TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Kolorizace černobílých obrázků pomocí neuronové sítě

Bakalářská práce

Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie

Autor práce: **Martin Poláček**
Vedoucí práce: prof. Ing. Jan Nouza, CSc.
Ústav informačních technologií a elektroniky

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Kolorizace černobílých obrázků pomocí neuronové sítě

Jméno a příjmení: **Martin Poláček**
Osobní číslo: M18000223
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav informačních technologií a elektroniky
Akademický rok: **2020/2021**

Zásady pro vypracování:

1. Cílem práce je prakticky se seznámit s problematikou enkodérů založených na neuronových sítích a aplikovat je na úlohu kolorizace (obarvení) černobílých obrázků.
2. Nastudujte si principy neuronových sítí a na nich založených enkodérů, a to zejména pro zpracování obrazových (dvourozměrných) dat. Na základě toho si proveďte vlastní implementaci takové neuronové sítě.
3. Vytvořte si co nejrozsáhlejší soubor barevných obrázků o normované velikosti 256 x 256 pixelů a ke každému z nich automaticky vygenerujte jeho černobílou verzi. Soubor rozdělte na trénovací, vývojovou a testovací část.
4. Vytvořte programy, které na trénovací sadě naučí neuronovou síť vybarvovat předložené černobílé obrázky. Vyzkoušejte různé přístupy a porovnejte výsledky na testovací sadě.

Rozsah grafických prací:	Dle potřeby dokumentace
Rozsah pracovní zprávy:	30 – 40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Schmitt, M., Hughes, L. H., Korner, M. and Zhu, X. X. Colorizing Sentinel-1 SAR images using a variational autoencoder conditioned on Sentinel-2 imagery. In:Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., Vol. XLII-2, 2018, pp. 1045–1051
[2] S. Indolia, A. Kumar, S.P. Mishra, P. Asopa. Conceptual understanding of convolutional neural network. A deep learning approach. Procedia Comput. Sci., 132 (2018), pp. 679-688
[3] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. Preprint http://arxiv.org (arXiv:2003.05991v1 [cs.LG]) 12 Mar 2020.

Vedoucí práce:	prof. Ing. Jan Nouza, CSc. Ústav informačních technologií a elektroniky
Datum zadání práce:	19. října 2020
Předpokládaný termín odevzdání:	17. května 2021

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 19. října 2020

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

11. května 2021 Martin Poláček

Kolorizace černobílých obrázků pomocí neuronové sítě

Abstrakt

Tato bakalářská práce se zabývá automatickým kolorováním černobílých obrázků pomocí autoenkodérů založených na neuronových sítích. Nejprve je popsán způsob kódování barevných a černobílých obrázků, dále jsou zmíněny nejčastěji používané barevné modely a formáty pro ukládání obrazové informace. Následně jsou stručně shrnuty metody automatické kolorizace vyvinuté v posledních letech. Čtvrtá a pátá kapitola je věnována neuronovým sítím. Jsou zejména zaměřeny na architektury a frameworky použité v práci, tedy konvoluční sítě a enkodéry. Důležitou součástí práce je vytvoření velmi rozsáhlého datasetu obsahujícího 150 000 obrázků, který je nezbytný pro trénování a testování neuronové sítě. V sedmé kapitole jsou zdokumentovány vlastní implementační a experimentální práce, které vedly k postupnému vývoji vlastního kolorizačního schématu. Praktickým výstupem je demonstrační program umožňující během necelé sekundy obarvit dodaný obrázek o rozměrech 256 x 256 pixelů, přičemž obrázky s jinými velikostmi jsou na tento rozměr automaticky upraveny.

Klíčová slova: kolorizace, neuronová síť, konvoluce, autoenkodér, Keras

Colorization of gray-scale images using neural networks

Abstract

This bachelor thesis deals with automatic coloring of grayscale images using autoencoders based on neural networks. First, the method of encoding color and grayscale images is described, then the most commonly used color models and formats for storing image information are mentioned, and then the methods of automatic colorization developed in recent years are briefly summarized. The fourth and fifth chapters are devoted to neural networks, focusing mainly on the architectures and frameworks used in the work, thus convolutional networks and encoders. An important part of the work is the creation of a very large dataset containing 150 000 images, which is necessary for training and testing of the neural network. The seventh chapter documents the implementation and experimental work, which led to the gradual development of its own colorization scheme. The practical output is a demonstration program that allows you to color the supplied image with dimensions of 256 x 256 pixels in less than a second, while images with other sizes are automatically adjusted to the size.

Keywords: colorization, neural network, convolution, autoencoder, Keras

Poděkování

Rád bych poděkoval panu profesoru Janu Nouzovi za vedení mé bakalářské práce a poskytnutí konzultací při řešení jednotlivých kroků návrhu neuronové sítě. Také bych rád poděkoval panu doktoru Jiřímu Jeníčkovi za poskytnutí přístupu k výpočetnímu clusteru.

Obsah

Seznam zkratek 9

1 Úvod **10**

2 Černobílé, barevné obrázky a jejich reprezentace **12**
2.1 Formáty obrázků 12
2.2 Barevné modely 13
2.2.1 RGB a CMYK 13
2.2.2 HSL (HSB) 14
2.2.3 Lab 14

3 Práce zabývající se problematikou kolorizace **16**
3.1 Kolorizace jako regresivní problém 16
3.2 Kolorizace pomocí CNN jako klasifikační problém 17
3.3 Kolorizace pomocí GAN 19

4 Neuronové sítě **20**
4.1 Neuron 20
4.2 Typy neuronových sítí 21
4.3 Aktivační funkce 21
4.3.1 Sigmoida 21
4.3.2 Tanh (hyperbolický tangens) 22
4.3.3 ReLU 22
4.3.4 Softmax 23
4.4 Plně propojená vrstva 23
4.5 Konvoluční vrstva 24
4.6 Pooling vrstva 24
4.6.1 Max-pooling vrstva 25
4.7 Unpooling vrstva 25
4.7.1 Upsampling vrstva 26
4.8 Transponovaná konvoluční vrstva 26
4.9 Algoritmus zpětné propagace 27
4.10 Autoenkodér 27

5 Výběr frameworku **29**
5.1 Instalace 29
5.2 Základní struktura 30

5.3 Ztrátová funkce (loss funkce) . . . 31
5.3.1 Střední kvadratická odchylka (MSE) . . . 31
5.3.2 Střední absolutní odchylka (MAE) . . . 32
5.4 Optimizér . . . 32
5.4.1 Metoda největšího spádu (gradient descent) . . . 32
5.4.2 Momentum SGD (stochastic gradient descent) . . . 32
5.4.3 RMSprop (root mean square propagation) . . . 33
5.4.4 Adaptive momentum (Adam) . . . 33
5.5 Předčasné ukončení . . . 33

6 Tvorba datasetu 34
6.1 Stažení dat . . . 34
6.2 Normalizace dat . . . 36
6.3 Augmentace dat . . . 36

7 Vlastní implementační a experimentální práce 37
7.1 Vyhodnocovací metrika . . . 37
7.2 Počáteční modely . . . 38
7.2.1 Návrh modelů . . . 38
7.2.2 Trénování modelů . . . 40
7.2.3 Druhý model . . . 42
7.3 Pooling pomocí konvolučních vrstev . . . 45
7.4 Transponované konvoluční vrstvy sloužící pro zvýšení rozměru . . . 48
7.5 Napojení klasifikátoru . . . 50
7.6 Výsledné porovnání . . . 52
7.7 Zodpovězení počátečních otázek . . . 55
7.8 Praktické aspekty . . . 56

8 Vytvořený demonstrační program 57

9 Závěr 59

Seznam obrázků 62

Použitá literatura 64

Seznam zkratek

TUL	Technická univerzita v Liberci
FM	Fakulta mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci
MIT	Massachusettský technologický institut
GAN	Generativní neuronová síť
CNN	Konvoluční neuronová síť
NN	Neuronová síť
JPEG	Joint photographic experts group
PNG	The portable network graphics
GPU	Grafická karta
CPU	Procesor
POOL	Sdružování
SSH	Secured shell
SFTP	Secured file transfer protocol
RAM	Random access memory
VRAM	Video random access memory
Tanh	Hyperbolický tangens
ReLU	Rectified linear unit
MSE	Střední kvadratická odchylka
MAE	Střední absolutní odchylka
SGD	Stochastic gradient descent
RMSprop	Root mean square propagation
Adam	Adaptive momentum

1 Úvod

Historie fotografie začíná v 19. století vytvořením prvního černobílého snímku francouzským vynálezcem (Joseph N. Niépce). Snímek byl pořízen za slunného dne a celková doba expozice činila 8 hodin. Ve stejném století objevil skotský fyzik James C. Maxwell fyzikální princip barevné fotografie. Na stejné plátno promítl tři černobílé snímky s použitím zeleného, modrého, červeného filtru a prokázal možnost aditivního míchání barev.

George Eastman vyrobil koncem 19. století fotografický film a představil první fotoaparát založený na chemickém principu. Až roku 1969 se podařilo vyvinout fotoaparát, který umožňoval digitálně zaznamenat obraz pomocí elektronických prvků fotoaparátu.

Na možnost zaznamenání obrazu zareagovali vědci vytvořením prvních černobílých elektronických televizí v první polovině 19. století. Následovalo zahájení pravidelných vysílání. Následně v 70. letech došlo k velkému rozšíření barevných televizí.

S rozvojem barevného obrazu se mnoho grafiků pokoušelo o kolorování černobílých snímků, popřípadě celých filmů. Proces kolorování spočíval v ručním obarvení snímků na základě úsudku a zkušeností grafika. Kolorování se do dnešní doby považuje za složitý proces. Pro dosažení přijatelných výsledků se skládá barevná složka fotografií z desítek až stovek vrstev, které se prolínají dohromady.

V posledních letech se stále více využívá technik automatického kolorování u historických černobílých fotografií a filmů. Příkladem mohou být webové aplikace zaměřené na kolorování soukromých historických fotografií[1], popřípadě již kolorované černobílé filmy[2].

Jako další způsob automatického kolorování se začínají využívat neuronové sítě, které mají velice dobré výsledky a zároveň po naučení sítě trvá proces kolorování pouze několik milisekund. Dosažení dobrých výsledků závisí na návrhu neuronové sítě a hlavně na velikosti a rozmanitosti dat, které jsou neuronové síti předkládána v průběhu učení.

Tato práce v první kapitole popisuje možné způsoby ukládání obrázku tak, aby bylo co nejsnadnější neuronovou síť navrhnou a následně také natrénovat. Následující kapitoly popisují základní principy neuronových sítí a způsob návrhu pro úlohu kolorizace černobílých obrázků. Práce dále pokračuje stanovením několika klíčových kategorií, které se nejvíce vyskytují mezi běžnými obrázky a popisem způsobu

[1] Webová aplikace sloužící pro kolorování libovolných obrázků https://imagecolorizer.com/colorize.html

[2] Příklad kolorování historických černobílých filmů https://www.youtube.com/watch?v=YZuP41ALx_Q

vytvoření kolekce dat pro trénování. Závěrečné kapitoly popisují návrh několika konvolučních neuronových sítí s rozdílnou strukturou, porovnání jejich výsledků na testovacích datech a vznik demonstračního programu pro ukázku funkčnosti vybrané nejlepší neuronové sítě.

2 Černobílé, barevné obrázky a jejich reprezentace

Černobílé i barevné obrázky mají specifickou strukturu ukládání. U černobílých digitálních obrázků se zaznamenává pouze jasová informace a pro každý pixel obrázku si stačí ukládat pouze jednu hodnotu. V případě barevných obrázků není možné (pouze jednou hodnotou) uložit jasovou i barevnou informaci obrázku. Využívá se aditivního míchání barev, pomocí kterého se jednotlivé složky barev sčítají, vytváří světlo o určité intenzitě a tím také barvu. Typicky se využívá míchání červené, zelené a modré barvy.

Pro popis libovolného barevného pixelu ve výsledku stačí mít uloženy pouze tři hodnoty (pro každou barvu jedna hodnota). Následná možnost převodu z barevného obrázku na černobílý je popsána následujícím vzorcem, který sečte hodnoty pro každou barvu v určeném poměru (30 % červená, 59 % zelená a 11 % modrá barva) pro zachování maximálního možného kontrastu obrázku.

$$I = 0.3R + 0.59G + 0.11B \tag{2.1}$$

Při převodu na černobílý obrázek se využívá zcela jednoznačný vztah, při kterém se ztrácí informace o barevné složce obrázku. Naopak zpětný převod na barevný obrázek jednoznačný není. Černobílý obrázek postrádá informaci o tom, jak by měla být vypočtena barevná složka a lze ji pouze odhadnout na základě jasových hodnot a kontextu celého obrázku.

2.1 Formáty obrázků

Pro ukládání obrázků se využívá několik rozdílných grafických formátů s rozdílnými vlastnostmi. Jednotlivé formáty mohou využívat ztrátovou nebo bezztrátovou kompresi dat, rozdílnou barevnou hloubku[1] a možnost průhledného pozadí, popřípadě animací.

Nejběžněji se používá **formát JPEG**. Jedná se o formát, který využívá ztrátovou kompresi a 24 bitovou barevnou hloubku. Celkem může obrázek obsahovat až 16 777 217 barev. Nepodporuje jakékoliv animace ani možnost průhledného pozadí.

Druhým velice populárním formátem je **formát PNG**. Využívá bezztrátovou kompresi a barevná hloubka může mít velikost až 32 bitů. Podporuje možnost průhledného pozadí a proto našel uplatnění hlavně ve webové grafice.

[1] Barevná hloubka udává, kolik bitů slouží pro uložení informace o barvě jednoho pixelu.

2.2 Barevné modely

Pro správné uložení obrázků lze použít mnoho barevných modelů. Jedná se o matematický model popisující barvy na základě jednotlivých složek (kanálů) modelu. Úspěšnost samotné kolorizace pomocí neuronové sítě závisí z části na tom, jaká data do sítě vstupují a následně vystupují. Lze určit kritéria, která by vhodný barevný model měl splňovat:

- Jeden z kanálů barevného modelu bude reprezentovat jasovou složku nezávisle na zbylých kanálech.
- Barevný model bude obsahovat co nejméně kanálů (pro zachování barev tři).
- Výstupní kanály budou nabývat hodnot ve stejném intervalu.

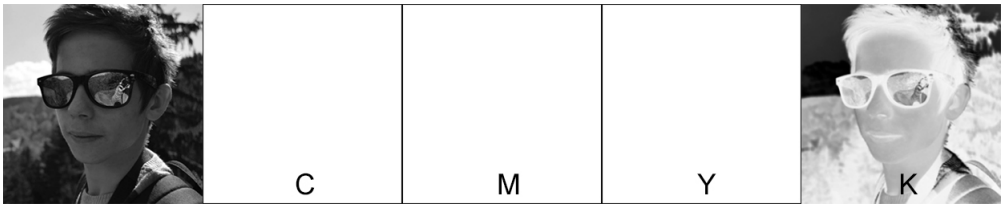
Mezi nejznámější barevné modely patří RGB a CMYK. Většina z fotografií tyto modely využívá a nabízí se tak jejich využití, aby nebylo nutné veškeré fotografie převádět do jiného barevného modelu.

2.2.1 RGB a CMYK

U barevného modelu RGB nabývají jednotlivé kanály hodnot vždy v intervalu 0 až 255. Model obsahuje pouze tři kanály, což je potřebné minimum. Problém nastává u reprezentace jasové složky, kdy jeden z kanálů by měl reprezentovat jasovou složku a po kolorování tak být neměnný. U RGB černobílý obrázek musí reprezentovat všechny tři kanály a není tak vhodné jeho použití. Zároveň výstup ze sítě nelze jakkoliv připojit k vstupním datům a využít vstupní jasovou složku [1].

Obrázek 2.1: Reprezentace černobílého obrázku v rámci barevného modelu RGB

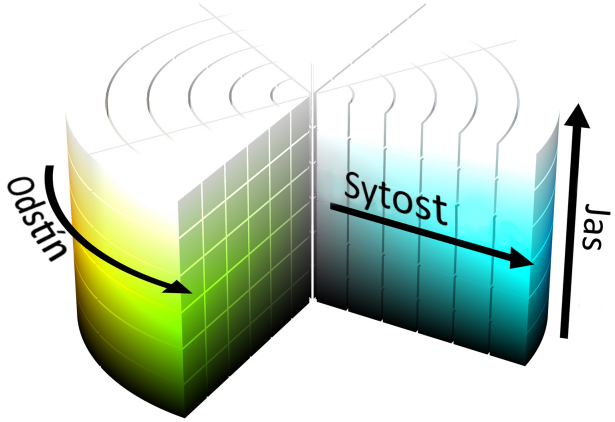
Barevný model CMYK nesplňuje požadavek třech kanálů. Oproti RGB má velkou výhodu v kanálu K, který reprezentuje jasovou složku. V případě zvolení tohoto barevného modelu, lze očekávat problémy s kvalitou kolorovaného obrázku, jelikož výstupem z neuronové sítě musí být tři kanály, nikoliv předpokládané dva.



Obrázek 2.2: Rozložení černobílého obrázku na jednotlivé kanály C, M, Y, K

2.2.2 HSL (HSB)

V počítačové grafice se velice často využívá barevný model HSL. Skládá se z odstínu (hue), kde se měří poloha na barevném kole (ve stupních 0° - 360°), dále sytost (saturation) vyjádřená v procentech 0 % (šedé odstíny) až 100 % (plná sytost). Poslední kanál zajišťuje jasovou složku (lightness) a nabývá hodnot 0 % až 100 %. Právě poslední zmíněný kanál sám dokáže vyjádřit černobílý obrázek.



Obrázek 2.3: Složení kanálů barevného modelu HSL [2]

Model se jeví jako velmi vhodný pro použití. Model se skládá pouze ze třech kanálů, jeden zajišťuje jasovou složku a zbylé dva zajišťují barevnou složku obrázku. Posledním požadavkem je stejný interval výstupních kanálů. Jako vstup do neuronové sítě slouží kanál vyjadřující jasovou složku (L). U zbylých dvou kanálů (výstup z neuronové sítě) nastává problém ve výstupních intervalech. Zatímco u saturace jsou to hodnoty mezi 0 % až 100 %, u odstínu 0° až 360° [1].

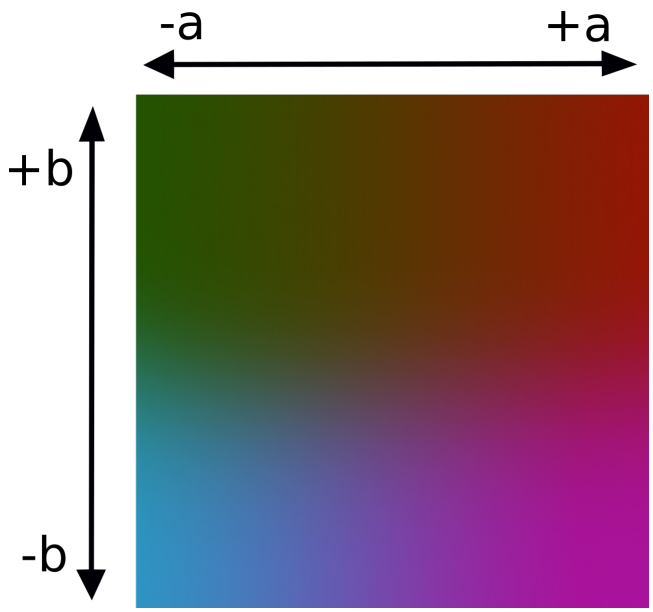
2.2.3 Lab

Posledním z vybraných modelů Lab (také CIELAB) se skládá ze třech kanálů. První kanál L zajišťuje jasovou složku a nabývá hodnot 0 - 100. Zbylé dva kanály nabývají vždy hodnot v intervalu -128 až 127:

- Kanál a - rozsah barev od zelené (-128) do červené (127)

- Kanál b - rozsah barev od modré (-128) do žluté (127)

Pomocí kanálů a, b lze vyjádřit téměř libovolnou barvu. Z popisu jednotlivých kanálů vyplývá, že splňují všechny vybrané podmínky. Skládá se pouze ze třech kanálů, jasovou složku lze vyjádřit kanálem L a barvu zbylými kanály (a, b). Zároveň oba výstupní kanály mají stejný interval hodnot [1].



Obrázek 2.4: Barevná paleta skládající se z kanálů a, b

3 Práce zabývající se problematikou kolorizace

Téma kolorizace černobílých obrázků spolu s rozvojem neuronových sítí začíná být populární. Během posledních let vzniklo několik zcela rozdílných přístupů k volbě barevných modelů, návrhu neuronových sítí a jejich trénování.

3.1 Kolorizace jako regresivní problém

Cheng, Yang a Sheng popisují ve svém odborném článku [3] neuronovou síť, která pracuje s barevným modelem YUV[1]. Na vstupu obdrží jasovou složku (Y) a jako výstup barevné složky U a V. Na problém kolorizace nahlíží jako na regresi, tedy na odhadování výstupní barvy na základě znalosti příznaků.

Neuronová síť funguje vždy pouze pro jeden pixel a skládá se ze vstupní vrstvy, kde každý pixel popisuje sada příznaků. Následně obsahuje tři skryté vrstvy, které jsou vzájemně plně propojené včetně napojení na výstupní vrstvu. Výstupní vrstva se skládá pouze ze dvou neuronů zastupující kanály U a V.

Po jednotlivých pixelech lze z neuronové sítě získat barevnou složku. Následně pro snížení šumu a zachování hran musí být aplikován bilaterální filtr. Výsledný obraz vzniká spojením chromatické a černobílé složky obrazu.

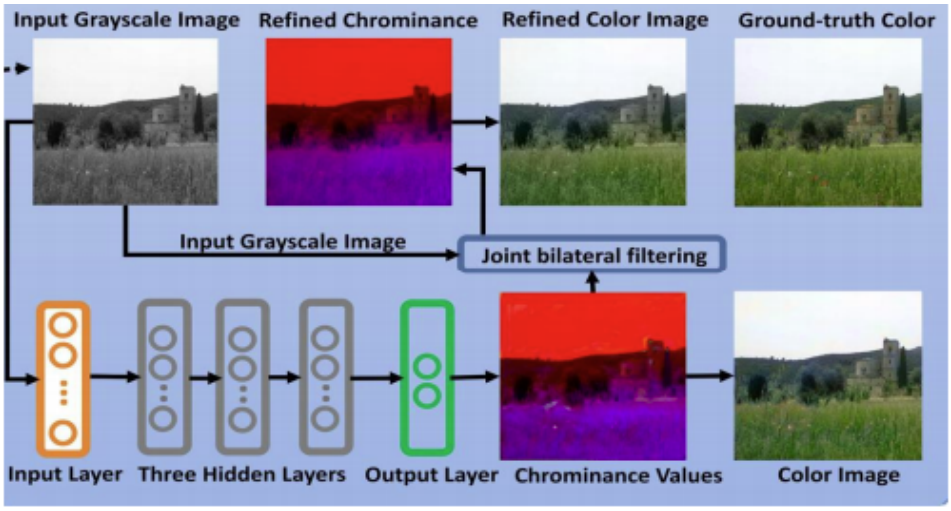
Autoři ve své práci nejprve vytvoří rozsáhlý dataset s různými motivy. Upozorňují také na možnost tvořících se artefaktů u objektů s velkým rozsahem barev. Z toho důvodu celý dataset rozdělují pomocí techniky adaptivního image clusteringu na několik kategorií, dle obsahu obrázku.

Pro každou z kategorií následně vytvoří neuronovou síť specializující se na určitou kategorii obrázku. Pro danou kategorii také vytvoří histogram, který u testovacích obrázků bude sloužit pro volbu správné neuronové sítě.

Autoři dělí vstupní příznaky obrázků do třech kategorií (low-level, mid-level, high-level). U každé kategorie představují několik metod pro správný výběr příznaků vstupního obrázku. Každá kategorie slouží pro jiný typ popisu obrázku od jednoduchých struktur (moře, obloha) až po ty komplikované (obličeje, lidé, zvířata).

Ve výsledku rozdělí 2344 obrázků do 33 kategorií a pro každou kategorii vytvoří neuronovou síť. Pro vyhodnocení využili následně 1519 obrázků. Vstupní vrstva obsahovala celkem 114 neuronů, kdy bylo využito několik metod pro volbu příznaků. Skryté vrstvy pak obsahovaly 52 neuronů a výstupní vrstva pouze požadované dva.

[1] Alternativní model k Lab. Jasovou složku vyjadřuje kanál Y a barvu U a V



Obrázek 3.1: Navržený postup kolorování obrázku [3]

Úspěšnost kolorizace shledávají autoři jako dobrou v případech, kdy u vstupního obrázku (na základě histogramu) je správně klasifikovaná kategorie.

Problém nastává u určení denní doby pořízení obrázku. S velkou pravděpodobností bude obrázek smysluplně kolorován. Nicméně v případě speciálních požadavků uživatele by bylo vhodné vytvořit více než pouze jednu univerzální neuronovou síť pro každou kategorii.

3.2 Kolorizace pomocí CNN jako klasifikační problém

V minulosti byl problém kolorizace černobílých obrázků rozdělen do tří kategorií [4].

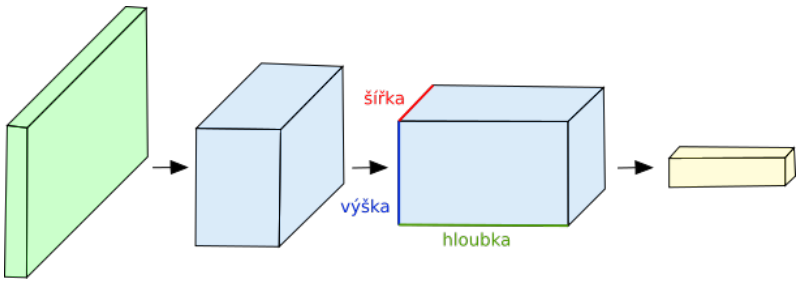
- Manuální kolorizace
- Semiautomatická kolorizace
- Pokročilá semiautomatická kolorizace

O manuální kolorizaci se jedná (v případě digitální fotografie) o nutnost kolorování každého z pixelů zvlášť za předpokladu využití lidského vidění pro odhad správné barvy.

V případě semiautomatické kolorizace usnadňuje proces kolorování segmentace určitých částí obrázků a následně jejich obarvení pomocí předdefinovaných barev, což ve většině případů stále nevede k požadovaným výsledkům. Vylepšení této techniky spočívá ve výběru podobného již barevného obrázku a na základě směrodatné odchylky výběru správné barvy ze vzorového obrázku. V roce 2010 bylo pro správný výběr vzorových obrázků využito k-means algoritmu, který obrázky dělí podle obsahu do shluků.

Pro zajištění plně automatické kolorizace využívá autor konvoluční neuronové sítě, které se skládají ze tří dimenzí (šířka, výška, hloubka). Jednotlivé vrstvy této

sítě pak mají za úkol transformovat 3D vstupní data na 3D výstupní data, na základě již naučených parametrů.

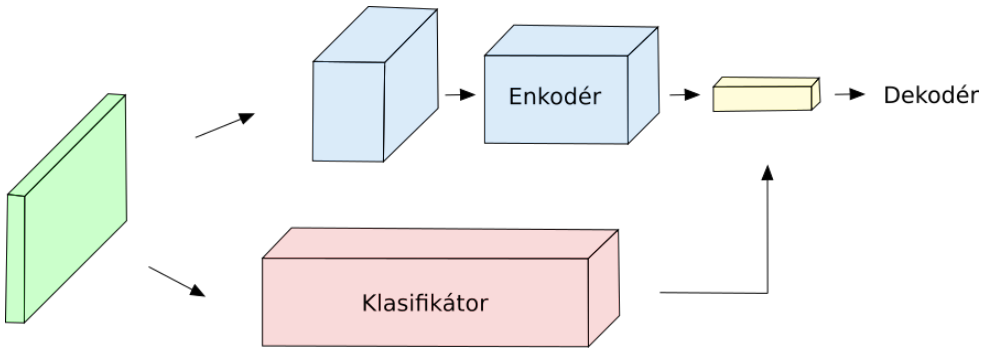


Obrázek 3.2: Extraktor příznaků

Obecnou architekturu modelu [4] neuronové sítě lze rozdělit na čtyři moduly. Vstupní data vstupují do extraktoru příznaků a zároveň do vstupní části již předtrénovaného klasifikátoru ResNetV2. Následně jsou výstupy obou modelů spojené pomocí fusion vrstvy[2], jejíž výstup se předá dále do kolorizéru.

Pro návrh výsledného modelu byl zvolen barevný model Lab. Jasová složka (L) slouží jako vstup do modelu a výstupem jsou barevné složky (a, b), které určují barvu jednotlivých pixelů.

Model se skládá z enkodéru a dekodéru. Enkodér slouží jako extraktor příznaků a skládá se z konvolučních vrstev. K výstupu z enkodéru se následně připojí výstup z již naučeného modelu pro generování příznaků ResNetV2. Dekodér zajišťuje rekonstrukci barevných složek pomocí konvolučních vrstev, jejichž rozměry odpovídají rozměrům vrstev u enkodéru, pouze v opačném pořadí. V případě nestandardního rozměru obrázku na vstupu koncová pooling vrstva zajistí zvětšení nebo zmenšení na požadované výstupní rozměry.



Obrázek 3.3: Připojení klasifikátoru na výstup z extraktoru příznaků

Jako aktivační funkce pro každou ze zvolených konvolučních vrstev byla zvolena funkce ReLU. Pouze u dekodéru v poslední vrstvě byl zvolen hyperbolický tangens s výstupními hodnotami v intervalu od -1 do 1. Tyto hodnoty lze následně snadno převést na standardní hodnoty barevného modelu Lab.

[2]Fusion vrstva převede vstupní data na vektor.

Výsledný model byl natrénován na poměrně malém datasetu, což ovlivnilo výsledné testování. Pro jednoduché obrázky (lesy, pláže) byly výsledky podobné vzorovým obrázkům. U složitějších obrázků docházelo k mírnému zabarvení, které se více blížilo původnímu černobílému obrázku než k požadovanému obarvení.

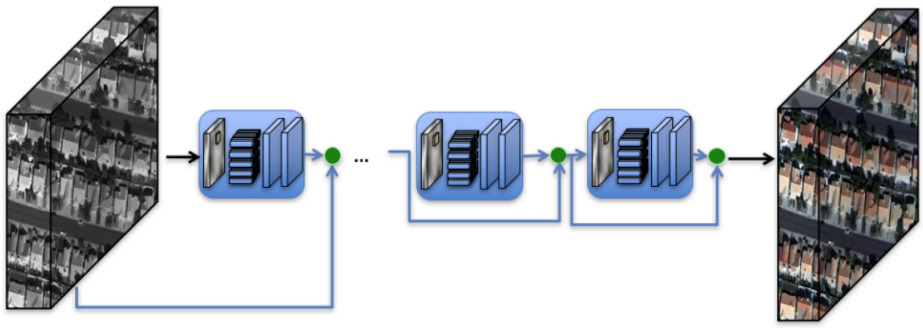
3.3 Kolorizace pomocí GAN

GAN síť [5] se skládá ze dvou modelů neuronových sítí. První z nich je generativní model, který se snaží ze vstupních dat vygenerovat správný výstupní obrázek. Druhý diskriminativní model má za úkol rozeznat, zda výstupní data z generativního modelu odpovídají těm požadovaným, nebo se jedná o nevyhovující výstup.

U zvoleného barevného modelu RGB musí generativní model vygenerovat ze vstupního černobílého obrázku všechny tři kanály (R, G, B). Diskriminativní model se naopak snaží naučit výpočet pravděpodobnosti, že výstupní obrázek odpovídá tomu požadovanému.

Generativní model nemá pouze (jako u konvolučních neuronových sítí) za cíl snížit ztrátu vypočtenou porovnáním požadovaného obrázku s vygenerovaným, ale zároveň také maximalizovat pravděpodobnost, že diskriminativní model nebude schopen rozeznat původní obrázek od výstupu z generativního modelu.

Autor zde vytváří neuronovou síť pro kolorizaci obrázků o rozměrech 64 x 64 pixelů. Generativní model se skládá z několika bloků. Každý blok obsahuje konvoluční vrstvy, jednu dekonvoluční vrstvu a jako aktivační funkci volí ReLU.



Obrázek 3.4: Návrh generativního modelu

Pro diskriminativní model je vstup také o velikosti 64 x 64 pixelů, obsahuje pět konvolučních vrstev zakončených plně propojenou vrstvou. Výstup vrací pouze jedno číslo (pravděpodobnost), zda se jedná o původní obrázek.

4 Neuronové sítě

4.1 Neuron

Základ pro každou neuronovou síť tvoří umělý neuron. K dispozici je celá řada variant modelů od jednodušších až po složité, které mohou připomínat neurony živých organismů. První a dodnes běžně používaný matematický model vytvořili roku 1943 McCulloh a Pitts.

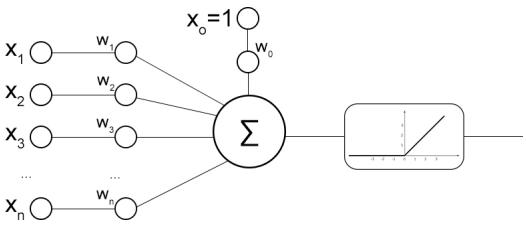
Neuron se skládá ze vstupní, funkční a výstupní části:

- **Vstupní část** obsahuje vektor vstupních dat a vah. Pomocí vah lze některá vstupní data upřednostnit, případně také potlačit.
- **Funkční část** provede součet vstupních dat vynásobených jednotlivými váhami. Následně provede jejich součet. K součtu hodnot přičteme bias (posunutí, které zajišťuje správnou funkci následné nelineární funkce). V posledním kroku aplikuje nelineární funkci.
- **Výstupní část** zajišťuje pouze přivedení dat na vstup následujícího neuronu.

Model lze matematicky popsat jako:

$$y = F(\sum_{n=0}^{C} x_n \cdot w_n) \tag{4.1}$$

kde x značí vstupní vektor, w vektor vah a F nelineární aktivační funkci. Bias je do vztahu včleněn jako váhový koeficient s indexem nula, který je násoben fiktivním příznakem s konstantní hodnotou 1 a indexem 0 [6].



Obrázek 4.1: Model neuronu

4.2 Typy neuronových sítí

Každá neuronová síť se skládá dohromady propojením několika neuronů a sítě se následně dělí podle způsobu zapojení:

- **Plně propojené sítě**, každý neuron je vždy propojen se všemi ostatními neurony a to i sám se sebou.
- **Vrstvové sítě** se skládají z několika skrytých vrstev. Každá vrstva může mít rozdílný počet neuronů. Musí obsahovat na začátku vstupní vrstvu a na konci výstupní.
- **Dopředné sítě** fungují na principu vrstvové sítě, pouze s jedním rozdílem. Výstupy každé vrstvy jsou vždy napojeny pouze na navazující vyšší vrstvu.
- **Konvoluční sítě** slouží nejčastěji pro zpracování obrazu. Fungují na principu dopředných sítí pouze s tím rozdílem, že se skládají z konvolučních vrstev, popřípadě pooling a unpooling vrstev pro dosažení nižší výpočetní náročnosti.
- **Rekurentní sítě** dokážou na rozdíl od všech ostatních typů sítí zachovávat informaci o předchozích datech ze vstupní sekvence a zohlednit ji ve výpočtech.

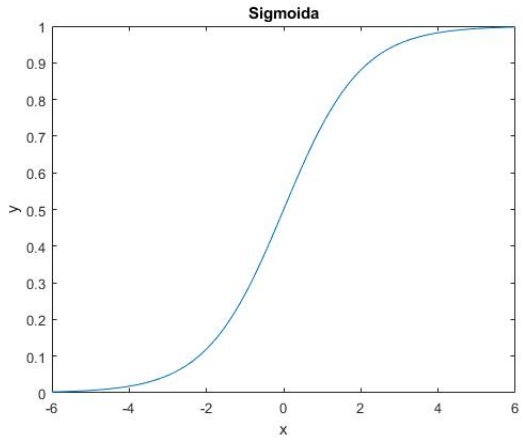
4.3 Aktivační funkce

Pro správné chování neuronu hraje klíčovou roli správně zvolená aktivační funkce [6]. Pro každý typ úlohy je vhodné použít jinou funkci. V hlubokých neuronových sítích lze i tyto funkce pro jednotlivé vrstvy kombinovat. Předpokladem každé aktivační funkce musí být její derivovatelnost tak, aby bylo možné na síť aplikovat algoritmus zpětné propagace.

4.3.1 Sigmoida

Sigmoida patří mezi nejběžnější a populární aktivační funkce. Převádí množinu reálných čísel do omezeného intervalu 0 až 1. Omezením intervalu zabraňuje nekontrolovatelnému růstu výstupních hodnot při případném velkém růstu hodnot jednotlivých vah. Tím zaručuje, že jednotlivé výstupní hodnoty nebudou mít mezi sebou velké rozdíly, což je také hlavní výhodou pro efektivní výpočty a správné fungování celé sítě [6].

$$F(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

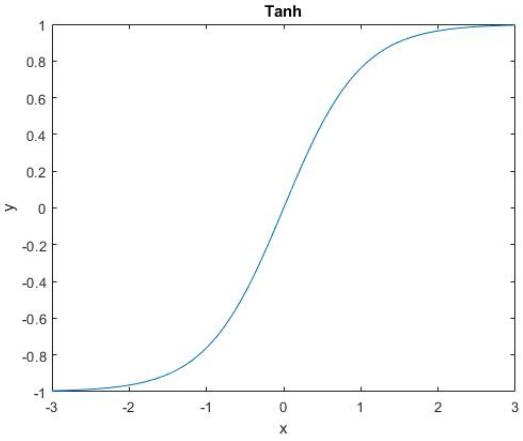


Obrázek 4.2: Průběh funkce sigmoida

4.3.2 Tanh (hyperbolický tangens)

Hyperbolický tangens má podobné vlastnosti jako sigmoida. Liší se pouze v hodnotách, které nabývá výstupní vektor x. Tyto hodnoty jsou vždy v intervalu od -1 do 1. Průměrná hodnota v tomto případě je 0 a tím zaručuje snadnou normalizaci. Funkci lze zároveň velice snadno aproximovat [6].

$$F(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{4.3}$$

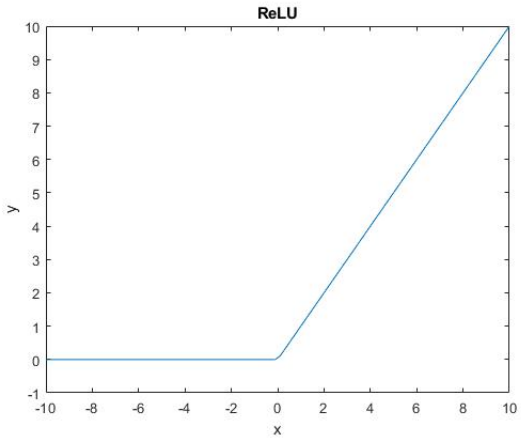


Obrázek 4.3: Průběh funkce hyperbolický tangens

4.3.3 ReLU

Funkce ReLU se hlavně u konvolučních neuronových sítí považuje za nástupce aktivační funkce sigmoid. Funkce transformuje všechny negativní hodnoty na hodnotu 0. Tato funkce navíc není výpočetně náročná, což zaručuje její rychlost. Navíc ji lze velmi snadno derivovat [6].

$$F(x) = max(0, x) \tag{4.4}$$



Obrázek 4.4: Průběh funkce ReLU

Jednou z nevýhod může být problém "dying ReLU". Jedná se o neurony, které v dopředném průchodu síti (pro libovolný vstup) vrací vždy hodnotu 0. Následně při zpětné propagaci vrací také gradient 0 a jednotlivé parametry již není možné trénovat. Jako řešení umírajících neuronů byla vytvořena funkce Leaky ReLU, která záporné hodnoty nenuluje, ale transformuje pomocí mírně rostoucí funkce [6].

4.3.4 Softmax

Softmax se nejčastěji využívá u klasifikačních úloh. Hodnoty jsou omezené v intervalu 0 až 1. Sečtením výstupního vektoru vznikne vždy v součtu hodnota 1, díky tomu každou hodnotu vektoru lze považovat za pravděpodobnost [6].

$$F(x_i) = \frac{e_i^x}{(\sum_{n=1}^{N} x_n^x)} \tag{4.5}$$

4.4 Plně propojená vrstva

Za plně propojenou vrstvu považujeme takovou vrstvu, kde vstupní data vrstvy jsou připojená na každý neuron zvolené vrstvy. Tyto vrstvy nejsou vhodné pro zpracování rozsáhlého počtu vstupních dat, vzhledem k počtu vah a množství výpočtů.

U každé vrstvy je možné si předem zvolit počet neuronů a tím také počet výstupních hodnot nezávisle na velikosti vstupních dat. V poslední době se plně propojené vrstvy využívají jako posledních několik vrstev neuronové sítě a formují koncové výstupní hodnoty.

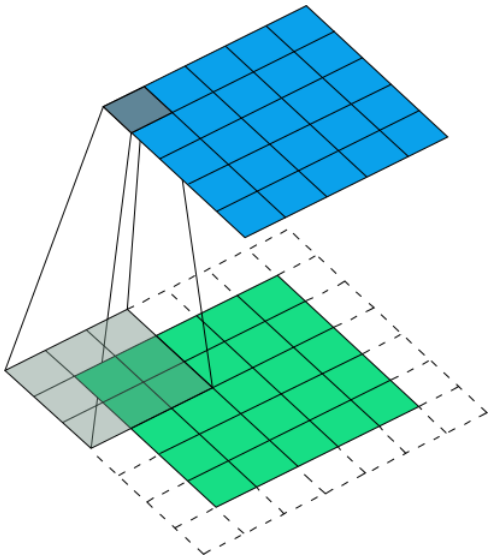
4.5 Konvoluční vrstva

Při zpracovávání obrázků pomocí neuronové sítě nastává zásadní problém velikosti obrázku (vstupních dat). Obrázky mají často více než 40000 pixelů pro každý kanál, přičemž barevný model se skládá vždy minimálně ze tří kanálů. Na vstup neuronové sítě by tak přišlo více jak 120 000 vstupních hodnot, což by mělo za příčinu obrovské množství parametrů.

Problém řeší konvoluční vrstvy. Tento typ vrstev nenačítá postupně pouze jednu vstupní hodnotu, ale části obrázku pomocí filtrů, které mají většinou rozměr 3 x 3 nebo 5 x 5. Na každou část se aplikuje skalární součin s váhami filtru, který následně ještě projde aktivační funkcí.

V každé konvoluční vrstvě jsou váhy sdílené. Pokud má vrstva filtr o rozměrech 3 x 3, jedná se dohromady pouze o 9 vah. Každá vrstva většinou obsahuje více filtrů a tedy více trénovatelných vah.

Konvoluce se provádí na všechny pozice obrázku. Klíčový je parametr posunu (také stride), který říká o kolik pixelů se výřez posune, často se tak jednotlivé výřezy překrývají.



Obrázek 4.5: Průběh konvoluce

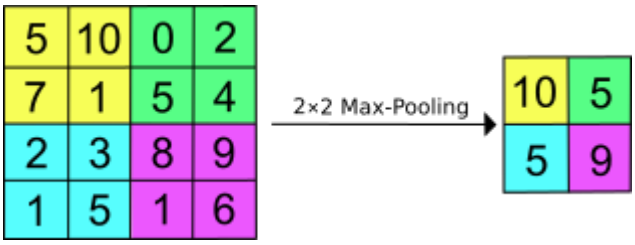
V případě využití pouze konvolučních vrstev (bez posunu) by vstup i výstup byl stále stejných rozměrů. Přesto u konvolučních neuronových sítí často vzniká požadavek na snížení vstupních rozměrů. V praxi tak jsou neuronové sítě složené z konvolučních vrstev, které jsou prokládány pooling vrstvami [7].

4.6 Pooling vrstva

Pooling vrstva se využívá, pokud je potřeba redukovat vstupní příznakovou mapu, aby se snížila náročnost učení, naopak se zvýšila informační hodnota příznaků a síť byla schopná generalizovat.

4.6.1 Max-pooling vrstva

Výpočetně nenáročnou a velice využívanou je max-pooling vrstva. Každá vrstva musí mít definovanou velikost jádra (výřez z původní příznakové mapy). Také vertikální a horizontální posun (o který se výřez posouvá). Ve většině případů se posun v obou směrech rovná právě velikosti jádra. Pro každou vybranou podoblast se vždy vybere maximální hodnota, která se zapíše do výstupní příznakové mapy. Na obrázku lze vidět příklad příznakové mapy o rozměrech 4 x 4, velikosti jádra 2 x 2 a posunu 2. Příznaková mapa se posunem dělí na čtyři podoblasti, kde je z každé vybrána maximální hodnota do výsledné příznakové mapy o poloviční rozměrech původní mapy.



Obrázek 4.6: Průběh max-pooling vrstvy

Tato vrstva není parametrická, nelze ji jakkoliv učit a v mnoha případech se tak nenávratně ztrácí klíčová informace pro následující vrstvy sítě.

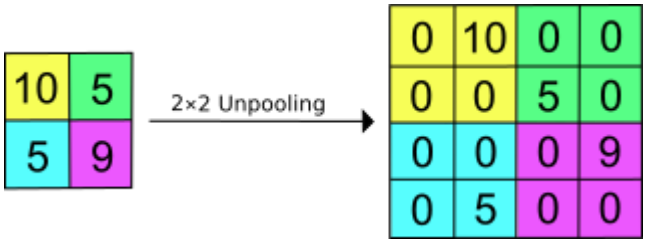
Aby bylo zamezeno ztrátě pro síť důležitých informací, využívá se pro snížení velikosti vstupní příznakové mapy konvoluční vrstva. Posun větší než jedna zajišťuje požadované omezení velikosti příznakové mapy s předpokladem, že je naučená tak, aby se důležité příznaky zachovaly [8].

4.7 Unpooling vrstva

Unpooling vrstva provádí inverzní operaci k pooling vrstvě. Tato operace má za úkol obnovení původní velikosti mapy příznaků a přibližné obnovení hodnot jednotlivých příznaků.

Její využití nacházíme u konvolučních neuronových sítí, kdy vstupní příznakovou mapu redukují pooling vrstvy a následné opětovné zvýšení rozměrů zajišťují unpooling vrstvy. Každá vrstva pro správné fungování musí mít zaznamenané pozice hodnot, které byly zvoleny u pooling vrstvy. V případě max-pooling vrstvy musí být zaznamenány pozice jednotlivých zvolených maximálních hodnot.

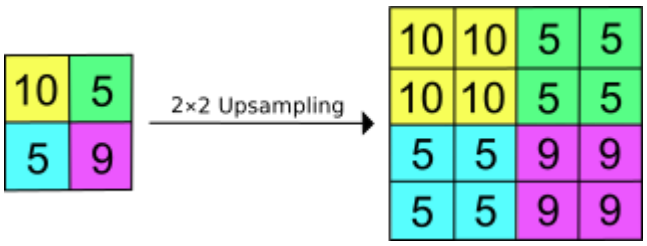
Na obrázku je zobrazen průběh rekonstrukce jednotlivých příznaků. Vrstva neobsahuje žádné naučitelné parametry. Jako jediné si pamatuje pozice původně zvolených příznaků u pooling vrstvy a zbytek hodnot vynuluje [8].



Obrázek 4.7: Průběh unpooling vrstvy

4.7.1 Upsampling vrstva

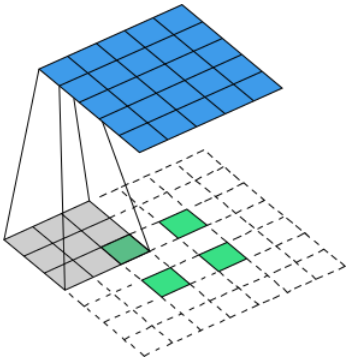
Úkolem upsampling vrstvy je rozšíření původní příznakové mapy. Tato vrstva není parametrická, nelze ji jakkoliv učit a zároveň není závislá na jakékoliv jiné vrstvě oproti pooling vrstvám. Celý princip rozšíření spočívá pouze v rozkopírování jedné hodnoty a rozšíření tak příznakové mapy na požadovaný rozměr. Vzhledem k nemožnosti učení vrstva není schopná zachovat jakýkoliv detail [8].



Obrázek 4.8: Průběh upsampling vrstvy

4.8 Transponovaná konvoluční vrstva

Transponovaná konvoluční vrstva (také dekonvoluční vrstva) je přímým opakem konvoluční vrstvy. Provádí vždy inverzní operaci ke konvoluci. Ve většině případů tato vrstva obsahuje také posun, čímž dochází k rozšíření vstupních rozměrů příznakové mapy. Na obrázku probíhá dekonvoluce, kdy z příznakové mapy o rozměrech 2 x 2 vrstva vygeneruje (na základě naučených parametrů) příznakovou mapu o rozměrech 5 x 5 [9].



Obrázek 4.9: Průběh transponované konvoluce

4.9 Algoritmus zpětné propagace

U algoritmu zpětné propagace se využívá způsob učení s učitelem. Síti se předkládá vstup a vždy také požadovaný výstup, které jsou mezi sebou porovnány.

Algoritmus zpětné propagace slouží pro trénování dopředných neuronových sítí. Před trénováním neuronové sítě jsou veškeré váhy jednotlivých vrstev inicializovány na náhodné hodnoty. Následně při každém dopředném průchodu neuronovou síti se spočítá výstup ze sítě. Následně se provede (na základě předem dané ztrátové funkce) porovnání výstupu ze sítě s požadovaným výstupem. Výstup ze ztrátové funkce představuje ztrátu (loss) sítě.

Ztráta se pak propaguje zpět do sítě a upravují se jednotlivé váhy vrstev tak, aby došlo ke zmenšení ztráty sítě. Tento postup se provádí tak dlouho, dokud nedojde k minimalizování ztráty sítě [10].

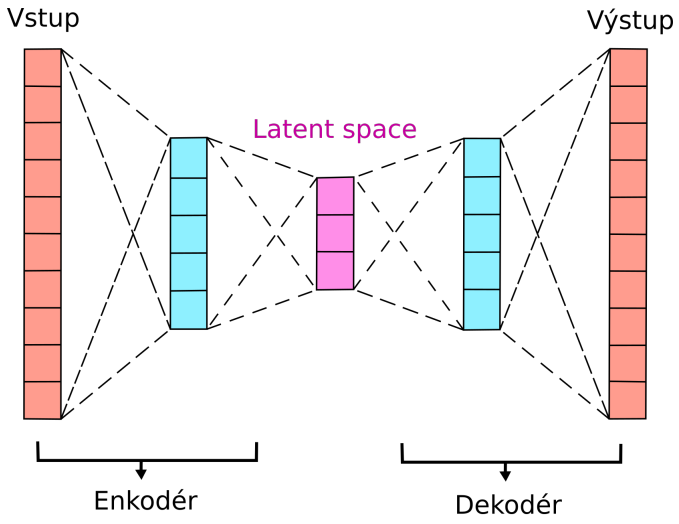
4.10 Autoenkodér

Autoenkodér je typickým příkladem dopředné neuronové sítě. Má za úkol transformovat vstupní hodnoty opět na tytéž hodnoty prostřednictvím několika transformací, přičemž cílem je nechat síť, aby se sama naučila tyto transformace. V první části (enkodér) dochází ke zmenšení velikosti příznakových map a zároveň zvýšení jejich počtu. V druhé části (dekodér) dochází ke zpětnému převodu příznakových map do původních rozměrů. Síť má za úkol se naučit pracovat tak, aby docházelo co k nejmenším rozdílům mezi vstupem a výstupem. Během učení jsou tyto rozdíly porovnávány pomocí ztrátové funkce. Následně algoritmus zpětné propagace zajistí úpravu vah v jednotlivých vrstvách.

Snížením rozměrů vrstev mezi vstupem a výstupem síť nedokáže pouze transformovat vstup na výstup, ale musí generalizovat. Generalizací následně síť dokáže zpracovat i data, které nikdy neviděla s velkou úspěšností.

Pojmem latent space se u autoenkodérů označuje datová reprezentace zkomprimovaných dat. Jedná se o nejdůležitější příznaky, které dokáží reprezentovat a charakterizovat vstupní data.

Princip autoenkodérů lze využít při tvorbě konvoluční neuronové sítě a vytvořit tak konvoluční autoenkodér. Enkodér tvoří konvoluční vrstvy a zároveň pooling vrstvy, které zajišťují snížení velikosti příznakových map. V druhé části je dekodér vytvořen zcela symetricky vůči enkodéru. Liší se pouze v nahrazení pooling vrstev unpooling, popřípadě upsampling vrstvami. Ty zajišťují zpětnou rekonstrukci příznakových map do původních rozměrů [11].



Obrázek 4.10: Schéma autoenkodéru

5 Výběr frameworku

Při tvorbě neuronových sítí máme na výběr ze dvou možností. První z nich je naprogramování si vlastních vrstev neuronové sítě, způsobu trénování a zpětné propagace. Tento způsob lze považovat v dnešní době prudkého rozvoje za značně neefektivní a samozřejmě také více náchylný na chyby. Druhou možností je využití již předpřipravených frameworků, které v sobě implementují vše potřebné pro tvorbu a učení neuronových sítí. Na vývojáře tak zbývá navržení architektury neuronové sítě, způsobu trénování a tvorba datasetu.

Pro tvorbu neuronových sítí je klíčový jazyk Python, pro který jsou vytvořeny tři základní frameworky:

- Tensorflow - vyvinut společností Google
- Pytorch - vyvinut společností Facebook
- Keras - rozhraní pro Tensorflow vyvinuté na univerzitě MIT

Vzhledem k předchozím zkušenostem s frameworkem Keras byl zvolen i pro tvorbu neuronové sítě v této bakalářské práci. Oproti samotnému Tensorflow (ze kterého vychází) má několik výhod [12]:

- Konzistentní rozhraní optimalizováno pro většinu běžných použití neuronových sítí.
- Přehledný a vypovídající výpis chybových hlášení.
- Architektura sítě se vytváří spojováním předdefinovaných bloků.
- Snadné vytváření nových bloků neuronové sítě.

5.1 Instalace

Samotný framework Keras se instaluje v rámci balíčku Tensorflow. Před instalací je nutné si vybrat, zda budoucí neuronová síť bude trénována pomocí procesoru nebo jedné či více grafických karet.

V případě trénování pomocí procesoru se volí instalační balíček “tensorflow”. V případě potřeby využití grafické karty, musí být splněno několik předpokladů:

- Nainstalována nejnovější CUDA[1] verze 11.1.
- Grafická karta s dostatečně velkou VRAM pro danou úlohu.
- Všechny grafické karty (na kterých bude trénováno) musí být stejné.

V případě splnění veškerých předpokladů lze instaloval balíček “tensorflow-gpu”.

5.2 Základní struktura

Před samotnou tvorbou architektury neuronové sítě a jejím trénování, musí být předpřipravená kolekce dat (dataset). Framework Keras požaduje jako formát datasetu Numpy[2] pole, kde datový typ jednotlivých hodnot je buď Float16 nebo Float32, v závislosti na požadované přesnosti uchovaných čísel. Rozměry dat datasetu také navíc určují vstupní rozměr první vrstvy neuronové sítě.

Při tvorbě modelu si lze vybrat ze dvou návrhových typů:

Sekvenční

Jedná se o jednodušší návrh architektury neuronové sítě, který ve většině návrhů dostačuje. V prvním kroku se inicializuje samotný sekvenční model, do kterého se následně vkládají vrstvy. Problém nastává v případě potřeby zapojení jednotlivých vrstev jinak než sekvenčně za sebou (např. paralelně). V tom případě se musí využít funkční návrhový typ.

Funkční

V případě funkčního návrhu má vývojář naprostou volnost v napojení jednotlivých vrstev. Lze tak jakkoliv spojit výstup a vstup libovolných vrstev (za předpokladu odpovídajících rozměrů vstupů a výstupů).

Při správně navrženém modelu si lze následně zobrazit architekturu celé sítě včetně počtu trénovatelných parametrů.

```
_________________________________________________________________
Vrstva (typ)                 Výstupní rozměry          Parametry
=================================================================
conv2d (Conv2D)              (None, 256, 256, 32)      320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      18496
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 256, 256, 32)      9248
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 256, 256, 16)      4624
```

[1] Software, který umožňuje spouštět vybrané programy napsané v C++ na grafické kartě
[2] Knihovna sloužící pro usnadnění práce s čísly v programovacím jazyce Python

```
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 256, 256, 2)       290
=================================================================
Počet parametrů: 32,978
Trénovatelných parametrů: 39,978
Netrénovatelných parametrů: 0
```

Dále je důležité nastavit správné parametry pro trénování [12]:

- Optimizér
- Velikost dávky - určuje, po průchodu jakého počtu obrázků budou upraveny parametry sítě
- Ztrátová funkce
- Počet epoch - počet průchodů celé trénovací sady neuronovou sítí
- Předčasné ukončení

5.3 Ztrátová funkce (loss funkce)

Pro porovnání výstupních hodnot a výpočet chyby mezi výstupem neuronové sítě a očekávaným výstupem se využívá ztrátová funkce. Cílem učení neuronové sítě je snaha vypočtenou ztrátu co nejvíce minimalizovat. Vypočtená ztráta se následně propaguje zpět neuronovou síti s cílem upravení vah.

Pro každý typ architektury (klasifikace, regrese, segmentace) se využívá rozdílných ztrátových funkcí. V této práci se jedná o regresi, pro kterou se typicky využívá různých typů odchylek.

5.3.1 Střední kvadratická odchylka (MSE)

Střední kvadratická odchylka vypočítá součet druhých mocnin rozdílů mezi skutečnými a predikovanými hodnotami pixelů. V případě velkých rozdílů (v predikovaných a skutečných datech) druhá mocnina ještě více posílí výslednou ztrátu [13].

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(d_j - x_j)^2 \tag{5.1}$$

Jednotlivé hodnoty v rovnici vyjadřují:

- d_j skutečná hodnota j-tého pixelu
- x_j predikovaná hodnota j-tého pixelu
- N počet vstupních pixelů

5.3.2 Střední absolutní odchylka (MAE)

Střední absolutní odchylka vypočítá součet absolutní hodnoty rozdílu mezi skutečnými a predikovanými hodnotami pixelů. Na rozdíl od MSE (v případě velkých rozdílů) nevygeneruje tak velikou ztrátu, což zhoršuje výsledné trénování neuronové sítě [13].

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|d_j - x_j| \tag{5.2}$$

5.4 Optimizér

Jako optimizér lze považovat algoritmus, který se u neuronové sítě stará (v průběhu učení) o nalezení nejoptimálnějších vah jednotlivých vrstev. Bližší vysvětlení jednotlivých optimzérů lze nalézt v odborných článcích [14, 15].

5.4.1 Metoda největšího spádu (gradient descent)

V optimálním případě by v každém kroku neuronová sít (na základě zpětné propagace) vypočítala gradient pro každou vrstvu. Gradient se následně přenásobí krokem učení[3] a odečte se od původních hodnot parametrů sítě. Proces se opakuje do té doby, než ztráta přestane klesat [14, 15].

$$\theta_{t+1} = \theta_t - \gamma * \triangle L(\theta_t) \tag{5.3}$$

Jednotlivé hodnoty v rovnici reprezentují:

- θ_t aktuální odhad parametrů
- γ velikost kroku učení
- $L(\theta_t)$ gradient (směr největšího vzrůstu) pro aktuální odhad parametrů

5.4.2 Momentum SGD (stochastic gradient descent)

V obecných případech všechny váhy sítě neovlivňují výslednou ztrátu stejně a samotné učení bude pomalé a neefektivní. První z řešení bylo pamatování si předchozího kroku úpravy parametrů a přičtení k aktuálně vypočtenému gradientu. Tím se výrazně zvýší rychlost nalezení globálního minima [14, 15].

$$\theta_{t+1} = \theta_t + \alpha \cdot v_t - \gamma * \triangle L(\theta_t) \tag{5.4}$$

Jednotlivé hodnoty v rovnici reprezentují:

- α hyperparametr (nutné určit)
- $\alpha \cdot v_t$ hybnost (přeškálovaná minulá úprava parametrů)

[3] Krok učení (learning rate) určuje, o jakou velikost se mění parametry sítě.

5.4.3 RMSprop (root mean square propagation)

V každém kroku se vypočítá gradient stejné jako u SGD. Krokem učení se následně upravuje pro každý parametr zvlášť a zároveň se sčítají jejich kvadráty. Pokud se v následných krocích gradient některého směru zvyšuje nerovnoměrně oproti jiným směrům, dochází k normalizaci tohoto směru.

$$u_{t+1} = \beta \cdot u_t + (1 - \beta) \cdot (\triangle L(\theta_t))^2 \tag{5.5}$$

Jednotlivé hodnoty v rovnici reprezentují:

- u_{t+1} průměrná norma gradientů
- β hyperparametr (nutné určit)

Při úpravě parametrů lze aplikovat postup stejný jako u SGD pouze s tím rozdílem, že vypočítaný gradient je normalizován tak, aby se gradient ve všech směrech posunul v průměru o stejný podíl [15].

$$\theta_{t+1} = \theta_t - \gamma * \frac{\triangle L(\theta_t}{\sqrt{u_{t+1} + \varepsilon}} \tag{5.6}$$

5.4.4 Adaptive momentum (Adam)

Jedná se o nejrozšířenější a nejlepší výchozí volbu při trénování neuronových sítí. Dochází ke kombinaci Momentum SGD a RMSprop. Ve většině případů funguje i s výchozím (zvolené frameworkem) nastavením parametrů [14, 15].

$$\theta_{t+1} = \theta_t - \gamma * \frac{v_{t+1}}{\sqrt{u_{t+1} + \varepsilon}} \tag{5.7}$$

5.5 Předčasné ukončení

Při malém datasetu a velké neuronové síti může docházet k přeučení sítě na trénovacích datech (tzv. overfitting). Typicky tak v určitém bodě vypočtená ztráta na validačních datech přestane klesat a naopak začne výrazně stoupat.

Aby bylo zabráněno tomuto stavu, využívá se předčasného zastavení, kdy se sleduje vypočtená hodnota ztráty na validačních datech a v případě, že ztráta začne stoupat, trénování se ukončí a zabrání se možnému přeučení sítě [16].

6 Tvorba datasetu

Pro správné trénování neuronové sítě je klíčové zvolit vhodné motivy obrázků tak, aby byla síť schopná kolorovat co nejrozsáhlejší množství obrázků. Pro tvorbu datasetu byly zvoleny následující kategorie:

- Portréty (ženy, muži, děti, herci)
- Krajiny (lesy, pláže, hory, květiny, vodopády, ostrovy)
- Zvířata (zoo, savana)
- Města (budovy, auta, ulice)

V odborných publikacích lze dohledat odkazy na velké množství již vytvořených datasetů. Některé jsou přímo připravené pro úlohy kolorizace obrázků, jiné obsahují pouze barevné obrázky, ze kterých se musí potřebné černobílé obrázky vytvořit. Příkladem může být dataset Image Colorization[1] vytvořen z celkem 25 000 obrázků o rozměrech 224 x 224 pixelů.

Zadání této bakalářské práce určuje rozměr obrázků 256 pixelů na šířku a 256 pixelů na výšku. Výše uvedený dataset by tak neodpovídal minimálním požadovaným rozměrům. Jiné datasety naopak měly požadovanou výšku a šířku několikanásobně větší a tím i větší požadavky na prostor na disku.

Vzhledem k těmto skutečnostem jsem zvolil jako nejjednodušší a nejrychlejší řešení vytvoření vlastního datasetu. Výhodou je úplná kontrola nad rozměry a počtem obrázků v jednotlivých kategoriích.

6.1 Stažení dat

Jako zdroj obrázků jsem zvolil webový server Google Images. Pro stahování většího množství obrázků ze serveru Google existuje již vytvořená Python knihovna "google_images_download".

Při inicializaci této knihovny se zvolí klíčové parametry, podle kterých budou obrázky vyhledány a následně také staženy.

- **Keywords** obsahuje seznam klíčových slov, která mají být zahrnuta při vyhledávání.

[1] Dataset vytvořen uživatelem shravankumar989 na portálu Kaggle.com

- **Format** určuje požadovaný datový formát každého souboru.
- **Limit** nastavuje, kolik obrázků má být pro zadaná klíčová slova staženo.
- **Size** slouží jako kritérium rozměru jednotlivých obrázků.
- **Aspect ratio** nastavuje poměr výšky a šířky vyhledávaných obrázků.
- **Color type** určuje, zda se má jednat o černobílý obrázek nebo plně barevný.
- **Type** zajišťuje, zda se jedná o fotografií nebo malbu.

Vzhledem k úpravám, které byly ze stran společnosti Google provedeny není možné (od roku 2020) vyhledat více jak 400 obrázků najednou. Původní knihovnu tak bylo nutné rozšířit o část, která bude vždy knihovnu inicializovat s číslovanými klíčovými slovy. Tím jsou z velké části zaručené rozdílné výsledky vyhledávání.

```
def downloadImages(numOfIters, keywords):
   instance = google_images_download()
      for num in range(numOfIters):
        instance.keywords = keywords + num
        instance.download()
        ...
```

Jednotlivé parametry vyhledávání jsem nastavil tak, aby co nejvíce odpovídala požadovaným datům pro trénování neuronové sítě a byly zvoleny následovně:

- **Format**: jpg, png (pro snadné načtení dat a podporu jednotlivých knihoven)
- **Limit**: 400 (maximální množství obrázků stažitelné jednou iterací)
- **Size**: medium (rozměry v rozsahu 300 - 500 pixelů)
- **Aspect ratio**: square (výška a šířka musí mít stejné rozměry).
- **Color type**: full-color (obrázek je požadován v barevné podobě, nikoliv černobílé)
- **Type**: photo (obrázek musí být fotografií)

Po stažení se musí provést kontrola, zda obrázky odpovídají veškerým požadovaným argumentům. Jednotlivé obrázky často mají metadata[2], která buď některý z argumentů vůbec neobsahují, popřípadě je špatně vyplněn.

Mezi nejčastější vady patří:

- Obrázek obsahuje vodoznak.
- Obrázek je černobílý.
- Výška nebo šířka obrázku neodpovídá minimálnímu rozměru 256 x 256 pixelů.

Pro konzistenci datasetu jsou veškeré obrázky obsahující některou z vad nevyhovující a dataset tyto obrázky nesmí obsahovat.

[2]Metadata jsou skrytá data, která poskytují doplňující informace o obrázku.

6.2 Normalizace dat

Pro uložení datového setu pro potřebu učení musí být data správně připravená a uložena. Tento proces se skládá z několika kroků:

- Změna velikosti obrázku na požadované rozměry.
- Převedení datasetu do správného barevného modelu.
- Rozdělení barevných kanálů.
- Vygenerování datasetu.

Pro předzpracování dat byl zvolen ImageDataGenerator sloužící pro načtení veškerých obrázků, jejich normalizaci a přeškálování na správné rozměry. Veškeré načtené obrázky jsou standardně v barevném modelu RGB. Jejich normalizace se zajistí vydělením každé hodnoty pixelu hodnotou 255 (maximální velikost). Tím se docílí rychlejších výpočtů (díky operacím s menšími rozsahy čísel).

Zároveň generátor přijímá jako vstupní parametr výšku a šířku vstupního obrázku. Na jeho základě podle potřeby obrázek zmenší nebo zvětší tak, aby odpovídal požadované velikosti.

Generátor jako výstup připraví list obsahující jednotlivé obrázky v barevném modelu RGB a s normalizovanými daty. V druhé kapitole této práce byl jako nejlepší barevný model zvolen barevný model Lab, jednotlivé obrázky je tak nutné převést. K převodu se využívá funkce rgb2lab. Pro uložení veškerých obrázků je využita knihovna Numpy, pomocí které se implementuje pole obrázků obsahující čtyři dimenze (počet obrázků, výška obrázku, šířka obrázku, počet kanálů).

Obdržené normalizované hodnoty jednotlivých pixelů jsou desetinná čísla. Jako datový typ pro uložení dat jsem proto zvolil Float16. Veškerá data se následně ukládají do souboru. Výsledný dataset obsahuje celkem 75 000 obrázků.

6.3 Augmentace dat

V případě, že dataset neobsahuje dostatečný počet obrázků, využívá se augmentace dat, neboli umělého rozšíření původního datasetu. V případě kolorizace je důležité mít co nejvíce odlišných obrázků, aby síť dokázala správně generalizovat.

Při augmentaci dat bylo zvoleno pouze horizontální překlopení. Výsledný počet obrázků v datasetu byl rozšířen na dvojnásobek, tedy na 150 000 obrázků.

7 Vlastní implementační a experimentální práce

Před samotným návrhem autoenkodéru jsem musel určit poměr, který rozdělí dostupná data (150 000 obrázků) na čtyři části:

- Trénovací data pro trénování samotné CNN **(75 %)**
- Validační data pro vyhodnocování ztráty CNN v průběhu učení **(10 %)**
- Validační data pro výběr nejlépe zvoleného modelu **(5 %)**
- Testovací data pro zjištění ztráty již natrénovaného modelu **(10 %)**

Zároveň bylo potřebné také určit, jestli se model bude trénovat na CPU nebo GPU. K dispozici byl počítač s operačním systémem Windows a specifikací:

- Procesor Intel i7-9700K
- Grafická karta Nvidia RTX 3070
- 32 GB 2667 MHz RAM

Počítač obsahoval již nainstalovaný software Anaconda určený pro distribuci programovacího jazyka Python. Doinstalovat jsem musel knihovnu "tensorflow-gpu", obsahující framework Keras a podporu trénování neuronové sítě pomocí GPU.

7.1 Vyhodnocovací metrika

Pro rozlišení přesnosti modelů, musí být zvolena vhodná metrika, pomocí které bude možné porovnat původní a výstupní obrázek. U klasifikačních úloh se snadno porovnají hodnoty vyjadřující odhadovanou a očekávanou třídu. Úspěšnost se následně vyjadřuje procentuálně.

V případě regrese není možné určit procentuální přesnost, protože výstupem z modelu není jedna konkrétní třída, ale hodnota z intervalu. Jako vyhodnocovací metrika proto slouží ztrátová funkce a modely se porovnávají pomocí ztrát. Čím nižší ztráta, tím je model přesnější.

7.2 Počáteční modely

V kapitole 3 (zmiňující práce jiných autorů) byl problém kolorizace vyřešen buď pouze jako regresivní problém, popřípadě navíc se zapojením klasifikátoru.

V případě regresivního modelu se model skládá z enkodéru, který z původního obrázku získá klíčové informace pro zpětnou rekonstrukci v dekodéru. Rozšíření o klasifikaci by spočívalo ve spojení výstupu z klasifikátoru a enkodéru. Tím by se dekodéru přidala navíc důležitá informace o typu obrázku a zjednodušil následný odhad barev v dekodéru.

Vzniklo několik otázek, které bylo nutné postupně ověřit:

- Potřebují obrázky o velikosti 256 x 256 pixelů klasifikační neuronovou síť pro správné kolorování?
- Jak nejvýhodněji snížit rozměry příznakových map v enkodéru?
- Jak nejvýhodněji zvýšit rozměry příznakových map v dekodéru?
- Lze neuronovou síť natrénovat na počítači s jednou GPU a 32 GB RAM?

V prvním kroku jsem se rozhodl pro tvorbu dvou nezávislých modelů, které vždy budou trénované zvlášť. Zároveň na nich bude možné porovnat dosažené výsledky. Modely se budou liší pouze v hloubce samotné neuronové sítě (v počtu vrstev).

7.2.1 Návrh modelů

U prvního modelu jsem využil znalost již vytvořených prací a model navrhl následovně:

- Sekvenční model - jednotlivé vrstvy budou na sebe navazovat v jedné sekvenci bez jakýchkoliv spojení navíc.
- Pro všechny konvoluční vrstvy jsem zvolil aktivační funkci ReLU s výjimkou poslední vrstvy, která jako aktivační funkci využije hyperbolický tangens z důvodu požadovaného výstupu v intervalu od -1 do 1.
- Rozměry příznakových map budou sníženy celkem čtyřikrát, vždy na polovinu předchozích rozměrů.

Pro snížení rozměru příznakových map jsem zvolil neparametrické max-pooling vrstvy. U max-pooling vrstvy navíc výstupní příznakové mapy neprochází aktivační funkcí. Pro správnou interpretaci zmenšených příznakových map následuje (za každou max-pooling vrstvou) konvoluční vrstva obsahující aktivační funkci. Pro zvětšení rozměru příznakových map jsem zvolil upsampling vrstvu. Výsledný navržený model vypadal následovně:

```
_________________________________________________________________
Vrstva (typ)                 Výstupní rozměry          Parametry
=================================================================
conv2d (Conv2D)              (None, 256, 256, 32)      320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 512)       1180160
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 256)       1179904
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 128)       295040
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 64, 64, 64)        73792
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 128, 128, 32)      18464
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 256, 256, 32)      0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 256, 256, 16)      4624
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 256, 256, 2)       290
=================================================================
Počet parametrů: 3,140,114
Trénovatelných parametrů: 3,140,114
Netrénovatelných parametrů: 0
```

Návrh byl navíc rozšířen o první konvoluční vrstvu, která má za cíl převést vstupní černobílý obrázek (obsahující pouze jednu příznakovou mapu) na 32 příznakových map. Tedy z rozměru 256 x 256 x 1 na rozměr 256 x 256 x 32.

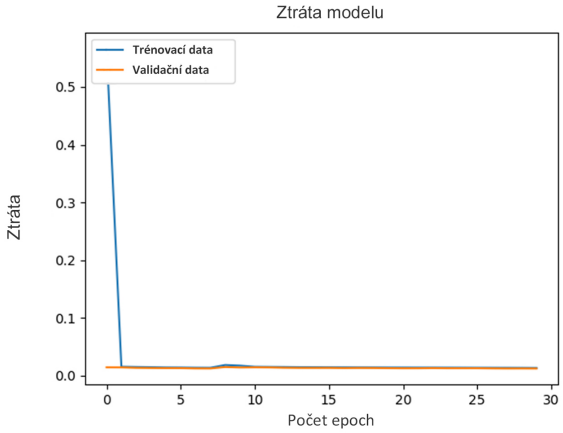
7.2.2 Trénování modelů

Následně jsem zvolil parametry trénování. Jako optimizér parametrů byl zvolen Adam s výchozím krokem učení 10^{-4}. Dále jako ztrátová funkce byla zvolena MSE vzhledem k očekávaným velkým rozdílů (v počátcích trénování) mezi původními a predikovanými hodnotami výstupních kanálů.

Pro trénování byla zvolená velikost dávky 128 (počet obrázků, po kterých se aktivuje algoritmus zpětné propagace) a počet epoch na 30. Výsledný počet spuštění alogiritmu zpětné propagace lze snadno získat následujícím vzorcem, kde o značí celkový počet obrázků pro trénování, d velikost dávky a e počet epoch.

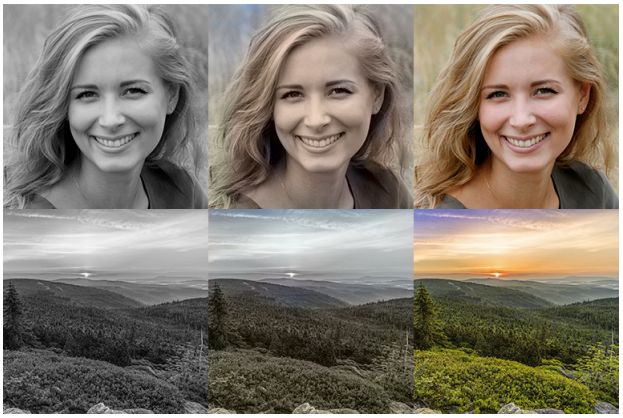
$$y = \frac{o}{d} \cdot e \tag{7.1}$$

S již navrženým modelem a zvolenými parametry pro trénování bylo možné trénování spustit. První pokus byl spuštěn na datasetu o velikosti 10 000 obrázků s následujícími výsledky:



Obrázek 7.1: Vývoj ztráty modelu v průběhu trénování prvního modelu

Z výsledku vyplývá patrný nedostatek obrázků, který vedl k neúspěšnému trénování. To dokazuje graf, kde ztráta neklesá, ale stagnuje na stejné hodnotě. Při pokusu kolorování černobílého obrázku lze pozorovat pouze nepatrné zabarvení.



Obrázek 7.2: Porovnání výstupů prvního trénovaného modelu s originálním obrázkem (zleva černobílý, kolorovaný a originální obrázek)

Při trénování dalších modelů jsem již využil celý dataset. Při opakovaném spuštění trénování však nastalo několik zásadních problémů:

- Nelze využít generátor dávek z frameworku (z důvodu velikosti datasetu).
- Nelze uložit celý dataset do RAM paměti (následuje pád programu).
- Doba trénování jedné epochy přesahuje třicet minut.

Generátor dávek

Generátor dávek vytvořený frameworkem nebylo možné (kvůli velikosti) využít pro vytvořený dataset. Vytvořil jsem si proto vlastní generátor, který při zavolání poskytuje určitý počet obrázků stanovený velikostí dávky. Zároveň pro zlepšení a zrychlení trénování byly obrázky v datasetu při každé nové epoše zamíchané. Stejný generátor slouží také pro generování validačních dávek.

```
def batch_generator(array_size, batch_size = 96, is_train=False):

    global L_train
    global ab_train
    global L_val
    global ab_val

    while True:
            if is_train:
                p = np.random.permutation(L_train.shape[0])
                L_train = L_train[p]
                ab_train = ab_train[p]
                X = L_train
                Y = ab_train
            else:
```

```
        X = L_val
        Y = ab_val

    for i in range(array_size):
        start = i*batch_size
        end = start + batch_size + 1
        yield X[start:end], Y[start:end]
```

Trénování pomocí GPU clusteru

Jediným možným řešením nedostatku RAM paměti a pomalého trénování bylo hardwarové vylepšení. Vzhledem k velikosti datasetu, nebylo možné se snažit o vylepšení výkonu na stávajícím počítači využitém pro trénování. Ústav informační technologie a elektroniky má k dispozici několik výpočetních GPU clusterů (s dostatečným výkonem) využitelných pro úlohy trénování neuronových sítí.

Po zažádání mi byl udělen přístup ke clusteru GPU2a o následující specifikaci:

- Procesor Intel Xeon E5-2590
- 8 grafických karet Nvidia GTX 1080 Ti
- 504 GB RAM

Ke clusteru lze přistupovat pouze využitím SSH[1] přístupu k příkazové řádce a SFTP[2]. Stejně jako u původního počítače bylo nutné nejdříve nainstalovat software Anaconda pro distribuci jazyka Python a následně balíček tensorflow-gpu. Cluster umožňuje využití paralelních výpočtů na všech grafických kartách současně. Paralelními výpočty dochází k výraznému zvýšení rychlosti trénování modelu.

Po vyřízení veškerých hardwarových nedostatků bylo již možné spustit trénování s následujícími výsledky:

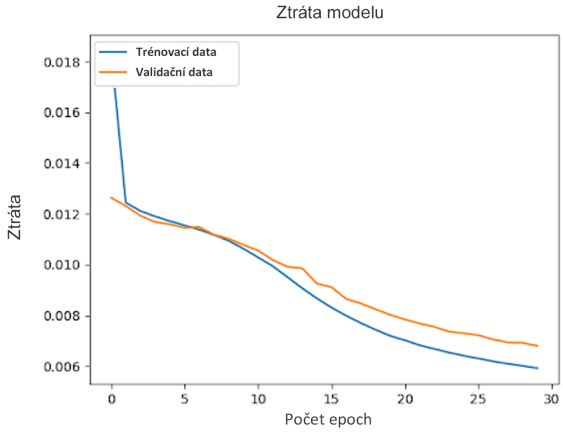
Z grafů jasně vyplývá postupné snižování ztráty jak u trénovacích, tak i validačních dat. Pokud jsem porovnal kolorované obrázky vygenerované z 7. a 30. epochy, lze pozorovat u 30. epochy více artefaktů, které ovlivňují výsledný barevný obrázek.

Tento jev je typickým příkladem přetrénování sítě (overfitting). V tomto případě nelze využít předčasného ukončení, protože nedochází ke zvýšení ztráty na validačních datech. Pro tento případ jsem využil druhý validační dataset, pomocí kterého jsem u modelu na každé epoše ověřil ztrátu. Nejmenší ztrátu měl model ze sedmé epochy, kde navíc dochází k protnutí validační a trénovací křivky ztráty.

7.2.3 Druhý model

Druhý model se strukturou téměř neliší od navrženého prvního modelu, pouze dojde ke zmenšení a následné zvětšení velikosti příznakových map celkem pětkrát. V původním modelu se velikost příznakových map snížila celkem čtyřikrát na rozměr 16

[1] Zabezpečený komunikační protokol.
[2] Zabezpečený protokol pro přenos souborů.



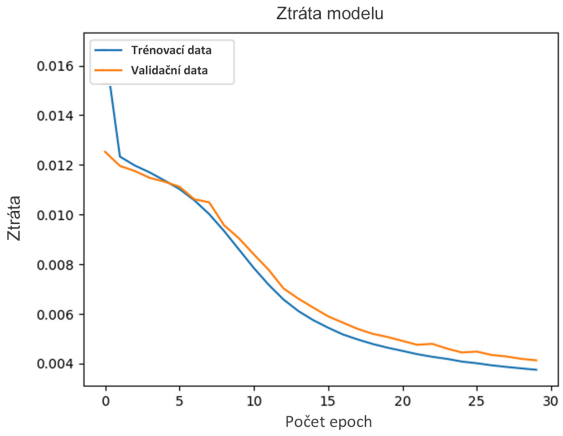
Obrázek 7.3: Vývoj ztráty v průběhu trénování prvního modelu na celém datasetu

Obrázek 7.4: Porovnání výstupů prvního modelu (trénovaného na celém datasetu) s originálním obrázkem (zleva černobíly obrázek, kolorovaný obrázek ze sedmé epochy, originální obrázek, kolorovaný obrázek z třicáté epochy)

x 16 pixelů. V aktuálním modelu se provede snížení pětkrát s výsledným rozměrem 8 x 8 pixelů. Do enkodéru jsem přidal max-pooling a konvoluční vrstvu, naopak do dekodéru upsampling a konvoluční vrstvu.

```
_________________________________________________________________
Vrstva (typ)                 Výstupní rozměry          Parametry
=================================================================
...
_________________________________________________________________
max_pooling2d_4 (MaxPooling2) (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_5 (Conv2D)             (None, 8, 8, 1024)        4719616
_________________________________________________________________
conv2d_6 (Conv2D)             (None, 8, 8, 512)         4719104
_________________________________________________________________
up_sampling2d (UpSampling2D)  (None, 16, 16, 512)       0
_________________________________________________________________
...
=================================================================
Počet parametrů: 12,578,834
Trénovatelných parametrů: 12,578,834
Netrénovatelných parametrů: 0
```

Přidáním pouze dvou parametrických vrstev došlo k čtyřnásobnému zvýšení trénovatelných parametrů. Model byl následně trénován na celém datasetu s následujícími výsledky:



Obrázek 7.5: Vývoj ztráty u většího modelu

Pomocí druhé validační sady byl jako nejlepší zvolen model z epochy 25. Pozorovat lze spojitost mezi počtem parametrů a dobou potřebnou pro učení. U prvního modelu se jednalo o sedm epoch, což přibližně odpovídá právě čtyřnásobku u aktuálního modelu.

Obrázek 7.6: Porovnání výstupů druhého modelu s originálním obrázkem (zleva černobíly obrázek, kolorovaný obrázek, originální obrázek, výstupní kanál a)

U modelu nedošlo ve ztrátě téměř k žádnému zlepšení v porovnání s menším modelem. V případě vizuálního porovnání obrázků z obou modelů, v aktuálním případě nejsou patrné žádné artefakty. Při pohledu na výstupní kanál (a) model již dokáže rozlišovat základní tvary. Výstupní kanál také obsahuje nežádoucí šachovnicovitý šum, který vzniká důsledkem využití neparametrických vrstev.

7.3 Pooling pomocí konvolučních vrstev

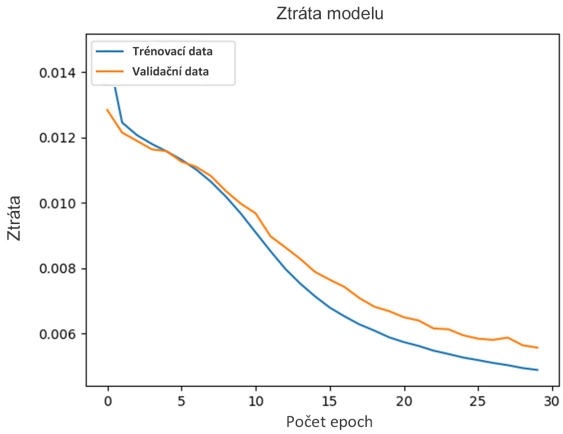
U minulých modelů lze pozorovat jasný dopad všech neparametrických vrstev na výstupní kanály v podobě šachovnicovitého šumu. Využité max-pooling vrstvy v enkodéru vždy vyberou pouze maximální hodnotu z předem určené příznakové mapy. Vícenásobným opakováním této vrstvy dochází ke ztrátě detailu a model dokáže rozpoznat pouze základní obrazce.

Jako řešení jsem využil konvoluční vrstvy, které při správně nastaveném posunu mají stejnou funkci jako pooling vrstvy. Největší výhodou jsou trénovatelné parametry těchto vrstev. Každá vrstva se dokáže naučit, jaké hodnoty na vstupu jsou klíčové pro reprezentaci obsahu obrázku a zisk co nejmenší ztráty.

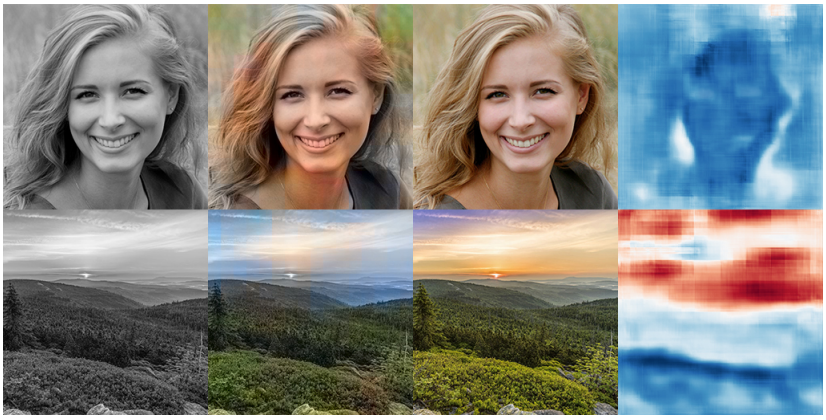
U obou modelů byly nahrazeny veškeré max-pooling vrstvy konvolučními vrstvami. Vzhledem k požadavku na snížení rozměru (u každé pooling vrstvy na polovinu) byl nastaven posun na 2 a počet filtrů stejný jako v předcházející vrstvě.

Menší model jsem trénoval s následujícími výsledky:

Nejlepší model pochází z epochy na průsečíků křivky ztráty validační a trénovací sady (sedmé epochy). U modelu nedošlo téměř k žádnému snížení ztráty, ale výstupní kanál (při porovnání s obrázky z minulého modelu) obsahuje více detailu. U prvního obrázku dokáže model rozpoznat místo pro oči a nekoloruje ho stejnou barvou jako celý obličej.

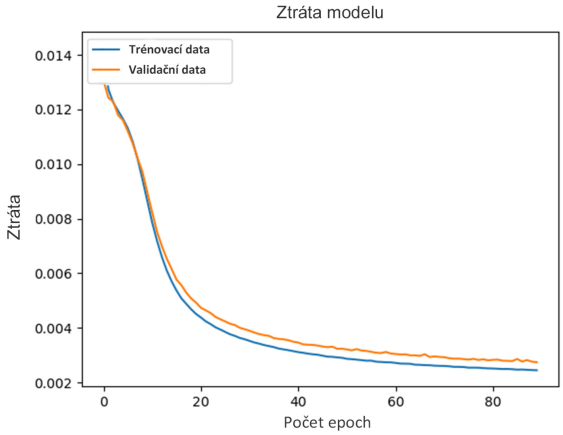


Obrázek 7.7: Vývoj ztráty menšího modelu po nahrazení pooling vrstev



Obrázek 7.8: Porovnání výstupů menšího modelu po nahrazení pooling vrstev s originálním obrázkem (pořadí stejné jako u předchozího obrázku)

U druhého modelu jsem předpokládal větší počet potřebných epoch pro úspěšné trénování v důsledku zvýšení počtu trénovatelných parametrů a zvolil jsem dvojnásobný počet epoch s následujícími výsledky:



Obrázek 7.9: Vývoj ztráty většího modelu po nahrazení pooling vrstev

Předpoklad většího počtu epoch se potvrdil. Na validační sadě byl vybrán model z epochy 34, což odpovídá čtyřnásobku potřebných epoch menšího modelu.

Obrázek 7.10: Porovnání výstupů většího modelu po nahrazení pooling vrstev s originálním obrázkem (pořadí stejné jako u předchozího obrázku)

U většího modelu se výstup vizuálně výrazně zlepšil. Již neobsahuje tak výrazný šachovnicovitý šum a dokáže rozpoznat větší detail. U prvního obrázku navíc model vynechal místo pro oči. Zároveň také u druhého obrázku není kolorována celá obloha modrou barvou, ale model rozpoznal slunce a výsledek se více blíží originálu.

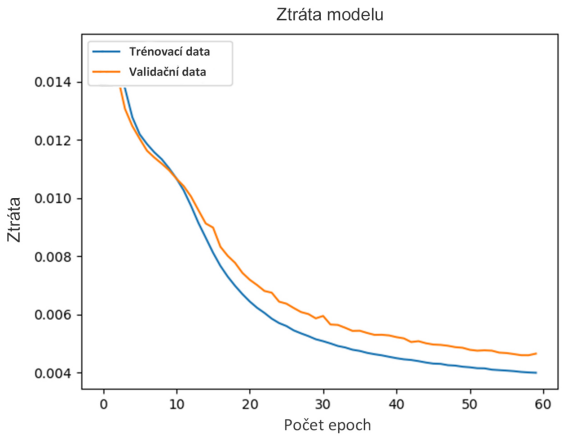
7.4 Transponované konvoluční vrstvy sloužící pro zvýšení rozměru

U minulých modelů došlo k mírnému zlepšení v oblasti šachovnicovitého šumu ve výstupních kanálech, ale stále je poměrně výrazný. U enkodéru proběhlo nahrazení všech neparametrických vrstev a není možnost, jak ho výrazně zlepšit. Problém tak nastává u dekodéru, kde na každou konvoluční vrstvu navazuje upsampling vrstva.

V minulých letech žádný z frameworků neobsahoval možnosti, jak efektivně nahradit tyto neparametrické upsampling vrstvy. V současné době (jako alternativa) slouží transponované konvoluční vrstvy s posunem. Jejich úkol je rozšířit původní příznakovou mapu na základě naučených parametrů.

Stejně jako u konvolučních vrstev, kde byl posun nastaven na 2, aby se snižoval vstupní rozměr na polovinu, tak nyní u transponovaných konvolučních vrstev, znamená nastavení posunutí o 2, rozšíření na dvojnásobný rozměr oproti vstupu.

V obou modelech jsem nahradil veškeré upsampling vrstvy transponovanými konvolučními vrstvami a trénoval s následujícími výsledky:



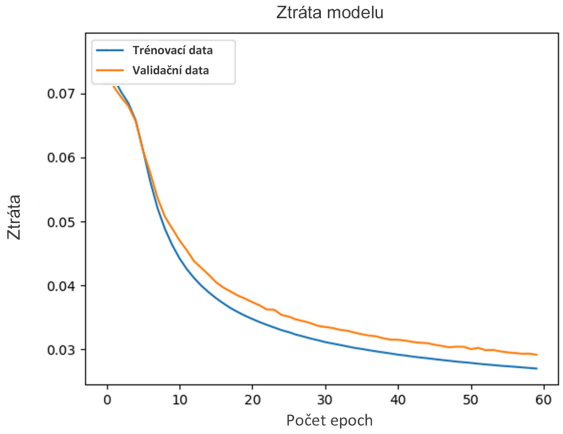
Obrázek 7.11: Vývoj ztráty menšího modelu po nahrazení upsampling vrstev

Jako nejlepší model byl tentokrát zvolen model z epochy číslo 12. Oproti předchozímu modelu byl potřeba (pro správné natrénování) téměř dvojnásobný počet epoch.

Stejné jako u minulého modelu jsou u prvního obrázku jasné obrysy obličeje včetně očí. Zároveň také došlo v mnoha částech obrázku k odstranění šachovnicovitého šumu.

Obrázek 7.12: Porovnání výstupů menšího modelu po nahrazení upsampling vrstev s originálním obrázkem (pořadí stejné jako u předchozího obrázku)

U druhého modelu jsem provedl stejné nahrazení veškerých upsampling vrstev a trénoval s následujícími výsledky:



Obrázek 7.13: Vývoj ztráty většího modelu po nahrazení upsampling vrstev

Nejlepší model pochází z epochy 58. Výstup ze sítě neobsahuje žádný šachovnicovitý šum. Naopak obsahuje ještě více detailu. Úplně poprvé jsou viditelné obrysy obličeje, očí a také vlasů.

Obrázek 7.14: Porovnání výstupů většího modelu po nahrazení upsampling vrstev s originálním obrázkem (pořadí stejné jako u předchozího obrázku)

7.5 Napojení klasifikátoru

U všech zatím navržených modelů CNN bylo na modelu, aby dokázal rozlišit o jaký typ obrázku se jedná (obličej, krajina, …). a v případě chybného rozpoznání typu obrázku docházelo zároveň k chybné kolorizaci.

Vhodným řešením je přivést na vstup dekodéru jak data z enkodérů, tak zároveň data o tom, o jaký typ obrázku se jedná. Pro tento účel slouží již předtrénované konvoluční neuronové sítě určené pro klasifikaci.

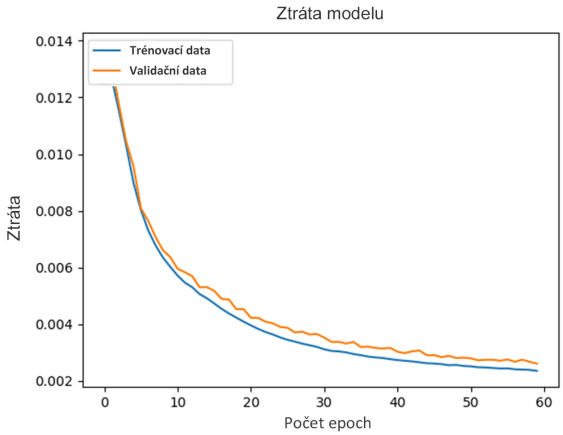
Pro určení typu obrázku jsem zvolil konvoluční neuronovou síť ResNet50 s již natrénovanými parametry na největším dostupném datasetu Imagenet. Jako vstup slouží RGB obrázek o velikost 256 x 256 pixelů a výstupem je vektor o velikosti 2048.

Problém nastává u vstupu do sítě, kdy síť požaduje obrázek o třech kanálech. V mém případě mám pouze černobílý obrázek. Snadným rozšířením jednokanálového černobílého obrázku na tříkanálový je připojení konvoluční vrstvy (obsahující tři filtry) na vstup předtrénované sítě.

Průchod sítě probíhá paralelně jak v enkodéru, tak v síti ResNet50. Výstupem z enkodéru jsou příznakové mapy a výstupem ze sítě ResNet50 vektor o velikost 2048. Napojení lze provést vytvořením 2048 příznakových map o stejném rozměru, jaký mají příznakové mapy vystupující z enkodérů. Každá příznaková mapa pak bude obsahovat stejné hodnoty, odpovídající pořadí ve vektoru příznaků. Následnuje spojení příznakových map z enkodéru a sítě ResNet.

Vzhledem k tomu, že část sítě funguje paralelně, již nebylo možné využít sekvenční přístup frameworku, ale musel se použít funkční přístup, který umožňuje u každé vrstvy zvolit, jaká data budou přivedena na vstup za předpokladu, že mají platný vstupní rozměr.

Následně bylo možné model trénovat. Veškeré parametry (které obsahuje síť ResNet50) byly uzamčeny, aby nedocházelo k jejich úpravě v průběhu trénovat a pro konkrétní obrázek tak vždy generovala stejná data. Trénování menšího modelu proběhlo s následujícími výsledky:



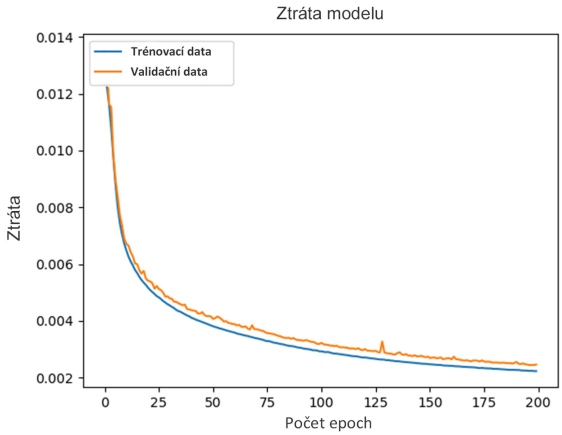
Obrázek 7.15: Vývoj ztráty menšího modelu po napojení klasifikátoru

U modelu již není možné pozorovat protnutí validační a trénovací křivky ztráty. Nelze tak ani odhadnout, z jaké epochy bude pocházet nejlepší model. Pomocí validační sady byl zvolen jako nejlepší model z epochy 44.

Obrázek 7.16: Porovnání výstupů menšího modelu po napojení klasifikátoru s originálním obrázkem (pořadí stejné jako u předchozího obrázku)

Při porovnání obrázků s předchozími modely se jedná o nejlepší výsledek jak vizuálně, tak také ve ztrátě modelu. Navíc síť díky správné klasifikaci dokáže rozpoznávat větší detaily. U prvního obrázku lze jasně určit, kde se nachází vlasy, oči a úplně poprvé rty, které jsou obarveny typickou růžovou barvou.

Stejné napojení sítě ResNet50 proběhlo i v případě většího modelu s následujícími výsledky:



Obrázek 7.17: Vývoj ztráty většího modelu po napojení klasifikátoru

Jako nejlepší model byl vybrán model z epochy 139 a výsledná ztráta je vyšší než u menšího modelu.

Obrázek 7.18: Porovnání výstupů většího modelu po napojení klasifikátoru s originálním obrázkem (pořadí stejné jako u předchozího obrázku)

Z obrázků lze pozorovat vizuální zhoršení. Barvy nejsou dostatečně syté jako u předchozích modelů a zároveň obrázek neobsahuje dostatečný detail, což dokazují rty u prvního obrázku, které mají pouze stejnou barvou jako jejich okolí.

7.6 Výsledné porovnání

Z každého trénování byl vybrán pomocí validační sady nejlepší model na základě ztráty modelu. V následující tabulce jsou porovnány veškeré modely evaluací na testovací sadě.

Model	Počet trénovatelných parametrů	Trénovací ztráta	Testovací ztráta
Základní menší model	3 140 114	0,0112	0,0146
Základní větší model	12 578 834	0,0047	0,0126
Menší + ConvPool	6 274 514	0,0110	0,0131
Větší + ConvPool	25 151 442	0,0041	0,0126
Menší + TransUnpool	7 058 354	0,0104	0,0124
Větší + TransUnpool	28 295 090	0,0313	0,0112
Menší + Klasifikátor	8 369 615	0,0031	0,0102
Větší + Klasifikátor	31 441 872	0,0032	0,0107

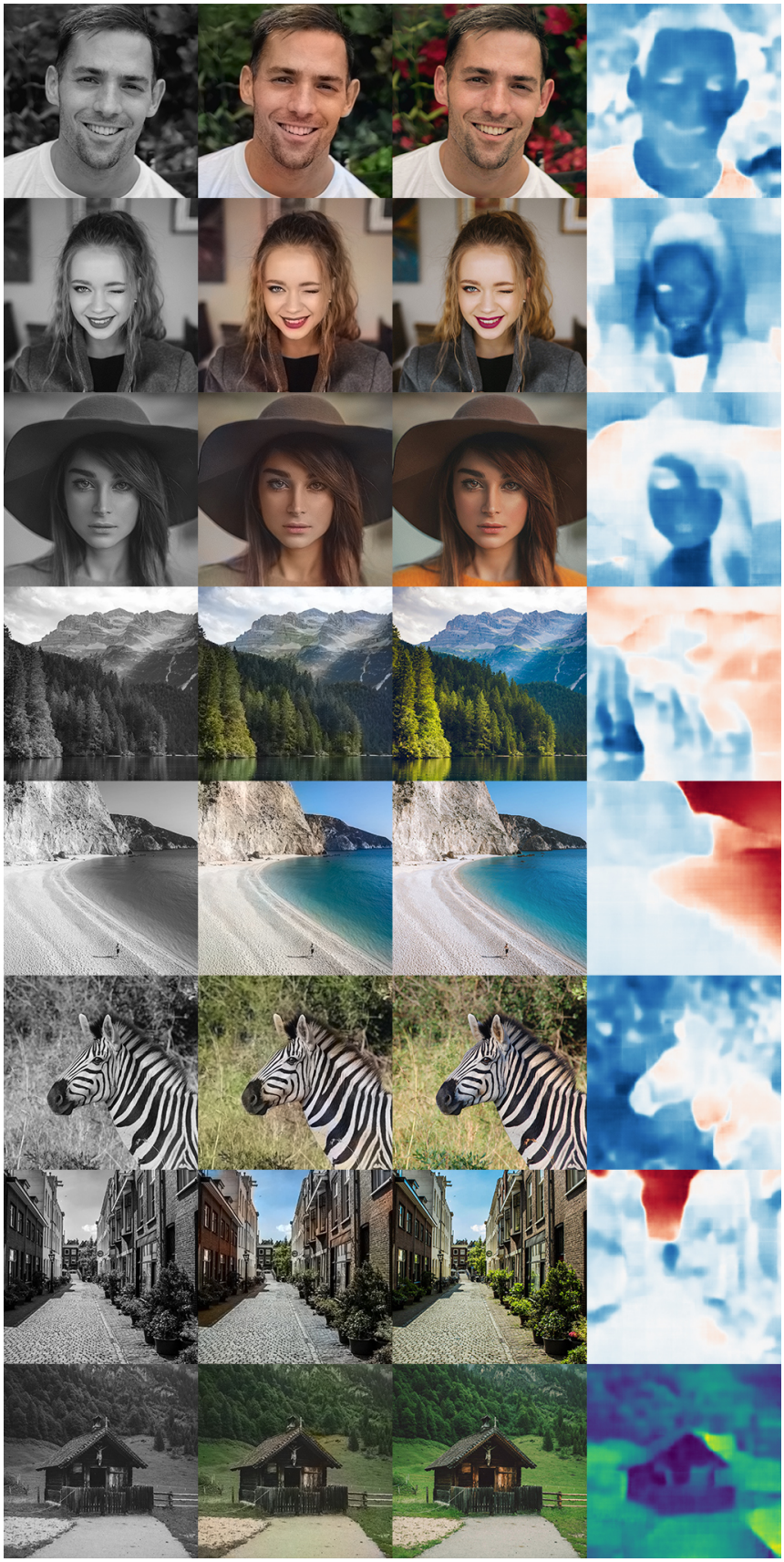
Tabulka 7.1: Porovnání ztráty vytvořených modelů

Z výsledků vyplývá, že každé rozšíření neuronové sítě o parametrické vrstvy, vedlo ke zmenšení ztráty na testovacích datech. Ve spojitosti se snižováním ztráty roste také počet parametrů modelů a tím i jejich velikost.

Ve všech případech byl větší model úspěšnější s výjimkou napojení na síť ResNet50, kdy menší model zaznamenal nejnižší ztrátu ze všech získaných modelů. Zároveň také jako jediný dokáže rozpoznat větší detail. Příkladem mohou být lidské rty, které jako jediný dokázal rozpoznat a správně obarvit.

Ztráta nejlepšího a nejhoršího modelu se ve výsledku liší pouze o 0,004. Důležité je však také brát v úvahu detail, který síť rozpozná díky využití parametrických vrstev.

Jako reprezentativní model byl vybrán menší model (4x snížena velikost) s napojením na předtrénovanou konvoluční síť ResNet50. Dosahuje vizuálně nejlepších výsledků a nejmenší ztráty na testovacích datech. Na následujících obrázcích jsou příklady úspěšného kolorování.



Obrázek 7.19: Ukázka úspěšného kolorování

Problém s kolorováním nastává v případě, kdy při trénování modelu nebyl k dispozici dostatečný počet obrázků dané kategorie, do které patří kolorovaný obrázek. Zároveň také v případě, pokud obrázek obsahuje více rozdílných objektů a napojený klasifikátor špatně klasifikuje kategorii, do které obrázek patří.

U kolorování obrázků (obsahující lidské postavy) často důsledkem nedostatku trénovacích obrázků dochází k záměně barvy oblečení. Pokud by takové obrázky byly následně porovnávány pomocí ztrátové funkce, hodnotil by se výsledek jako nepovedená kolorizace. V případě vizuálního porovnání se kolorizace hodnotí jako nepovedená, pouze pokud barvy objektů neodpovídají jejich přirozeným barvám. U oblečení (z vizuálního hlediska) záměna barev chybou není.

Na následujících obrázcích je možné vidět výsledky, kdy síť špatně klasifikovala daný obrázek a barvy neodpovídají původnímu obrázku:

Obrázek 7.20: Ukázka neúspěšných kolorizací

Postupným upravováním počátečního modelu jsem nahradil neparametrické pooling vrstvy parametrickými a postupně se také snižovala ztráta modelů a omezoval vznik šachovnicovitého šumu na výstupu.

7.7 Zodpovězení počátečních otázek

Během praktické práce vyplynuly odpovědi na všechny počáteční otázky. Už u trénování prvního modelu došlo na hardwarový nedostatek. V důsledku velké kolekce

dat bylo během trénování potřeba přibližně 170 GB RAM a 8 GPU pro trénovací časy v řádu jednotek hodin. Zároveň pro trénování samotné sítě nebyla (pro výrazné snížení ztráty modelu) potřeba klasifikační síť. Výsledky i tak byly velice uspokojivé. Připojením klasifikátoru došlo především k odstranění artefaktů v obraze a zvýšení detailu.

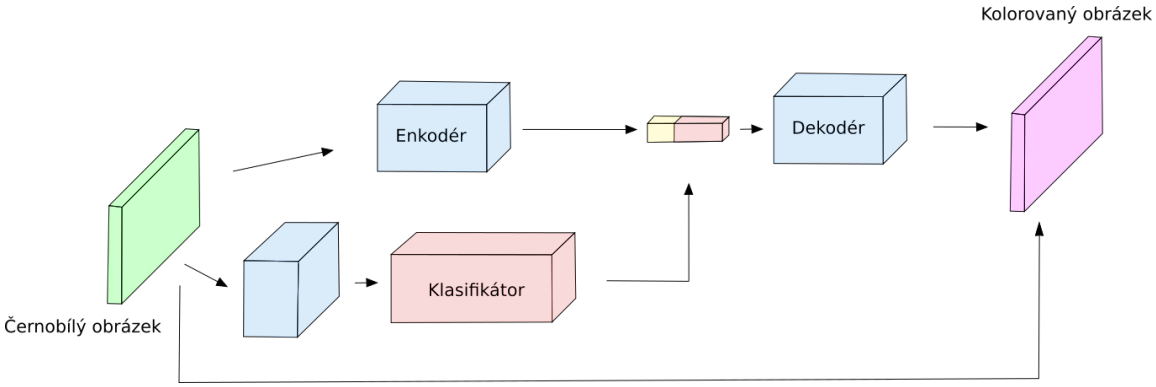
Nahrazením počátečních neparametrických vrstev došlo k výraznému snížení šachovnicovitého šumu. Pro snížení rozměrů příznakových map dosáhla nejlepších výsledků konvoluční vrstva s posunem. Naopak pro zvýšení rozměrů příznakových map nejlepších výsledků dosahovala parametrická transponovaná konvoluční vrstva s posunem.

7.8 Praktické aspekty

Z hlediska časové a výpočetní náročnosti nejdéle trvá trénování celé sítě. Následné kolorování jednotlivých obrázků již není výpočetně náročné a trvá okolo jedné sekundy. Navíc pro výpočty není potřeba grafická karta, ale pouze procesor. Díky tomu kolorování lze spustit téměř na jakémkoliv počítači.

8 Vytvořený demonstrační program

Pro ukázku fungování vytvořené konvoluční neuronové sítě jsem vytvořil jednoduchý demonstrační program. Program dokáže otevřít libovolný obrázek formátu JPEG nebo PNG. Následně převede obrázek do barevného modelu Lab a oddělí jasovou složku pro případ, kdy by byl otevřen barevný obrázek místo černobílého. Síť byla navíc připravena pro obrázky o velikosti 256 x 256 pixelů a jiný rozměr není přípustný. Pokud obrázek neodpovídá požadovaným rozměrům, dojde k jeho zmenšení nebo zvětšení.

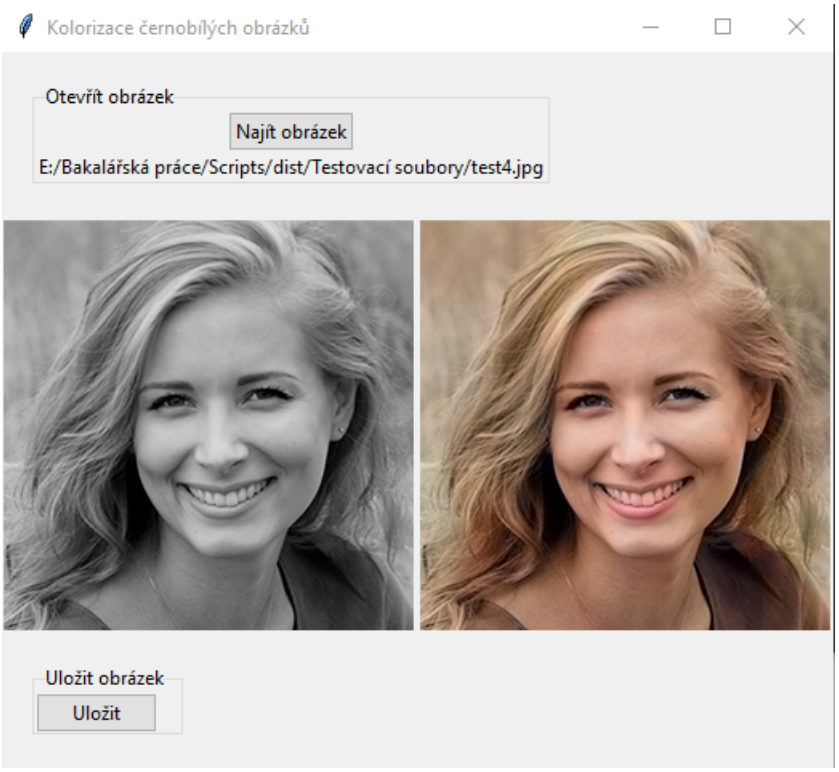


Obrázek 8.1: Vnitřní architektura modelu

Jasová složka obrázku následně projde skrze celou paralelní neuronovou síť. Výstupní barevná složka je připojena k jasové složce a následně zobrazeno porovnání vstupního a výstupního kolorovaného obrázku z programu.

Ovládání celého programu zajišťuje velice jednoduché a intuitivní GUI. V prvním kroku se vybere obrázek ve formátu JPEG nebo PNG. Následně program zajistí kompletní průchod sítí a porovnání vstupního a výstupního kolorovaného obrázku. Výsledný kolorovaný obrázek lze uložit ve formátu JPEG.

Pro spuštění programu nejsou vyžadovány žádné nadstandartní požadavky. Program obsahuje veškeré požadované knihovny, které jsou klíčové pro správné fungování programu. Navíc pro výpočet kolorovaného obrázku je možné využít CPU a není požadována GPU. Výsledná velikost programu je 855 MB a z toho parametry natrénované sítě zabírají 375 MB.



Obrázek 8.2: Ukázka demonstračního programu

9 Závěr

Tato bakalářská práce se zabývá automatickým kolorováním černobílých obrázků pomocí autoenkodérů založených na neuronových sítích. Nejdříve byly představeny způsoby reprezentace černobílých a barevných obrázků. Také porovnány jejich výhody a nevýhody pro následné využití při trénování neuronové sítě.

Třetí kapitola se věnuje již vytvořeným pracím na téma kolorizace černobílých obrázků, problematice získávání klíčových příznaků ze vstupního obrázku a možnosti využití neuronových sítí pro řešení problému kolorizace.

Ve čtvrté kapitole jsou popsány základní principy fungování neuronových sítí. Byly představeny jednotlivé vrstvy využitelné pro tvorbu konvolučních neuronových sítí a jejich možnost zapojení. Navazující pátá kapitola popisuje výběr frameworku pro trénování a zvolení jednotlivých parametrů tak, aby trénování neuronové sítě bylo co nejefektivnější.

Následující šestá kapitola popisuje proces výběru klíčových kategorií obrázků, vytvoření kolekce dat o velikosti 150 000 obrázků a přípravu dat pro samotné trénování neuronové sítě. S vytvořenou kolekcí dat vznikly první dva návrhy konvoluční neuronové sítě pro kolorizaci. Překvapivě měl již počáteční model dobré výsledky na testovacích datech. Následovalo několik rozšíření a nahrazení neparametrických vrstev s výslednou ztrátou 0,0102. Dále bylo zjištěno, že pro získání dostatečného detailu je nutné napojit na vytvořenou konvoluční neuronovou síť klasifikátor.

Připojení klasifikátoru na stávající konvoluční neuronovou síť vyžadovalo využití paralelní struktury zapojení sítě. Navíc způsob implementace připojení klasifikátoru nebylo možné dohledat v žádné odborné literatuře a v průběhu experimentální práce vznikl efektivní způsob připojení.

Trénování jednotlivých modelů konvoluční neuronové sítě vyžadovalo velký hardwarový výkon. Pro trénování jednoho modelu bylo vždy využito všech osm grafických karet výpočetního clusteru, aby bylo možné model trénovat v řádech hodin a nikoliv dnů. Naopak samotné použití již natrénovaného modelu nevyžaduje grafickou kartu a doba kolorování jednoho obrázku se pohybuje okolo jedné sekundy.

Výsledkem práce je demonstrační program, který využívá vytvořený model neuronové sítě a během sekundy dokáže vybraný vstupní obrázek kolorovat s možností následného uložení. Program obsahuje veškeré potřebné knihovny pro spuštění na libovolném počítači.

Tato práce může sloužit jako předloha pro pokročilejší techniky kolorování, především rozšíření o adaptivní velikost obrázku vstupujícího do sítě, popřípadě kolorování černobílých videí a filmů. Zároveň by bylo možné zkoumat samotnou neuronovou síť, zda neexistují jiné metody, kterými by byl obrázkům dodán dostatečný detail i bez využití klasifikátorů.

Seznam obrázků

2.1 Reprezentace černobílého obrázku v rámci barevného modelu RGB . 13
2.2 Rozložení černobílého obrázku na jednotlivé kanály C, M, Y, K . . . 14
2.3 Složení kanálů barevného modelu HSL [2] 14
2.4 Barevná paleta skládající se z kanálů a, b 15

3.1 Navržený postup kolorování obrázku [3] 17
3.2 Extraktor příznaků 18
3.3 Připojení klasifikátoru na výstup z extraktoru příznaků 18
3.4 Návrh generativního modelu 19

4.1 Model neuronu 20
4.2 Průběh funkce sigmoida 22
4.3 Průběh funkce hyperbolický tangens 22
4.4 Průběh funkce ReLU 23
4.5 Průběh konvoluce 24
4.6 Průběh max-pooling vrstvy 25
4.7 Průběh unpooling vrstvy 26
4.8 Průběh upsampling vrstvy 26
4.9 Průběh transponované konvoluce 27
4.10 Schéma autoenkodéru 28

7.1 Vývoj ztráty modelu v průběhu trénování prvního modelu 40
7.2 Porovnání výstupů prvního trénovaného modelu s originálním obrázkem (zleva černobílý, kolorovaný a originální obrázek) 41
7.3 Vývoj ztráty v průběhu trénování prvního modelu na celém datasetu 43
7.4 Porovnání výstupů prvního modelu (trénovaného na celém datasetu) s originálním obrázkem (zleva černobíly obrázek, kolorovaný obrázek ze sedmé epochy, originální obrázek, kolorovaný obrázek z třicáté epochy) 43
7.5 Vývoj ztráty u většího modelu 44
7.6 Porovnání výstupů druhého modelu s originálním obrázkem (zleva černobíly obrázek, kolorovaný obrázek, originální obrázek, výstupní kanál a) 45
7.7 Vývoj ztráty menšího modelu po nahrazení pooling vrstev 46
7.8 Porovnání výstupů menšího modelu po nahrazení pooling vrstev s originálním obrázkem (pořadí stejné jako u předchozího obrázku) 46

7.9 Vývoj ztráty většího modelu po nahrazení pooling vrstev 47
7.10 Porovnání výstupů většího modelu po nahrazení pooling vrstev s originálním obrázkem (pořadí stejné jako u předchozího obrázku) 47
7.11 Vývoj ztráty menšího modelu po nahrazení upsampling vrstev 48
7.12 Porovnání výstupů menšího modelu po nahrazení upsampling vrstev s originálním obrázkem (pořadí stejné jako u předchozího obrázku) . 49
7.13 Vývoj ztráty většího modelu po nahrazení upsampling vrstev 49
7.14 Porovnání výstupů většího modelu po nahrazení upsampling vrstev s originálním obrázkem (pořadí stejné jako u předchozího obrázku) . 50
7.15 Vývoj ztráty menšího modelu po napojení klasifikátoru 51
7.16 Porovnání výstupů menšího modelu po napojení klasifikátoru s originálním obrázkem (pořadí stejné jako u předchozího obrázku) 51
7.17 Vývoj ztráty většího modelu po napojení klasifikátoru 52
7.18 Porovnání výstupů většího modelu po napojení klasifikátoru s originálním obrázkem (pořadí stejné jako u předchozího obrázku) 52
7.19 Ukázka úspěšného kolorování 54
7.20 Ukázka neúspěšných kolorizací 55

8.1 Vnitřní architektura modelu 57
8.2 Ukázka demonstračního programu 58

Použitá literatura

[1] SUBRAMANYAM, Muthukumar. AN ANALYSIS OF COLOUR SPACE. *ResearchGate*. 2014, s. 8. Dostupné také z: https://www.researchgate.net/publication/273794022_AN_ANALYSIS_OF_COLOUR_SPACE.

[2] JUAN JOSÉ MARTÍN-SOTOCA. *The HSL model*. 2019. Dostupné také z: https://www.researchgate.net/figure/a-Color-wheel-of-hue-b-The-HSL-model-Creative-Commons_fig2_335024102.

[3] ZEZHOU CHENG Qingxiong Yang, Bin Sheng. Deep Colorization. *Arxiv*. 2016, s. 12. Dostupné také z: https://arxiv.org/abs/1605.00075.

[4] KALYAN, Ajay. Image Colorization Using Convolutional Neural Networks. *SSRN*. 2019, s. 5. Dostupné také z: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3441712.

[5] COHN, Lee. Artificial Colorization of Grayscale Satellite Imagery via GANs. *Medium*. 2017, s. 10. Dostupné také z: https://medium.com/the-downlinq/artificial-colorization-of-grayscale-satellite-imagery-via-gans-part-1-79c8d137e97b.

[6] LIU, Jizhao. A Novel Neuron Model of Visual Processor. *Arxiv*. 2020, s. 35. Dostupné také z: https://arxiv.org/pdf/2104.07257.pdf.

[7] SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks. *Towards data science*. 2020, s. 7. Dostupné také z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[8] OMID E. DAVID, Nathan Netanyahu. Painter Classification Using Deep Convolutional Autoencoders. *ResearchGate*. 2016, s. 9. Dostupné také z: https://www.researchgate.net/publication/306081538_DeepPainter_Painter_Classification_Using_Deep_Convolutional_Autoencoders.

[9] ANWAR, Aqeel. What is Transposed Convolutional Layer? *Towards data science*. 2020, s. 5. Dostupné také z: https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11.

[10] AL-MASRI, Anas. How Does Back-Propagation in Artificial Neural Networks Work? *Towards data science*. 2019, s. 10. Dostupné také z: https:

//towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7.

[11] DOR BANK Noam Koenigstein, Raja Giryes. Autoencoders. *Arxiv*. 2020, s. 22. Dostupné také z: https://arxiv.org/abs/2003.05991.

[12] FCHOLLET. Keras: Developer guides. *Keras.io*. 2019. Dostupné také z: https://keras.io/guides/.

[13] GROVER, Prince. 5 Regression Loss Functions All Machine Learners Should Know. *Heartbeat*. 2018, s. 11. Dostupné také z: https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0.

[14] DOSHI, Sanket. Various Optimization Algorithms For Training Neural Network. *Heartbeat*. 2019, s. 7. Dostupné také z: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

[15] RUDER, Sebastian. An overview of gradient descent optimization algorithms. *Ruder.io*. 2016, s. 28. Dostupné také z: https://ruder.io/optimizing-gradient-descent/index.html#rmsprop.

[16] BROWNLEE, Jason. A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks. *Machine Learning Mastery*. 2018, s. 10. Dostupné také z: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/.