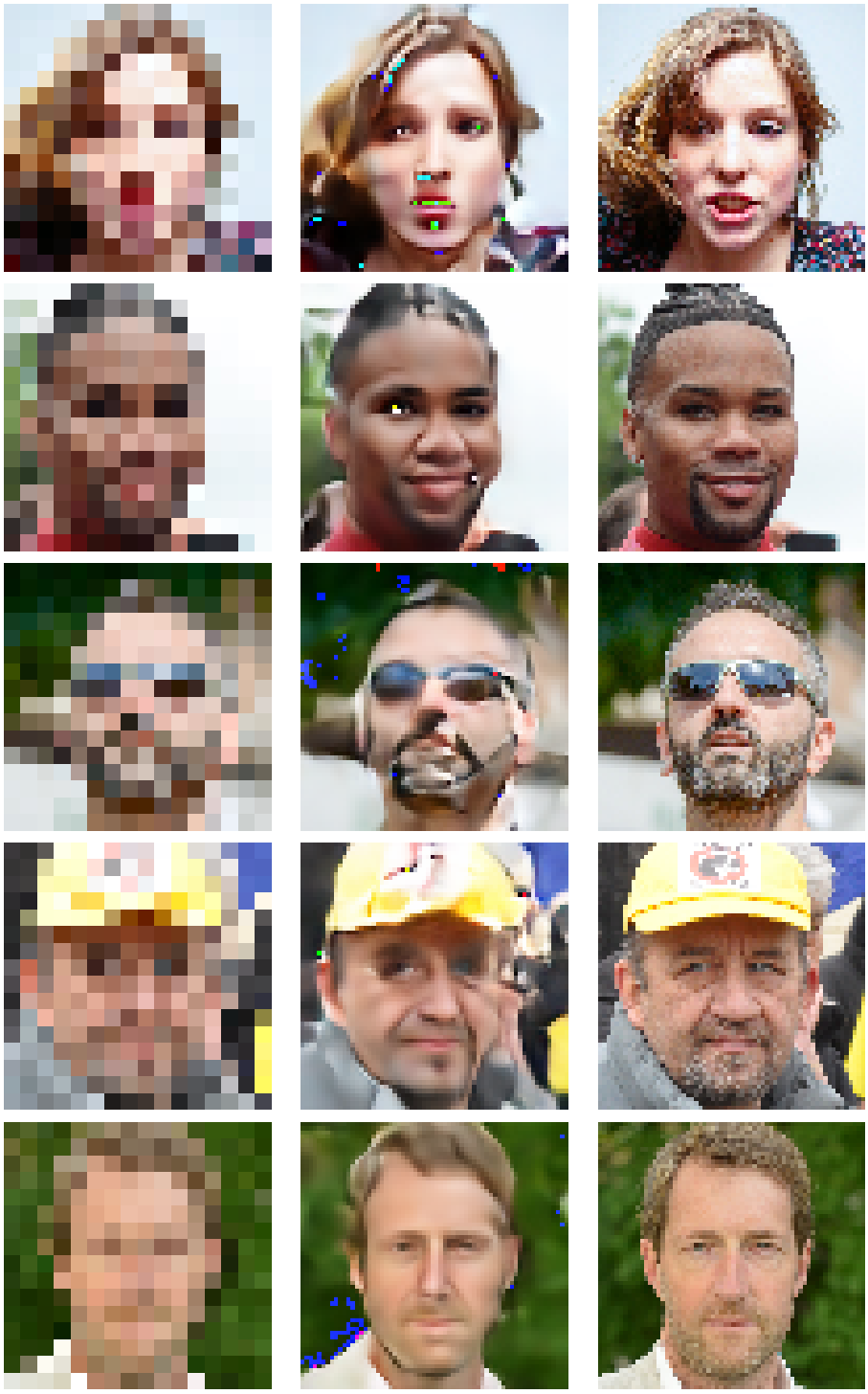
(a) (b) (c)

Obrázek A.4: Snímky s rozlišením 16×16 pixelů rekonstruované pomocí ResNet modelu s pozměněnou strukturou propojení reziduálních bloků natrénovaného na 15 000 snímcích. (a) jsou vstupní snímky, (b) jsou rekonstruované snímky a (c) je *ground-truth*.



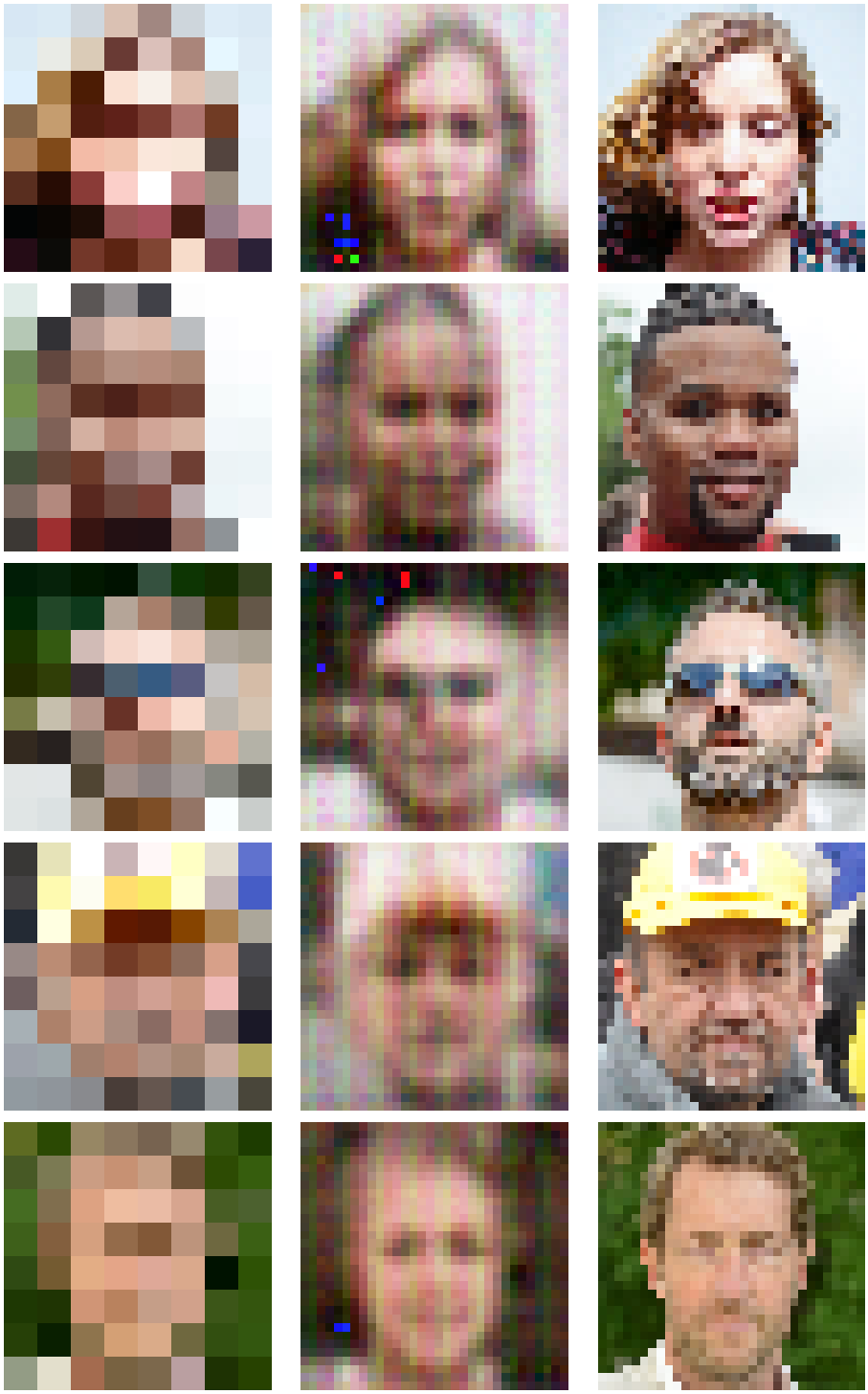
(a) (b) (c)

Obrázek A.5: Snímky s rozlišením 8×8 pixelů rekonstruované pomocí ResNet modelu s pozměněnou strukturou propojení reziduálních bloků natrénovaného na 30 000 snímcích. (a) jsou vstupní snímky, (b) jsou rekonstruované snímky a (c) je *ground-truth*.



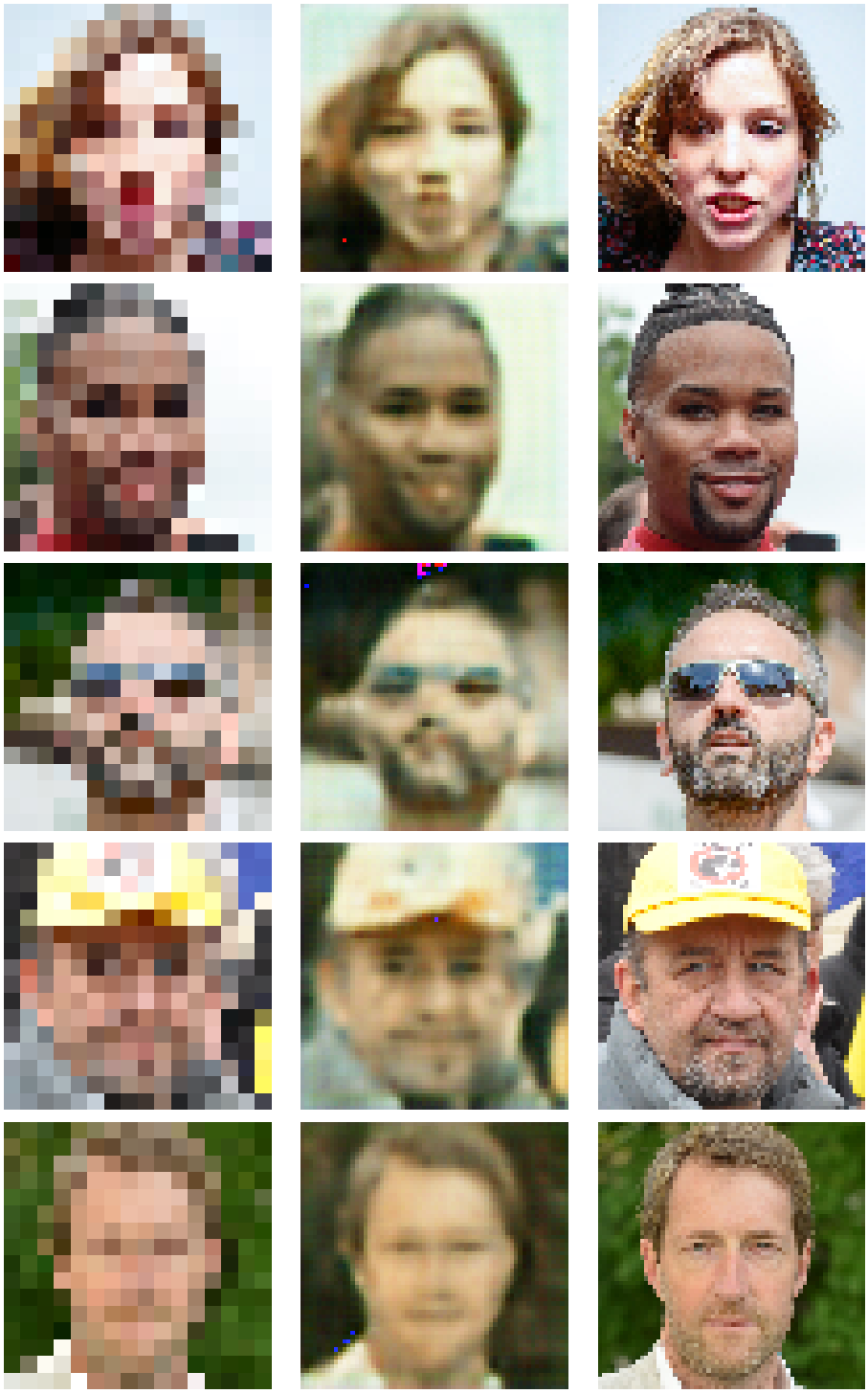
(a) (b) (c)

Obrázek A.6: Snímky s rozlišením 16×16 pixelů rekonstruované pomocí ResNet modelu s pozměněnou strukturou propojení reziduálních bloků natrénovaného na 30 000 snímcích. (a) jsou vstupní snímky, (b) jsou rekonstruované snímky a (c) je *ground-truth*.



(a) (b) (c)

Obrázek A.7: Snímky s rozlišením 8×8 pixelů rekonstruované pomocí GAN modelu natrénovaného na 15 000 snímcích. (a) jsou vstupní snímky, (b) jsou rekonstruované snímky a (c) je *ground-truth*.



(a) (b) (c)

Obrázek A.8: Snímky s rozlišením 16 × 16 pixelů rekonstruované pomocí GAN modelu trénovaného na 15 000 snímcích. (a) jsou vstupní snímky, (b) jsou rekonstruované snímky a (c) je *ground-truth*.

Příloha B

Obsah datového média

Přiložené datové médium obsahuje následující adresářovou strukturu:

- **src/** – zde jsou obsaženy veškere zdrojové kódy implementovaných modelů a aplikace s grafickým uživatelským rozhraním.
  - trained_models/ – v tomto adresáři jsou uloženy natrénované modely neuronových sítí.
- **src_doc/** – tento adresář obsahuje zdrojové kódy textové části práce.
- **doc/** – tady je uložena textová část práce ve formátu pdf.
- **dataset/** – v tomto adresáři je uložen dataset – jak trénovací tak evaluační část.
  - LR_8_8/ – trénovací vstupní snímky s rozlišením 8×8 pixelů.
  - LR_16_16/ – trénovací vstupní snímky s rozlišením 16×16 pixelů.
  - HR_32_32/ – trénovací originální snímky s rozlišením 32×32 pixelů.
  - HR_64_64/ – trénovací originální snímky s rozlišením 64×64 pixelů.
  - eval_8_8/ – evaluační vstupní snímky s rozlišením 8×8 pixelů.
  - eval_16_16/ – evaluační vstupní snímky s rozlišením 16×16 pixelů.
  - eval_32_32/ – evaluační originální snímky s rozlišením 32×32 pixelů.
  - eval_64_64/ – evaluační originální snímky s rozlišením 64×64 pixelů.
- **output/** – v tomto adresáři jsou k nalezení příklady výstupu jednotlivých modelů.
  - resnet_model/ – výstup z původního ResNet modelu.
  - resnet_groups_model/ – výstup z pozměněného ResNet modelu.
  - gan_model/ – výstup z GAN modelu.
- README.md – popis všech částí.