

ŽILINSKÁ UNIVERZITA V ŽILINE

Fakulta riadenia a informatiky

DIPLOMOVÁ PRÁCA

Bc. Marek Zaťko

Zvyšovanie rozlíšenia obrázkov pomocou GAN

Vedúci práce: Ing. Peter Tarábek PhD.

Registračné číslo: 1093/2019

Žilina 2020

ŽILINSKÁ UNIVERZITA V ŽILINE

Fakulta riadenia a informatiky

DIPLOMOVÁ PRÁCA

Informatika

Bc. Marek Zaťko

Zvyšovanie rozlíšenia obrázkov pomocou GAN

Žilinská univerzita v Žiline

Fakulta riadenia a informatiky

Školiace pracovisko: Katedra matematických metód a operačnej analýzy

Žilina, 2020

Pod'akovanie

Rád by som sa pod'akoval všetkým, ktorí mi pomáhali pri tvorbe tejto záverečnej práce. Vďaka patrí predovšetkým Ing. Petrovi Tarábekovi PhD. za správne usmerňovanie, cenné rady a pripomienky.

Ďakujem

ABSTRAKT

ZAŤKO, Marek: *Zväčšovanie rozlíšenia obrázkov pomocou GAN*. [Diplomová práca]. – Žilinská univerzita v Žiline. Fakulta Riadenia a Informatiky; Katedra matematických metód a operačnej analýzy. – Vedúci: Ing. Peter Tarábek PhD. – Žilina FRI UNIZA, 2020. Počet strán: 106

Hlavným cieľom práce je preskúmať metódu zväčšovania rozlíšenia obrázkov pomocou neurónových sietí, konkrétnie architektúru Generatívnych konfrontačných sietí (GAN). V ďalších častiach práce sme sa zamerali na implementáciu konkrétnych GAN modelov vo frameworku TensorFlow a následne sme túto metódu zväčšovania porovnávali s inými známymi metódami zväčšovania rozlíšenia.

Ako súčasť práce bola vytvorená aj aplikácia na generovanie datasetov textových obrázkov, na ktorých sme následne realizovali niektoré experimenty zväčšovania rozlíšenia s využitím technológie Optického rozpoznávania znakov (OCR), konkrétnie implementácie Tesseract.

Kľúčové slová: zväčšovanie rozlíšenia, rozlíšenie, OCR, Tesseract, MNIST, GAN, neurónové siete, rozpoznávanie obrazu, TensorFlow, Python, Generatívne konfrontačné siete

ABSTRACT

ZAŤKO, Marek: *Enhancing Low Resolution Images using GAN* [Diploma thesis]. – University of Žilina. Faculty of Management Science and Informatics; Department of Mathematical Methods and Operations Research. – Supervisor: Ing. Peter Tarábek PhD. – Žilina FRI UNIZA, 2020. Number of pages: 106

The main goal of the thesis is to research the method of image resolution upscaling using neural networks, specifically the Generative adversarial networks (GAN) architecture. In the other parts of thesis, we focused on implementing specific GAN architectures using TensorFlow libraries and then we compared the model with others image resolution upscaling methods.

We also developed an application for text image dataset generation on which we then conducted image resolution upscaling experiments with the use of Optical Character Recognition (OCR), specifically the Tesseract implementation.

Key words: image upscaling, upsampling, interpolation, resolution, OCR, Tesseract, GAN, MNIST, neural networks, image recognition, TensorFlow, Python, Generative adversarial networks

Obsah

Úvod	1
1 Formulácia problému	2
1.1 Reprezentácia obrázka.....	4
2 Existujúce metódy zväčšovania rozlíšenia.....	6
2.1 Metóda najbližšieho suseda.....	9
2.2 Bilineárna interpolácia	10
2.3 Bikubická interpolácia.....	11
2.4 LANCZOS interpolácia.....	12
3 Upsampling založený na neurónových sietiach	15
3.1 Umelé neurónové siete	15
3.1.1 McCullochov-Pittsov umelý neurón.....	15
3.1.2 Perceptrón.....	17
3.1.3 Umelý neurón	17
3.1.4 Viacvrstvové neurónové siete.....	18
3.1.5 Trénovanie umelých neurónových sietí.....	20
3.2 Konvolučné neurónové siete	25
3.2.1 Konvolúcia	25
3.2.2 Transponovaná konvolúcia.....	27
3.2.3 Pooling.....	29
3.3 Klasifikátor.....	30
3.4 Generatívne konfrontačné siete	32
3.4.1 Chybová funkcia diskriminátora	34
3.4.2 Chybová funkcia generátora.....	35
3.4.3 Tréningový cyklus GANu	35
3.4.4 Deep Convolution GAN (DCGAN)	36
4 Implementácia.....	39
4.1 Python.....	39
4.2 TensorFlow.....	39
4.3 Tenzor.....	40
4.4 Použité datasety	40
4.4.1 Dataset MNIST, Fahion MNIST	41
4.4.2 Aplikácia na tvorbu datasetov obrázkov obsahujúcich text	41
4.4.3 Dataset Text_Set	43
4.5 Architektúra a implementácia použitých modelov.....	43
4.5.1 Architektúra a implementácia klasifikátorov MNIST a FMNIST	43
4.5.2 Architektúry a implementácia GANu.....	46

4.6	Implementácia optického rozpoznávania znakov (OCR).....	52
5	Experimenty so zväčšovaním rozlíšenia pomocou GAN.....	53
5.1	Tréning klasifikátorov	53
5.1.1	Použité metriky na porovnávanie obrázkov	54
5.1.2	Tréning GANov.....	55
5.2	Percepčná chyba generátora	57
5.3	Dvojnásobné zväčšenie rozlíšenia.....	58
5.3.1	GENERATOR_K2 s klasifikátorom K – MNIST , trénovaný na oboch chybových funkciách.....	59
5.3.2	GENERATOR_K2 s klasifikátorom K – FMNIST	63
5.3.3	GENERATOR_K2 trénovaný na datasete Text_Set	67
5.3.4	GENERATOR_K2 trénovaný na rozšírenom datasete Text_Set	74
5.4	Štvornásobné zväčšovanie rozlíšenia	79
5.4.1	GENERATOR_K4 s klasifikátorom K – MNIST	79
5.4.2	GENERATOR_K4 s klasifikátorom K – FMNIST	83
5.4.3	GENERATOR_K4 trénovaný na datasete Text_Set	88
5.5	Zhrnutie experimentov	90
Záver	93	
Zoznam použitej literatúry	94	
Zoznam príloh	96	
Prílohy	97	
Príloha A: Priebeh tréningu klasifikátorov	98	
Príloha B: Príklady štvornásobného zväčšenia MNIST generátorom GENERATOR_K2 a ostatnými zväčšovacími metódami	99	
Príloha C: Vývoj zväčšených vybraných obrázkov počas tréningu štvornásobného zväčšenia MNIST pre obe chybové funkcie generátora	105	
Príloha D: Štvornásobné zväčšenie náhodných obrázkov datasetu FMNIST metódou najbližšieho suseda	106	

Zoznam obrázkov

Obrázok 1 Najjednoduchší upsampling, kde neznáme hodnoty pixelov nahradíme hodnotou 0, zväčšovací faktor $k = 2$	3
Obrázok 2 Obrázok rozložený na jednotlivé farebné kanály	5
Obrázok 3 Znázornenie problematiky zväčšenia obrázka	6
Obrázok 4 (a) Známe hodnoty $g(u)$ (b) Funkcia $f(x)$, ktorú sa snažíme aproximovať z hodnôt $g(u)$ <i>Zdroj:</i> [BB16].....	7
Obrázok 5 Aproximačné funkcie gx , (a) Metóda najbližšieho suseda (b) Lineárna interpolácia <i>Zdroj:</i> [BB16].....	8
Obrázok 6 Príklad interpolácie najbližšieho suseda zväčšovacím faktorom $k = 16$, so zobrazenou mriežkou, žltým vyznačená „ <i>jagged edge</i> “, na grafe znázornený interpolačný kernel	10
Obrázok 7 Bilineárna interpolácia obrázka	11
Obrázok 8 Príklad bikubickej interpolácie	12
Obrázok 9 Príklad Lanczosovej interpolácie, $\alpha = 3$	13
Obrázok 10 Porovnanie interpolačných metód.....	14
Obrázok 11 McCulloch-Pittsov model neurónu pre funkciu <i>OR</i>	16
Obrázok 12 Schéma umelého neurónu	18
Obrázok 13 Všeobecná architektúra doprednej siete.....	19
Obrázok 14 Model dropoutu, (a) aktívne všetky neuróny, (b) niektoré neuróny neaktívne	23
Obrázok 15 Ilustrácia výpočtu dvoch hodnôt v konvolúcii	26
Obrázok 16 Príklad konvolúcie s kernelom k na detekciu hrán	26
Obrázok 17 Príklad transoformácie kernela k na konvolučnú maticu	28
Obrázok 18 Príklad výpočtu feature mapy pomocou konvolučnej matice	28
Obrázok 19 Príklad výpočtu konvolúcie pomocou transponovanej konvolučnej matice....	29
Obrázok 20 Vizualizácia problematiky generatívnych modelov.....	32
Obrázok 21 Schéma fungovania generatívnych konfrontačných sietí.....	34
Obrázok 22 Príklad obrázkov z datasetov (a) MNIST (b) FMNIST	41
Obrázok 23 Volba parametrov generovania v aplikácií.....	42
Obrázok 24 Rozhranie výberu formátov a príklad vygenerovaných dát	42
Obrázok 25 Príklad obrázkov datasetu <i>Text Set</i>	43

Obrázok 26 Vývoj úspešnosti klasifikácie zväčšeného testovacieho datasetu generátorom <i>GENERATOR_K2</i> na klasifikátore $K - MNIST$ pri oboch chybových funkciách generátora	59
Obrázok 27 Vývoj chybových funkcií pre generátor a diskriminátor pri tréningu <i>GENERATOR_K2</i> na datasete MNIST.....	60
Obrázok 28 Vývoj hodnôt porovnávacích metrík pri tréningu <i>GENERATOR_K2</i> na klasifikátore $K - MNIST$	61
Obrázok 29 Príklad zväčšených obrázkov generátorom po každom epochu tréningu	62
Obrázok 30 Vývoj úspešnosti klasifikácie zväčšeného testovacieho datasetu generátorom <i>GENERATOR_K2</i> pri oboch chybových funkcií, klasifikovanom na $K - FMNIST$	64
Obrázok 31 Vývoj metrík v tréningoch <i>GENERATOR_2</i> na $K - FMNIST$ pri oboch chybových funkciách	65
Obrázok 32 Vývoj zväčšovania náhodných obrázkov z testovacieho datasetu FMNIST, generované generátorom <i>GENERATOR_2</i> trénovanom na percepčnej chybovej funkci... <td>66</td>	66
Obrázok 33 Vývoj SSIM zväčšeného testovacieho datasetu <i>Text_Set</i> generátorom <i>GENERATOR_K2</i> trénovaným s klasickou chybovou funkciou.....	68
Obrázok 34 Vývoj zväčšovania vybraných obrázkov z testovacieho datasetu <i>Text_Set</i> , generátorom <i>GENERATOR_K2</i> počas tréningu	69
Obrázok 35 Vývoj metrík počas tréningu <i>GENERATOR_2</i> na datasete <i>Text_Set</i>	70
Obrázok 36 Porovnanie zväčšenia pomocou <i>GENERATOR_K2</i> a interpolácie LANCZOS3 na datasete <i>Text_Set</i>	73
Obrázok 37 Vývoj SSIM zväčšeného rozšíreného testovacieho datasetu <i>Text_Set</i> generátorom <i>GENERATOR_K2</i> trénovaným s klasickou chybovou funkciou.....	75
Obrázok 38 Počty nesprávne rozpoznaných písmen na zväčšených obrázkoch pomocou #1 <i>GENERATOR_K2</i> a LANCZOS3 interpolácie	77
Obrázok 39 Počty nesprávne rozpoznaných čísel a medzere na zväčšených obrázkoch pomocou #1 <i>GENERATOR_K2</i> a LANCZOS3 interpolácie	77
Obrázok 40 Porovnanie zväčšenia pomocou #1 <i>GENERATOR_K2</i> a interpolácie LANCZOS3 na rozšírenom datasete <i>Text_Set</i>	78
Obrázok 41 Vývoj úspešnosti klasifikácie zväčšeného testovacieho datasetu generátorom <i>GENERATOR_K4</i> pri oboch chybových funkcií, klasifikovanom na $K - MNIST$	79
Obrázok 42 Vývoj metrík v tréningoch <i>GENERATOR_K4</i> na $K - MNIST$ pri oboch chybových funkciách	80

Obrázok 43 Vývoj štvornásobného zväčšovania náhodných obrázkov z testovacieho datasetu MNIST, generované generátorom <i>GENERATOR_K4</i> trénovanom na klasickej chybovej funkcií počas tréningu	82
Obrázok 44 Vývoj úspešnosti klasifikácie zväčšeného testovacieho datasetu generátorom <i>GENERATOR_K4</i> pri oboch chybových funkcií, klasifikovanom na <i>K – FMNIST</i>	83
Obrázok 45 Vývoj metrík v tréningoch <i>GENERATOR_K4</i> na <i>K – FMNIST – 2</i> pri oboch chybových funkciách	84
Obrázok 46 Porovnanie zväčšenia náhodne vybraných obrázkov z testovacej množiny FMNIST pre oba prípady chybových funkcií.....	86
Obrázok 47 Porovnanie zväčšenia náhodne vybraných obrázkov z testovacej množiny FMNIST ostatnými zväčšovacími metódami	87
Obrázok 48 Porovnanie zväčšenia pomocou <i>GENERATOR_K4</i> a interpolácie LANCZOS5 na datasete <i>Text_Set</i>	89

Zoznam tabuliek

Tabuľka 1 Architektúra $K - F/MNIST$	44
Tabuľka 2 Architektúra $DISCRIM$	47
Tabuľka 3 Architektúra $Generator_K2$	49
Tabuľka 4 Architektúra $GENERATOR_K4$	50
Tabuľka 5 Dosiahnuté presnosti klasifikátorov	54
Tabuľka 6 Obory hodnôt a význam hodnôt jednotlivých metrík	55
Tabuľka 7 Rekonštrukčné chyby $GENERATOR_K2$ pri oboch chybových funkciách	60
Tabuľka 8 Hodnoty metrík dvojnásobného zväčšenia testovacieho datasetu MNIST	63
Tabuľka 9 Rekonštrukčné chyby pre obe chybové funkcie	64
Tabuľka 10 Hodnoty metrík dvojnásobného zväčšenia testovacieho datasetu FMNIST	67
Tabuľka 11 Percento úspešne rozpoznaných textov pre pôvodný a zmenšený testovací dataset	71
Tabuľka 12 Percento úspešne rozpoznaných textov zo zväčšeného testovacieho dataset jednotlivými metódami	72
Tabuľka 13 Percento úspešne rozpoznaných textov pre pôvodný a zmenšený testovací dataset	74
Tabuľka 14 SSIM a výsledok rozpoznávania Tesseractom na zväčšených obrázkoch pomocou najlepších troch generátorov vybratých na základe SSIM	75
Tabuľka 15 SSIM indexy a výsledky rozpoznávania OCR na zväčšených obrázkoch testovacieho rozšíreného datasetu $Text_Set$	76
Tabuľka 16 Rekonštrukčné chyby generátorov pri štvornásobnom zväčšení testovacieho datasetu MNIST	80
Tabuľka 17 Hodnoty metrík štvornásobného zväčšenia testovacieho datasetu MNIST	81
Tabuľka 18 Rekonštrukčné chyby oboch generátorov pri štvornásobnom zväčšení testovacieho datasetu FMNIST	84
Tabuľka 19 Hodnoty metrík štvornásobného zväčšenia testovacieho datasetu FMNIST	85
Tabuľka 20 Hodnoty metrík štvornásobného zväčšenia testovacieho datasetu $Text_Set$	88
Tabuľka 21 Porovnanie úspešnosti klasifikácie dvojnásobného a štvornásobného zväčšenia rozlíšenia testovacieho datasetu MNIST	91
Tabuľka 22 Porovnanie úspešnosti klasifikácie dvojnásobného a štvornásobného zväčšenia rozlíšenia testovacieho datasetu Fashion MNIST	92

Úvod

V súčasnosti zaznamenávame ohromné napredovanie vo všetkých oblastiach strojového učenia a to i napriek tomu, že väčšina týchto techník je známa už niekoľko desiatok rokov. Napríklad koncepcia umelých neurónových sietí bola známa už v roku 1943. Dôvodom tohto súčasného napredovania je hlavne stále väčšia a dostupnejšia výpočtová sila, ktorá je v tejto oblasti klúčová a to najmä v oblasti hlbokých neurónových sietí.

Hlboké neurónové siete, ktorých princíp a fungovanie predstavíme v neskorsích častiach práce, sú v súčasnosti využívané v mnohých oblastiach. Ako príklad môžeme uviesť rozpoznávanie tváre, kompresia dát, autonómne riadenie vozidiel, personalizácia reklám, aplikácia obrazových filtrov, inteligentné doplnenie kódu a mnoho ďalších.

V práci sa zameriavame na jednu konkrétnu aplikáciu neurónových sietí, ktorou je zväčšovanie rozlíšenia obrázkov. Konkrétnie sa zameriame na jednu z mnohých architektúr sietí, ktoré dokážu tento proces realizovať. Táto architektúra nesie názov Generatívne konfrontačné siete (*angl. Generative Adversarial Networks, skr. GAN*). Práca je teda zameraná na preskúmanie možností využitia tohto prístupu. V úvodných kapitolách čitateľovi prezentujeme samotný problém zväčšovania rozlíšenia a takisto predstavíme niektoré existujúce používané metódy, ktoré v záverečných častiach práce porovnáme práve so zväčšovaním rozlíšenia založenom na spomínaných generatívnych konfrontačných sietiach. Toto porovnávanie budeme realizovať na troch rôznych úlohách, kde v prvých dvoch prípadoch pôjde o zväčšovanie rozlíšenia najpoužívanejších referenčných datasetoch MNIST a Fashion MNIST. V tretej úlohe sa venujeme využitiu zväčšovania rozlíšenia v problematike rozpoznávania textov pomocou optického rozpoznávania znakov (OCR). Ako súčasť práce takisto vznikla aplikácia, ktorá umožňuje generovanie datasetov obrázkov, na ktorých sa nachádza text. V práci využijeme túto aplikáciu na tvorbu datasetu, ktorý simuluje formát evidenčných čísel vozidiel (EČV).

1 Formulácia problému

Zväčšovanie rozlíšenia zohráva kľúčovú rolu v úlohách, ktoré vyžadujú väčší obrázok ako je k dispozícii. Ako príklad môžeme uviesť problematiku rozpoznávania tváre pomocou neurónových sietí. Architektúra takejto siete je navrhnutá na konkrétnie vstupné rozlíšenie obrázka. Ak by sme teda chceli rozpoznať tvár z menšieho obrázka, musí existovať metóda, ktorá toto rozlíšenie dokáže zväčšiť. Takisto napríklad v počítačových hrách je podstatne menej výpočtovo náročné zväčšiť už zrenderovaný frame¹ ako renderovať väčší frame vo väčšom rozlíšení.

Matematicky teda hľadáme predpis (metódu)

$$\Psi(O_P, k_w, k_h) = O_{SR} \quad (1.1)$$

, kde O_P predstavuje originálny obrázok, teda množinu pixelov

$$O_P = \{p_{00}, p_{01}, \dots, p_{0w-1}, p_{10}, p_{11}, \dots, p_{h-1w-1}\} \quad (1.2)$$

s rozlíšením $P = w * h$, kde w je šírka obrázka a h výška obrázka. O_{SR} je zväčšený obrázok s pixelmi

$$O_{SR} = \{p_{00}, p_{01}, \dots, p_{0k_w * w - 1}, p_{10}, p_{11}, \dots, p_{k_h * (h-1)k_w * (w-1)}\} \quad (1.3)$$

s rozlíšením $SR = k_w * w + k_h * h$, kde hodnoty $k_w, k_h \in Q^+ - \{1; 0\}$ nazývame škálovacie faktory (*angl. scaling factors*).

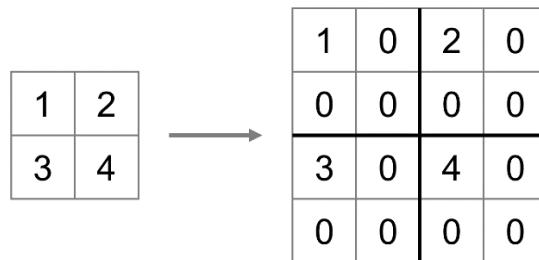
V prípade problematiky zväčšovania rozlíšenia teda uvažujeme len $k_w, k_h \in Q^+ > 1$. V práci sa zaoberáme len faktormi, kde $k = k_w = k_h$, teda takým zväčšením ktoré zachováva pomer šírky a výšky obrázka (*aspect ratio*).

Na prvý pohľad by sa zdalo, že faktor k by mal nadobúdať len celočíselné hodnoty, pretože obrázok nemôže mať neceločíselné rozlíšenie. Akýkoľvek neceločíselný faktor k , ale môžeme vyjadriť ako zlomok $k = \frac{p}{q}$, kde $p, q \in Z^+$, ak by sme teda chceli obrázok O_P škálovať o takýto faktor, stačí ho najskôr zväčšiť faktorom p a následne zmenšiť faktorom q .

¹ Jeden snímok animácie

Predpis Ψ sa nazýva škálovacia metóda, upscaling metóda, zväčšovacia metóda alebo upsampling. Ak hovoríme o resampling metóde, máme na mysli taký predpis Ψ , ktorým sa dá realizovať aj zväčšenie aj zmenšenie obrázku O_p , teda že predpis Ψ je definovaný aj pre faktory $k_w, k_h \in (0; 1)$.

Takto definovaný predpis je ale príliš všeobecný, a vyhovuje mu aj prípad, keby všetky hodnoty pixelov, ktoré nepoznáme vo zväčšenom obrázku nahradíme hodnotou nula, čo v praxi znamená čiernu farbu.



Obrázok 1 Najjednoduchší upsampling, kde neznáme hodnoty pixelov nahradíme hodnotou 0, zväčšovací faktor $k = 2$

Citateľovi musí byť jasné, že takýto predpis je prakticky nepoužiteľný. Musíme teda definovať niektoré požiadavky, ktoré od zväčšeného obrázka a samotného predpisu očakávame, podľa [Witt05] je týchto požiadaviek deväť :

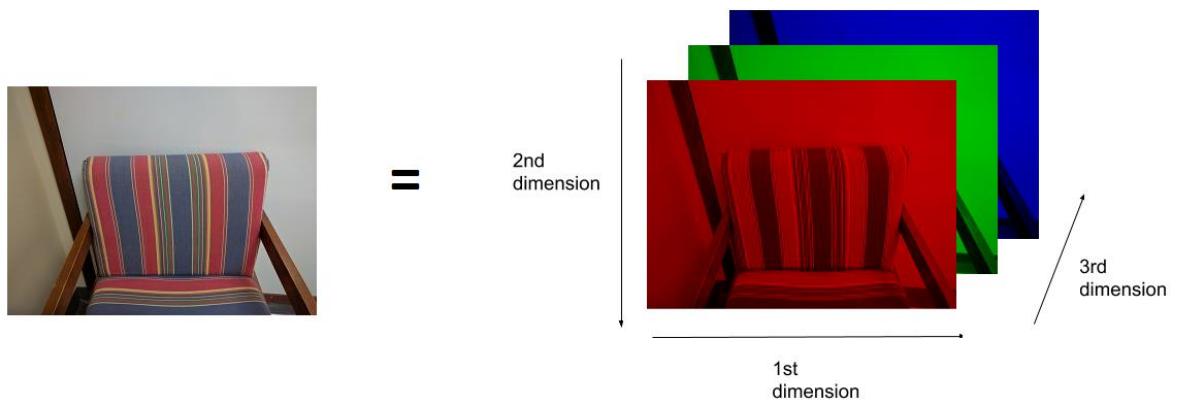
1. Geometrická invariancia – Interpolačná metóda by mala zachovávať geometriu a relatívne veľkosti všetkých objektov v obrázku.
2. Invariancia kontrastu – Metóda by mala zachovávať rovnakú úroveň osvetlenia objektov a celkový kontrast obrázku.
3. Bezšumovosť – Metóda by nemala do obrázku pridávať šum alebo iné nežiadúce artefakty.
4. Zachovávanie hrán – Metóda by mala zachovávať všetky hrany objektov v obrázku, prípadne by ich mala ešte zaostrovať.
5. Aliasing – Metóda by mala produkovať len hladké hrany.
6. Zachovávanie textúr – Metóda by nemala rozmazávať alebo vyhľadzovať textúry objektov.
7. Over-smoothing – Metóda by nemala produkovať prehnane hladké objekty a prehnane „hranaté“ objekty.

8. Aplikáčné požiadavky – Metóda by mala produkovať výsledky primerané k druhu vstupných obrázkov. Napríklad od fotografii očakávame realistické zväčšenie, ale napríklad pri zväčšovaní medicínskych obrázkov očakávame ostré a výrazné hrany a podobne.
9. Senzitivita na parametre obrázka – Metóda by nemala príliš zohľadňovať parametre, ktoré sú iné v každom obrázku.

Tieto požiadavky samozrejme nie sú nikde oficiálne definované, jedná sa len o subjektívny názor autora a preto je na čitateľovi, ktorým z nich bude priklaďať akú dôležitosť. Predpis Ψ , ktorý by vyhovel všetkým týmto požiadavkám neexistuje, a keby aj existoval, určite by vzhľadom na jeho výpočtovú zložitosť neboli použiteľný v praktických úlohách. V práci sa teda budeme venovať predpisom, ktoré vyhovujú len niektorým z uvedených požiadaviek.

1.1 Reprezentácia obrázka

Obrázok je teda množina pixelov. Táto množina býva v pamäti reprezentovaná ako trojdimenzionálne pole, kde veľkosť prvej dimenzie je šírka obrázka w , veľkosť druhej dimenzie je výška obrázka h a veľkosť tretej dimenzie je počet farebných kanálov obrázka c . V prípade klasického farebného obrázka sú tieto farebné kanály tri. Červený(R), zelený(G) a modrý(B), kde každá z ich hodnôt je niekde v rozsahu $<0; 255>$, teda vyjadrujú ako veľmi červený, zelený a modrý je daný pixel. Tieto tri farebné kanály tvoria základné farebné spektrum ľudského oka a preto je ich rôznymi intenzitami vyjadriteľné celé viditeľné farebné spektrum.



Obrázok 2 Obrázok rozložený na jednotlivé farebné kanály

Zdroj: <https://datacarpentry.org/image-processing/03-skimage-images/>

V prípade čiernobieleho obrázka existuje len jeden farebný kanál, ktorý nám hovorí „ako veľmi biely“ je daný pixel. Hodnota nula predstavuje absolútnu čiernu a hodnota 255 absolútну bielu farbu.

Obrázok teda môžeme určiť trojicou $[w, h, c]$. Toto označenie sa používa v rôznych knižniciach určených pre strojové učenie a taktiež je základným stavebným kameňom knižnice, ktorú v práci používame a ktorú predstavíme v neskorsích kapitolách.

Ak teda hovoríme o čiernobielom obrázku so šírkou 28 a výškou 28, môžeme tento obrázok jednoducho označiť ako $[28, 28, 1]$.

2 Existujúce metódy zväčšovania rozlíšenia

V tejto kapitole si predstavíme niektoré vybrané najpoužívanejšie upsampling metódy založené na matematických metódach, ktoré budeme v experimentoch porovnávať s našou implementáciou založenou na Generatívnych konfrontačných sietiach. Spôsoby porovnávania ako aj vysvetlenie našej metódy predložíme v ďalších kapitolách.

Všetky prezentované metódy majú jednu spoločnú vlastnosť a to, že sa dajú použiť aj na downsampling, teda aj v prípadoch, kde škálovací faktor $k \in (0; 1)$. Uvedené metódy sú teda resampling metódami. Keďže v práci sa sústredíme len na upsampling, metódy predstavíme len na príkladoch zväčšenia.



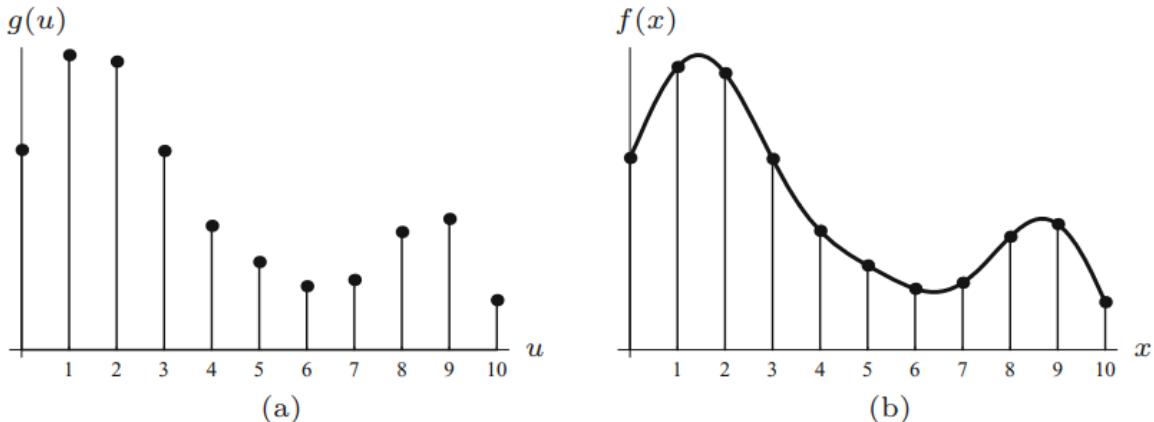
Obrázok 3 Znázornenie problematiky zväčšenia obrázka

Zdroj: <https://www.cambridgeincolour.com/tutorials/image-interpolation>

Na Obrázok 3 vidíme všeobecný príklad zväčšenia rozlíšenia. Celá problematika zväčšovania rozlíšenia je teda o riešení otázky akými hodnotami nahradíť neznáme pixely na zväčšenom obrázku. Pre zjednodušenie najskôr ilustrujeme problém a jeho možné riešenia pre jeden rozmer.

Metódy akými sa rieši uvedený problém vychádzajú z rôznych riešení interpolácie. Interpolácia je jedna z úloh číselnej analýzy, v ktorej ide o nájdenie pôvodnej spojitej funkcie $f(x), x \in R$, ktorej poznáme len niektoré diskrétné hodnoty. Túto diskrétnu funkciu označíme ako $g(u), u \in Z$. Funkciu, ktorá vznikne touto aproximáciou funkcie $f(x)$ označíme ako $\tilde{g}(x)$ [BB16]. Keďže obrázok je vo svojej podstate len funkcia dvoch

premenných (x, y) , ktorej hodnota v tomto bode udáva farbu pixela na tejto pozícii, môžeme riešenia interpolácie aplikovať aj na obrázok.



Obrázok 4 (a) Známe hodnoty $g(u)$ (b) Funkcia $f(x)$, ktorú sa snažíme approximovať z hodnôt $g(u)$ *Zdroj:* [BB16]

Jedným z najjednoduchších a najtriviálnejších spôsobov akým approximovať funkciu $f(x)$ je taká, že neznámu hodnotu funkcie \tilde{g} v bode x nahradíme najbližšou známou hodnotou funkcie g . Formálne teda

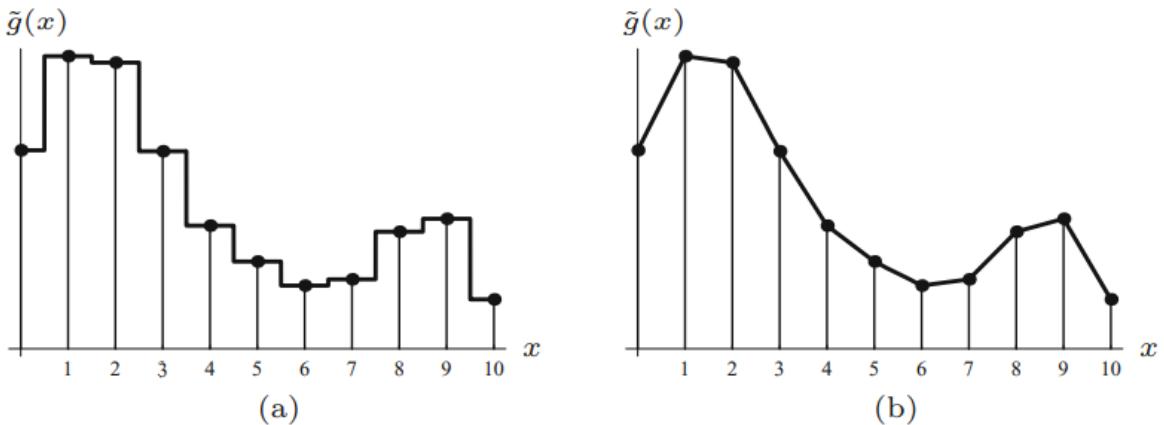
$$\tilde{g}(x) = g(u_x) \quad (2.0)$$

,kde u_x je najbližšie celé číslo k reálnemu číslu x , teda $u_x = \text{round}(x) = \lfloor x + 0.5 \rfloor$. Tento najtriviálnejší spôsob sa nazýva metóda najbližšieho suseda. [BB16]

Ďalšou z možností je takzvaná lineárna interpolácia, pri ktorej sa neznáme hodnota v bode x vypočíta ako pozícia na lineárnej funkcií ktorá spája dva najbližšie známe body, teda body u_x a $u_x + 1$. Teda

$$\tilde{g}(x) = g(u_x) * (1 - (x - u_x)) + g(u_x + 1) * (x - u_x) \quad (2.1)$$

Potom approximačné funkcie $\tilde{g}(x)$, ktoré vznikli týmito metódami sú znázornené na Obrázok 5



Obrázok 5 Aproximačné funkcie $\tilde{g}(x)$, (a) Metóda najbližšieho suseda (b) Lineárna interpolácia *Zdroj:* [BB16]

Spomínané, a ďalšie metódy riešenia interpolácie sa dajú generalizovať a implementovať pomocou konvolúcie známej diskrétnej funkcie $g(x)$ so spojitou funkciou $k(x)$, ktorú nazývame interpoláčny kernel. Konvolúcia funkcií w a g je potom definovaná ako

$$\tilde{g}(x) = [k * g](x) = \sum_{u=-\infty}^{\infty} k(x-u) * g(u) \quad (2.2)$$

Pre každú neznámu hodnotu $\tilde{g}(x)$, teda vypočítame takúto konvolúciu. Kernel k , určuje váhy jednotlivým známym hodnotám funkcie g na základe ich vzdialosti $x - u$ od interpolovanej hodnoty v bode x v takejto nekonečnej sume. Potom na základe vhodnej definície funkcie w dokážeme realizovať rôzne approximácie. Takáto definícia konvolúcie je jednoducho rozšíriteľná do dvoch dimenzií, teda do priestoru obrázkov ako

$$\tilde{g}(x, y) = [k * g](x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} k(x-i, y-j) * g(i, j) \quad (2.3)$$

V tomto prípade teda approximujeme v bode (x, y) na základe všetkých známych hodnôt $g(i, j)$, ktorým kernel k priradí váhu na základe ich vzdialenosť od tohto approximovaného bodu. V prípade obrázkov sú týmito známymi hodnotami $g(j, j)$ farby jednotlivých pixelov v obrázku v pôvodnom rozlíšení a approximovanými hodnotami $\tilde{g}(x, y)$ sú neznáme hodnoty vo zväčšenom obrázku. Keďže všetky používané kernely k , sú v prípade interpolácie obrázkov separabilné, je možné kernel $k(x, y)$ vyjadriť ako

$$k(x, y) = k(x) * k(y) \quad (2.4)$$

Čo znamená, že pri interpolácii takýmto kernelom stačí v prvom kroku interpolovať podľa osi x a následne podľa y . V doméne obrázkov to znamená, že najskôr môžeme interpolovať riadky obrázka a následne stĺpce podľa definície jednorozmernej konvolúcie (2.2). Teda problém 2-D interpolácie sa pomocou takéhoto separabilného kernelu k dá zjednodušiť na dve 1-D interpolácie [YG+07]. Na poradí týchto jednorozmerných interpolácií nezáleží, v oboch prípadoch dosiahneme rovnaký výsledok. V prípade obrázka s viacerými farebnými kanálmi sa tento postup aplikuje na každý farebný kanál zvlášť.

V ďalších podkapitolách prezentujeme najpoužívanejšie interpolačné kernely k na interpoláciu obrázkov. Nebudeme sa zaoberať jednotlivými implementačnými detailmi ako napríklad interpolácia hodnoty v rohu obrázku a podobne, nakoľko to nie je zámerom práce. Predložíme len definíciu a tvar jednotlivých interpolačných kernelov, a ukážeme ich slabé alebo silné stránky.

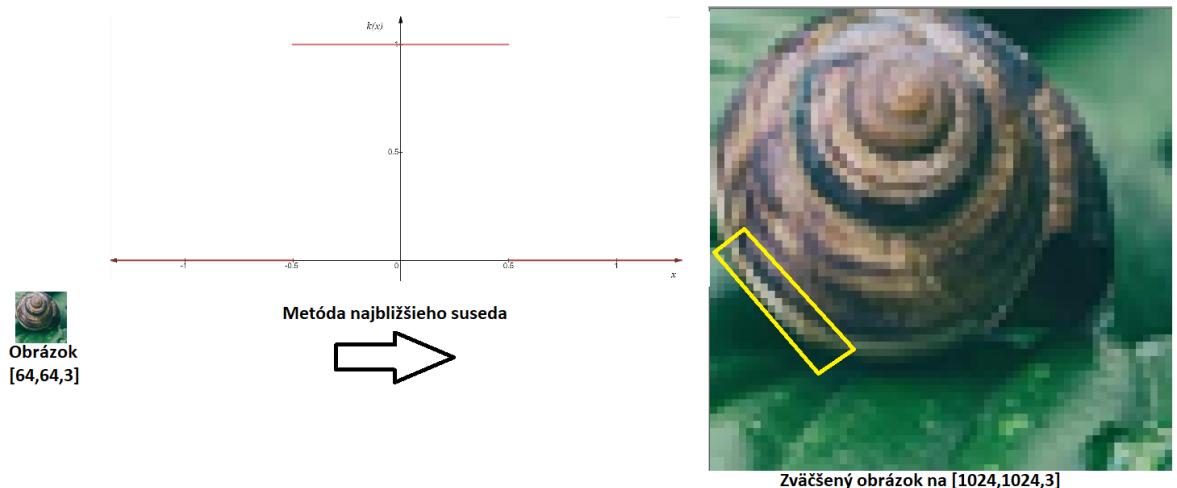
2.1 Metóda najbližšieho suseda

Metóda najbližšieho suseda (angl. *Nearest Neighbor*) je najjednoduchšia interpolačná metóda, kde hodnota neznámeho pixela je rovnaká, ako hodnota jeho najbližšieho susedného známeho pixela. Interpolácia pomocou metódy najbližšieho suseda je jediná, ktorá do interpolovaného obrázka nevnáša žiadne nové hodnoty (farby). Iný pohľad, akým sa dá pozerať na fungovanie tejto metódy je taký, že ak zväčšujeme obrázok napríklad faktorom $k = 2$, tak každý pixel pôvodného obrázka sa skopíruje na oblasť 2×2 v zväčšenom obrázku.

Interpolačný kernel k je v prípade metóde najbližšieho suseda teda definovaný ako

$$k(x) = \begin{cases} 1 & ; ak |x| \leq 0.5 \\ 0 & ; inak \end{cases} \quad (2.5)$$

,kde premenná x znamená vzdialenosť interpolovaného pixela, teda neznámeho pixela, ktorého hodnotu nepoznáme, od známeho pixela.



Obrázok 6 Príklad interpolácie najbližšieho suseda zväčšovacím faktorom $k = 16$, so zobrazenou mriežkou, žltým označená „jagged edge“, na grafe znázornený interpoláčny kernel

Vďaka tomu, že metóda najbližšieho suseda len kopíruje pixely, zachováva tak ostrosť obrázku. Jednými z nevýhod sú takzvané jagged edges (*voľný preklad* nehladké hrany), ktoré sa prejavujú tým, že hrany zväčšeného obrázky majú tvar „schodov“. Ďalšou nevýhodou je, že výsledný obrázok vyzerá moc „pixelovo“ (*angl. pixelated*), teda, moc „hranato“. Metóda najbližšieho suseda je výpočtovo rýchla metóda interpolácie a využíva sa hlavne v „pixelovom“ umení alebo len pri veľmi malých zväčšeniach (*angl. pixel art*).

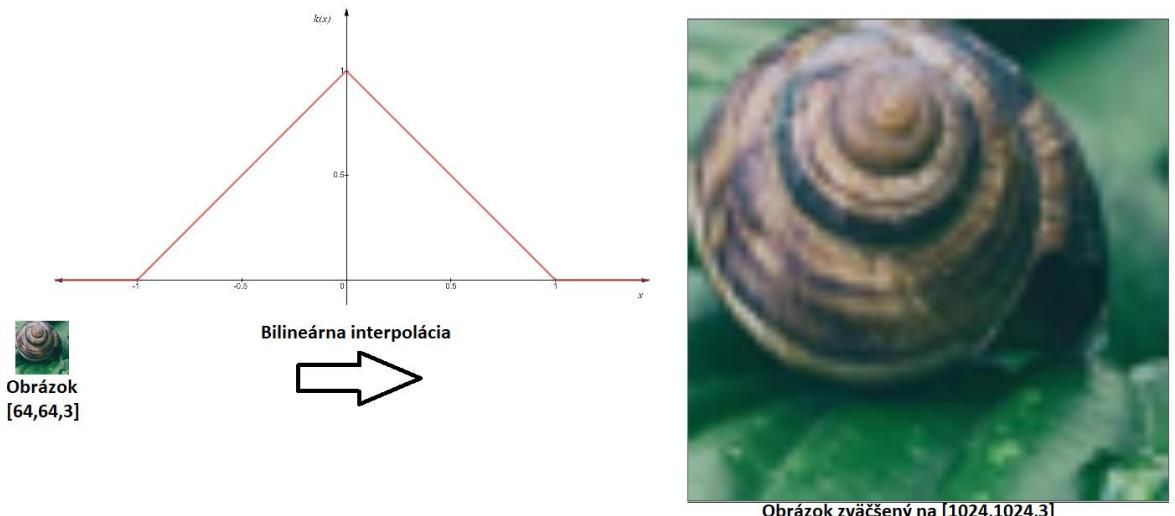
2.2 Bilineárna interpolácia

Interpoláčny kernel k bilineárnej interpolácie má tvar trojuholníka, vďaka čomu je viac hladší ako kernel metódy najbližšieho suseda, čo sa priamo prejavuje napohľad „hladším“ zväčšeným obrázkom. Artefakt jagged hrán (viď Obrázok 6) je pri použití bilineárnej interpolácie menej prítomný ako v prípade metóde najbližšieho suseda.

Kedže kernel bilineárnej interpolácie má nenulové hodnoty v intervale $< -1; 1 >$, znamená to, že určuje váhy dvom najbližším susedným známym pixelom. Kedže, ako bolo spomínané, interpolácia v prípade 2-D je separabilná na dve 1-D interpolácie, vo výsledku to znamená, že bilineárna interpolácia berie do úvahy 2×2 okolie známych pixelov od toho approximovaného.

Interpoľačný kernel bilineárnej interpolácie je potom definovaný ako

$$k(x) = \begin{cases} 1 - |x| & ; ak |x| \leq 1 \\ 0 & ; inak \end{cases} \quad (2.6)$$



Obrázok 7 Bilineárna interpolácia obrázka

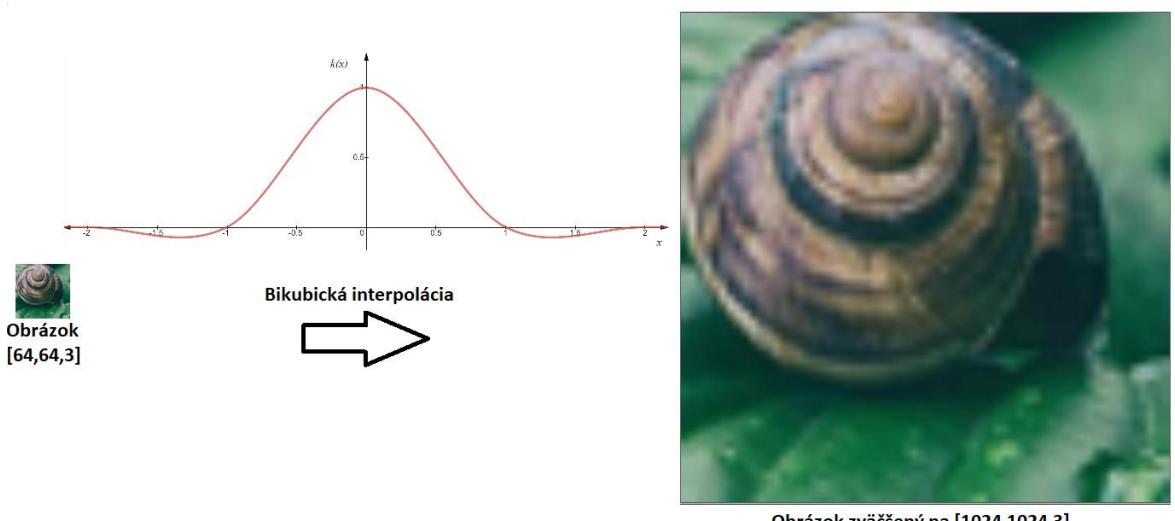
Bilineárny kernel sa všeobecne využíva hlavne pri zväčšovaní malých textov, ktorým sa okrem iného venujeme v experimentálnej časti práce.

2.3 Bikubická interpolácia

Bikubická interpolácia (angl. *Bicubic interpolation*) je rozšírením bilineárnej interpolácie tak, že pokým v lineárnej interpolácii sa na výpočet hodnoty neznámeho pixelu použili jeho 2x2 okolie (štvrť susedné známe pixely), tak v prípade bikubickej interpolácie používame jeho 4x4 okolie (šesťnásť známych susedných pixelov). Kernel k je v tomto prípade definovaný tak, aby bližším pixelom bola priradená väčšia váha $k(x)$, kde x je spomínaná vzdialenosť interpolovaného pixela k známemu pixelu.

Bikubická interpolácie typicky obsahuje ešte jeden voliteľný parameter a , ktorý sa v každej implementácii lísi. Napríklad v toolkite Matlab $a = -0.5$ a v prípade OpenCV $a = -0.75$. Interpoľačný kernel k je v tomto prípade definovaný ako

$$k(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1 & \text{ak } x \leq 1 \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a & \text{ak } 1 < x < 2 \\ 0 & \text{inak} \end{cases} \quad (2.7)$$



Obrázok 8 Príklad bikubickej interpolácie

Táto metóda interpolácie je najviac používaná a aplikovateľná na všetky druhy obrázkov. Metóda má veľmi dobrý pomer jej výsledkov a výpočtovej zložitosti, a preto je v niektorých softvérových nástrojoch na prácu s obrazom práve bikubická metóda interpolácie nastavená ako predvolená (Photoshop, CorelDraw Graphics suite).

2.4 LANCZOS interpolácia

Interpolácia pomocou Lanczosovho interpolačného kernela je z pomedzi všetkých prezentovaných interpolácií najviac výpočtovo zložitá, ale všeobecne sa považuje za jednu z najlepších. Takisto ako bikubická interpolácia, aj Lanczosov kernel obsahuje voliteľný parameter a . V tomto prípade však hodnota parametra a určuje veľkosť okolia známych pixelov, ktoré budú použité na výpočet hodnoty toho neznámeho (interpolovaného). V práci

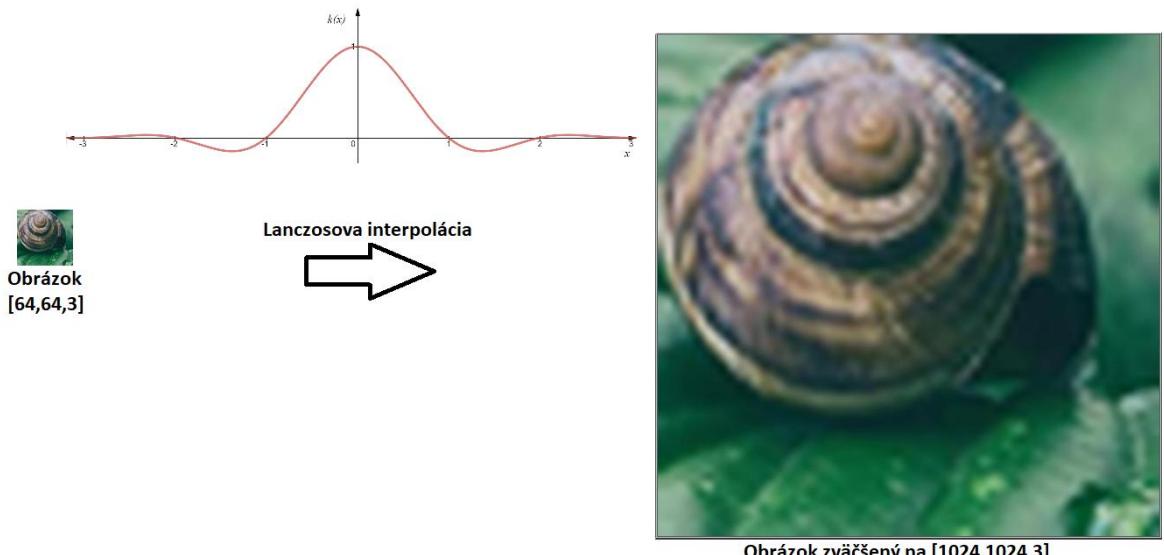
sa budeme zaoberať dvoma najpoužívanejšími, kde $a = 3$ a $a = 5$. Budeme ich nazývať LANCZOS3 respektíve LANCZOS5, teda LANCZOS3 používa 3×3 okolie a LANCZOS5 5×5 okolie známych pixelov, ktorým následne kernel k pridelí váhu na základe ich vzdialenosťi od interpolovaného pixelu, ktorá bude použitá v samotnom výpočte tejto hodnoty, teda spomínanej konvolúcie (2.2).

Lanczosov interpolačný kernel vychádza z ohraničenia inak teoreticky ideálneho filtra, ktorý neprepustí signál vyšších frekvencií (*angl. low pass filter*) [Turk90]. Týmto ideálnym filtrom je funkcia *sinc* definovaná ako

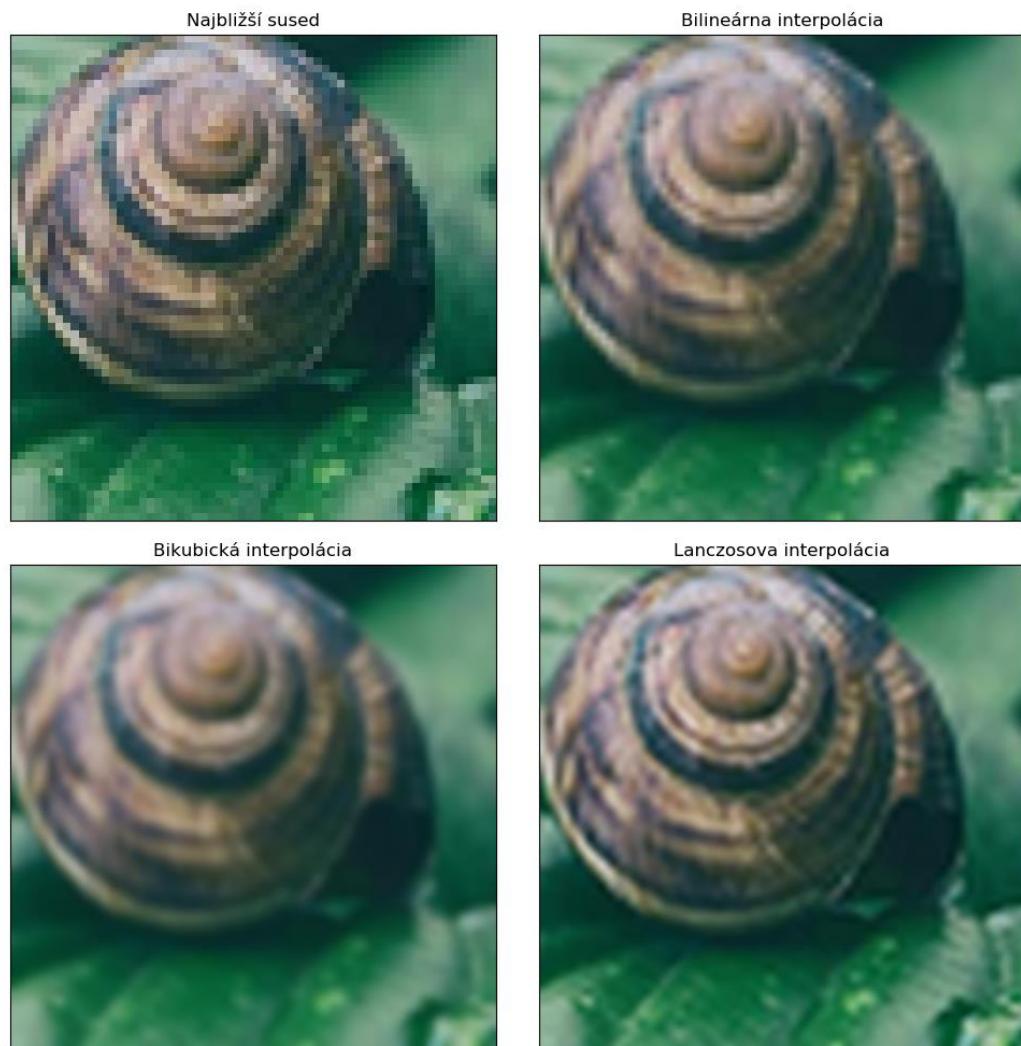
$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \quad x \neq 0 \quad (2.8)$$

Lanczosov kernel k je potom definovaný ako

$$k(x) = \begin{cases} \text{sinc}(x) * \text{sinc}\left(\frac{x}{a}\right) & ; ak -a < x < a \\ 0 & ; inak \end{cases} \quad (2.9)$$



Obrázok 9 Príklad Lanczosovej interpolácie, $a = 3$



Obrázok 10 Porovnanie interpolačných metód

3 Upsampling založený na neurónových sietiach

V tejto kapitole uvedieme riešenie formulovaného problému zväčšovania rozlíšenia založené na neurónových sietiach, predstavíme použitú architektúru a čitateľa uvedieme do problematiky a princípu fungovania neurónových sietí.

3.1 Umelé neurónové siete

Umelé neurónové siete (*angl. Artificial neural networks*) sú jedným z hlavných nástrojov používaných v oblasti strojového učenia (*angl. machine learning*). Princíp fungovania umelých neurónových sietí vychádza z biologického princípu fungovania mozgu, ktorý sa na základe vonkajších stimulov dokáže v čase učiť a riešiť tak stále zložitejšie úlohy. Učenie ľudského mozgu je založené na prepojení malých buniek v mozgovej kôre, tieto bunky sa nazývajú neuróny.

Neuróny v nervovej tkanine sú navzájom prepojené a komunikujú pomocou elektrických impulzov. Každý neurón tieto elektrické impulzy príjme cez vlákna nazývané dentrity. Následne v some, ktoré sa chápe ako jadro neurónu, tieto elektrické signály spracuje a transformuje na iný signál. Tento spracovaný signál neurón následne cez vlákno nazývané axón sprostredkuje iným neurónom, ktoré sú s týmto axónom prepojené časťou nazývanou synapsa. Problematika „učenia sa“ biologického mozgu je veľmi zložitá a venuje sa jej vedecký obor nazývaný neurobiológia. Veľmi zjednodušene povedané sa mozog učí zmenou intenzít spojení neurónov, prípadne vytváraním nových alebo zánikom existujúcich spojení.

Tento zjednodušený model sa po prvýkrát snažili matematicky formulovať Warren McCulluch a Walter Pitts v roku 1943. V ich článku [MP43] sa pokúsili demonštrovať to, že Turingov stroj môže byť implementovaný v konečne dlhej sieti navzájom prepojených výpočtových jednotiek založených na abstrakcii fungovania neurónu v biologickom mozgu.

3.1.1 McCullochov-Pittsov umelý neurón

Nimi prezentovaný model [MP43] neurónu funguje na princípe dvoch funkcií f a g , kde g predstavuje sčítanie vstupov x takéhoto neurónu, čo je v analógií so spomínanými dentritami a somou. Následne na základe výstupu tohto spracovania, funkcia f spraví rozhodnutie o výstupe neurónu y na základe prahovej hodnoty θ . Vstupné hodnoty

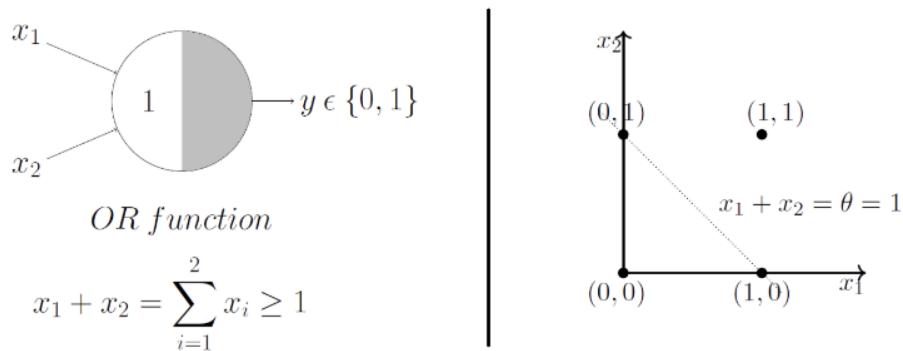
x_0, x_1, \dots, x_n , ako aj výstupná hodnota y , sú z množiny $\{0,1\}$. Formálne teda je McCulloch-Pittsov model neurónu definovaný ako

$$g(x) = \sum_{i=0}^n x_i \quad ; x_i \in \{0 ; 1\} \quad (3.0)$$

$$f(x) = \begin{cases} 1; & x \geq \theta \\ 0; & x < \theta \end{cases} \quad (3.1)$$

$$y = f(g(x)) \quad (3.2)$$

Takto definovaná funkcia dokáže reprezentovať niektoré z logických operácií na základe vhodného nastavenia prahovej hodnoty θ . V prípade, ak chceme reprezentovať funkciu *OR* je hodnota $\theta = 1$ a pre logickú funkciu *AND* je to hodnota $\theta = 2$.



Obrázok 11 McCulloch-Pittsov model neurónu pre funkciu *OR*

Zdroj:[Chan18]

Na Obrázok 11 vidíme, že všetky body (dvojice) (x_1, x_2) , kde výstup funkcie $OR(x_1, x_2) = 1$, sa nachádzajú na alebo nad priamkou. Dvojica $(0,0)$ pre ktorú je výstupom funkcie *OR* hodnota 0 sa nachádza pod priamkou. Tento fakt sa nazýva lineárna separabilita. Teda, že jednotlivé kategórie výstupov, v tomto prípade 0 a 1, vieme rozdeliť jedinou priamkou. V prípade viacerých dimenzií, teda pri väčšom množstve vstupov x_i to nebude priamka, ale viacrozmerná rovina. V prípade funkcie *XOR* lineárna separabilita neplatí, a teda jeden takto definovaný neurón nestačí na reprezentovanie tejto funkcie.

Model, ktorí predložili McCulloch a Pitts v roku 1943 bol abstrakciou vtedajších poznatkov neurobiológie. V tejto dobe sa ešte nevedelo o takzvanom Hebbovom [Hebb49] pravidle. Tento fakt o fungovaní mozgu prezentovaný v roku 1949 znamenal okrem iného aj potrebu zakomponovania do formálneho modelu neurónu k jednotlivým vstupom x_i aj

váhy w_i , čo viedlo k myšlienke dodnes používaného modelu neurónu, nazývaného perceptrón.

3.1.2 Perceptrón

Perceptrón, prezentovaný v roku 1958 Frankom Rosenblattom v článku [Rose58], je zovšeobecnením McCulloch-Pittsoveho modelu s rešpektovaním Hebbového pravidla [Hebb49]. Zovšeobecnenie modelu spočíva v rozšírení intervalu možných hodnôt vstupov x_i a výstupu y na obor reálnych čísel. Zakomponovanie Hebbového pravidla znamenalo priradenie ku každému vstupu x_i jeho váhu w_i .

Formálne je teda model perceptrónu zapísaný ako

$$g(x) = \sum_{i=0}^n w_i x_i ; \quad x_i, w_i \in R \quad (3.3)$$

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \quad (3.4)$$

$$y = f(g(x) + b) \quad (3.5)$$

Ďalšie zovšeobecnenie pôvodného modelu (3.0 – 3.2) spočíva vo funkcií f , ktorá sa nazýva aktivačná. Aktivačná funkcia je funkcia, ktorá transformuje vstup x do nejakého intervalu. V prípade pôvodného perceptrónu je ňou funkcia (3.4), ktorá sa nazýva binary step, a ktorá sa využíva pri binárnej klasifikácii. Binárna klasifikácia priradí vstupu x jednu z dvoch možných tried, napríklad. Takto definovaný model je možné trénovať na binárnu klasifikáciu algoritmom popísaným v [Rose58] a preto sa v dnešnom ponímaní pojmom perceptrón skôr spája s týmto tréningovým algoritmom.

3.1.3 Umelý neurón

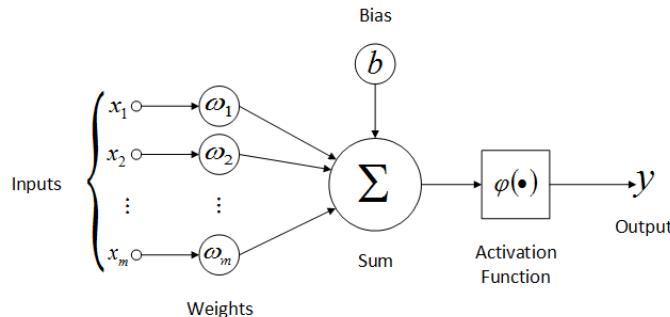
Dnešný model neurónu používaný v umelých neurónových sieťach je formálne definovaný ako

$$g(x) = \sum_{i=0}^n w_i x_i , \quad x_i, w_i \in R \quad (3.6)$$

$$y = \varphi(g(x) + b) \quad (3.7)$$

, kde hodnoty x_i sú vstupy do neurónu, hodnota y je výstupom neurónu a b je takzvaný bias, ktorý má za následok posunutie krivky aktivačnej funkcie φ , čo v končenom dôsledku do

modelu prináša schopnosť väčšej flexibility posunov n-rozmernej roviny pri riešení lineárne separovateľných problémov. Schéme takéhoto formálneho neurónu je teda



Obrázok 12 Schéma umelého neurónu

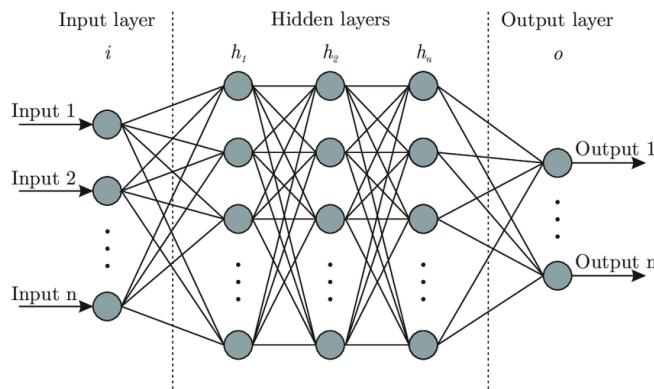
Takýto model aj napriek väčšej flexibilite a možnosti výberu vhodnej aktivačnej funkcie stále nedokáže riešiť lineárne neseparovateľné problémy. Riešenie tohto problému spočíva vo vzájomnom vrstvení a prepojení takto formulovaných neurónov [MP69].

3.1.4 Viacvrstvové neurónové siete

Minsky a Papert v článku [MP69] ukázali, že lineárne neseparovateľné problémy sa dajú riešiť zestrojením orientovaného acyklického grafu, kde jednotlivé vrcholy budú predstavovať práve spomínané umelé neuróny (3.6, 3.7). Prvá vrstva takto zestrojeného grafu sa nazýva vstupná vrstva (*angl. input layer*) a posledná vrstva sa nazýva výstupná vrstva (*angl. output layer*). Stredné vrstvy grafu nazývame skryté vrstvy (*angl. hidden layers*).

Vstupná vrstva je zložená z n vstupných vrcholov, ktorých úlohou je sprostredkovanie vstupného vektora $X = \{x_1, x_2, \dots, x_n\}$ ďalšej vrstve. Každý vrchol vstupnej vrstvy je teda prepojený so všetkými vrcholmi ďalšej vrstvy. Vrcholy tejto ďalšej vrstvy sú potom prepojené so všetkými vrcholmi nasledujúcej vrstvy. Týmto princípom pokračujeme až do výstupnej vrstvy, ktorej výstupné hodnoty predstavujú výstupný vektor $Y = \{y_1, y_2, \dots, y_n\}$. Takto organizovaný graf teda predstavuje zložitú funkciu, ktorá mapuje vstupný vektor X na korešpondujúci výstupný vektor Y . Vrstvu neurónov, ktoré majú spojenia so všetkými neurónmi ďalšej vrstvy, nazývame plne prepojená vrstva (*angl. fully connected layer*).

Takto zestrojenému grafu hovoríme dopredná neurónová sieť (*angl. Feed-Forward neural network*).



Obrázok 13 Všeobecná architektúra doprednej siete

Zdroj: [BG+17]

Jeden z najklúcovejších dôkazov v celej problematike umelých neurónových sietí predložil Cybenko v [Cyb89], kde matematicky dokázal, že takáto neurónová siet s len jednou skrytou vrstvou s konečným počtom neurónov dokáže byť univerzálnym approximátorom, teda že pri voľbe vhodných parametrov (váh a biasov) dokáže siet approximovať akúkoľvek n -rozmernú spojitú funkciu. Cybenko tento dôkaz predložil len pre sigmoidné aktivačné funkcie a pre jednu konečne dlhú skrytú vrstvu. Následne Kurt Hornik v [KH-91] dokázal, že tento fakt o univerzálnom approximátore nezáleží na zvolenej aktivačnej funkcií φ , ale vychádza z vzájomného prepojenia vrstiev. V roku 2017 v článku [LP+17] bolo ukázané, že akúkoľvek spojitú funkciu n vstupných premenných dokáže approximovať viacvrstvová siet so šírkou $n + 1$ s použitím aktivačných funkcií ReLU (Rectified linear unit).

Aktivačné funkcie ReLU a hyperbolický tangens Tanh

Rectified linear unit (skr. ReLU) je najpoužívanejší typ aktivačných funkcií pre množstvo typov neurónových sietí. Funkcie ReLU ukazujú rýchlejšiu konvergenciu pri tréningu a takisto sú menej náročné na výpočet ako predtým používané sigmoidné funkcie. V práci využívame dve konkrétnie ReLU funkcie a to

- Klasická ReLU – definovaná ako $\varphi(x) = \max(0, x)$ (3.8)

$$\bullet \text{ Leaky ReLU – definovaná ako } \varphi(x) = \begin{cases} x & ; x > 0 \\ 0.01x & ; \text{ inak} \end{cases} \quad (3.9)$$

V práci takisto využívame funkciu hyperbolického tangensu (*d'alej tanh*). Funkcia \tanh má na rozdiel od funkcií ReLU menší obor hodnôt, konkrétnie $(-1, 1)$. Aktivačná funkcia \tanh

sa preto používá najmä vo výstupnej vrstve neurónovej siete, teda keď chceme aby výstup siete pochádzal z tohto intervalu. Funkcia \tanh je definovaná ako

$$\tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \quad (3.10)$$

3.1.5 Trénovanie umelých neurónových sietí

Ako sme spomínali vyššie, dôkazy o univerzálnom approximátore vychádzali z predpokladu vhodne zvolených váh a biasov jednotlivých spojení na základe funkcie, ktorú sa snažíme pomocou neurónovej siete approximovať. Toto vhodné nastavenie váh však nie je prakticky možné, keďže tieto váhy vopred nepoznáme. Preto je nutný proces, v ktorom sa na základe vstupných a ich korešpondujúcich výstupných vektorov budú upravovať váhy siete tak, aby sa výstup siete stále viac a viac podobal výstupu, ktorý od siete očakávame. Neurónová sieť sa týmto spôsobom dokáže naučiť approximovať aj veľmi zložitú funkciu, ktorá vyjadruje vzťah medzi vstupmi x a výstupmi y , ale ktorej tvar nepoznáme a nie je možné ho jednoducho odvodiť alebo analiticky vypočítať. Tomuto prístupu k postupnej úprave váh neurónovej siete hovoríme tréning.

Takéto učenie, kde pre každý vstupný vektor poznáme jeho očakávaný výstup sa v terminológií strojového učenia nazýva učenie s učiteľom (angl. supervised learning). Ako príklad môžeme uviesť problém klasifikácie zvierat, teda vstupný vektor môže byť obrázok psa a od neurónovej siete očakávame, že výstupom bude trieda zvieraťa, ktoré sa na obrázku nachádza, teda v tomto prípade pes. Problém klasifikácie ešte predstavíme bližšie v ďalších kapitolách.

Algoritmus spätného šírenia chýb

Jeden z prvých, ale v súčasnosti stále najpoužívanejším algoritmom pre tréning neurónovej siete, pri učení s učiteľom je algoritmus spätného šírenia chýb (angl. a ďalej Backpropagation).

Základná myšlienka Backpropagation algoritmu bola predstavená už v roku 1960 v článku [KH60], ktorý sa venoval teórií riadenia. Pre trénovanie viacvrstvových neurónových sietí bol algoritmus prezentovaný až v roku 1986 v článku [RHW86].

Algoritmus Backpropagation vychádza z tréningového pravidla nazývaného delta pravidlo (angl. delta rule alebo Widrow-Hoff learning rule, LMS rule), pre jednovrstvové siete, teda napríklad prezentovaný perceptrón (viď 3.1.2).

Backpropagation je algoritmus na vypočítanie gradientu chybovej funkcie (*angl. Loss function, Error function, Cost function, Objective function*) doprednej neurónovej siete, kde chybová funkcia $Loss(y_e, y_c)$ je funkcia vyjadrujúca rozdiel medzi očakávaným výstupom siete y_e a reálnym výstupom siete y_c pri vstupe x . Snahou tréningového procesu je teda minimalizovanie hodnoty $Loss$. Backpropagation algoritmus počíta gradienty v opačnom smere akým je realizované spomínané mapovanie vstupného vektora x na výstupný vektor y , teda backpropagation algoritmus začína vo výstupnej vrstve a končí v prvej skrytej vrstve siete. Týmto princípom sa vynechávajú redundantné výpočty a algoritmus je tak efektívny a realizovateľný aj pre hlboké neurónové siete, teda siete s viac ako jednou skrytou vrstvou.

Funkcia $Loss$ musí byť diferenciovateľná, teda musí existovať jej derivácia $Loss'$. Backpropagation algoritmus, analyticky a efektívne podľa metód dynamického programovania a na základe reťazového pravidla (*angl. chain-rule*) vypočíta parciálne derivácie chybovej funkcie podľa každého parametra neurónovej siete, teda vypočíta hodnoty

$$\frac{\partial Loss}{\partial w_{ij}^k} \quad (3.11)$$

,ktoré tvoria spomínany gradient. Hodnota w_{ij}^k , predstavuje váhu neurónu j vo vrstve k , pre vstup z neurónu i z predchádzajúcej vrstvy. Na základe hodnôt týchto parciálnych derivácií potom podľa iných rôznych prístupov k samotnej úprave váh siete dokážeme zlepšovať jej aproximačné schopnosti, a to na základe skutočnosti, že gradient principiálne vyjadruje smer najväčšieho rastu funkcie, v našom prípade teda chybovej funkcie. Čo znamená, že v prípade minimalizácie chybovej funkcie, teda ak chceme menší rozdiel medzi hodnotami y_e a y_c musíme váhy siete w_{ij}^k meniť v opačnom smere gradientu. Proces zmeny týchto váh je optimalizačná oblast', ktorá sa neustále vyvíja a vznikajú stále novšie a sofistikovanejšie techniky. V práci využívame metódu ADAM, ktorá vychádza z najznámejšej metódy gradient descent a ktorú predstavíme neskôr.

Algoritmus a problémy tréningu neurónovej siete

Ak sme si už predstavili spôsob výpočtu parciálnych derivácií chybovej funkcie podľa váh siete, potom tréning je iteratívny proces dvoch krokov vykonávaných v cykle nazývanom tréningový cyklus (*angl. training loop*).

1. Dopredné šírenie – predstavuje krok v ktorom neurónová sieť vypočíta výstupný vektor y zo vstupného vektora x na základe súčasne nastavených parametrov siete.
2. Úprava parametrov – v tomto kroku sa na základe výstupného vektora siete a očakávaného výstupu vypočíta hodnota chybovej funkcie. Následne sa pomocou backpropagation algoritmu vypočítajú spomínané parciálne derivácie chybovej funkcie, ktoré sa pomocou zvoleného optimalizačného algoritmu, napr. ADAM využijú vo výpočtoch nových hodnôt parametrov siete v ďalšej iterácii tréningu.

Tento tréningový cyklus pokračuje až pokým nie je splnená určená podmienka. Touto podmienkou väčšinou býva počet epoch (vysvetlené nižšie) tréningu alebo dosiahnutá zvolená hodnota chybovej funkcie, poprípade nejaká iná dosiahnutá hodnota vybranej metriky.

Ked'že hodnoty parciálnych derivácií a teda aj hodnota gradientu chybovej funkcie v iterácii tréningu sú počítané na základe konkrétneho vstupného vektora x , ideálnym prípadom by bolo keby vstupný vektor x obsahoval všetky tréningové dát. Toto však nie je v bežnom prípade možné, pretože tréningové množiny typicky obsahujú veľa dát a tak si vyžadujú veľa pamäti na jej uloženie. Ked'že z dôvodov paralelizácie výpočtov potrebných na fungovanie neurónových sietí sú objekty neurónovej siete uložené v pamäti grafickej karty, ktorá je typicky rádovo menšia ako iné pamäte s väčšou odozvou dostupnosti dát, nie je možné celú tréningovú množinu uložiť priamo na pamäť grafickej karty.

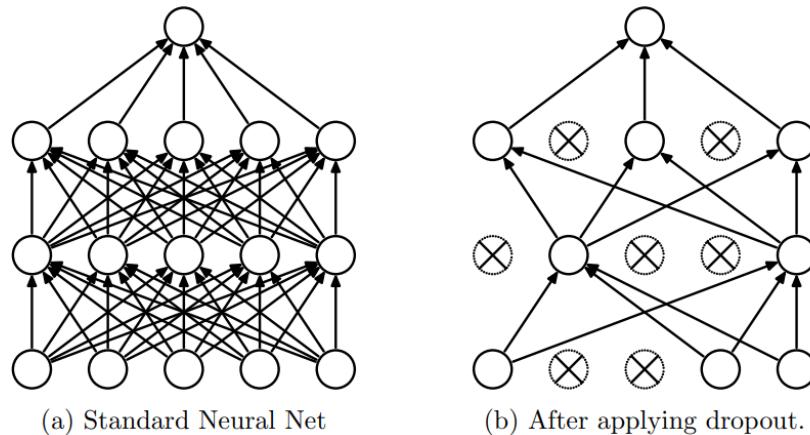
Na riešenie tohto kapacitného problému sa dátá tréningovej množiny rozdeľujú do dávok nazývaných batche. Každý batch teda obsahuje niekoľko prvkov tréningovej množiny (typicky mocnina 2), ktoré voláme samples. Každý sample teda obsahuje páru (vstupné dát : očakávaný výstup). V tomto prípade v tréningovom algoritme v kroku 1 je vstupným vektorom x celý takýto batch.

Gradient chybovej funkcie v každej iterácii tréningu je teda počítaný na základe vstupného batchu dát. Ked'že tento batch dát neobsahuje všetky prvky tréningovej množiny, vähy v kroku 2 sú upravované len na základe tej časti. Preto je dôležité aby vstupným vektorom bol v každej iterácii tréningu iný batch a tak sa v procese tréningu obsiahla celá variabilita tréningovej množiny. Moment ked' týmto iteratívnym procesom prejde celá tréningová množina, teda všetky batche, sa nazýva epocha.

Počet epoch tréningu typicky býva základným parametrom tréningu a záleží od zložitosti skrytej závislosti medzi vstupmi a výstupmi siete a od jej zvolenej architektúry. Jedným z problémov výberu vhodného počtu epoch je takzvané pretrénovanie(*angl.* overfitting). Pretrénovanie nastáva vtedy, keď hodnota chybovej funkcie je súčasťou malá, ale len v prípade tréningovej množiny. Sieť sa teda veľmi dobre naučila modelovať zložitú závislosť medzi vstupom a výstupom, ale pri testovacej množine zlyháva. Testovacia množina obsahuje dátá určené na overovanie aproximačných schopností siete (validovanie), teda obsahuje dátá, ktoré siet' v procese tréningu „nevidela“. Jav pretrénovania typicky nastáva pri veľkom množstve epoch tréningu. Problém pretrénovania je teda veľmi nežiadúci jav, ktorý sa skúma aj v súčasnosti, čo viedie k vzniku rôznych techník na jeho zamedzenie.

Zamedzenie pretrénovaniu pomocou dropoutu

Jednou z možností ako znížiť nežiadúci efekt pretrénovania, ktorú v práci využívame, je princíp nazývaný dropout. Dropout je jedna z regularizačných techník pre hlboké neurónové siete predstavená v [SH+14]. Princíp dropoutu je jednoduchý. V každej iterácii tréningového cyklu sú náhodne zvolené neuróny deaktivované, teda ich váhy sú nastavené v tejto iterácii tréningu na hodnotu 0. Táto deaktivácia je realizovaná len počas tréningu. Počas validovania siete sú prítomné vo výpočte všetky neuróny siete.



Obrázok 14 Model dropoutu, (a) aktívne všetky neuróny, (b) niektoré neuróny neaktívne

Dropout býva zvyčajne implementovaný pre zvolenú vrstvu neurónov, teda vybranej vrstve sa určí podiel náhodných neurónov, ktoré budú v tejto iterácii tréningu aktívne. Napríklad dropout s hodnotou 0.5, znamená, že 50% náhodne vybraných neurónov bude aktívnych v danej iterácii tréningu.

Adaptive moment estimation (ADAM)

Adaptívny odhad momentu (*dalej ADAM*) [KB15], je jedna z najpoužívanejších a revolučných optimalizačných gradientových metód navrhnutých špeciálne pre hlboké neurónové siete, teda siete s aspoň jednou skrytou vrstvou. Adam vo všeobecnosti dosahuje rýchlejšiu konvergenciu v tréningu a vychádza zo štatistickej teórie, kde moment n-tého rádu náhodnej veličiny X je definovaný ako

$$m_n = E[X^n] \quad (3.12)$$

, kde $E[X^n]$ je stredná hodnota náhodnej veličiny X . Gradient chybovej funkcie L v jednej iterácii môže byť chápány ako náhodná premenná, pretože v každej iterácii tréningu je počítaný na základe iných vstupných dát. Adam využíva dve premenné m_t a v_t , ktoré sa po každej iterácii menia podľa vzťahov

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (3.13)$$

$$v_t = \beta_2 m_{t-1} + (1 - \beta_2) g_t^2 \quad (3.14)$$

Hodnoty m_t a v_t matematicky vyjadrujú necentrovaný odhad momentu prvého (stredná hodnota) a druhého rádu (necentrovaný rozptyl) a hodnoty β_1 a β_2 predstavujú voliteľné parametre, vo väčšine implementácií nastavených na 0.9 a 0.999. Premenná g_t vyjadruje gradienty chybovej funkcie vypočítané podľa princípu prezentovaného v predchádzajúcej kapitole.

Následne sa podľa vzťahov

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (3.15)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (3.16)$$

vypočítajú centrované odhady momentov prvého a druhého rádu \hat{m}_t a \hat{v}_t , ktoré sa následne podľa vzťahu

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \delta} \hat{m}_t \quad (3.17)$$

použijú na výpočet váhovej matice θ_{t+1} , ktorá obsahuje nové hodnoty váh w_{ij}^k neurónovej siete v ďalšej iterácii tréningu. Hodnota η sa nazýva learning rate a je ďalším parametrom algoritmu. Hodnota $\delta = 10^{-8}$.

3.2 Konvolučné neurónové siete

Dopredné neuronové siete, o ktorých sme doteraz hovorili, fungujú dobre na klasifikačné problémy založené na vopred známych definovaných vzorov (vlastnosti, charakteristik, črt, *angl. a d'alej features*). Ako príklad môžeme uviesť situáciu, keby na základe informácií ako počet asistencií, počet gólov, nahrávok a podobne určujeme pozíciu hráča vo futbale, teda či ide o obrancu, útočníka alebo brankára. Takáto situácia pri práci s obrazom nie je možná, pretože vopred nepoznáme features, ktoré vystihujú daný obrázok, napríklad pri obrázku auta nevieme jasne povedať prečo vieme, že na obrázku je auto, môže to byť tým, že sa na obrázku nachádzajú štyri kruhy, ktoré pripomínajú kolesá alebo tým, že vo vnútri sedia ľudia, ktorí sú pripútaní bezpečnostným pásom a podobne. Tento problém okrem iného riešia práve konvolučné neurónové siete.

Konvolučné neurónové siete sú jednou z mnohých rôznych architektúr neurónových sietí. Tieto siete sú špecificky navrhnuté na prácu s viacdimenzionálnymi vstupmi, kde medzi jednotlivými vzorkami tohto vstupu existujú závislosti, ktoré nevieme vopred definovať alebo určiť. Preto sa konvolučné siete najviac používajú v úlohách, kde vstupom do siete je obrazová informácia.

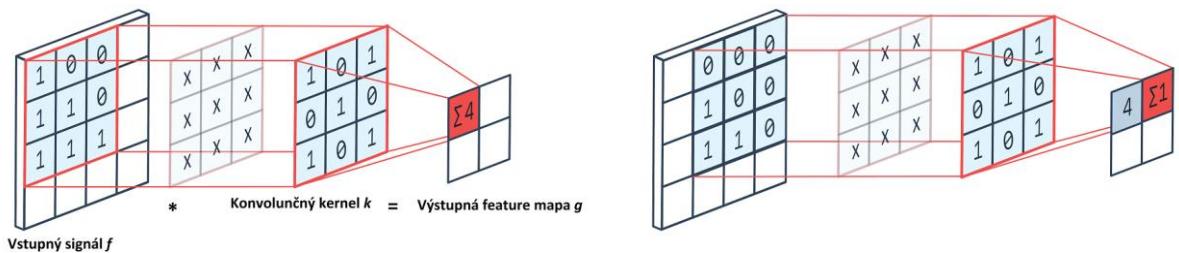
3.2.1 Konvolúcia

Ako názov vypovedá, tento typ sietí vychádza z princípu matematickej operácie konvolúcie, ktorú sme predstavili v (2.2). V tomto prípade však konvolučný kernel k nebude spojitá funkcia, ale bude ňou matica k . Konvolúcia je potom v tomto prípade definovaná ako

$$g[m, n] = (f * h)[m, n] = \sum_i \sum_j f[m - i, n - j] * k[i, j] \quad (3.18)$$

Takto definovaná konvolúcia bude operáciou formálneho neurónu z kapitoly 3.1.3. Teda funkcia neurónu g nebude v tomto prípade vážená suma (3.6), ale bude to vyššie definovaná konvolúcia.

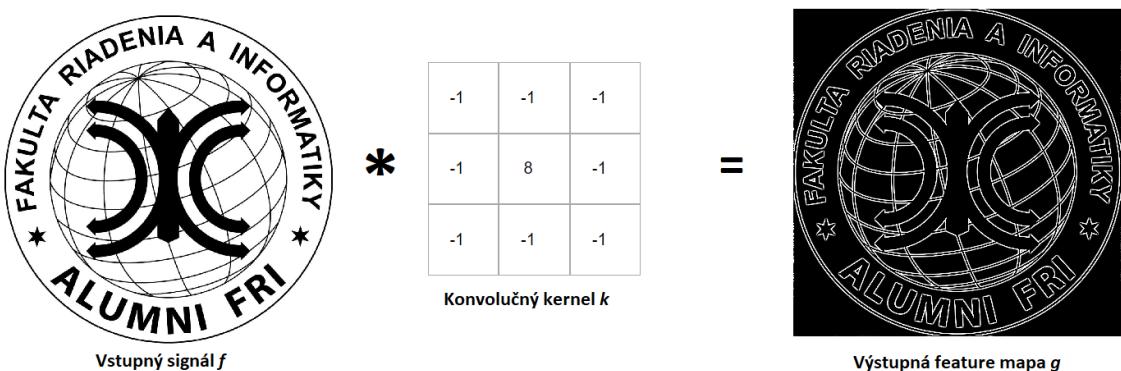
Pre lepšiu ilustráciu je operácia konvolúcie „posúvanie“ kernela k po celom vstupe neurónu



Obrázok 15 Ilustrácia výpočtu dvoch hodnôt v konvolúcií

Zdroj: <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-convolution-block>

Výstup konvolúcie g sa nazýva mapa vzorov (*angl. a ďalej feature mapa*), ktorá pri zvolení vhodného konvolučného kernela k obsahuje hodnoty, ktoré reprezentujú charakteristické črty vstuпу. Ako príklad môžeme uviesť známy konvolučný kernel pre detekciu hrán



Obrázok 16 Príklad konvolúcie s kernelom k na detekciu hrán

Na Obrázok 16 je vstupom do konvolúcie obrázok s jedným farebným kanálom a teda sa používa len jeden konvolučný kernel k . Pre prípad s viacerými farebnými kanálmi sa použije pre každý farebný kanál jeden filter.

Ako vidíme, klúčovým faktorom sú hodnoty kernelu. Backpropagation algoritmus však umožňuje vypočítanie parciálnych derivácií chybovej funkcie aj podľa hodnôt takého kernelu, vďaka čomu vie následne optimalizačný algoritmus (napríklad prezentovaný ADAM) pre zmenu váh, teda v tomto prípade hodnôt kernelu, zmeniť tieto hodnoty tak, aby hodnota chybovej funkcie klesala, teda „rozdiel“ medzi výstupom siete a očakávaným

výstupom bol menší. Čo v konečnom dôsledku znamená, že hodnoty kernelu budú nastavené tak, aby vytvárali feature mapy obsahujúce charakteristické črty vstupnej triedy obrázkov.

Veľkosť posúvania kernelu k je voliteľný parameter, ktorý sa nazýva *stride*. Takisto ako vidíme z ilustrácie, stred kernela k môžeme „priložiť“ len k niektorým pixelom vstupu f tak, že celý kernel bude zakomponovaný vo výpočte. Ak chceme, aby bol kernel aplikovateľný na všetky hodnoty vstupu, je potreba zväčšiť tento vstup. Schéma takého zväčšenia sa nazýva padding a je ďalším voliteľným parametrom konvolúcie. Väčšinou sa používajú dva spôsoby paddingu a to

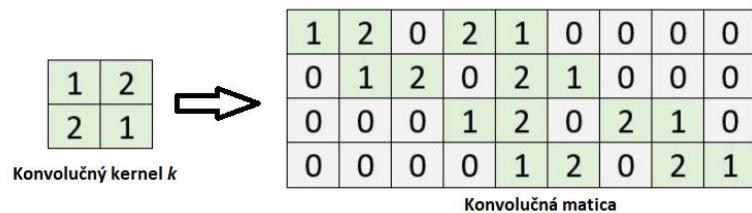
- Valid padding – vstup nie je nijako zväčšený, teda veľkosť výstupnej feature mapy bude menšia ako veľkosť vstupu
- Same padding – vstup sa rozšíri o hodnoty 0 o toľko, aby výstupná feature mapa mala rovnakú veľkosť ako vstup

V implementáciách sa pre celú vrstvu takýchto neurónov volia jednotné parametre. Vrstvu týchto neurónov nazývame konvolučná vrstva. Konkrétnie hodnoty parametrov, veľkosti kernelov a podobne predstavíme pri prezentovaní implementácie v ďalších kapitolách.

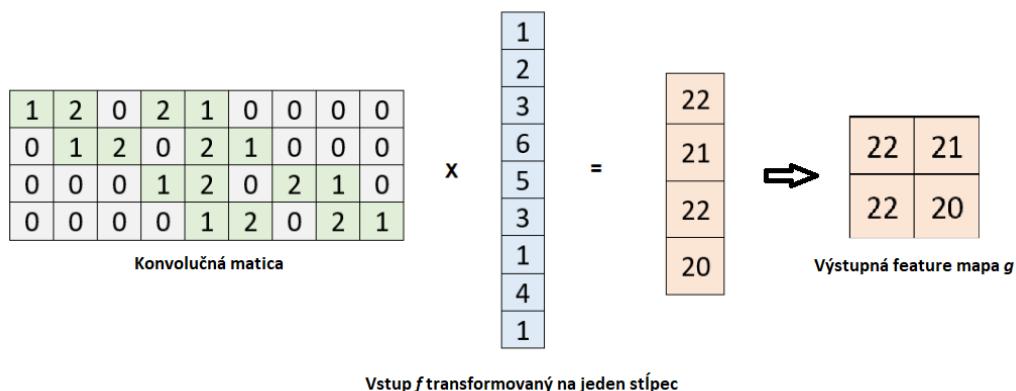
3.2.2 Transponovaná konvolúcia

Z princípu fungovania konvolúcie znázornenom na Obrázok 15 vidíme, že operácia konvolúcie znižuje veľkosť dimenzií vstupu. V niektorých úlohách však potrebujeme opačný efekt, teda vstupu potrebujeme zväčšiť veľkosť jeho dimenzií. Jednou z najflexibilnejších možností ako tento proces realizovať, je operácia transponovanej konvolúcie (*inak* inverzná konvolúcia, *angl.* transpose convolution, *alebo nesprávne, ale používane* dekonvolúcia).

Klasická konvolúcia prezentovaná v predchádzajúcej kapitole je väčšinou implementovaná ako násobenie matíc, kde sa z konvolučného kernela k vytvorí väčšia matica ktorá sa nazýva konvolučná matica

Obrázok 17 Príklad transformácie kernela k na konvolučnú maticu

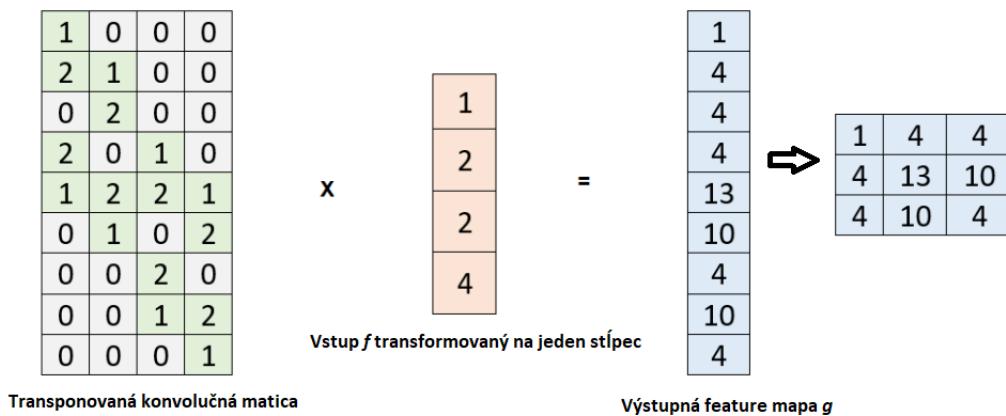
Rozostupy hodnôt medzi nulami, počet riadkov a stĺpcov konvolučnej matice záležia na spomínaných voliteľných parametroch konvolúcie stride a padding. Následne na výpočet feature mapy g stačí vstup f transformovať do matice s jedným stĺpcom a vynásobiť s takouto konvolučnou maticou.



Obrázok 18 Príklad výpočtu feature mapy pomocou konvolučnej matice

Na Obrázok 18 vidíme, že veľkosť vstupu bola 3×3 a výstupná feature mapa mala veľkosť 2×2 , čo zodpovedá tvrdeniu, že konvolúcia zmenšuje dimenzie obrázka.

Ak ale transponujeme konvolučnú maticu a vynásobíme ju vstupom o veľkosti 2×2 , tak opäť dostaneme veľkosť výstupnej feature mapy 3×3 . Teda opačným procesom ku konvolúcií je konvolúcia s transponovanou konvolučnou maticou, odtiaľ názov transponovaná konvolúcia.



Obrázok 19 Príklad výpočtu konvolúcie pomocou transponovanej konvolučnej matice

Pomocou tohto prístupu teda vieme zväčšovať dimenzie vstupe rovnakým princípom ako zmenšovať, stačí len použiť maticu s inými rozmermi. Vďaka tomu, že váhy kernelu, teda aj konvolučnej matice sú trénovateľné, vieme jej hodnoty zlepšovať tak aby hodnota chybovej funkcie klesala, čo nám v konečnom dôsledku umožňuje zväčšovať dimenzie vstupe tak, aby sa výstup čo najviac podobal očakávanému výstupu a hodnoty kernelu boli nastavené tak, aby produkovali feature mapy, ktoré zodpovedajú zväčšeným features vstupnej domény.

Takéto zväčšovanie dimenzií vstupe logicky a priamo má využitie v problematike upsamplingu obrázkov. Preto práve operácia transponovanej konvolúcie bude základným stavebným kameňom neurónových sietí, ktoré v práci využívame na riešenie tohto problému.

Upsampling metódy prezentované na začiatku práce fungujú na základe konštantných vzorcov, ktoré vôbec nesúvisia s typom obrázkov, ktoré zväčšovali. Transponovaná konvolúcia s nimi ale priamo súvisí, pretože hodnoty kernelu k sú trénované podľa konkrétnej tréningovej množiny, teda napríklad siet' učená na základe obrázkov mačiek bude mať iné hodnoty konvolučných kernelov k ako siet' trénovaná na obrázkoch psov. Tento fakt znamená, že zväčšovanie na základe transponovanej konvolúcie je viac flexibilnejšie ako zväčšovanie na základe spomínaných upsampling metódach.

3.2.3 Pooling

Pooling predstavuje operáciu podobnú konvolúcii, teda posúvanie okna hodnôt po vstupe. V prípade konvolúcie sme volali toto okno kernel, v prípade poolingu ho nazývame filter. Tento filter typicky v implementáciách býva veľkosti 2x2 a v každom kroku sa posúva

o 2 pozície, teda spomínaný parameter stride má hodnotu 2. Takýto filter postupne aplikujeme na časti vstupu o veľkosti 2×2 a na každú takúto časť aplikujeme konkrétnu formu poolingu. Medzi najviac používané formy poolingu patria

- Average pooling – výstupná hodnota je priemer hodnôt okna
- Max pooling – výstupná hodnota je maximálna hodnota okna

Tento prístup má okrem zníženia vplyvu zmeny pozícii vstupných features za následok aj redukovanie dimensií vstupu z dôvodu, že vychádza z konvolúcie. Znamená to, že v ďalších vrstvách siete, ktoré nasledujú po poolingu budú dátá menej rozmerné, čo urýchli výpočty a teda aj proces tréningu. Preto sa v niektorých architektúrach veľmi hlbokých neurónových sietí pooling používa len pre vlastnosť tohto zrýchlenia.

V práci využívame pooling len v architektúre klasifikátorov, konkrétnie jeho formu Max pooling.

3.3 Klasifikátor

Ako v kapitole prezentujeme, neurónová siet' je vlastne veľká zložitá nelineárna funkcia, ktorá robí mapovanie vstupného vektora x na výstupný vektor y . Tieto vektory môžu byť principiálne ľubovoľne veľké a obsahujú závislosť, ktorú sa pomocou neurónovej siete snažíme odhaliť a tak aproximovať skutočnú funkciu, ktorá reprezentuje túto závislosť.

Jednou z konkrétnych využití tohto prístupu je klasifikácia. Klasifikácia je proces, v ktorom vstupu priradujeme kategóriu (triedu, *angl. label*) do ktorej tento vstup patrí. Teda napríklad vstupným vektorom x budú hodnoty pixelov x_i a výstupným vektorom y budú hodnoty y_i vyjadrujúce pravdepodobnosť, že vstupný vektor x patrí do kategórie i , teda, že na obrázku je napríklad mačka, pes, líška alebo iná kategória. Následne stačí vybrať najväčšiu hodnotu z vektora y . Index tejto najväčšej hodnoty predstavuje určenú výslednú triedu vstupného vektora x .

K dosiahnutiu toho, aby na výstupe neurónovej siete boli pravdepodobnosti, teda hodnoty z intervalu $<0,1>$, ktorých súčet je 1, je potrebné zvoliť vhodnú aktivačnú funkciu pre výstupnú vrstvu siete. Najpoužívanejšia takáto funkcia je funkcia softmax, ktorá je definovaná ako

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_{j=0}^n e^{y_j}} \quad (3.19)$$

Po aplikovaní funkcie softmax na každú jednu zložku výstupného vektora y stačí následne vybrať spomínanú najväčšiu hodnotu. Index tejto najväčšej hodnoty potom predstavuje číslo výslednej kategórie pre vstup x .

Ako bolo prezentované, iterácia tréningu neurónovej siete vychádza z hodnoty chybovej funkcie, ktorej hodnota určuje rozdiel medzi očakávaným výstupom siete t a skutočným výstupom siete y pri konkrétnom vstupnom vektoru x . V prípade klasifikátora, je výstupom y spomínaný pravdepodobnostný vektor a preto chybová funkcia musí v tomto prípade vyjadrovať rozdiel medzi rozdeleniami pravdepodobností. Jednou z najpoužívanejších takýchto funkcií je krízová entropia (*angl. cross entropy*) definovaná pre diskrétné hodnoty ako

$$H(t, y) = - \sum_{i=0}^k t[i] * \log y[i] \quad (3.20)$$

Hodnota H , teda vyjadruje ako veľmi odlišné sú pravdepodobnostné vektory t a y . Vektor t predstavuje vektor s k zložkami, v ktorom $k - 1$ hodnôt je 0, a jedna hodnota je 1. Hodnota 1 je práve na pozícii skutočnej kategórie vstupu klasifikátora, teda vektora x . Takto konštruovaný vektor nazývame one-hot vektor. Hodnota k potom predstavuje počet možných kategórií do ktorých sa snažíme vstup zaradiť.

V prípade $k = 2$, teda, v prípade, že vektoru x priradujeme jednu z dvoch možných tried, hovoríme o binárnej klasifikácii. V prípade binárnej klasifikácie sa krízová entropia transformuje na binárnu krízovú entropiu BCE

$$BCE(t, y) = -t * \log(y) - (1 - t) * \log(1 - y) \quad (3.21)$$

V tomto prípade očakávaný výstup t je len jedna hodnota 0 alebo 1, ktorá vyjadruje či vstup patrí do kategórie 0 alebo 1.

Vzťah binárnej krízovej entropie pre binárnu klasifikáciu je možné prepísať aj na tvar

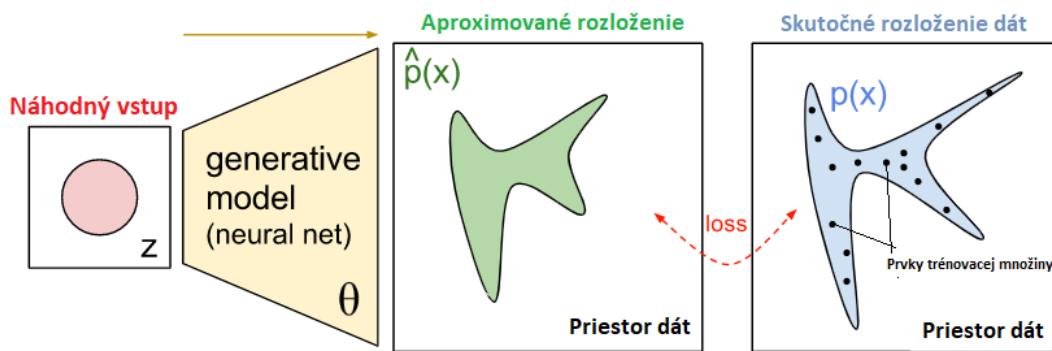
$$BCE(t, y) = \begin{cases} -\log(y) & \text{ak } t = 1 \\ -\log(1 - y) & \text{ak } t = 0 \end{cases} \quad (3.22)$$

Binárnu krízovú entropiu v práci využívame ako časť chybovej funkcie generatívneho modelu určeného na realizovanie zväčšovanie rozlíšenia, ktorý predstavíme v ďalšej časti.

3.4 Generatívne konfrontačné siete

Vhodne parametrizovaná neurónová sieť, teda sieť s vhodnými hodnotami váh, ktorá obsahuje transponované konvolučné vrstvy by teoreticky dokázala realizovať upsampling obrázkov, ale ako sa ukázalo tréning takejto jednej siete by bol neefektívny a v niektorých prípadoch nerealizovateľný. Dôvodom je to, že zväčšovanie rozlíšenia spadá do problematiky generovania dát. V oblasti generovania dát pomocou neurónových sietí je snahou vytváranie nových neexistujúcich dát, ktoré nie sú obsiahnuté v tréningovej množine siete. Príkladom je napríklad generovanie nových dizajnov oblečenia. Tréningová množina je v tomto prípade zložená z veľkého počtu obrázkov oblečenia.

Matematicky teda hľadáme také parametre siete θ , ktoré aproximujú skutočné rozdelenie $p(x)$ prvkov z tréningovej množiny (*angl. training samples*) $x_0, x_1, x_2, \dots x_n$. Skutočný predpis alebo tvar rozdelenia $p(x)$ teoreticky obsahuje všetky možné prvky z danej domény, a preto nie je prakticky možné ho analyticky vypočítať alebo nejako odvodiť. V analógií s generovaním nových dizajnov oblečenia rozdelenie $p(x)$ obsahuje všetky možné takéto dizajny.



Obrázok 20 Vizualizácia problematiky generatívnych modelov

Zdroj: <https://openai.com/blog/generative-models/>

Neurónová sieť s parametrami θ teda generuje dátá z approximačného rozdelenia $\hat{p}(x)$, ktoré sa v každej iterácii tréningu neurónovej siete porovnáva so skutočným rozdelením dát tréningovej množiny $p(x)$, kde sa následne na základe vypočítanej hodnoty chybovej funkcie (loss) upravujú parametre siete θ tak, aby sa rozloženia viac podobali, teda hodnota *loss* bola nižšia. Takýmto iteratívnym procesom vieme docieliť, že sieť sa naučí generovať nové prvky z rozloženia, ktoré je approximáciou skutočného rozdelenia z ktorého pochádzajú dátá tréningovej množiny. Ako je jasné a logické takáto jedna neurónová sieť, ktorá by

dokázala skutočne approximovať rozloženie dát by musela obsahovať veľmi veľa parametrov θ , čo by znamenalo, že tréning takejto siete by bol neprakticky dlhý a siet' by nebola použiteľná v reálnom čase. Tento problém viedol k iným sofistikovanejším myšlienkom k tréningu a architektúram neurónových sietí určených na generovanie dát.

Doteraz najpoužívanejšia architektúra a prístup k tréningu neurónových sietí určených na generovanie dát boli predstavené v [GA+14] s názvom Generative Adversarial Networks (*slov.* Generatívne Konfrontačné *siete*, *skrt.* a ďalej GAN) a považujú sa za najlepšiu myšlienku v oblasti strojového učenia za posledných 20 rokov.

GAN sa skladajú z dvoch rozdielnych modelov neurónových sietí, a to konkrétnie generátor a diskriminátor:

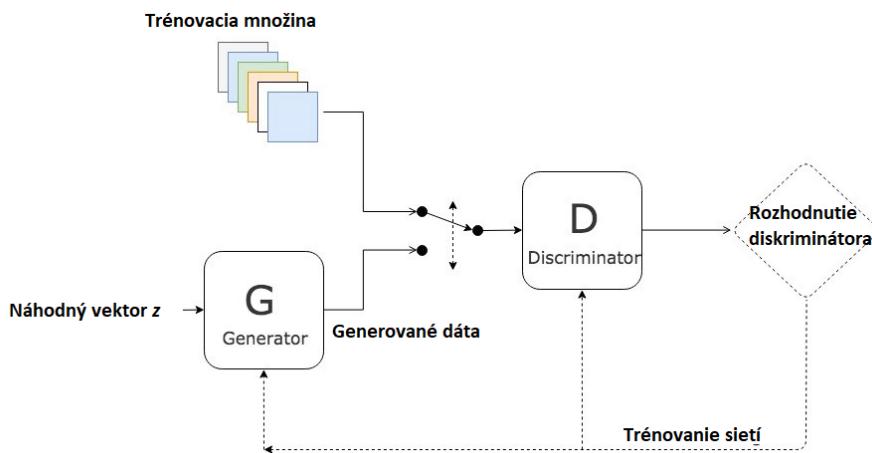
- Generátor G – úlohou generátora je generovanie „falošných“ dát, ktoré vyzerajú ako tie z tréningovej množiny. Formálnejšie je teda úlohou generátora approximovať spomínané skutočné rozdelenie tréningovej množiny $p(x)$. Operácia $G(z)$ predstavuje výstup generátora pri vstupe z , týmto vstupom do generátora býva vo väčšine prípadov Gaussovský náhodný vektor. Rozdelenie z ktorého dáta generátor generuje budeme označovať ako $p(z)$
- Diskriminátor D – úlohou diskriminátora je rozlišovať medzi dátami z tréningovej množiny a dátami, ktoré vznikli generátorom. Výstupom diskriminátora je teda skalár $D(x)$, ktorý vyjadruje pravdepodobnosť, že vstup x pochádza z tréningovej množiny a nie z generátora G . Diskriminátor D , teda plní úlohu binárneho klasifikátora, teda, že vstupu x priradí jednu z dvoch možných tried. V prípade ak vstup x pochádza z tréningovej množiny tak hodnota $D(x)$ by mala v optimálnom prípade byť 1 a v prípade ak vstup x bol generovaný generátorom G , tak hodnota $D(x)$ by mala byť 0. Hodnota $D(G(z))$ teda udáva pravdepodobnosť, že výstup generovaný generátorom G je reálny obrázok, a teda, že patrí do skutočného rozdelenia dát tréningovej množiny $p(x)$.

V prípade takto definovaných modelov je potom generátor G trénovaný aby „oklamal“ diskriminátor, teda hodnota $D(G(z))$ bola čo najbližšia hodnote 1.

Formálne teda hľadáme riešenie

$$\min_G \max_D V(D, G) = E_{x \sim p(x)} [\log(D(x))] + E_{z \sim p(z)} [\log(1 - D(G(z)))] \quad (3.23)$$

Teoretické riešenie tejto min-max hry existuje v bode $p(z) = p(x)$, teda keď rozdelenie generovaných dát bude rovnaké ako skutočné rozdelenie dát tréningovej množiny, a keď diskriminátor nebude vedieť rozlíšiť medzi generovanými a reálnymi dátami, teda keď hodnoty $D(x)$ budú blízke hodnote 0.5. Tento bod sa nazýva Nashove equilibrium, ktoré sa v problematike GANov neustále skúma a vznikajú tak sofistikovanejšie metódy ako toto equilibrium prakticky dosiahnuť.



Obrázok 21 Schéma fungovania generatívnych konfrontačných sietí

Zdroj:

<https://medium.com/sicara/keras-generative-adversarial-networks-image-deblurring-45e3ab6977b5>

Na reprezentáciu min-max problému (3.23) a tréningu sietí v základnej myšlienke GANov, ktorú v práci využívame sa používajú dve chybové funkcie, jedna pre diskriminátor a druhá pre generátor.

3.4.1 Chybová funkcia diskriminátora

Ako bolo prezentované, úlohou diskriminátora je rozlišovať medzi výstupom generátora $G(z)$ a reálnymi dátami tréningovej množiny x . Z dôvodov prezentovaných v predchádzajúcich častiach práce je v prípade tréningu použitý batch náhodných vektorov z , a reálnych dát x . Od diskriminátora očakávame, že jeho výstup D , bude v prípade reálnych vstupných dát x hodnota 1, teda očakávame, že $D(x) = 1$, a v prípade generovaných dát $G(z)$ bude výstup D hodnota 0, teda $D(G(z)) = 0$. Takéto očakávanie je matematicky

vyjadriteľné ako maximalizovanie logaritmu pravdepodobnosti pre reálne dátu x a logaritmu opačnej pravdepodobnosti pre generované dátu $G(z)$ zapísané ako

$$\text{maximalizuj } D_{loss} = \log(D(x)) + \log(1 - D(G(z))) \quad (3.24)$$

tento maximalizačný tvar je možné prepísať na minimalizačný pridaním znamienka

$$\text{minimalizuj } D_{loss} = -(\log(D(x)) + \log(1 - D(G(z)))) \quad (3.25)$$

Táto minimalizačná funkcia je vyjadriteľná a väčšinou implementovaná pomocou binárnej krízovej entropie (*BCE*) predstavenou v predchádzajúcej časti práce, potom chybová funkcia diskriminátora je

$$\text{minimalizuj } D_{loss} = BCE(1, D(x)) + BCE(0, D(G(z))) \quad (3.26)$$

3.4.2 Chybová funkcia generátora

Úlohou generátora je na základe náhodného vektora z generovať taký výstup $G(z)$, aby výstup diskriminátora pri takomto generovanom vstupe, teda hodnota $D(G(z))$ bola 1. Teda aby diskriminátor rozhadol, že generované dátu $G(z)$ pochádzajú z neznámeho rozdelenia $p(x)$, ktoré sa generátorom snažíme approximovať. Táto požiadavka na generátor je podobne ako v prípade diskriminátora vyjadriteľná ako minimalizácia logaritmu opačnej pravdepodobnosti $D(G(z))$. Minimalizačná chybová funkcia generátora je potom

$$\text{minimalizuj } G_{loss} = \log(1 - D(G(z))) \quad (3.27)$$

Prepisaná pomocou binárnej krízovej entropie (*BCE*) ako

$$\text{minimalizuj } G_{loss} = BCE(1, D(G(z))) \quad (3.28)$$

3.4.3 Tréningový cyklus GANu

Ak sme si už predstavili princíp fungovania GANov a chybové funkcie ich modelov, tréning GANov je potom iteratívny proces:

1. Z tréningovej množiny vyber batch X tréningových dát pre túto iteráciu tréningu, $X = \{x^{(1)}, \dots, x^{(m)}\}$
2. Vytvor batch náhodných vektorov $Z = \{z^{(1)}, \dots, z^{(m)}\}$
3. Generátor G podľa jeho súčasných parametrov (váh) vypočíta výstup $G(Z)$
4. Diskriminátor D podľa jeho súčasných parametrov (váh) vypočíta výstupy $D(X)$ a $D(G(Z))$
5. Vypočítajú sa hodnoty G_{loss} a D_{loss}
6. Na základe hodnôt chybových funkcií G_{loss} a D_{loss} , backpropagation algoritmus vypočíta parciálne derivácie týchto funkcií podľa parametrov generátora respektívne diskriminátora
7. Optimalizačný algoritmus na úpravu váh upraví váhy generátora a diskriminátora podľa hodnôt parciálnych derivácií vypočítaných v kroku 6

Ako bolo spomínané pri všeobecnom tréningovom cykle neurónových sietí, tréning aj v tomto prípade končí pri dokončení určeného množstva epoch tréningu, alebo pri dosiahnutej hodnote zvolenej metriky na validačnej (testovacej) sade datasetu. V niektorých prípadoch je tréningový cyklus GANov upravený tak, aby váhy generátora boli upravované niekoľko násobne viac ako váhy diskriminátora, teda sa kladie väčší dôraz na trénovanie generátora ako diskriminátora. V práci využívame prístup rovnakej priority generátora aj diskriminátora, teda parametre oboch sietí sú upravované rovnako veľa krát.

3.4.4 Deep Convolution GAN (DCGAN)

Hlboké konvolučné generatívne konfrontačné siete (skrt. DCGAN) [RM16] sú priamym využitím myšlienky GAN, kde architektúra generátora je zložená s vrstiev transponovanej konvolúcie a architektúra diskriminátora sa skladá z konvolučných vrstiev. Práve architektúra DCGAN tvorí základnú architektúru modelov generátora a diskriminátora, ktoré v práci využívame na realizáciu upsamplingu obrázkov. Architektúru DCGAN takisto okrem operácií konvolúcie a transponovanej konvolúcie využíva aj operáciu normalizovania dávok (*angl. a ďalej* batch normalization) a operáciu max-pooling (viď. 3.2.3), ktorú v architektúre takisto využívame.

Batch normalization

Tréningové dáta bývajú pred realizovaním samotného tréningového procesu väčšinou upravované aby mali určité vlastnosti. Proces tejto úpravy vstupných dát sa nazýva

preprocessing. Jedným z mnohých nástrojov preprocessingu vstupných dát býva normalizácia. Normalizácia je proces zmeny hodnôt vstupných dát tak, aby sa všetky nachádzali v rovnakom intervale alebo pochádzali z rovnakého rozdelenia, typicky normálneho.

Ked'že každou iteráciou tréningu sa výstupy(aktivácie) jednotlivých neurónov v sieti menia dôsledkom zmeny váh, a aktivácie neurónov v ďalšej vrstve sú viazané na výstup predchádzajúcej vrstvy, znamená to, že každá vrstva je vystavená neustálej zmene rozdelenia z ktorého pochádzajú jej vstupy. Tento fakt znamená, že každá vrstva sa v každej iterácii tréningu musí okrem iného, prispôsobovať aj zmene tohto rozdelenia. Tento jav je však nežiadúci, pretože jediným spôsobom, akým sa siet' prispôsobuje je práve zmena váh, ktorá je v tomto prípade ovplyvňovaná aj zmenou tohto vstupného rozdelenia, čo v konečnom dôsledku komplikuje a spomaľuje tréningový proces. Matematicky teda hľadáme spôsob ako zamedziť príliš veľkej zmene kovariancie výstupov jednotlivých vrstiev medzi jednotlivými iteráciami tréningu.

Najpoužívanejšia metóda akou toto realizovať je normalizácia dávok (*angl. batch normalization*). Ked'že v tréningovom procese typicky býva v jednej iterácií jeden batch tréningových dát, tak aj výstupy všetkých neurónov produkujú batche o rovnakom počte aktivácií. Snahou je teda hodnoty aktivácií batchov neurónov upraviť tak, aby pochádzali z rovnakého intervalu, teda pre každý batch aktivácií z rovnakej vrstvy vypočítame strednú hodnotu μ_B

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i \quad (3.29)$$

, odchýlku σ_B

$$\sigma_B = \sqrt{\frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2} \quad (3.30)$$

a následne normalizujeme podľa vzťahu

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (3.31)$$

Nové hodnoty aktivácií y_i získame škálovaním a posunutím hodnôt \hat{x}_i podľa vzťahu

$$y_i = \gamma \hat{x}_i + \beta \quad (3.32)$$

Parametre γ a β sú pre každú vrstvu, kde využívame normalizáciu dávok trénovateľnými parametrami, teda siet' sa v procese tréningu naučí ich vhodne hodnoty pre danú vrstvu.

Konštanta ϵ je vo výpočte zakomponovaná len kvôli numerickej stabilité a jej hodnota býva ľubovoľné veľmi malé číslo.

4 Implementácia

V tejto časti práci predstavíme použité nástroje, vybraný programovací jazyk a takisto predstavíme nástroj vytvorený na tvorbu datasetov, ktoré využijeme v experimentálnej časti práce.

4.1 Python

Python je v súčasnosti jeden z najviac populárnych programovacích jazykov. Najväčším dôvodom je práve súčasný trend strojového učenia a dátovej vedy, ktoré v posledných rokoch zaznamenali obrovský pokrok. Python ponúka veľké množstvo frameworkov a knižníc určených práve na tieto oblasti. Python takisto ponúka veľké množstvo populárnych nástrojov na tvorbu desktopových ale aj webových aplikácií, nástrojov na vytváranie skriptov určených na automatizáciu a mnoho ďalšieho. Jeho relatívne jednoduchá syntax v kombinácií s týmito dostupnými nástrojmi sú dôvodom jeho neustále rastúcej popularite a dôvodom jeho voľby v práci.

4.2 TensorFlow

TensorFlow je open-source knižnica od developerov divízie zameranej na umelú inteligenciu, Google Brain. TensorFlow ponúka rozsiahly framework a nástroje na riešenie problémov strojového učenia a teda aj jeho časti v ktorej sa práci zameriavame, teda neurónových sietí. TensorFlow ponúka podrobň dokumentáciu a keďže patrí do prvej trojice najpoužívanejších frameworkov určených na strojové učenia, existuje mnoho článkov, videí a blogov, ktoré vysvetľujú jeho princípy. Knižnica TensorFlow je napísaná v jazyku C++, ale existujú obaľovacie nástroje na jeho použitie aj v iných jazykoch, napríklad v spomínanom Pythone, ktoré volajú konkrétné metódy napísané v jazyku C++.

TensorFlow funguje na princípe „lenivej“ evaluácie (*angl. lazy evaluation*), čo znamená, že TensorFlow najskôr vytvorí výpočtový graf zadaného matematického výrazu, a až v prípade nutnosti hodnoty v takomto vrchole grafu sa realizuje samotný výpočet. Tento prístup je ľahko paraleлизovaný, pretože stačí v takomto grafe nájsť cesty, ktorých výpočty nesúvisia (teda cesty v grafe nie sú previazané) čo znamená, že výpočty na týchto cestách môžu byť paraleлизované. Práve paraleлизácia je nevyhnutnou podmienkou pre realizovanie

strojového učenia a obzvlášť neurónových sietí, pretože parametrov v sieti a tak aj potrebných výpočtov je veľmi veľa.

Tvorba takéhoto výpočtového grafu v TensorFlowe je pomerne ťažká programátorská úloha, a preto vznikajú rôzne abstrakcie od tohto prístupu pri ktorých sa programátor nemusí zaoberať implementačnými detailmi tvorby a evaluácie takéhoto grafu. Jednou a najpoužívanejšou knižnicou na abstrakciu tohto prístupu je Keras. V práci využívame kombináciu klasického TensorFlow a Keras.

4.3 Tenzor

Základným stavebným kameňom väčšiny knižníc určených na strojové učenie, vrátane prezentovaného TensorFlow, je tenzor.

Tenzor je generalizovanie myšlienky vektora a matice do vyšších dimenzií. Teda pokial' vektor a matica reprezentujú jedno, respektíve dvoj rozmernú štruktúru, tenzor vo všeobecnosti reprezentuje n-rozmernú štruktúru. Tenzor je typicky implementovaný ako n-rozmerné pole homogénnych premenných, teda premenných rovnakého dátového typu.

Takéto n-rozmerné pole teda môžeme označiť ako $[d_0, d_1, \dots, d_n]$, kde n predstavuje počet dimenzií tenzora, a hodnoty d_0, d_1, \dots, d_n predstavujú počet prvkov v dimenzií $0, 1 \dots n$. Napríklad maticu rozmerov 3x3, môžeme označiť ako tenzor [3,3] alebo ako tenzor [3,3,1], čo znamená, že ku konkrétnej hodnote takejto matice sa dostaneme dvoma, respektívne troma indexami. Obrázok s troma farebnými kanálmi o veľkosti 255x255 je potom reprezentovaný tenzorom [255,255,3]. Dávka (batch) takýchto obrázkov veľkosti 16 je potom tenzor $T = [16, 255, 255, 3]$. Hodnotu p -tého pixela r -tého riadku a c -tého stĺpca k -teho obrázku v takomto tenzore potom získame ako $v = T[k][r][c][p]$

4.4 Použité datasety

Na validáciu a samotné trénovanie neurónových sietí je potrebný dataset. Dataset je množina dát rozdelená na tréningovú a testovaciu (validačnú) množinu. Tréningová množina býva typicky niekoľkonásobne väčšia ako validačná. Pri neurónových sieťach platí, že čím rozmernejšie sú vstupné dátá, tým je potrebná väčšia sieť, teda viac potrebnej grafickej pamäte a tréning takejto siete potom býva rádovo dlhší. Preto sa v domácich podmienkach väčšinou používajú datasety obsahujúce relatívne malé, ale nie jednoduché

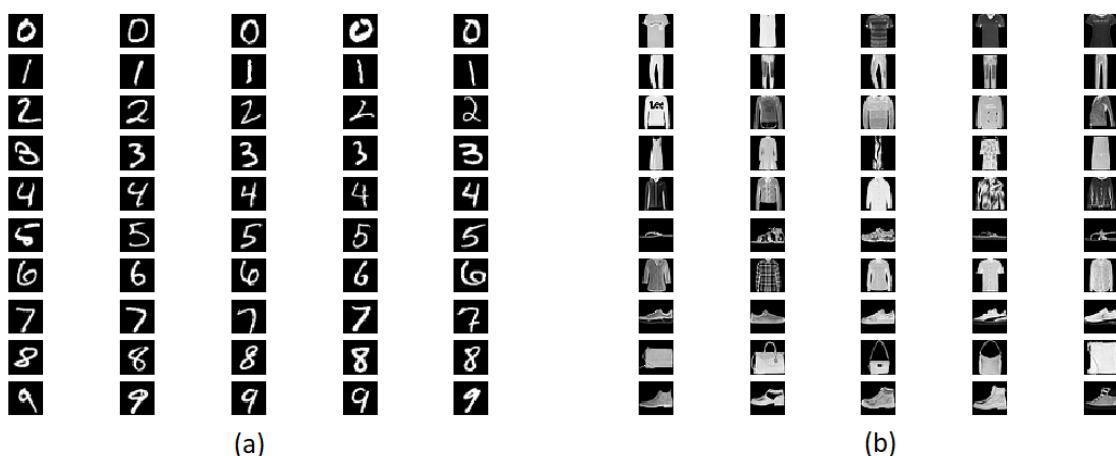
dáta. Jednými z najpoužívanejšími voľne dostupnými datasetami, ktoré týmto vlastnostiam vyhovujú sú datasety MNIST a Fashion MNIST, čo je dôvodom ich použitia v práci.

4.4.1 Dataset MNIST, Fahion MNIST

Dataset MNIST je relatívne starý, ale stále používaný dataset, na ktorom sa vykonávajú rôzne experimenty typicky spojené s neurónovými sieťami. MNIST obsahuje 70 000 čiernobielych obrázkov o veľkosti 28x28, na ktorých sa nachádzajú rukou písané čísllice 0 až 9. Každý obrázok je pritom anotovaný, teda je k nemu priradená trieda do ktorej patrí. Každý obrázok teda obsahuje aj informáciu aká číslica sa na obrázku nachádza.

Dataset MNIST je typicky rozdelený do tréningovej množiny veľkosti 60 000 obrázkov a testovacej množiny veľkosti 10 000. Tréningová množina datasetu MNIST je teda tenzor [60000,28,28,1] a testovacia množina tenzor [10000,28,28,1].

Dataset Fashion MNIST (*ďalej FMNIST*) má rovnaké rozmery ako dataset MNIST čo znamená, že pri jeho použití nie je potrebná úprava architektúry siete navrhnutej na dataset MNIST. Dataset FMNIST sa považuje za komplikovanejší ako MNIST, pretože na jednotlivých obrázkoch tohto datasetu sa nachádzajú čiernobiele obrázky oblečenia kategórií 0 až 9.



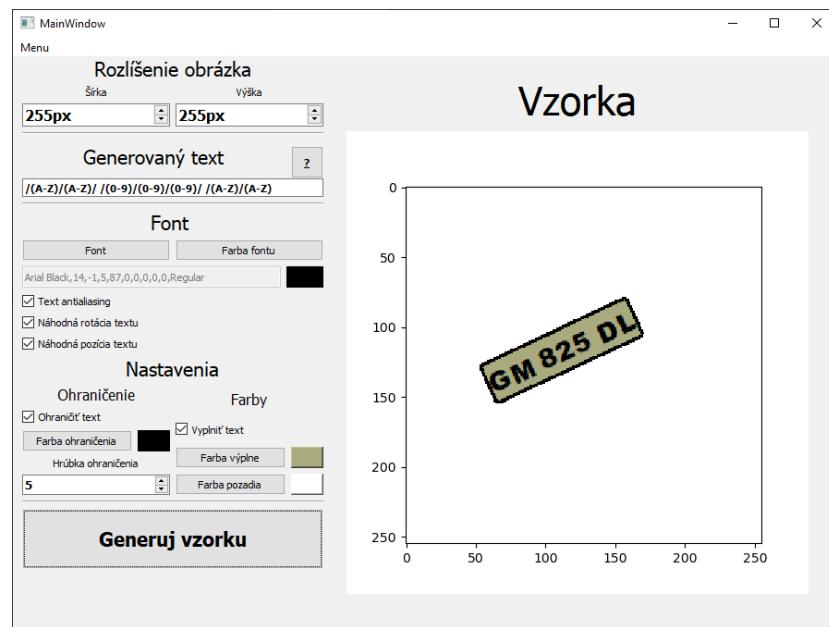
Obrázok 22 Príklad obrázkov z datasetov (a) MNIST (b) FMNIST

4.4.2 Aplikácia na tvorbu datasetov obrázkov obsahujúcich text

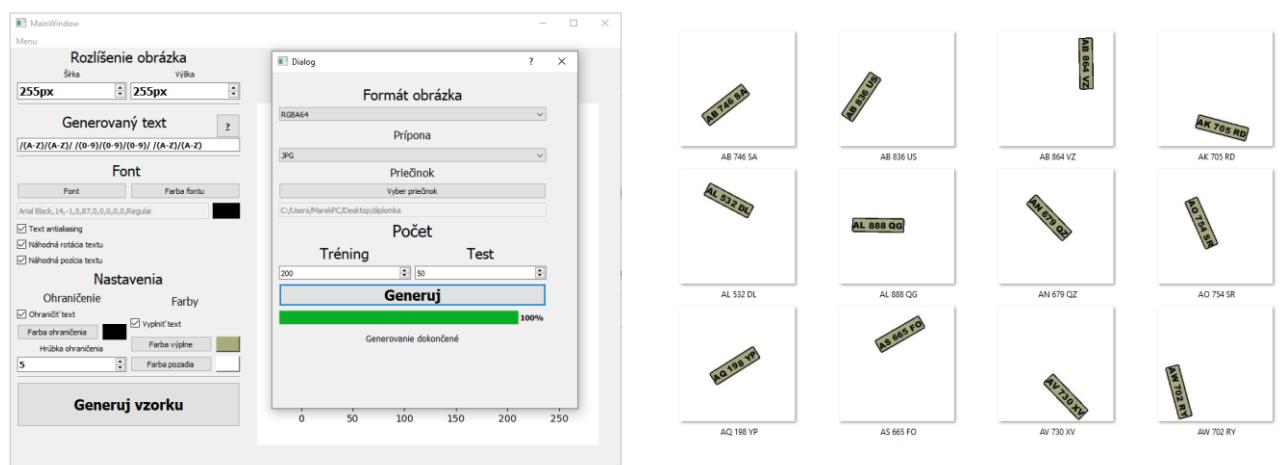
Ako súčasť práce taktiež vznikla aplikácia na tvorbu datasetov obrázkov, ktoré obsahujú text.

Užívateľské rozhranie aplikácie je takisto napísané v jazyku Python s využitím frameworku Qt, ktorý je v prípade Pythonu implementovaný v knižnici PyQt. Na samotné vykreslovanie a generovanie obrázkov boli využité známe knižnice OpenCV a Matplotlib.

Aplikácia podporuje širokú škálu voliteľných fontov, rôzne nastavenie farieb textu, pozadia, ohraničenia textu, nastavenie náhodnej rotácie textu v obrázku, náhodnej pozície, a jednoduchým princípom vysvetleným v aplikácií sa určí formát generovaného textu. Aplikácia tak vytvorí anotovaný dataset podľa zadaných parametrov, anotácie, teda v tomto prípade text, ktorý sa na konkrétnom obrázku nachádza je názov súboru tohto obrázka.



Obrázok 23 Voľba parametrov generovania v aplikácii



Obrázok 24 Rozhranie výberu formátov a príklad vygenerovaných dát

4.4.3 Dataset *Text_Set*

Ako bolo uvedené v úvode práce, v experimentálnej časti sa venujeme aj zväčšovaniu obrázkov na ktorých je text. Konkrétnie budeme zväčšovať text, ktorý má formát EČV. Ked'že neexistuje žiadny vhodný voľne dostupný anotovaný dataset EČV, využijeme prezentovanú aplikáciu na tvorbu datasetu, ktorý bude obsahovať obrázky s textom tohto formátu. Konkrétnie to sú čiernobiele obrázky veľkosti 112x112, kde v strede obrázku sa nachádza náhodne generovaný text vo formáte EČV písaný fontom Arial Black Regular veľkosti 12 s aktívnym anti aliasingom. Text je písaný čierou farbou na bielom pozadí. Tento dataset má veľkosť 10 000 tréningových obrázkov a 5000 testovacích a budeme ho v označovať názvom *Text Set*.



Obrázok 25 Príklad obrázkov datasetu *Text Set*

4.5 Architektúra a implementácia použitých modelov

Ako bolo prezentované, na realizáciu zväčšovania rozlíšenia pomocou neurónových sietí využívame generatívne modely neurónových sietí. V práci sa zaoberáme konkrétnym generatívnym modelom GAN, prezentovaným vyššie. Takisto implementujeme jednu architektúru klasifikátora, ktorý bude slúžiť ako forma metriky na hodnotenie úspešnosti zväčšovania rozlíšenia. Presný postup predstavíme v experimentálnej časti. Význam a hodnoty parametrov jednotlivých vrstiev siete sme prezentovali čitateľovi v predchádzajúcich kapitolách. Architektúry sú prezentované vo forme tabuľky, postupne od vstupnej vrstvy do výstupnej vrstvy.

4.5.1 Architektúra a implementácia klasifikátorov MNIST a FMNIST

Ked'že datasety MNIST a FMNIST majú rovnaké parametre, teda vstupný tenzor je pri oboch datasetoch rovnaký, budeme používať identickú architektúru klasifikátorov pre oba datasety.

Klasifikátor $K - F/MNIST$

Architektúra $K - FMNIST$ predstavuje architektúru na klasifikáciu datasetu MNIST a FMNIST. Siet' teda kategorizuje jednotlivé obrázky na jej vstupe do 10 kategórií. Ak budeme v experimentálnej časti hovoriť o klasifikátore $K - MNIST$, máme na mysli architektúru $K - F/MNIST$ natrénovanú na datasete MNIST. Klasifikátor $K - FMNIST$ je takisto označenie tejto architektúry, ale natrénovanej na datasete FMNIST.

Architektúra $K - F/MNIST$ je teda

Tabuľka 1 Architektúra $K - F/MNIST$

Vrstva	Parametre vrstvy							
Vstupná vrstva	-							
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu	Padding	Aktivačná funkcia			
	128	3x3	1	Same	ReLU			
Batch Normalization	-							
MaxPooling	Veľkosť filtra							
	2x2							
Dropout	Podiel aktívnych neurónov							
	0.3							
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu	Padding	Aktivačná funkcia			
	64	3x3	1	Same	ReLU			
Batch Normalization	-							
MaxPooling	Veľkosť filtra							
	2x2							
Dropout	Podiel aktívnych neurónov							
	0.3							
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu	Padding	Aktivačná funkcia			
	32	3x3	1	Same	ReLU			
Batch Normalization	-							
MaxPooling	Veľkosť filtra							
	2x2							
Dropout	Podiel aktívnych neurónov							
	0.3							
Plne prepojená	Počet neurónov		Aktivačná funkcia					
	64		ReLU					
Dropout	Podiel aktívnych neurónov							
	0.2							
Plne prepojená	Počet neurónov		Aktivačná funkcia					
	32		ReLU					
Dropout	Podiel aktívnych neurónov							
	0.1							
Plne prepojená (Výstupná vrstva)	Počet neurónov		Aktivačná funkcia					
	10		SoftMax					

Klasifikátor $K - F/MNIST$ implementujeme ako triedu `K_FMNIST`, ktorá obsahuje a abstrahuje všetky potrebné funkcie na jeho fungovanie. Implementácia modelu vo frameworku Keras je jednoduchá, najskôr stačí definovať typ modelu neurónovej siete, ktorá v našom prípade klasická dopredná sieť, teda sieť bez cyklov. V Kerase sa dopredná sieť nazýva sekvenčná (Sequential). Následne stačí postupne pridávať jednotlivé vrstvy v smere od vstupnej po výstupnú.

```
# Vráti model neurónovej siete
def get_model(self):
    if self.model_made:
        return self.model
    model = tf.keras.Sequential()

    model.add(
        keras.layers.Conv2D(filters=128, kernel_size=3, padding='same', activation='relu',
input_shape=self.input_size))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.MaxPooling2D(pool_size=2))
    model.add(keras.layers.Dropout(0.3))

    model.add(keras.layers.Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.MaxPooling2D(pool_size=2))
    model.add(keras.layers.Dropout(0.3))

    model.add(keras.layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.MaxPooling2D(pool_size=2))
    model.add(keras.layers.Dropout(0.3))

    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.2))
    model.add(keras.layers.Dense(32, activation='relu'))
    model.add(keras.layers.Dropout(0.1))
    model.add(keras.layers.Dense(10, activation='softmax'))
    self.model_made = True
    self.model = model
    return self.model
```

Z takto vytvoreného modelu je potom jednoduché získať výstupný vektor

```
# Vráti výstup neurónovej siete
# input - vstup do neurónovej siete(modelu)
# training = True - ak výstup počítame vo fáze tréningu siete
def output(self,input,training=True):
    model = self.get_model()
    return model(inputs, training=training)
```

Ako bolo prezentované, pri klasifikácií ako chybovú funkciu používame krížovú entropiu.

Kedže možných kategórií v datasetoch MNIST a FMNIST je 10, musíme použiť kategorickú krížovú entropiu. Výpočet hodnoty tejto chybovej funkcie potom implementujeme ako

```
# Vráti hodnotu chybovej funkcie, v tomto prípade Kategorická krížová entropia
# real_labels - one hot vektor, skutočnej kategórie vstupu, napríklad pre číslicu 5
[0,0,0,0,0,1,0,0,0,0]
# my_labels - pravdepodobnosťny vektor, ktorý je výstupom tejto siete pre konkrétny vstup, teda
tentto vektor bol získaný metódou output()
def loss(self,real_labels, my_labels):
    cross_entropy = tf.keras.losses.CategoricalCrossentropy()
    loss = cross_entropy(real_labels,my_labels)
    return loss
```

Následne vo fáze tréningu takéhoto modelu, na základe hodnôt TensorFlow objektu **GradientTape** a hodnoty chybovej funkcie vypočítame konkrétnie hodnoty gradientov na základe ktorých dokáže optimalizačný algoritmus ADAM, implementovaný ako

```
# Vytvorí ADAM optimalizátor na úpravu váh modelu na základe gradientov
self.optimizer = tf.keras.optimizers.Adam(0.001) # 0.001 - parameter learning rate
```

upravovať váhy (trénovateľné premenné) neurónovej siete tak, aby bola chybová funkcia minimalizovaná. Implementované ako

```
# Použije gradienty nachádzajúce sa v GradientTape tape na úpravu váh modelu podľa konkrétneho
# optimalizačného algoritmu, v tomto prípade ADAM
# tape - GradientTape, ktorý obsahuje pohľad na gradienty tohto modelu
# loss - hodnota chybovej funkcie, ktorá sa použije na výpočet hodnôt konkrétnych gradientov
def calc_apply_gradients(self,tape,loss):
    grads = tape.gradient(loss,self.get_model().trainable_variables)
    self.optimizer.apply_gradients(zip(grads,self.get_model().trainable_variables))
```

GradientTape sa vytvára v tréningovom cykle klasifikátora implementovanom ako trieda **Classifier_Trainer**.

Výstup poslednej konvolučnej vrsty architektúry $K - F/MNIST$ využijeme v experimentálnej časti práce na definovanie inej chybovej funkcie generátora, ktorú následne budeme porovnávať s pôvodnou chybovou funkciami, prezentovanou v teoretickej časti práce. Vo frameworku Keras sa pre získanie výstupu akejkoľvek skrytej vrstvy siete musí definovať pohľad na pôvodný model siete. Tvorbu tohto pohľadu implementujeme ako

```
# Vytvorí iný pohľad na celkový model, ktorého výstup bude výstup z layer_num vrstvy
# layer_num - index vrstvy modelu
def make_hidden_layer_model(self,layer_num):
    if self.layer_output_model_made:
        return self.layer_output_model

    model = self.get_model()
    output_layer = model.layers[layer_num]
    output_model = tf.keras.Model(inputs=model.input,outputs=output_layer.output)
    self.layer_output_model = output_model
```

Konkrétny výstup tohto pohľadu na model, získame metódou

```
# Vráti výstup (aktivácie) pohľadu na model vytvoreného v metóde make_hidden_layer_model
# input - vstupný vektor do neurónovej siete.
def hidden_layer_output(self,input):
    return self.layer_output_model(input)
```

4.5.2 Architektúry a implementácia GANu

Implementácia GANu je založená na rovnakom princípe ako implementácia klasifikátora. Každá architektúra je implementovaná v triede s rovnakým názvom ako názov architektúry.

Architektúra *DISCRIM*

Architektúra *DISCRIM* predstavuje architektúru diskriminátora. Ako bolo prezentované, diskriminátor robí rozhodutie či vstupné dátá boli generované generátorom, vtedy očakávame od diskriminátora hodnotu 0 a naopak či vstupné dátá pochádzali z tréningovej množiny, v tomto prípade očakávame výstup 1. Preto diskriminátor musí mať na výstupe len jednu hodnotu pre každý vstup z dávky (batchu), čomu zodpovedá výstupný tenzor $[BS, 1]$. Vstupným tenzorom je v tomto prípade tenzor obsahujúci batch obrázkov veľkosti $H * W$ s jedným farebným kanálom, čomu zodpovedá tenzor tvaru $[BS, W, H, 1]$.

Tabuľka 2 Architektúra *DISCRIM*

Vrstva	Parametre vrstvy						
Vstupná vrstva	-						
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia		
	256	2x2	2	same	Leaky Relu		
Batch Normalization	-						
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia		
	128	3x3	1	Same	Leaky Relu		
Batch Normalization	-						
Dropout	Podiel aktívnych neurónov						
	0.3						
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia		
	64	3x3	1	Same	Leaky Relu		
Batch Normalization	-						
Dropout	Podiel aktívnych neurónov						
	0.3						
Plne prepojená	Počet neurónov			Aktivačná funkcia			
	128			ReLU			
Dropout	Podiel aktívnych neurónov						
	0.3						
Plne prepojená	Počet neurónov			Aktivačná funkcia			
	64			ReLU			
Dropout	Podiel aktívnych neurónov						
	0.2						
Plne prepojená (výstupná vrstva)	Počet neurónov			Aktivačná funkcia			
	1			-			

Implementácia je realizovaná rovnakým spôsobom ako pri klasifikátore teda existuje samostatná trieda, ktorá obsahuje všetky potrebné metódy na fungovanie diskriminátora

s názvom `DISCRIM`. Takisto obsahuje rovnakú funkciu `calc_apply_gradients`, a funkciu `get_output`, ktorá vráti výstup pri konkrétnom vstupe. Jediný rozdiel je len v definovaní architektúry a chybovej funkcie. V prípade diskriminátora je chybová funkcia implementovaná pomocou binárnej krízovej entropie, ktorú sme prezentovali v teoretickej časti práce.

```
# Vráti hodnotu chybovej funkcie diskriminátora
# real_output - výstup diskriminátora, ak na vstupe boli dátá z tréningovej množiny = D(X)
# fake_output - výstup diskriminátora, ak na vstupe boli dátá generované generátorom = D(G(z))
def loss(self,real_output, fake_output):
    cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

Konkrétnе použitie týchto metód ukážeme pri implementácii tréningového cyklu GANov.

Architektúra *GENERATOR_K2*

Kedžže generátor je model, ktorý v našom prípade bude realizovať zväčšovanie rozlíšenia obrázku na jeho vstupe, nemôže byť vstupom do generátora náhodný vektor z ako sme doteraz prezentovali. V našom prípade bude vstupom do generátora obrázok, ktorý model generátora bude zväčšovať. Kedžže v práci experimentujeme s dvoma rôznymi zväčšovacími faktormi, konkrétnie $k = 2$ a $k = 4$, teda realizujeme dvojnásobné a štvornásobné zväčšovanie rozlíšenia, znamená to, že potrebujeme dve rôzne architektúry generátora.

Architektúra *GENERATOR_K2* je architektúra generátora, ktorý bude realizovať dvojnásobné zväčšenie rozlíšenie, pri datasete MNIST a FMNIST to bude zväčšenie $14 \times 14 \rightarrow GENERATOR_K2 \rightarrow 28 \times 28$ a pri datasete *Text_Set* to bude zväčšenie $56 \times 56 \rightarrow GENERATOR_K2 \rightarrow 112 \times 112$. Ako bolo prezentované v predchádzajúcich častiach, samotné zväčšenie realizujeme pomocou operácie transponovanej konvolúcie, ktorá je typicky implementovaná ako samostatná vrstva s voliteľnými parametrami, ktorých význam sme takisto predstavili. Za každou vrstvou transponovej konvolúcie nasleduje typicky normalizácia dávok. Z takto zväčšeného tenzora potom operáciami konvolúcie v kombinácií s normalizáciou dávok postupne zmenšujeme veľkosť štvrtej dimenzie tenzora, teda počet feature máp, ktoré vznikli transponovanou konvolúciou.

Aktivačnou funkciou poslednej vrstvy je spomínaný hyperbolický tangens, teda výstupné hodnoty (hodnoty pixelov výstupných obrázkov) sú v rozsahu $< -1; 1 >$, čo je zároveň aj rozsah normalizovaných dát, ktoré do siete vstupujú. Kedžže v experimentálnej

časti sa zaobráme len čiernobielymi obrázkami, teda obrázkami s jedným farebným kanálom, veľkosť štvrtej dimenzie výstupného tenzora musí byť jedna, teda výstupná konvolučná vrstva má len jeden kernel. Týmto prístupom získame na výstupe batch zväčšených obrázkov, teda tenzor veľkosti $[BS, W * 2, H * 2, 1]$, kde W je pôvodná šírka a H je pôvodná výška obrázkov vo vstupnom batchi veľkosti BS .

Tabuľka 3 Architektúra *Generator_K2*

Vrstva	Parametre vrstvy				
Vstupná vrstva	-				
Transponovaná konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	256	2x2	2	Valid	Leaky Relu
Batch Normalization	-				
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	128	2x2	1	Same	Leaky Relu
Batch Normalization	-				
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	64	2x2	1	Same	Leaky Relu
Batch Normalization	-				
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	32	2x2	1	Same	Leaky Relu
Batch Normalization	-				
Konvolučná vrstva (Výstupná vrstva)	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	1	2x2	1	Same	tanh

Všetky modely generátorov sú takisto implementované v samostatných triedach. Pre architektúru *GENERATOR_K2* je to trieda *GENERATOR_K2*, ktorá obsahuje všetky potrebné metódy pre fungovanie generátora. Metódy sú totožné ako v prípade predchádzajúcich modelov. Jediný rozdiel je v definícii chybovej funkcie, ktorú sme prezentovali v kapitole venovanej GANom. Chybová funkcia generátora, je takisto ako v prípade diskriminátora implementovaná pomocou binárnej krízovej entropie

```
# Vráti hodnotu chybovej funkcie generátora
# fake_output - výstup diskriminátora ak na jeho vstupe bol výstup z generátora, hodnota D(G(z))
def loss(self, fake_output):
    cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

Architektúra ***GENERATOR_K4***

GENERATOR_K4 je architektúra generátora, ktorý realizuje štvornásobné zväčšenie rozlíšenie vstupných obrázkov veľkosti $W * H$, teda vykonáva transformáciu $(W) * (H) \rightarrow GENERATOR_K4 \rightarrow (4 * W) * (4 * H)$ čo znamená potrebu dvoch vrstiev trasponovej konvolúcie.

Architektúra *GENERATOR_K4* sa od architektúry *GENERATOR_K2* lísi pridaním jednej vrstvy transponovanej konvolúcie, po ktorej nasleduje normalizácia dávok a vrstva klasickej konvolúcií

Tabuľka 4 Architektúra *GENERATOR_K4*

Vrstva	Parametre vrstvy				
Vstupná vrstva	-				
Transponovaná konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	256	2x2	2	Valid	Leaky Relu
Batch Normalization	-				
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	128	4x4	1	Same	Leaky Relu
Batch Normalization	-				
Transponovaná konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	256	5x5	2	Same	Leaky Relu
Batch Normalization	-				
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	64	2x2	1	Same	Leaky Relu
Batch Normalization	-				
Konvolučná vrstva	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	32	2x2	1	Same	Leaky Relu
Batch Normalization	-				
Konvolučná vrstva (Výstupná vrstva)	Počet kernelov	Veľkosť kernelov	Posun kernelu (stride)	Padding	Aktivačná funkcia
	1	2x2	1	Same	tanh

Implementácia tréningového cykla GANov

Trieda *GAN_Best_Classified_Trainer* trénuje architektúru GANu metódou najlepšej klasifikácií na klasifikátore, ktorý dostane ako parameter. Princíp tréningu najlepšej klasifikácie predstavíme v experimentálnej časti práce.

Trieda obsahuje rovnaké metódy ako trieda pre trénovanie klasifikátorov. Rozdiel je len v potrebe viacerých datasetov, keďže v tomto prípade je potrebná prítomnosť dvoch verzií každého datasetu, jedna pôvodná a druhá, ktorá obsahuje zmenšené obrázky. Podstatný rozdiel je v metóde `_train_step`, v ktorej prebieha tréning na konkrétnych batchoch zmenšených a pôvodných dát z tréningovej množiny

```
# Realizuje tréningový krok trénovania GANov
# real_images - batch pochádzajúci z pôvodnej tréningovej množiny
# gen_input - batch vstupných dát generátora, teda v prípade zväčšovania rozlíšenia batch zmenšenej
# tréningovej množiny
def _train_step(self,real_images,gen_input):
    with tf.GradientTape() as generator_tape , tf.GradientTape() as discrim_tape:
        # Výstup generátora. Hodnota G(z)
        gen_imgs = self.generator.output(gen_input,is_training=True)

        # Výstup diskrimátora ak na vstupe bol batch z pôvodnej tréningovej množiny
        # Hodnota D(x)
        real_output = self.discriminator.output(real_images,is_training=True)

        # Výstup diskrimátora ak na vstupe bol batch generovaný generátorom
        # Hodnota D(G(z))
        fake_output = self.discriminator.output(gen_imgs,is_training=True)

        # Hodnota chybovej funkcie generátora
        gen_loss = self.generator.loss(fake_output)

        # Hodnota chybovej funkcie diskriminátora
        discrim_loss = self.discriminator.loss(real_output,fake_output)

    # Úprava váh generátora a diskriminátora
    self.generator.calc_apply_gradients(generator_tape,gen_loss)
    self.discriminator.calc_apply_gradients(discrim_tape,discrim_loss)
    return gen_loss,discrim_loss
```

Trieda `GAN_Best_Classified_Trainer` takisto obsahuje aj metódy na sledovanie priebehu tréningu, kde po každom epochu tréningu, generátor zväčší zmenšený testovací dataset na pôvodné rozlíšenie, ktorý sa následne porovnáva s pôvodným, nezmenšeným datasetom. Toto porovnávanie realizujeme na základe hodnôt rôznych metrík, ktoré takisto predstavíme v experimentálnej časti.

Okrem triedy `GAN_Best_Classified_Trainer` sme implementovali aj triedu `GAN_Best_Classified_Trainer_Perceptual_loss`, ktorá má totožnú implementáciu. Rozdiel je však v úprave konečnej chybovej funkcie generátora práve v metóde `_train_step`. Chybovú funkciu perceptual loss predstavíme taktiež v experimentálnej časti práce.

4.6 Implementácia optického rozpoznávania znakov (OCR)

Ako bolo spomínané v úvode práce, niektoré experimenty zväčšovania rozlíšenia budeme hodnotiť na základe výsledku algoritmu, ktorý rozpoznáva v obrázku text. Takémuto algoritmu hovoríme optické rozpoznávanie znakov (*angl. Optical Character Recognition skr. a ďalej OCR*). Algoritmus OCR rôznymi matematickými úpravami rozpoznáva vo vstupnom obrázku text. Výstupom algoritmu je v optimálnom prípade textový reťazec, ktorý sa na vstupnom obrázku nachádza.

Implementácia takéhoto algoritmu je zložitá úloha, ktorá presahuje zámer tejto práce a tak v práci využijeme jednu z mnohých existujúcich algoritmov OCR s názvom Tesseract, ktorý je pre jazyk python implementovaný v knižnici pytesseract.

V práci sme na komunikáciu s pytesseractom implementovali triedu **Tesseract**. Trieda **Tesseract** takisto obsahuje aj metódy preprocessingu (predspracovania) vstupného obrázku tak, aby pytesseract fungoval najlepšie možne vzhľadom na formát vstupného obrázka. Presné použité metódy preprocessingu predstavíme v experimentálnej časti.

5 Experimenty so zväčšovaním rozlíšenia pomocou GAN

V prvej časti experimentov prezentujeme dosiahnuté výsledky trénovania architektúry klasifikátora, ktoré v nasledujúcej časti budú slúžiť ako jedna z konkrétnych porovnávacích metrík pre zväčšené obrázky GANom. Takisto predstavíme spôsob tréningu GANov, predstavíme a porovnáme inú chybovú funkciu generátora a čitateľovi takisto predstavíme spôsob a použité metriky hodnotenia výsledkov zväčšovania rozlíšenia.

5.1 Tréning klasifikátorov

Ako bolo prezentované, v práci používame jednu architektúru klasifikátora, ktorú označujeme ako $K - F/MNIST$. Túto architektúru trénujeme jedenkrát na datasete MNIST a druhýkrát na datasete Fashion MNIST (FMNIST). Ak teda hovoríme o natrénovanom $K - FMNIST$, máme na mysli architektúru $K - F/MNIST$ natrénovanú na datasete FMNIST. Konkrétnie architektúry sietí sú prezentované v predchádzajúcej kapitole.

Klasifikátor trénujeme spôsobom skorého zastavenia, teda po každej epoche tréningu, tréovaný klasifikátor klasifikuje celý testovací dataset, ktorý má v prípade datasetov MNIST a FMNIST veľkosť 10 000 obrázkov. Výsledkom tejto klasifikácie je podiel $presnosť = \frac{\text{počet správne klasifikovaných obrázkov}}{\text{počet všetkých obrázkov}}$ (angl. accuracy). Hodnotu accuracy vypočítame po každej epoche tréningu a výsledkom trénovania bude siet s takými váhami, ktoré mala po skončení epochy, po ktorej dosiahla najväčšiu hodnotu accuracy.

Pri tréningu klasifikátorov sme zaznamenávali aj hodnotu chybovej funkcie, ktorá sa v každej iterácii tréningu počíta na základe konkrétneho batchu dát z tréningového datasetu, výsledok klasifikácie tohto batchu chybová funkcia porovná so skutočnými kategóriami jednotlivých obrázkov v batchi. Chybová funkcia v prípade klasifikátora $K - F/MNIST$ je kategorická krízová entropia, prezentovaná v kapitole o klasifikátoroch. Klasifikátor sme trénovali v oboch prípadoch osem epoch.

Počas tréningu sme zaznamenávali aj hodnoty chybovej funkcie a takisto výsledok klasifikácie testovacieho datasetu po každom epoche, priebehy týchto hodnôt sú k dispozícii v Príloha A.

Architektúra $K - FMNIST$, trénovaná na datasete MNIST dosiahla najväčšiu presnosť po vykonaní 6 epoch a to 0.9921.

Klasifikátor $K - FMNIST$, teda architektúra $K - F/MNIST$, trénovaná a validovaná na datasete FMNIST, dosiahol presnosť 0.906. Dôvodom menšej hodnoty je práve väčšia zložitosť datasetu FMNIST oproti datasetu MNIST.

Pre prehľadnosť uvádzame dosiahnuté presnosti aj v tabuľke

Tabuľka 5 Dosiahnuté presnosti klasifikátorov

Názov klasifikátora	Dosiahnutá presnosť (accuracy) v %
$K - MNIST$	99.21%
$K - FMNIST$	90.06%

Natrénované klasifikátory použijeme pri tréningu GANov, ktoré realizujú zväčšovanie rozlíšenia.

5.1.1 Použité metriky na porovnávanie obrázkov

Aby sme dokázali hodnotiť obrázky, ktoré vzniknú GANom alebo inými metódami na zväčšenie rozlíšenia, musíme definovať funkciu, ktorej výstup nám povie ako podobné sú si dva obrázky. Ako je zrejmé, nemôže existovať jediná funkcia, ktorá by túto podobnosť vyjadrovala, pretože sa nedá jasne definovať čo myslíme pod pojmom podobnosť obrázkov. Napríklad v prípade porovnávania obrázkov dvoch psov, môžeme povedať, že obrázky sú podobné, pretože sa na nich nachádza pes, ale môžeme aj povedať, že nie sú si podobné a to z dôvodu, že jeden pes sa nachádza na obrázku na jednom mieste, ale na druhom obrázku je tento pes na inom mieste.

Jednou z najtriviálnejších možností ako vyjadriť podobnosť dvoch obrázkov môže byť formou vzdialenosnej metriky, ktorá vypočíta vzdialenosť (rozdiel) medzi dvoma pixelmi na rovnakej pozícii. Potom výsledná hodnota takejto metriky je aritmetický priemer takýchto vzdialenosí medzi všetkými párami pixelov. V práci používame dve rôzne vzdialenosné metriky a to euklidovskú vzdialenosť a strednú kvadratickú chybu (*angl. mean square error, skr. MSE*). Keďže všetky tri tieto metriky vyjadrujú určitú formu vzdialnosti, môžeme povedať, že menšia hodnota akejkoľvek takejto metriky znamená väčšiu podobnosť medzi obrázkami a naopak.

Ďalšou z možností ako vyjadriť podobnosť medzi dvoma obrázkami je formou funkcií, ktoré vznikli pre tento účel. Jednou z takýchto funkcií je index štruktúrnej podobnosti, skrátene SSIM (*angl. Structural SIMilarity index*) [WZ04]. Index SSIM nadobúda hodnoty v intervale $< -1; 1 >$, kde väčšia hodnota sa dá interpretovať ako väčšia podobnosť. Hodnota 1 je dosiahnuteľná iba v prípade zhodných obrázkov.

Druhou funkciou, ktorú v práci používame ako metriku na porovnávanie, je PSNR (*angl. Peak signal-to-noise ratio*), vysvetlené v [HZ10], ktorá vychádza z oblasti spracovania signálov. Keďže obrazová informácia sa dá interpretovať ako signál, PSNR sa používa aj na porovnávanie dvoch obrázkov. Hodnota PSNR nemá uzavretý obor hodnôt ako SSIM, ale keďže v jeho vzorci je v menovateli stredná kvadratická chyba, väčšia hodnota PSNR znamená väčšiu podobnosť medzi obrázkami.

Pre prehľadnosť uvádzame použité metriky a interpretácie ich hodnôt do tabuľky

Tabuľka 6 Obory hodnôt a význam hodnôt jednotlivých metrík

Metrika	Obor hodnôt	Interpretácia hodnôt
Euklidovská vzdialenosť	R	Malá hodnota znamená veľkú podobnosť
Stredná kvadratická chyba (MSE)	R	Malá hodnota znamená veľkú podobnosť
SSIM	$< -1; 1 >$	Veľká hodnota znamená veľkú podobnosť
PSNR	R	Veľká hodnota znamená veľkú podobnosť

5.1.2 Tréning GANov

Ako bolo spomínané, pri trénovaní klasifikátora dokážeme jednoznačne určiť, kedy je model natrénovaný najlepšie.

V prípade trénoania GANov neexistuje jednoznačná metrika, ktorá nám povie v ktorej iterácii tréningu je model najlepší. Jednou z možností môže byť dosiahnutá minimálna alebo maximálna hodnota definovaných chybových funkcií pre diskriminátor, respektívne generátor. Keďže hodnoty týchto chybových funkcií v procese tréningu neustále podliehajú zmenám, kde zmena jednej hodnoty má priamy vplyv na druhú, čo vyplýva z princípu konfrontačného tréningu, čiže nevypovedajú jednoznačne o kvalite generovaných obrázkoch, nie je ani tento prístup k hodnoteniu tréningu najvhodnejší.

Ďalším spôsobom môže byť hodnota nejakej zo spomínaných porovnávacích metrík, kde po každom epochu tréningu porovnáme skutočné (originálne) dátá s tými

generovanými. Tento spôsob má však taktiež nejasnosti, pretože neexistuje jednoznačná metrika, ktorá dokáže dostatočne vyjadriť rozdiel medzi dvoma obrázkami danej domény, v jednom prípade môže byť vhodnou vzdialenosnou metrikou euklidovská a v druhom prípade napríklad stredná kvadratická chyba a v iných prípadoch napríklad index SSIM.

V práci preto využijeme metriku založenú na úspešnosti klasifikácie na vopred natrénovanom klasifikátore. Spôsob trénovania jednej konkrétnej architektúry GAN v práci je teda nasledovný

1. *Tréning klasifikátora na danej tréningovej množine* – natrénujeme potrebnú architektúru klasifikátora podľa datasetu, ktorý budeme zväčšovať.
2. *Zmenšenie tréningového a testovacieho datasetu* – tréningový aj testovací dataset na ktorom sme trénovali klasifikátor v kroku 1, zmenšíme na rozlíšenie, ktoré budeme následne zväčšovať GANom. Teda napríklad v prípade, ak chceme GAN natrénovať na zväčšovanie rozlíšenia pre rukou písané čísllice, zmenšíme tréningový aj testovací dataset MNIST na rozlíšenie z ktorého chceme realizovať zväčšovanie.
3. *Tréning GANu na zmenšenom datasete* – podľa princípov prezentovaných v teoretickej časti trénujeme diskriminátor aj generátor. Jednotlivé vstupy generátora pri tréningu budú pochádzať práve zo zmenšeného tréningového datasetu. Po každej epoche takéhoto tréningu, generátor zväčší zmenšený testovací dataset na pôvodné rozlíšenie, ktorý následne klasifikátor natrénovaný v kroku 1 klasifikuje. Výsledok takejto klasifikácie je potom naša forma metriky po každom epochu tréningu. Teda po každom epochu tréningu generátor zväčší celý testovací dataset, ktorý sa klasifikuje na vopred natrénovanom klasifikátore. Výsledkom tréningu potom bude taký generátor a diskriminátor, ktorý dosiahol najlepší výsledok tejto klasifikácie spomedzi všetkých epoch tréningu.

V experimentoch budeme predpokladať, že tento princíp trénovania založeného na presnosti klasifikácie zväčšených obrázkov nám umožňuje spomedzi epoch tréningu vybrať lepšie riešenie (natrénovaný generátor) ako pri použití nejakej inej metriky alebo hodnôt chybových funkcií. Tento predpoklad má svoje opodstatnenie pretože klasifikátor

natrénovaný kroku 1 sa naučí skryté závislosti nezmenšeného (originálneho) tréningového datasetu, na základe ktorých vie s určitou presnosťou určiť, čo sa na obrázku nachádza, respektíve do akej kategórie patrí. Môžeme teda povedať, že natrénovaný klasifikátor vie „ako by mali vyzerat“ obrázky, aby ich dokázal správne klasifikovať. Ak teda takto natrénovaný klasifikátor správne určí, čo sa na obrázku zväčšenom generátorom nachádza (správne klasifikuje), môžeme o takomto obrázku povedať, že bol zväčšený správne, respektíve, že pri zväčšení nedošlo k takej strate pôvodnej informácie, ktorá by spôsobila, že obrázok nedokáže klasifikátor správne klasifikovať.

Týmto princípom teda získame natrénovaný GAN, ktorý bude realizovať zväčšovanie rozlíšenia z rozlíšenia, na ktoré sme v kroku 2 zmenšili dataset na rozlíšenie pôvodného datasetu.

Okrem meniacich sa hodnôt klasifikácie a hodnôt chybových funkcií pre generátor a diskriminátor, pri tréningu GANov zaznamenávame aj hodnoty metrík (predstavené v 5.1.1), ktoré hodnotia podobnosť zväčšených obrázkov s pôvodnými obrázkami po každom epochu, čo nám umožňuje sledovať priebeh vývoja týchto hodnôt a hodnotiť tak samotné výsledky zväčšovania po jednotlivých epochách tréningu.

5.2 Percepčná chyba generátora

Ked'že chybová funkcia generátora, ktorú sme prezentovali v teoretickej časti práce, vo svojej definícii neobsahuje žiadnu zložku, ktorá by mala súvis s konkrétnou doménou tréningového datasetu, môže nastáť situácia, že pri tréningu sa siet' nedokáže dostatočne dobre a včas naučiť approximovať určité črty, ktoré vystihujú danú tréningovú doménu, čo v konečnom dôsledku môže znamenať horšie výsledky a väčší počet potrebných epoch tréningu. Jedným zo spôsobov zamedzenia tohto efektu je práve zmena chybovej funkcie generátora tak, aby v sebe zahŕňala nejakú reprezentáciu črt konkrétnej domény. Nami prezentovaná zmena chybovej funkcie vychádza zo svojho času „state of the art“ článku, ktorý sa venuje zväčšovaniu rozlíšenia [LT+17]. Autori v ňom predstavili novú chybovú funkciu generátora, ktorá sa nazýva percepčná chyba (perceptual loss), vďaka ktorej dosiahli v tom čase najlepšie výsledky.

Princíp percepčnej chyby spočíva v zakomponovaní jednej zložky do pôvodnej chybovej funkcie generátora. Touto pridanou zložkou sú aktivácie vybranej vrstvy vopred natrénovaného klasifikátora, ktorý ako bolo spomínané, je naučený ako majú vyzerat' jeho

vstupy, aby ich dokázal správne klasifikovať, čo znamená, že ak na vstupe klasifikátora bude obrázok, ktorý klasifikátor vie správne klasifikovať, tak aj aktivácie všetkých jeho vrstiev (teda jednotlivé výstupy skrytých vrstiev) budú podstatne odlišné od tých, ktoré by boli prítomné, ak by na vstupe bol obrázok, ktorý klasifikátor nedokáže správne klasifikovať. Preto je touto zložkou percepčnej chyby práve rozdiel medzi aktiváciami vybranej skrytej vrstvy ak bol na vstupe obrázok zväčšený generátorom a pôvodný obrázok (nezmenšený). Potom percepčnú chybu definujeme ako

$$Perc_{loss} = G_{loss} + \alpha * MSE(A_{class}(G(X_{LR})), A_{class}(X_{HR})) \quad (5.0)$$

, kde G_{loss} predstavuje pôvodnú chybovú funkciu generátora a $A_{class}(G(X_{LR}))$ predstavuje aktivácie (výstup) zvolenej skrytej vrstvy klasifikátora $class$, ak na vstupe bol $G(X_{LR})$, teda obrázok X_{LR} zväčšený generátorom G . $A_{class}(X_{HR})$ naopak predstavuje aktivácie zvolenej skrytej vrstvy A_{class} , ak na vstupe bol pôvodný nezmenšený obrázok X_{HR} . Funkcia MSE je stredná kvadratická chyba. Konštanta α slúži len na stlmenie vplyvu tejto zložky, pretože hlavnou úlohou generátora je stále generovať také obrázky, ktoré diskriminátor označí za pravé, čo je vyjadrené v pôvodnej chybovej funkcií G_{loss} . V [LT+17] autori použili $\alpha = 10^{-3}$, ale keďže v ich prípade bolo použitých viac skrytých vrstiev klasifikátora ako v našom prípade, použijeme $\alpha = 10^{-2}$. V experimentoch sa teda zameriame aj na porovnanie týchto dvoch chybových funkcií generátora.

5.3 Dvojnásobné zväčšenie rozlíšenia

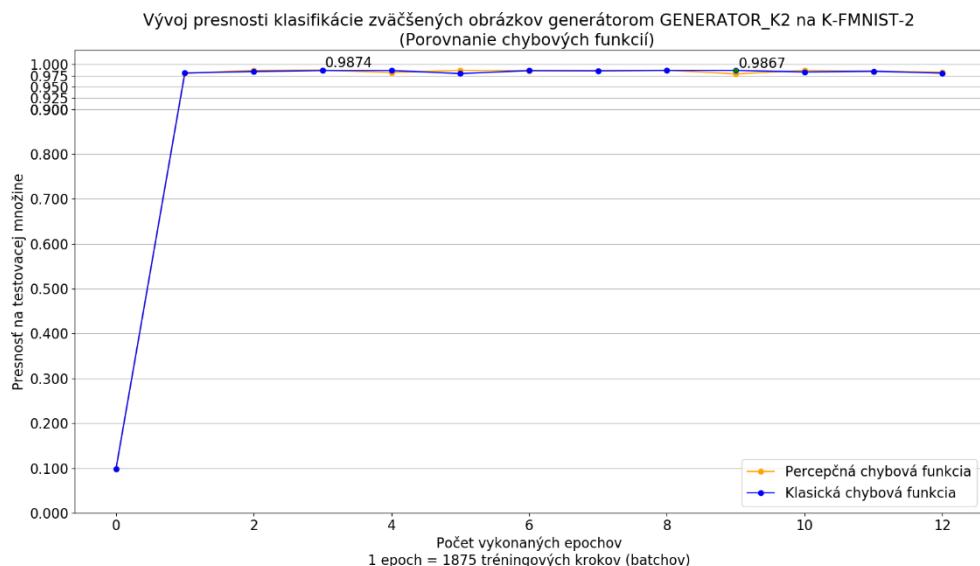
V tejto časti čitateľovi prezentujeme priebeh tréningu a dosiahnuté výsledky architektúry generátora *GENERATOR_K2* pri oboch prezentovaných chybových funkciách na datasetoch MNIST a Fashion MNIST a takisto výsledky na dataseite *Text_Set*.

Ked'že architektúra diskriminátora ostáva počas všetkých experimentov rovnaká, budeme rozdeľovať experimenty podľa použitej architektúry generátora.

5.3.1 *GENERATOR_K2* s klasifikátorom *K – MNIST*, trénovaný na oboch chybových funkciách

Predkladáme priebeh tréningu GANu s architektúrou generátora *GENERATOR_K2* a diskriminátora *DISCRIM* založeného na princípe najlepšej klasifikácie (vid' 5.1.2) na klasifikátore *K – MNIST* pri oboch prezentovaných chybových funkcií generátora, teda pri klasickej (3.4.2) a aj percepčnej chybovej funkcií (5.2). Pre jednoduchšie porovnanie prezentujeme priebehy oboch tréningov v jednom grafe.

Počas tréningu sú zaznamenané všetky hodnoty prezentovaných metrík, ako aj priebeh hodnôt chybových funkcií v jednotlivých iteráciách tréningu. Výsledkom tohto trénovania sú teda modely s takými váhami, ktoré dosiahli najväčšiu presnosť zväčšeného testovacieho datasetu na klasifikátore spomedzi všetkých epoch tréningu. Počet epoch pre všetky prezentované tréningy je nastavený na 12.



Obrázok 26 Vývoj úspešnosti klasifikácie zväčšeného testovacieho datasetu generátorom *GENERATOR_K2* na klasifikátore *K – MNIST* pri oboch chybových funkciách generátora

Z grafu vidíme, že v prípade tréningu s percepčnou chybovou funkciou, až 98.74% zväčšených obrázkov bolo klasifikovaných správne už po tretej epoce tréningu GANu. Keďže klasifikátor *K – MNIST* mal presnosť na pôvodnom nezmenšenom testovacom datasete 99.21%, môžeme povedať, že *GENERATOR_K2* trénovaný s percepčnou chybovou funkciou nedokázal „správne“ rekonštruovať $99.21 - 98.74 = 0.47\%$ obrázkov. Túto hodnotu budeme označovať ako rekonštrukčnú chybu generátora. Takisto vidíme, že

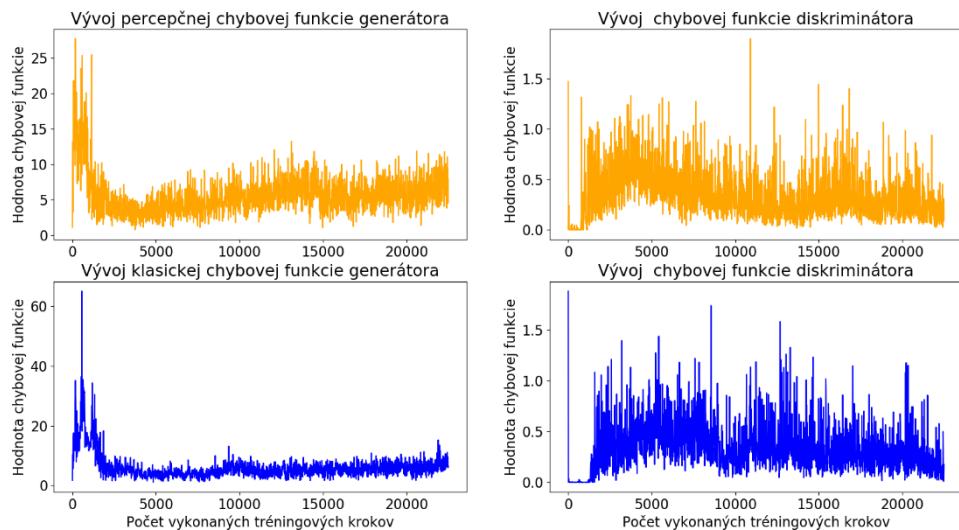
pred akýmkoľvek tréningom, teda po epochu 0, bola úspešnosť klasifikácie len cca 10%, čo znamená, že generátor zväčšoval obrázky tak, že klasifikátor rozhodoval náhodne, keďže existuje 10 možných kategórií obrázkov. V prípade tréningu s klasickou chybovou funkciou generátora bola dosiahnutá presnosť 98.67%, čomu zodpovedá rekonštrukčná chyba 0.54%.

Pre prehľadnosť budeme udávať tieto hodnoty do tabuľky

Tabuľka 7 Rekonštrukčné chyby GENERATOR_K2 pri oboch chybových funkciách

GENERATOR_K2 trénovaný na	Výsledok klasifikácie	Presnosť K-MNIST	Rekonštrukčná chyba
Klasická chybová funkcia	0,9867	0,9921	0,0054
Percepčná chybová funkcia	0,9874		0,0047

Na ďalších grafoch prezentujeme vývoj hodnôt chybových funkcií pre generátor a diskriminátor, ktoré sú počítané na základe konkrétneho batchu dát z tréningovej množiny. Veľkosť batchu je v prípade tréningových dát 32.

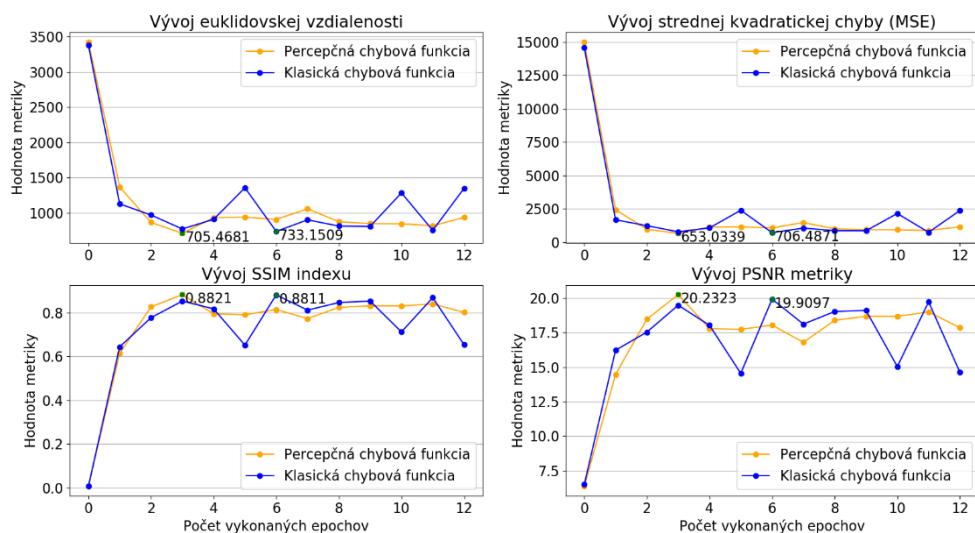


Obrázok 27 Vývoj chybových funkcií pre generátor a diskriminátor pri tréningu GENERATOR_K2 na datasete MNIST

Kedže GAN sa snaží minimalizovať chybové funkcie, na grafe vidíme ich klesajúce tendencie (vid' teoretickú časť neurónových sietí a GANov). Takisto na Obrázok 27 si môžeme všimnúť, že interval, v ktorom sa pohybuje hodnota percepčnej chybovej funkcie generátora, je menší ako v prípade klasickej chybovej funkcie.

Takisto si môžeme všimnúť aj relatívne veľké zmeny hodnôt chybovej funkcie diskriminátora. Tento jav nastáva najmä dôsledkom neustálej zmeny vstupného rozdelenia dát, ktoré logicky nastáva pri tréningu, pretože jednotlivé batche dát sú z tréningového datasetu konštruované náhodne.

Predkladáme aj vývoj priemerných hodnôt spomínaných porovnávacích metrík, ktoré po každej epoche porovnávali celý zväčšený testovací dataset s pôvodným (nezmenšeným) testovacím datasetom.

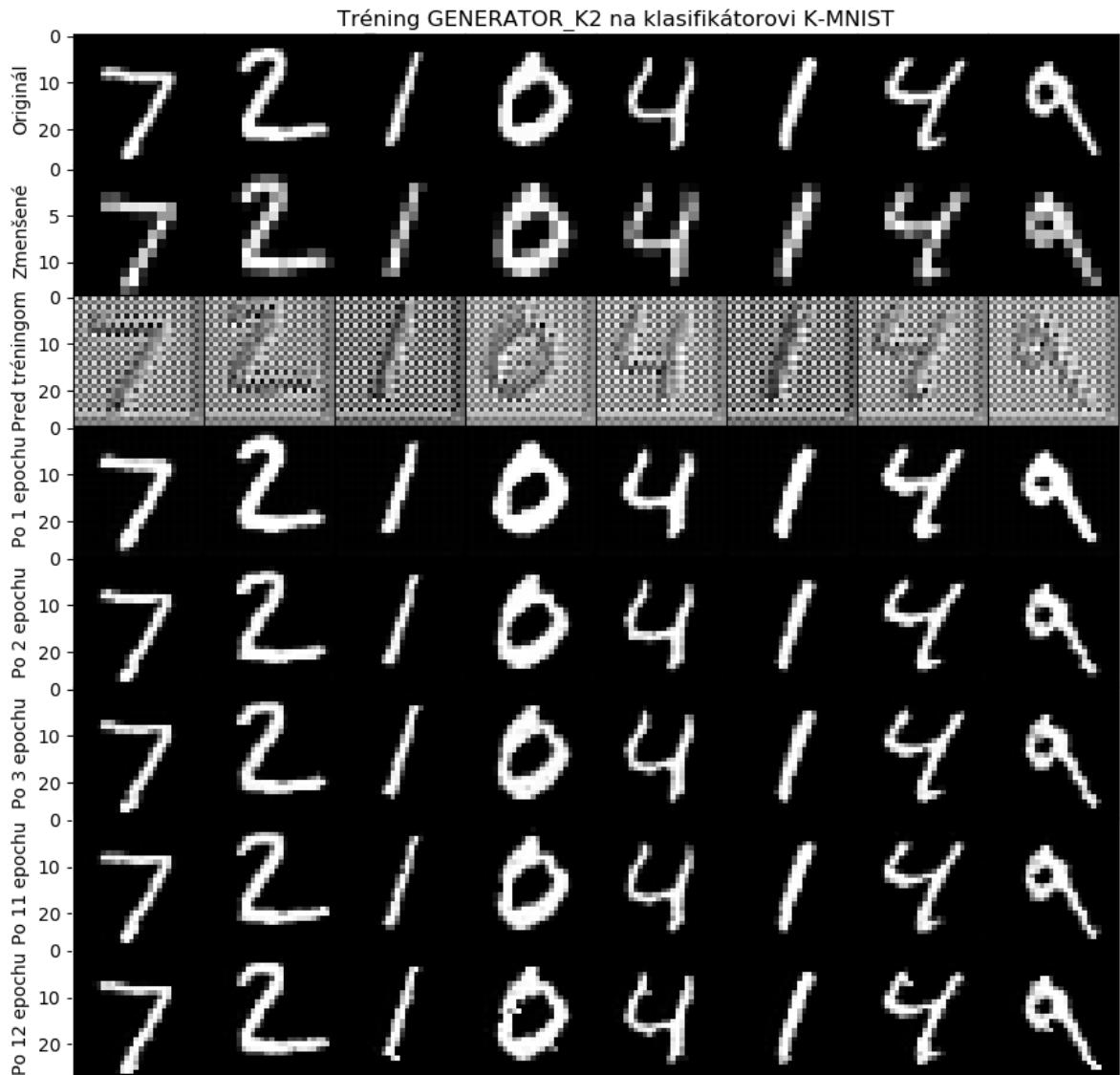


Obrázok 28 Vývoj hodnôt porovnávacích metrík pri tréningu *GENERATOR_K2* na klasifikátore *K – MNIST*

Na Obrázok 28 vidíme, že najlepšie hodnoty metrík v prípade percepčnej chybovej funkcie boli dosiahnuté po skončení epochy 3, čo zodpovedá aj najlepšiemu výsledku klasifikácie na Obrázok 26, ktorý bol v prípade percepčnej chybovej funkcie dosiahnutý taktiež po epoche 3. V prípade klasickej chybovej funkcie generátora toto pozorovanie neplatí, keďže vidíme, že najlepšie hodnoty metrík v tomto prípade boli prítomné po epoche 6, ale najlepší výsledok klasifikácie bol prítomný po epoche 9.

Takisto čitateľovi prezentujeme aj priebeh tréningu s percepčnou chybovou funkciou z pohľadu vybraných zväčšených obrázkov. Na prvom riadku sa nachádzajú obrázky z originálnej nezmenšenej testovacej množiny. V druhom riadku vidíme tieto obrázky zmenšené na rozlíšenie 14x14, ktoré následne po každej epoche zväčšíme generátorom.

Tieto zväčšené obrázky sú postupne uložené do riadkov podľa toho, po koľkej epoche tréningu boli generátorom zväčšované. Na zvislej osi sú znázornené veľkosti obrázkov v pixeloch.



Obrázok 29 Príklad zväčšených obrázkov generátorom po každom epochu tréningu

Vidíme, že dvojnásobné zväčšenie rozlíšenia funguje vzhľadom na všetky predložené hodnoty metrík a obrázkov veľmi dobre, ale porovnajme tieto výsledky ešte s inými metódami zväčšovania rozlíšenia prezentovanými v prvých kapitolách práce. Teda zväčšíme ten istý zmenšený testovací dataset týmito metódami na pôvodné rozlíšenie, a rovnako ako v prípade takéhoto zväčšenia GANom, necháme tento zväčšený dataset klasifikovať klasifikátorom $K - MNIST$. Rovnako vypočítame aj hodnoty porovnávacích metrík

Tabuľka 8 Hodnoty metrík dvojnásobného zväčšenia testovacieho datasetu MNIST

Zväčšovacia metóda	Výsledok klasifikácie K-MNIST	Euklid ovská vzdialenosť	Stredná kvadratická chyba (MSE)	SSIM index	PSNR
GENERATOR_K2 (Pecrp loss)	0,9874	705,47	653,03	0,88	20,23
GENERATOR_K2	0,9867	804,61	847,61	0,853	19,09
Metóda najbližšieho suseda	0,9747	904,59	1063,74	0,805	18,04
LANCZOS5	0,9729	656,67	563,24	0,873	20,83
LANCZOS3	0,9702	665,15	577,57	0,797	20,72
Bikubická interpolácia	0,9559	717,68	671,49	0,863	20,07
Bilineárna interpolácia	0,8567	855,87	950,79	0,775	18,50

Vidíme, že vo výsledku klasifikácie na $K - MNIST$, **GENERATOR_K2** prekonal všetky uvedené metódy v oboch prípadoch chybových funkcií, ale napríklad najmenšiu euklidovskú vzdialenosť dosiahla metóda LANCZOS5, ktorá okrem tejto metriky prekonala **GENERATOR_K2** aj vo všetkých ostatných, okrem spomínamej klasifikácie a indexu SSIM. Výsledky sa nedajú jednoznačne interpretovať a povedať tak, ktorá metóda je lepšia alebo horšia, pretože každá zo spomínaných metrík podobnosť počíta inak a preto je na čitateľovi, ktorej metrike pridelí akú váhu.

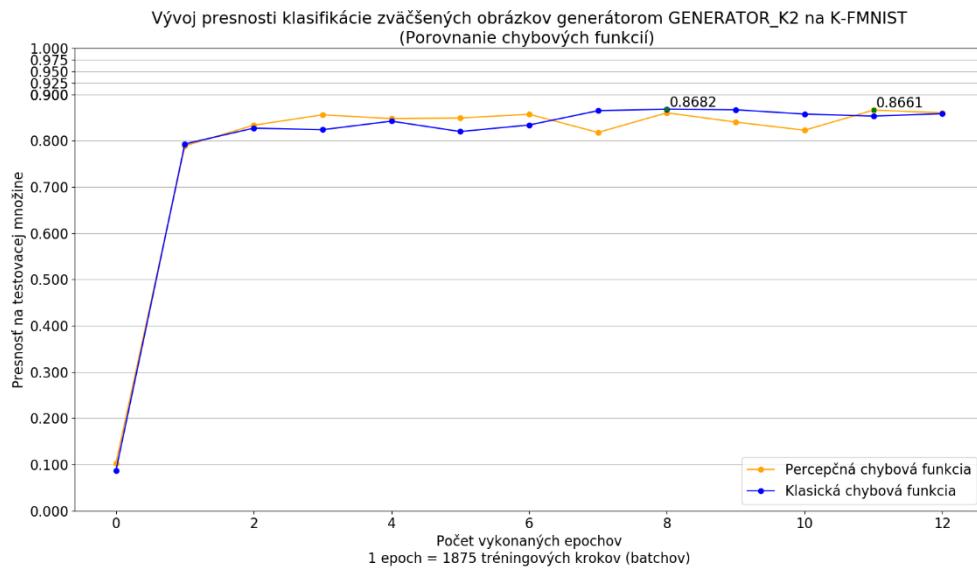
Výsledky metód sú si relatívne podobné, pretože obrázky zmenšené na rozlíšenie 14x14 obsahujú stále dostatok informácie na jeho dobrú rekonštrukciu do pôvodného rozlíšenia. Preto v ďalších častiach, v ktorých sa budeme venovať štvornásobnému zväčšeniu, predpokladáme väčšie rozdiely v hodnotách medzi jednotlivými metódami zväčšenia.

5.3.2 **GENERATOR_K2** s klasifikátorom $K - FMNIST$

V nasledujúcom experimente porovnáme klasickú chybovú funkciu generátora s percepčnou chybovou funkciou na dvojnásobnom zväčšení datasetu FMNIST. Takisto porovnáme výsledky oboch trénovaní s výsledkami zväčšovacích metód a hodnôt ich korešpondujúcich metrík.

Dodržíme rovnaký postup ako v predchádzajúcich experimentoch, teda trénujeme princípom najlepšej klasifikácie zväčšeného datasetu na klasifikátore, ktorý je tomto prípade klasifikátor $K - FMNIST$ natrénovaný na presnosť 90.06%.

Ako v predchádzajúcich experimentoch, trénujeme 12 epoch a výsledkom trénovalia je taký model, ktorý dosiahol najväčšiu presnosť klasifikácie testovacieho datasetu, ktorý generátor po každom epochu zväčší na pôvodné rozlíšenie.



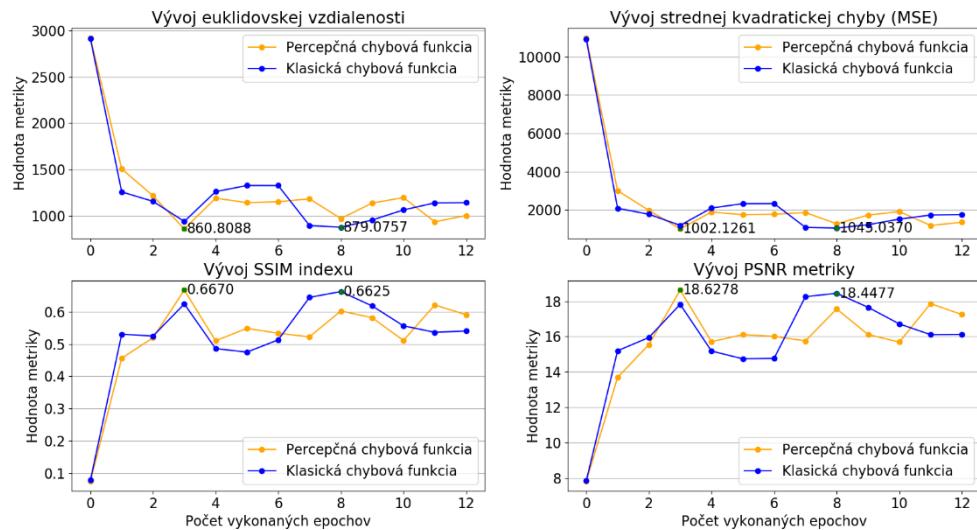
Obrázok 30 Vývoj úspešnosti klasifikácie zväčšeného testovacieho datasetu generátorom *GENERATOR_K2* pri oboch chybových funkcií, klasifikovanom na $K - FMNIST$

Z grafov vyplýva, že rekonštrukčná chyba, teda rozdiel medzi presnosťou klasifikátora a výsledkom klasifikácie zväčšeného testovacieho datasetu je

Tabuľka 9 Rekonštrukčné chyby pre obe chybové funkcie

GENERATOR_K2 trénovaný na	Výsledok klasifikácie	Presnosť K-FMNIST	Rekonštrukčná chyba
Klasická chybová funkcia	0,8682	0,9006	0,0324
Percepčná chybová funkcia	0,8661		0,0345

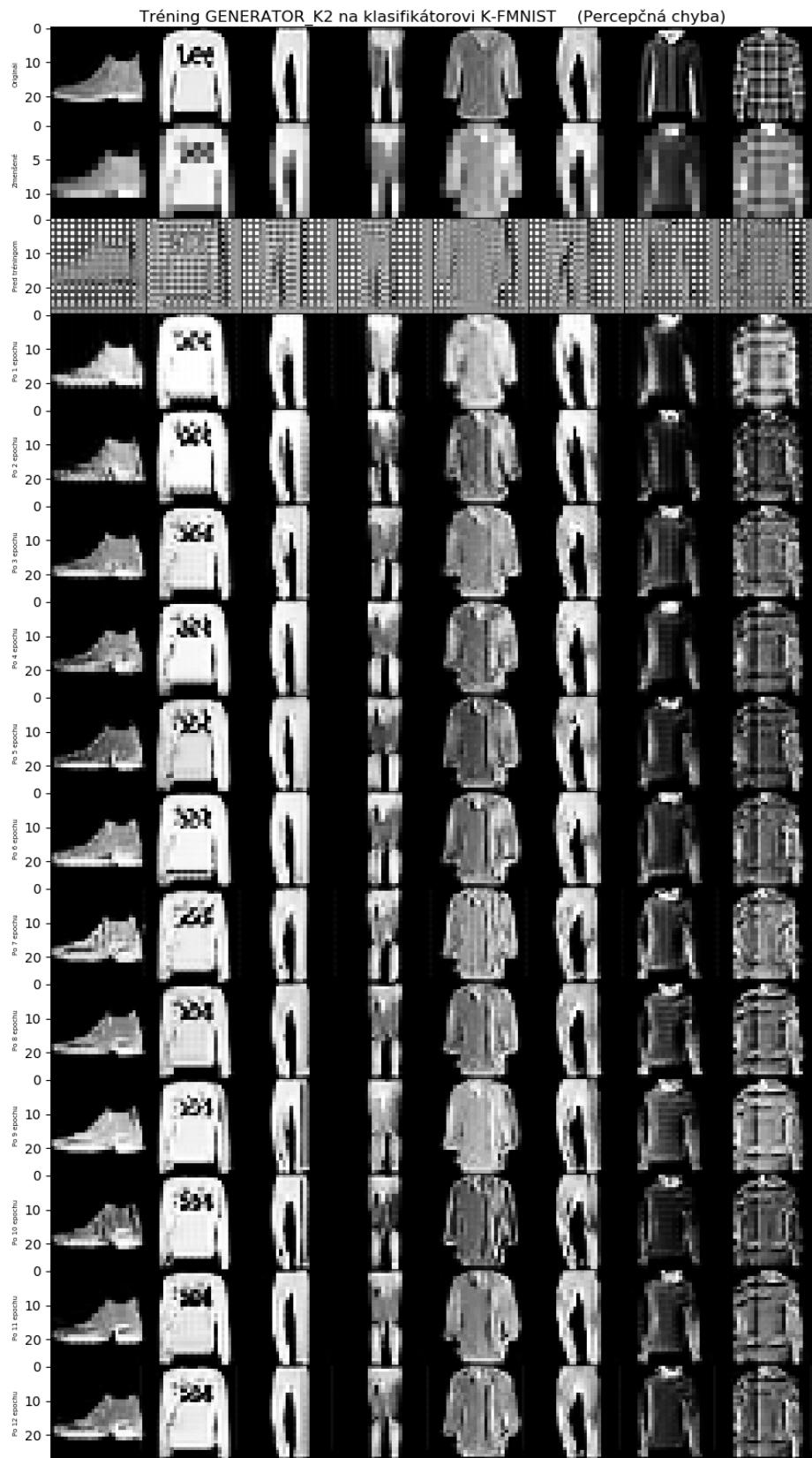
Vidíme, že rekonštrukčná chyba je v prípade dvojnásobného zväčšenia testovacieho datasetu FMNIST menšia v prípade klasickej chybovej funkcií generátora. Toto platilo aj v experimentoch dvojnásobného zväčšenia datasetu MNIST.



Obrázok 31 Vývoj metrík v tréningoch *GENERATOR_2* na $K - FMNIST$ pri oboch chybových funkciách

Z vývoja metrík počas tréningu vidíme, že v prípade klasickej chybovej funkcie generátora boli najlepšie hodnoty metrík dosiahnuté po rovnakom počte epoch ako výsledok najlepšej klasifikácie. Naopak, v prípade percepčnej chybovej funkcií generátora sa najlepšie hodnoty metrík podarilo získať už po epochu 3, čo je o 8 epoch skôr ako najlepšia dosiahnutá hodnota klasifikácie, ktorá bola prítomná po epochu 11. Teda opäť platí zistenie z predchádzajúceho experimentu, že percepčná chybová funkcia pri dvojnásobnom zväčšení testovacieho datasetu má za následok rýchlejšie dosiahnutie najlepších hodnôt prezentovaných porovnávacích metrík.

Prikladáme ešte vývoj zväčšenia vybraných obrázkov z testovacieho datasetu FMNIST počas tréningu s percepčnou chybovou funkciou generátora. Kedže rozdiely medzi jednotlivými epochami sú viac výrazné ako v prípade datasetu MNIST, prezentujeme v tomto prípade zväčšenie po každej z dvanásťich epoch tréningu



Obrázok 32 Vývoj zväčšovania náhodných obrázkov z testovacieho datasetu FMNIST, generované generátorom *GENERATOR_2* trénovanom na percepčnej chybovej funkcií

Takisto ako v predchádzajúcom experimente, porovnáme natrénované modely generátorov s ostatnými zväčšovacími metódami

Tabuľka 10 Hodnoty metrík dvojnásobného zväčšenia testovacieho datasetu FMNIST

Zväčšovacia metóda	Výsledok klasifikácie K-FMNIST	Euklidovská vzdialenosť	Stredná kvadratická chyba (MSE)	SSIM index	PSNR
GENERATOR_K2	0,8682	879,08	1045,04	0,663	18,45
GENERATOR_K2 (Percep loss)	0,8661	937,48	1182,72	0,621	17,86
Metóda najbližšieho suseda	0,8619	876,43	1020,91	0,723	18,40
LANCZOS3	0,8554	782,25	811,55	0,732	19,38
LANCZOS5	0,8542	788,54	824,57	0,718	19,31
Bikubická interpolácia	0,8436	791,46	831,23	0,732	19,28
Bilineárna interpolácia	0,7826	849,99	957,19	0,681	18,65

V tomto prípade sú hodnoty porovnávacích metrík horšie pre generátor trénovaný s percepčnou chybou, na rozdiel od zväčšovania datasetu MNIST, kde tieto hodnoty boli lepšie práve pre percepčnú chybovú funkciu.

Žiadna z ostatných zväčšovacích metód neprekonała výsledok klasifikácie generátorov. Najbližšie bola metóda najbližšieho suseda. Najlepšie hodnoty porovnávacích metrík dosiahla metóda LANCZOS3.

5.3.3 *GENERATOR_K2* trénovaný na datasete *Text_Set*

V tomto experimente predkladáme výsledok trénovalia GANu pri generátore s architektúrou *GENERATOR_K2*, trénovaného na datasete *Text_Set*. Teda generátor v tomto prípade realizuje zväčšenie $56 \times 56 \rightarrow GENERATOR_K2 \rightarrow 112 \times 112$.

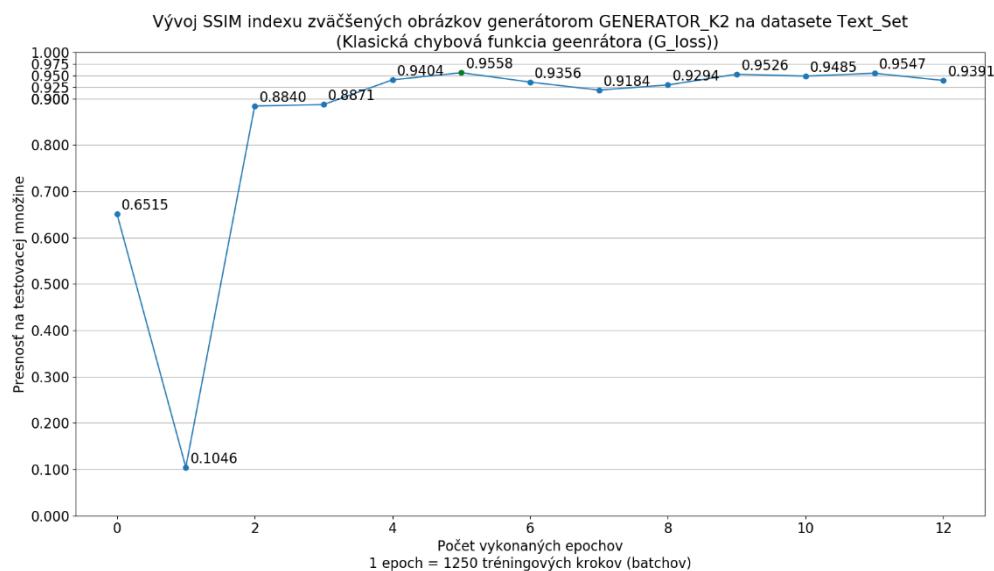
Vzhľadom na to, že kategórie obrázkov v datasete *Text_Set* sú samotné reťazce nachádzajúce sa na obrázku, nebudeme v tomto prípade používať metriku presnosti na vopred nami natrénovanom klasifikátore, ale použijeme spomínaný algoritmus OCR, konkrétnie prezentovaný Tesseract. Výsledkom takejto metriky teda bude percento úspešne rozpoznaných obrázkov, respektívne správne rozpoznaných textov nachádzajúcich sa na

týchto zväčšených obrázkoch. Keďže algoritmus Tesseract má veľkú výpočtovú zložitosť, nebudeme priebeh tréningu hodnotiť na základe tejto presnosti, ale vyberieme jednu zo spomínaných porovnávacích metrík.

Vzhľadom na všetky doteraz prezentované hodnoty metrík sa najlepšou metrikou, ktorá najviac odráža výsledky korešpondujúcich klasifikácií, zdá byť index SSIM, čo je dôvodom jeho voľby ako tréningovej metriky pre trénovanie GANov na datasete *Text_Set*.

Teda doteraz používaný tréningový princíp založený na najlepšej klasifikácii upravíme tak, že po každom epochu namiesto hodnoty presnosti zväčšeného testovacieho na vopred natrénovanom klasifikátore, budeme počítať index SSIM medzi zväčšeným testovacím datasetom a pôvodným (nezmenšeným) testovacím datasetom.

Rovnako ako v prechádzajúcich experimentoch s datasetmi MNIST a FMNIST, predkladáme vývoj tréningovej metriky, teda v tomto prípade spomínaného indexu SSIM



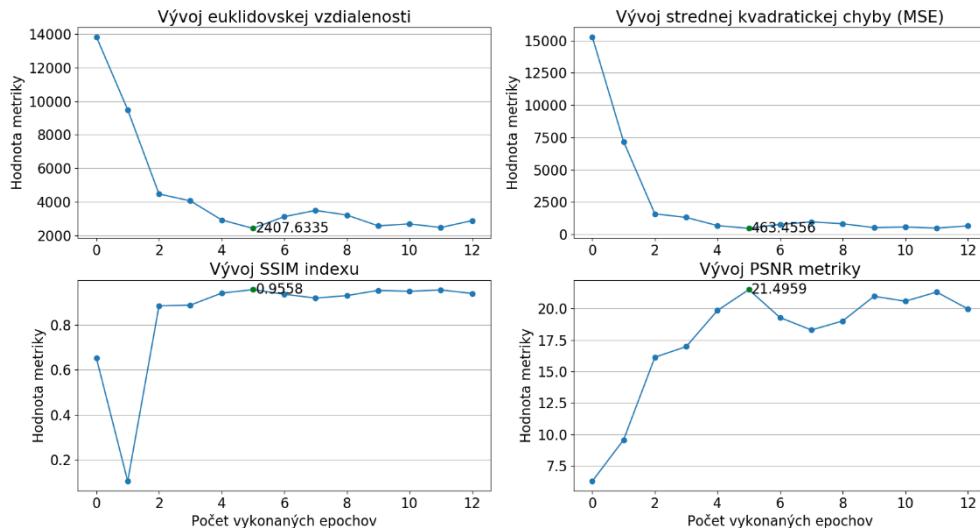
Obrázok 33 Vývoj SSIM zväčšeného testovacieho datasetu *Text_Set* generátorom *GENERATOR_K2* trénovaným s klasickou chybovou funkciou

Vidíme, že index SSIM bol počas tréningu veľmi veľký, čo ale v tomto prípade nemusí automaticky svedčiť aj o kvalite zväčšeného textu, keďže ten tvorí len malú časť obrázka, čo znamená, že podstatná časť indexu SSIM je v tomto prípade počítaná len z pozadia, ktoré je v tomto prípade celé biele, čo znamená jednoduchú approximáciu pre generátor. Takisto si môžeme všimnúť, že po skončení prvého epochu bol SSIM index výrazne nižší. Toto je dôsledok toho, že generátor začal tréning tak, že pozadie approximoval ako šedé, čo môžeme vidieť na Obrázok 34 na ktorom opäť prezentujeme vývoj zväčšovania počas tréningu

Tréning GENERATOR_K2 na datasete TextSet								
Pred tréningom	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
Zmenšené								
Originál								
Po epochu 1	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
...	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH
	QV 582 ZL	WZ 014 BC	OF 084 AN	HX 096 XT	ZW 650 KH	UJ 469 RZ	ZO 091 QE	OM 536 YH

Obrázok 34 Vývoj zväčšovania vybraných obrázkov z testovacieho datasetu *Text_Set*, generátorom *GENERATOR_K2* počas tréningu

Z vývoja si môžeme všimnúť, že jednotlivé rozdiely medzi výsledkami sú zanedbateľné a preto sa nedá z pohľadu na zväčšené obrázky jednoznačne určiť úspešnosť zväčšenia v jednotlivých epochách. Preto predkladáme ešte vývoj metrík



Obrázok 35 Vývoj metrík počas tréningu *GENERATOR_2* na datasete *Text_Set*

Vidíme, že najlepšie hodnoty ostatných metrík, takisto ako metrika SSIM, boli najlepšie po epoche 5. Výsledkom tréningu je teda model po epoche 5. Takisto vidíme, že hodnoty metrík sa počas jednotlivých epoch moc nemenia a teda lepším ukazovateľom úspešnosti bude spomínaný algoritmus OCR.

Algoritmus OCR predpokladá určité vlastnosti vstupného obrázka na jeho najlepšie možné fungovanie. Niektoré tieto predpokladané vlastnosti dokážeme docieliť pomocou preprocessingu vstupného obrázka. V prípade implementácie Tesseract je jedna z takýchto vlastností taká, že text na vstupnom obrázku, respektívne všetky pixely, ktoré tvoria text, sú maximálne čierne (hodnota 0) a pozadie, na ktorých sa tento text nachádza je maximálne biele (hodnota 255). V prípade pôvodného datasetu *Text_Set* sa hodnoty všetkých pixelov nachádzajú v intervale $<0; 255>$, kde hodnoty pixelov textu sa nachádzajú niekde v hornej polovici tohto intervalu, čo znamená, že na dosiahnutie spomínamej vlastnosti na správne fungovanie Tesseractu stačí vykonať jednoduchú transformáciu hodnôt

$$p(x, y) = \begin{cases} 255, & \text{src}(x, y) \geq 127 \\ 0, & \text{inak} \end{cases} \quad (5.1)$$

, kde $p(x, y)$ predstavuje novú hodnotu pixelu na pozícii (x, y) a $src(x, y)$ udáva pôvodnú hodnotu tohto pixela. V literatúre a dokumentáciách sa takáto funkcia nazýva *threshold*. Na všetky obrázky vstupujúce do Tesseractu, aplikujeme teda transformáciu p .

Tesseract obsahuje veľké množstvo parametrov, ktoré sa nastavia podľa informácií, ktoré o vstupnom obrázku vieme. Jednou z takýchto parametrov je takzvaný *whitelist*. *Whitelist* predstavuje zoznam znakov, ktoré vieme, že tvoria rozpoznávaný text. V prípade datasetu *Text_Set*, ktorý simuluje formát EČV, sú teda možnými znakmi číslice a veľké písmena. Nastavením tohto parametra dokážeme uľahčiť Tesseractu prácu pri rozpoznávaní, a dosiahnuť tak lepší výsledok rozpoznávania.

Tesseract teda vráti rozpoznaný textový reťazec a keďže vieme, že text na obrázku má formát EČV, teda text je vo formáte ČČ_PPP_ČČ, kde _ predstavuje medzeru, Č číslicu a P písmeno, môžeme výstupný reťazec Tesseractu v prípade, že nie je v tomto formáte, transformovať na tento formát a zlepšiť tak celkovú úspešnosť. V prípade, ak sa vo výstupnom reťazci na pozícii na ktorej vieme, že patrí číslica, nachádza písmeno alebo naopak, vykonáme jednoduchý preklad podľa predpisov

$$\begin{aligned} (0 \leftrightarrow "0") \quad (1 \leftrightarrow "I") \quad (2 \leftrightarrow "Z") \\ (4 \leftrightarrow "Z") \quad (5 \leftrightarrow "S") \quad (8 \leftrightarrow "B") \end{aligned}$$

Teda napríklad, ak sa na pozícii, kde vieme, že má byť číslica, nachádza písmeno B, nahradíme ho číslicou 8, pretože sa mu najviac vizuálne podobá. Rovnaký princíp potom aplikujeme na každý znak výsledného reťazca a získame tak finálny výstupný rozpoznaný reťazec, ktorý porovnáme so skutočným textom, ktorý poznáme vďaka tomu, že dataset *Text_Set* je anotovaný. Týmto spôsobom potom vypočítame percento správne rozpoznaných reťazcov, čo nám umožní porovnať ostatné metódy zväčšovania s GANom.

Najskôr uvádzame presnosť OCR na pôvodnom nezmenšenom testovacom datasete s rozlíšením 112x112 a zmenšenom datasete s rozlíšením 56x56

Tabuľka 11 Percento úspešne rozpoznaných textov pre pôvodný a zmenšený testovací dataset

Dataset	% správne rozpoznaných (OCR)
Pôvodný (112x112)	91%
Zmenšený (56x56)	0.08%

Na Tabuľka 11 si môžeme všimnúť, že Tesseract nerozpoznal správne takmer žiadny text na zmenšených obrázkoch, čo je spôsobené samotnou implementáciou Tesseraetu, ktorá nie je stavaná na takéto malé rozlíšenia textu.

Dosiahnuté úspešnosti rozpoznávania zväčšených obrázkov pomocou jednotlivých metód sú

Tabuľka 12 Percento úspešne rozpoznaných textov zo zväčšeného testovacieho dataset jednotlivými metódami

Zväčšovacia metóda	Presnosť OCR
LANCZOS3	69,64%
Bikubická interpolácia	67,38%
LANCZOS5	65,8%
Bilineárna interpolácia	44,68%
Generator_K2	43,56%
Metóda najbližšieho suseda	2,88%

Vidíme, že zväčšenie generátorom *GENERATOR_K2* je porovnatelné s bilineárnom interpoláciou, ale oproti ostatným metódam zlyháva o $\pm 24\%$, čo v porovnaní s výsledkom nezmenšeného datasetu znamená o 47.44% menej rozpoznaných obrázkov.

Ked'že v tomto experimente výsledky jednotlivých metód vyzerajú takmer identicky a rozdiel je na prvý pohľad viditeľný len na výsledkoch OCR, hodnoty metrík neuvádzame. Z tohto dôvodu neuvádzame ani všetky príklady zväčšenia všetkých metód, uvedieme len pári príkladov najlepšej metódy, teda LANCZOS3, v porovnaní s *GENERATOR_K2*

Príklad zväčšených obrázkov Test_Set generátorom GENERATOR_K2 a metódou LANCZOS3

LANCZOS3		generátor		Zmenšené		Originál		LANCZOS3	
IQ 330 GC	EI 592 KL	GW 362 EO	EU 126 QN	IT 212 JY	RI 394 MM	FT 940 ZX	SP 437 ID	IQ 330 GC	EI 592 KL
IQ 330 GC	EI 592 KL	GW 362 EO	EU 126 QN	IT 212 JY	RI 394 MM	FT 940 ZX	SP 437 ID	IQ 330 GC	EI 592 KL
IQ 330 GC	EI 592 KL	GW 362 EO	EU 126 QN	IT 212 JY	RI 394 MM	FT 940 ZX	SP 437 ID	IQ 330 GC	EI 592 KL
IQ 330 GC	EI 592 KL	GW 362 EO	EU 126 QN	IT 212 JY	RI 394 MM	FT 940 ZX	SP 437 ID	IQ 330 GC	EI 592 KL
YO 585 YT	EP 050 WI	ZU 054 MY	TN 816 NP	ZZ 526 LQ	IZ 959 NQ	CS 097 GB	EN 105 TR	YO 585 YT	EP 050 WI
YO 585 YT	EP 050 WI	ZU 054 MY	TN 816 NP	ZZ 526 LQ	IZ 959 NQ	CS 097 GB	EN 105 TR	YO 585 YT	EP 050 WI
YO 585 YT	EP 050 WI	ZU 054 MY	TN 816 NP	ZZ 526 LQ	IZ 959 NQ	CS 097 GB	EN 105 TR	YO 585 YT	EP 050 WI
YO 585 YT	EP 050 WI	ZU 054 MY	TN 816 NP	ZZ 526 LQ	IZ 959 NQ	CS 097 GB	EN 105 TR	YO 585 YT	EP 050 WI
VT 502 TU	LA 461 HS	EJ 135 SD	LV 939 YL	NG 622 ZC	KS 090 RQ	GY 756 CJ	OP 059 SJ	VT 502 TU	LA 461 HS
VT 502 TU	LA 461 HS	EJ 135 SD	LV 939 YL	NG 622 ZC	KS 090 RQ	GY 756 CJ	OP 059 SJ	VT 502 TU	LA 461 HS
VT 502 TU	LA 461 HS	EJ 135 SD	LV 939 YL	NG 622 ZC	KS 090 RQ	GY 756 CJ	OP 059 SJ	VT 502 TU	LA 461 HS
VT 502 TU	LA 461 HS	EJ 135 SD	LV 939 YL	NG 622 ZC	KS 090 RQ	GY 756 CJ	OP 059 SJ	VT 502 TU	LA 461 HS

Obrázok 36 Porovnanie zväčšenia pomocou GENERATOR_K2 a interpolácie LANCZOS3 na datasete *Text_Set*

Z príkladov môžeme vidieť, že v niektorých prípadoch zväčšenia generátorom nie je text úplný a výrazný, respektíve v niektorých zväčšených znakoch môžeme vidieť biele miesta, čo môže spôsobovať pozorovaný horší výsledok rozpoznávania oproti iným zväčšovacím metódam. Príčinou tejto pozorovanej neúplnosti zväčšených znakov môže byť to, že tréningový dataset *Text_Set* obsahuje len 10 000 obrázkov a tak sa môže stať, že testovací dataset veľkosti 5 000, na ktorej realizujeme validáciu môže obsahovať pomerne veľa obrázkov, na ktorých sa znaky nachádzajú na pozíciah, ktoré neboli dostatočne obsiahnuté v tréningovom datase, čo spôsobilo, že generátor nemal dostatočnú príležitosť sa naučiť ich vhodné zväčšenie. Inou príčinou môže byť samotná dĺžka tréningového procesu, ktorá bola v tomto experimente 12 epoch, čo môže byť v prípade zložitosti datasetu *Text_Set* málo.

Preto v nasledujúcom experimente rozšírime dataset *Text_Set* tak, aby obsahoval viac obrázkov a takisto predĺžime tréningový proces.

5.3.4 GENERATOR_K2 trénovaný na rozšírenom datase *Text_Set*

V tomto experimente je snahou napraviť spomínané možné nedostatky experimentu 5.3.3.

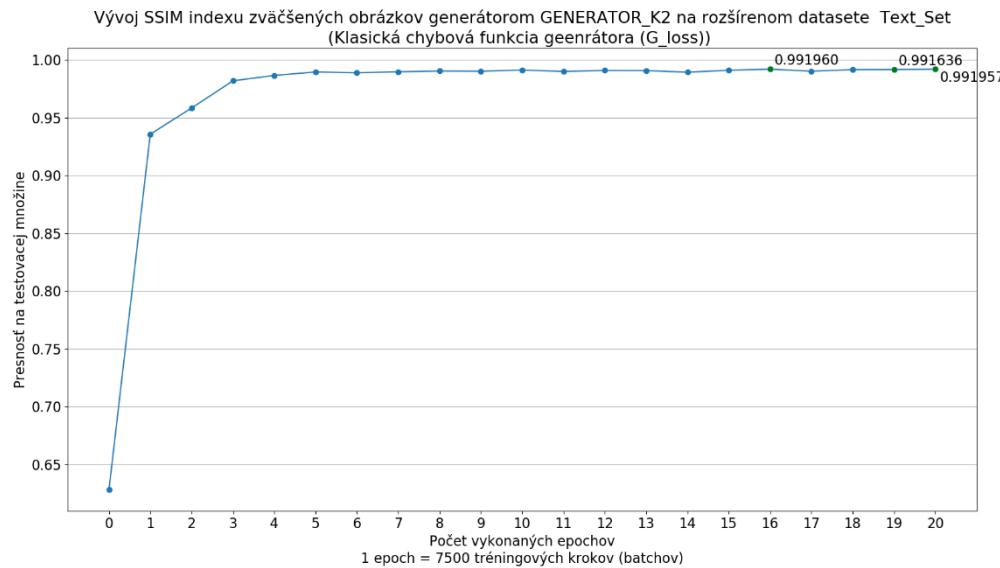
Rozšírime teda dataset *Text_Set*. Všetky parametre obrázkov zostanú rovnaké, ale prezentovanou aplikáciou necháme v tomto prípade generovať 60 000 tréningových a 10 000 testovacích obrázkov. Rovnako ako v predchádzajúcom experimente, najskôr prezentujeme výsledky rozpoznávania Tesseractu na pôvodnom a zmenšenom testovacom datase. Spôsob preprocessingu a úpravy výstupného rozpoznaného textu zostávajú totožné ako v predchádzajúcom experimente

Tabuľka 13 Percento úspešne rozpoznaných textov pre pôvodný a zmenšený testovací dataset

Rozšírený dataset	% správne rozpoznaných (OCR)
Pôvodný (112x112)	90.49%
Zmenšený (56x56)	0.03%

Na Tabuľka 12 opäť vidíme, že Tesseract opäť zlyháva pri rozpoznávaní textov malých rozmerov.

Tréning v prípade tohto experimentu predĺžime na 20 epoch a takisto vyberieme až tri najlepšie epochy na základe tréningovej metriky SSIM, ktorá bola použitá aj v predchádzajúcom experimente. Výberom troch najlepších epoch, ktorých výsledky rozpoznávania porovnáme medzi sebou, môžeme lepšie odpovedať na otázku či index SSIM spoľahlivo odráža výsledok rozpoznania textov na zväčšených obrázkoch.



Obrázok 37 Vývoj SSIM zväčšeného rozšíreného testovacieho datasetu *Text_Set* generátorom *GENERATOR_K2* tréovaným s klasickou chybovou funkciou

Z vývoja indexu SSIM počas tréningu znázorneného na Obrázok 37 vyberáme tri najlepšie generátory a to konkrétnie po epoche 16, 20 a 19. Pre každý vybratý generátor realizujeme rovnaké rozpoznávanie textov na zväčšených obrázkoch testovacieho datasetu pomocou Tesseractu ako v experimente 5.3.3., teda takisto použijeme rovnaký preprocessing a aj úpravu výstupného rozpoznaného reťazca.

Tabuľka 14 SSIM a výsledok rozpoznávania Tesseractom na zväčšených obrázkoch pomocou najlepších troch generátorov vybratých na základe SSIM

Generátor	SSIM	% správne rozpoznaných (OCR)
#1 GENERATOR_K2	0,991960	83,56
#2 GENERATOR_K2	0,991957	74,97
#3 GENERATOR_K2	0,991636	80,64

Na Tabuľka 14 si môžeme všimnúť výrazné zlepšenie správne rozpoznaných textov oproti experimentu 5.3.3. Pozrime sa ale ešte na výsledky ostatných metód.

Tabuľka 15 SSIM indexy a výsledky rozpoznávania OCR na zväčšených obrázkoch testovacieho rozšíreného datasetu *Text_Set*

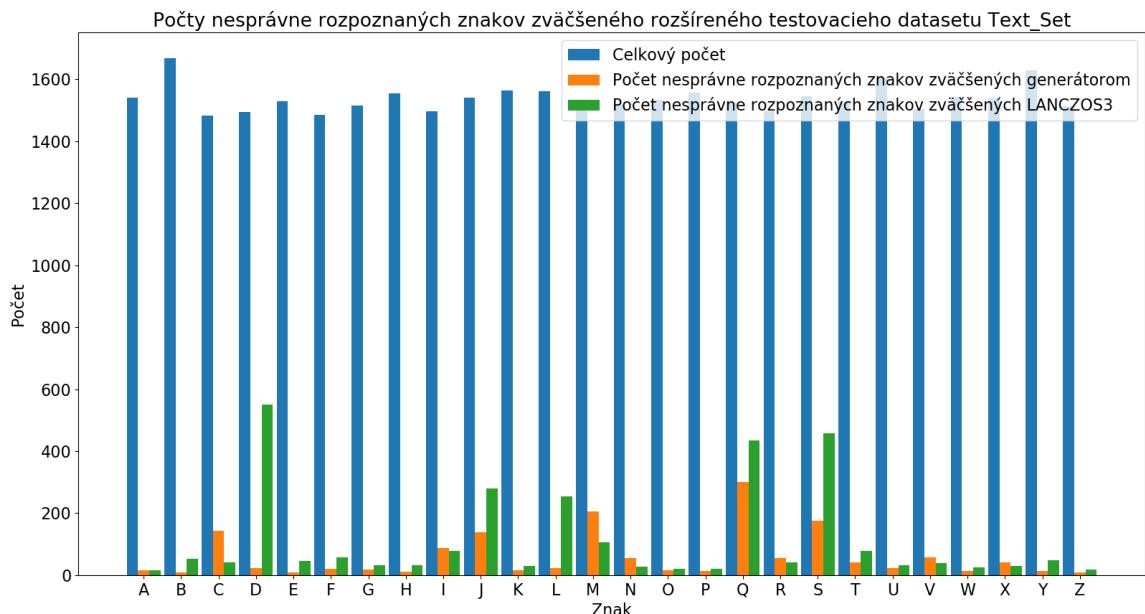
Zväčšovacia metóda	SSIM	% správne rozpoznaných (OCR)
#1 GENERATOR_K2	0,991960	83,56
#3 GENERATOR_K2	0,991636	80,64
#2 GENERATOR_K2	0,991957	74,97
LANCZOS3	0,958455	68,75
Bikubická interpolácia	0,954128	66,6
LANCZOS5	0,959466	65,24
Bilineárna interpolácia	0,943133	44,88
Metóda najbližšieho suseda	0,937459	2,71

Z Tabuľka 15 vyplýva, že všetky tri najlepšie generátory výrazne prekonali v tomto experimente ostatné zväčšovacie metódy. V prípade generátora, ktorý dosiahol najlepší index SSIM pozorujeme až o $83.56 - 68.75 = 14.81\%$ viac správne rozpoznaných textov oproti najlepšej zväčšovacej metódy LANCZOS3, čo je v prípade testovacieho datasetu veľkosti 10 000 až o 1481 obrázkov viac.

Teda môžeme konštatovať, že rozšírenie datasetu a predĺženie tréningu výrazne pomohlo k dosiahnutiu lepšieho zväčšenia pre prípad rozpoznávania textov Tesseractom na datasete *Text_Set* a tieto výsledky dokonca výrazne prekonali ostatné zväčšovacie metódy, čo v prípade s nerozšíreným datasetom (experiment 5.3.3) neplatilo.

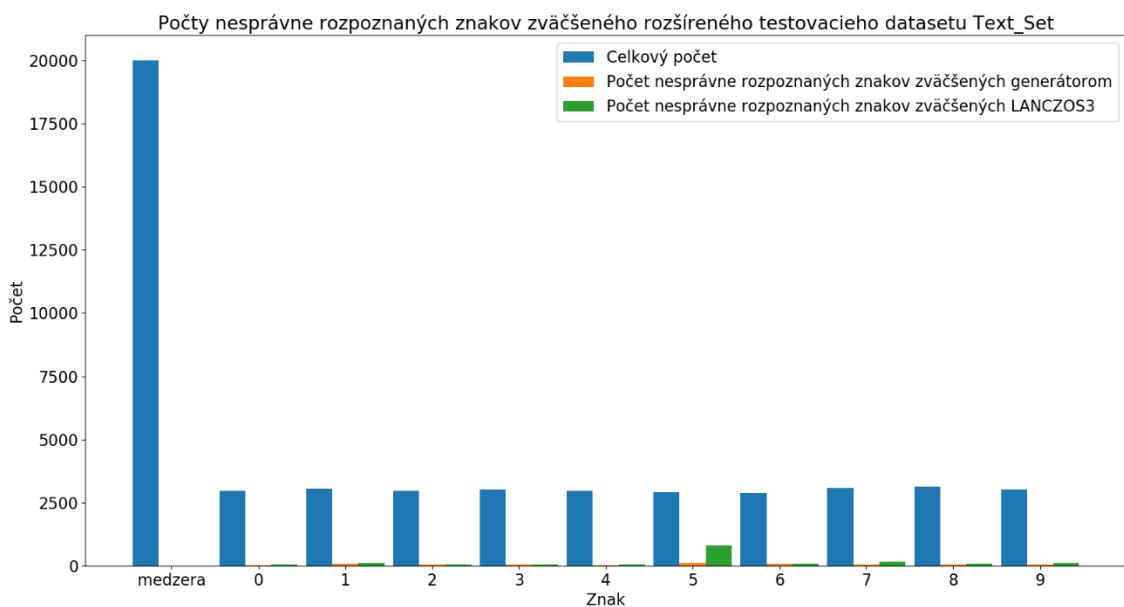
V prípade odpovede na otázku, či index SSIM spoľahlivo odráža výsledok správne rozpoznaných textov na zväčšených obrázkoch, z Tabuľka 15 vyplýva, že väčší index SSIM znamená lepší výsledok rozpoznania, ale len na pomerne veľkých rozdieloch hodnôt tohto indexu. V prípade malého zväčšenia alebo zmenšenia indexu SSIM nemôžeme teda konštatovať, že výsledok rozpoznania bude lepší, respektíve horší, pretože napríklad z Tabuľka 15 vidíme, že len veľmi malá zmena hodnôt SSIM pre #1 *GENERATOR_K2* a #2 *GENERATOR_K2* spôsobila rozdiel až 8.59%.

Pre lepšie porovnanie, prečo zväčšovanie pomocou generátorov dosiahlo lepšie výsledky ako ostatné metódy, predkladáme ešte počty nesprávne nerozpoznaných znakov vo zväčšených obrázkoch generátorom #1 *GENERATOR_K2* a interpoláciou LANCZOS3. Modré stĺpce predstavujú celkový počet výskytov jednotlivých znakov v skutočných textoch, ktoré sa na obrázkoch nachádzajú a ktoré poznáme z anotácie datasetu.



Obrázok 38 Počty nesprávne rozpoznaných písmen na zväčšených obrázkoch pomocou #1 GENERATOR_K2 a LANCZOS3 interpolácie

Z Obrázok 38 vidíme, že počty nesprávne rozpoznaných písmen sú takmer vo všetkých prípadoch väčšie v prípade zväčšenia pomocou LANCZOS3 interpolácie, čo sa logicky odráža na výsledku celkového rozpoznávania prezentovaného v Tabuľka 15. Zaujímavým pozorovaním je, že zväčšovanie písmen C, M, N, I a X pomocou generátora je menej úspešné ako zväčšovanie týchto písmen pomocou LANCZOS3 interpolácie. V ostatných prípadoch písmen platí opak, teda zväčšovanie generátorom bolo viac úspešné.



Obrázok 39 Počty nesprávne rozpoznaných čísel a medzere na zväčšených obrázkoch pomocou #1 GENERATOR_K2 a LANCZOS3 interpolácie

V prípade zväčšovania číslic a medzier z Obrázok 39 vidíme, že najväčší problém pri zväčšovaní pomocou oboch porovnávaných metód robila číslica 5, čo je očakávané, keďže spomedzi všetkých číslic je jej tvar najkomplexnejší.

Predkladáme ešte na porovnanie náhodne vybrané zväčšené obrázky testovacieho rozšíreného datasetu *Text_Set* pomocou generátora #1 *GENERATOR_K2* a LANCZOS3 interpolácie.

Príklad zväčšených obrázkov *Test_Set* generátorom #1 *GENERATOR_K2* a metódou LANCZOS3

Originál	BC 918 EP	KX 246 RO	HQ 099 MV	YW 598 XO	SQ 679 CU	BH 814 BY	OI 017 VA	LV 629 LN
Zmenšené	BC 918 EP	KX 246 RO	HQ 099 MV	YW 598 XO	SQ 679 CU	BH 814 BY	OI 017 VA	LV 629 LN
generátor	BC 918 EP	KX 246 RO	HQ 099 MV	YW 598 XO	SQ 679 CU	BH 814 BY	OI 017 VA	LV 629 LN
LANCZOS3	BC 918 EP	KX 246 RO	HQ 099 MV	YW 598 XO	SQ 679 CU	BH 814 BY	OI 017 VA	LV 629 LN
Originál	GQ 354 GS	CB 048 BT	EB 963 MB	QW 437 PE	HR 212 SV	JQ 387 MD	FA 235 YL	GV 272 AA
Zmenšené	GQ 354 GS	CB 048 BT	EB 963 MB	QW 437 PE	HR 212 SV	JQ 387 MD	FA 235 YL	GV 272 AA
generátor	GQ 354 GS	CB 048 BT	EB 963 MB	QW 437 PE	HR 212 SV	JQ 387 MD	FA 235 YL	GV 272 AA
LANCZOS3	GQ 354 GS	CB 048 BT	EB 963 MB	QW 437 PE	HR 212 SV	JQ 387 MD	FA 235 YL	GV 272 AA
Originál	HB 598 ZS	DE 103 TU	VC 401 NB	FN 518 BP	AW 522 GV	GY 446 VB	OB 792 NV	KO 163 TG
Zmenšené	HB 598 ZS	DE 103 TU	VC 401 NB	FN 518 BP	AW 522 GV	GY 446 VB	OB 792 NV	KO 163 TG
generátor	HB 598 ZS	DE 103 TU	VC 401 NB	FN 518 BP	AW 522 GV	GY 446 VB	OB 792 NV	KO 163 TG
LANCZOS3	HB 598 ZS	DE 103 TU	VC 401 NB	FN 518 BP	AW 522 GV	GY 446 VB	OB 792 NV	KO 163 TG

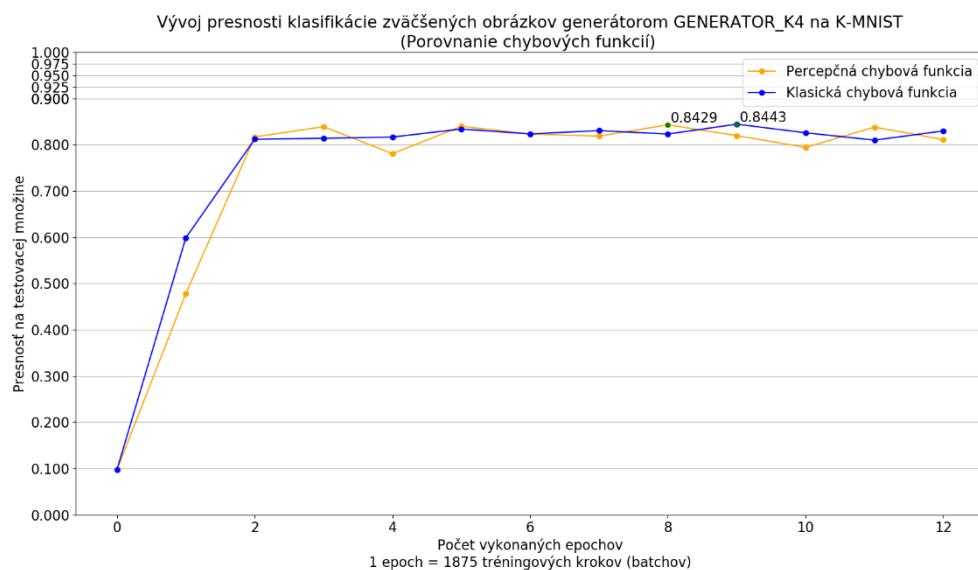
Obrázok 40 Porovnanie zväčšenia pomocou #1 *GENERATOR_K2* a interpolácie LANCZOS3 na rozšírenom datasete *Text_Set*

5.4 Štvornásobné zväčšovanie rozlíšenia

V tejto časti predstavíme výsledky a priebehy tréningov architektúry *GENERATOR_K4*, ktorý realizuje štvornásobné zväčšenie rozlíšenia. Experimenty sú rovnaké ako v prípade dvojnásobného zväčšenia, teda porovnávame prezentované chybové funkcie generátora a takisto porovnávame hodnoty porovnávacích metrík pre ostatné metódy zväčšovania.

5.4.1 *GENERATOR_K4* s klasifikátorom *K – MNIST*

Ked'že generátor v tomto prípade realizuje štvornásobné zväčšenie rozlíšenia, teda robí transformáciu $7 \times 7 \rightarrow \text{GENERATOR_K4} \rightarrow 28 \times 28$, znamená to, že v tomto prípade existuje väčšie množstvo neznámych pixelov, ktorých hodnoty sa generátor musí naučiť reprezentovať, čo znamená, že môžeme očakávať horšie hodnoty porovnávacích metrík a takisto aj menšiu úspešnosť klasifikácie.



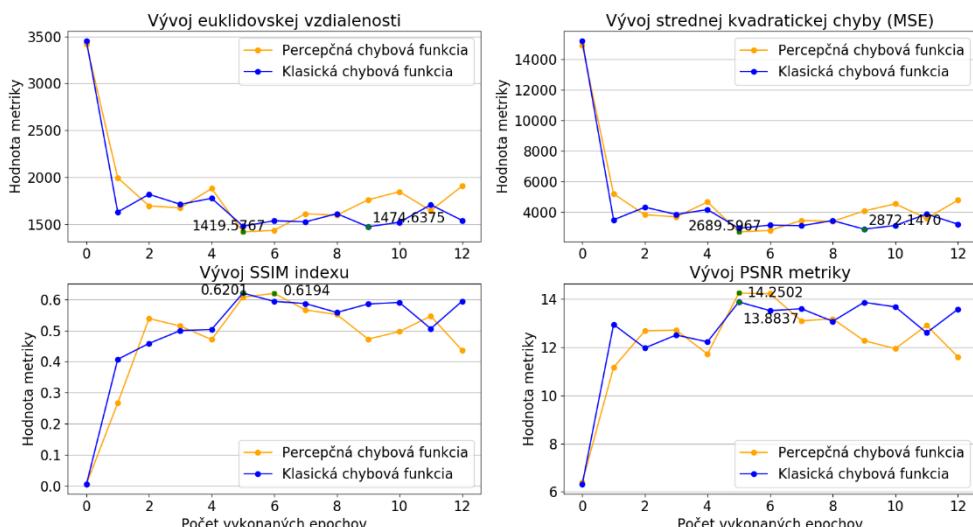
Obrázok 41 Vývoj úspešnosti klasifikácie zväčšeného testovacieho datasetu generátorom *GENERATOR_K4* pri oboch chybových funkcií, klasifikovanom na *K – MNIST*

Ked'že klasifikátor *K – MNIST* je natrénovaný s presnosťou 99.21%, znamená to nasledujúce rekonštrukčné chyby generátora *GENERATOR_K4*

Tabuľka 16 Rekonštrukčné chyby generátorov pri štvornásobnom zväčšení testovacieho datasetu MNIST

GENERATOR_K4 trénovaný na	Výsledok klasifikácie	Presnosť K-MNIST	<u>Rekonštrukčná chyba</u>
Klasická chybová funkcia	0,8443	0,9921	0,1478
Percepčná chybová funkcia	0,8429		0,1492

Z vyššie uvedených hodnôt vyplýva, že v oboch prípadoch chybových funkcií generátora, môžeme konštatovať $\pm 15\%$ zhoršenie klasifikácie, čo môžeme interpretovať tak, že okolo 15% obrázkov bolo zväčšených tak, že chyba aproximácie bola taká veľká, že klasifikátor nedokázal správne rozpoznať číslicu na obrázku.



Obrázok 42 Vývoj metrík v tréningoch *GENERATOR_K4* na *K – MNIST* pri oboch chybových funkciách

Z vývoja hodnôt jednotlivých metrík vidíme, že prvýkrát nastala situácia, že najlepšie hodnoty metrík neboli prítomné po skončení rovnakej epochy. Teda napríklad vidíme, že pre percepčnú chybovú funkciu generátora bola najlepšia hodnota indexu SSIM dosiahnutá po epochu 6, ale najlepšie hodnoty ostatných metrík tejto chybovej funkcie boli dosiahnuté po epochu 5.

Kedže ale trénujeme spôsobom najlepšej klasifikácie, výsledok tréningu na základe klasickej chybovej funkcie generátora je siet' po epochu 9 a v prípade percepčnej chybovej funkcie je to siet' po epochu 8. Takže opäť pozorujeme, že percepčná chybová funkcia mala

za následok, že najlepšia hodnota tréningovej metriky bola dosiahnutá skôr ako v prípade klasickej chybovej funkcie generátora . Porovnajme ešte výsledné generátory s ostatnými zväčšovacími metódami, ktoré v tomto prípade realizujú štvornásobné zväčšenie rozlíšenie.

Tabuľka 17 Hodnoty metrík štvornásobného zväčšenia testovacieho datasetu MNIST

Zväčšovacia metóda	Výsledok klasifikácie K-MNIST	Euklidovská vzdialenosť	Stredná kvadratická chyba (MSE)	SSIM index	PSNR
GENERATOR_K4	0,8443	1474,64	2872,14	0,59	13,87
GENERATOR_K4 (Percep loss)	0,8429	1597,82	3377,15	0,55	13,18
Metóda najbližšieho suseda	0,6629	1596,85	3306,78	0,45	13,09
LANCZOS5	0,5892	1471,53	2809,97	0,40	13,80
LANCZOS3	0,5812	1434,08	2669,90	0,42	14,02
Bikubická interpolácia	0,5253	1397,01	2534,31	0,44	14,25
Bilineárna interpolácia	0,3366	1415,47	2599,64	0,42	14,14

Z výsledkov vidíme, že spomedzi ostatných zväčšovacích metód sa najviac k najlepšiemu výsledku klasifikácie priblížila metóda najbližšieho suseda, ktorá však nedosiahla najlepšie hodnoty ostatných metrík z pomedzi ostatných zväčšovacích metód. Vidíme, že metrika euklidovskej vzdialenosť a stredná kvadratická chyba neodrážajú dobre to, čo od porovnávacích metrík očakávame, pretože napríklad pre výsledky metódy bikubickej interpolácie sú ich hodnoty najlepšie, ale SSIM a PSNR v tomto prípade najlepšie nie sú. Hodnoty SSIM a PSNR sú najlepšie v prípade klasickej chybovej funkcie, ktorá dosiahla aj najlepší výsledok klasifikácie zväčšených obrázkov.

Takisto ako v predchádzajúcich experimentoch, predkladáme aj vývoj zväčšovania generátorom trénovaného na základe klasickej chybovej funkcie po jednotlivých epochách tréningu



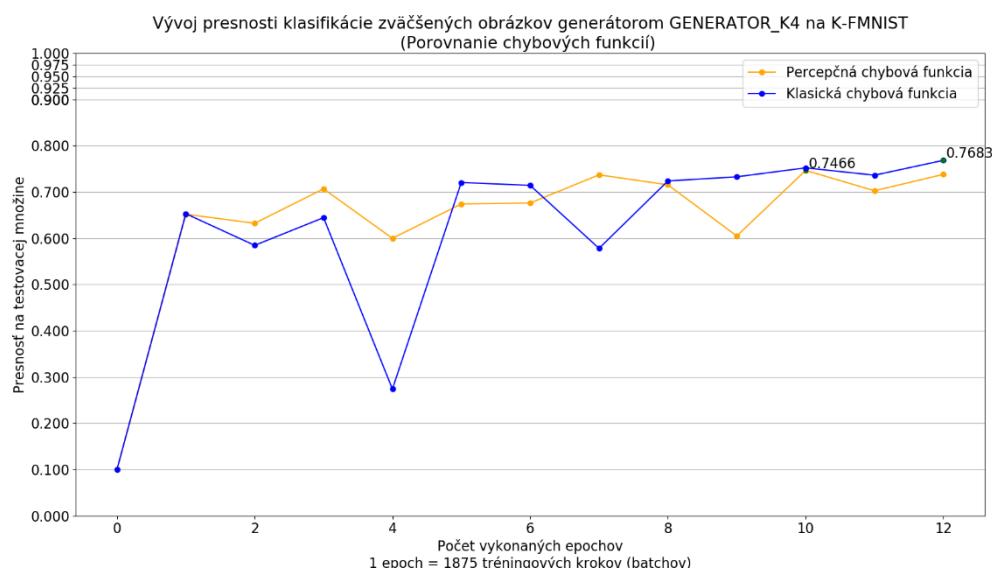
Obrázok 43 Vývoj štvornásobného zväčšovania náhodných obrázkov z testovacieho datasetu MNIST, generované generátorom *GENERATOR_K4* trénovanom na klasickej chybovej funkcií počas tréningu

Vidíme, že zväčšované obrázky s rozlíšením 7×7 už neobsahujú veľa detailov a preto mal generátor v tomto prípade väčšiu možnosť' niektoré detaľy viac approximovať', čo môžeme vidieť napríklad na Obrázok 43 po skončení epochy 11 (predposledný riadok obrázkov) na predposlednom obrázku, kde generátor vytvoril na spodnej časti číslice 4 slučku.

Príklady štvornásobného zväčšenia MNIST natrénovaným generátorom a ostatnými zväčšovacími metódami sa nachádzajú v Príloha B.

5.4.2 GENERATOR_K4 s klasifikátorom $K - FMNIST$

V tomto experimente porovnáme priebeh a výsledky trénovania generátora *GENERATOR_K4*, ktorý bude realizovať štvornásobné zväčšenie datasetu Fashion MNIST. Ako referenčný klasifikátor teda použijeme natrénovaný $K - FMNIST$ s presnosťou 90.06%. Takisto ako v predchádzajúcich experimentoch, trénujeme dvoma rôznymi chybovými funkciemi generátora.

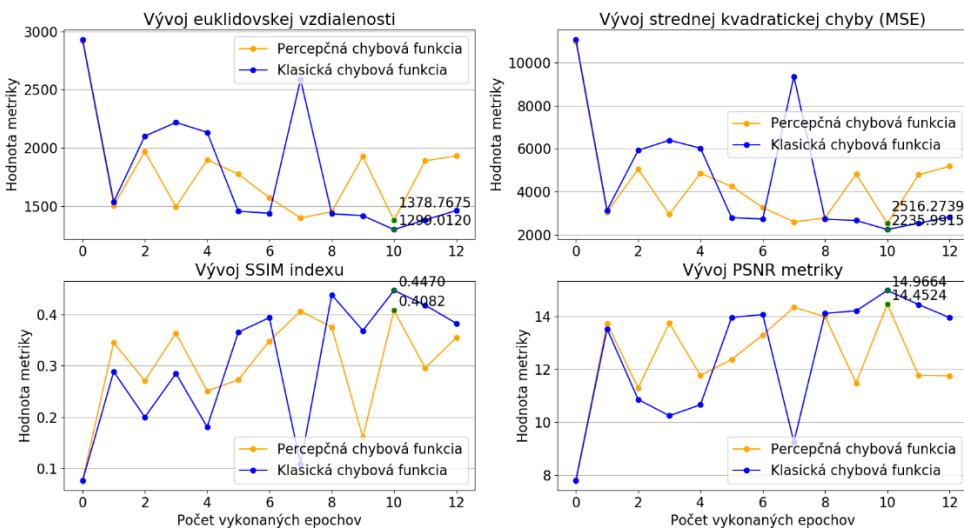


Obrázok 44 Vývoj úspešnosti klasifikácie zväčšeného testovacieho datasetu generátorom *GENERATOR_K4* pri oboch chybových funkcií, klasifikovanom na $K - FMNIST$

V prípade štvornásobného zväčšovania datasetu Fashion MNIST môžeme vidieť najväčšie skoky v hodnotách klasifikácie zväčšovaného testovacieho datasetu spomedzi doteraz prezentovaných experimentov. Tieto skoky sú menej výrazné v tréningu s percepčnou chybovou funkciou, ktorá najlepší výsledok klasifikácie 0.7456 dosiahla o jeden epoch skôr ako pri tréningu s klasickou chybovou funkciou, ktorej výsledok bol 0.7683. Čo znamená rekonštrukčné chyby

Tabuľka 18 Rekonštrukčné chyby oboch generátorov pri štvornásobnom zväčšení testovacieho datasetu FMNIST

GENERATOR_K4 trénovaný na	Výsledok klasifikácie	Presnosť K-FMNIST-2	<u>Rekonštrukčná chyba</u>
Klasická chybová funkcia	0,7683	0,9006	0,1323
Percepčná chybová funkcia	0,7466	0,9006	0,154



Obrázok 45 Vývoj metrík v tréningoch *GENERATOR_K4* na *K – FMNIST – 2* pri oboch chybových funkciách

Z vývoja metrík zobrazeného na Obrázok 45 vidíme, že v oboch prípadoch chybových funkcií boli najlepšie hodnoty dosiahnuté po epoche 10. Opäť sledujeme, že pri štvornásobnom zväčšení rozlíšenia datasetu FMNIST sa hodnoty metrík výrazne menia počas celého tréningu, čo v prípade predchádzajúcich experimentov neplatilo v takej veľkej miere. Toto sledovanie je spôsobené zložitosťou štvornásobného zväčšenia datasetu FMNIST, keďže jednotlivé obrázky s rozlíšením 7x7, ktoré generátor zväčšuje na 28x28 neobsahujú skoro žiadne detaily, a tak je generátor nútený v každej iterácii tréningu tieto detaily approximovať, keďže táto approximácia musí byť v závislosti na zložitosti datasetu FMNIST veľká, odráža sa to vo veľkých skokoch hodnôt metrík a teda aj v relatívne veľkých zmenách zväčšených obrázkov počas tréningu.

Vývoje zväčšených vybraných obrázkov počas tréningu pre oba prípady chybovej funkcie sú k dispozícii v Príloha C. Na obrázku v tejto prílohe si môžeme všimnúť, že generátor

s percepčnou chybovou funkciou mal pri konci tréningu tendenciu generovať obrázky, na ktorých je zväčšovaný objekt biely.

Takisto prezentujeme porovnanie výsledných natrénovaných generátorov s ostatnými zväčšovacími metódami

Tabuľka 19 Hodnoty metrík štvornásobného zväčšenia testovacieho datasetu FMNIST

Zväčšovacia metóda	Výsledok klasifikácie K-FMNIST	Euklidovská vzdialenosť	Stredná kvadratická chyba (MSE)	SSIM index	PSNR
GENERATOR_K4	0,7683	1461,66	2825,89	0,38	13,94
GENERATOR_K4 (Percep loss)	0,7466	1378,77	2516,27	0,41	14,45
Metóda najbližšieho suseda	0,7167	1418,12	2682,21	0,42	14,24
LANCZOS3	0,7083	1283,40	2202,31	0,43	15,12
LANCZOS5	0,7008	1309,25	2297,38	0,42	14,96
Bikubická interpolácia	0,6861	1251,72	2091,16	0,46	15,33
Bilineárna interpolácia	0,6062	1248,73	2078,12	0,45	15,35

Z výsledkov porovnávacích metrík opäť vidíme, že z hľadiska úspešnosti klasifikácie sa generátorom najviac priblížila metóda najbližšieho suseda, ako to bolo v prípade štvornásobného zväčšovania datasetu MNIST. Najlepšie hodnoty metrík dosiahla bikubická interpolácia.

Ked'že hodnoty všetkých metrík sú v tomto prípade pri všetkých metódach relatívne blízke, predkladáme na porovnanie aj náhodne vybrané obrázky zväčšené všetkými metódami

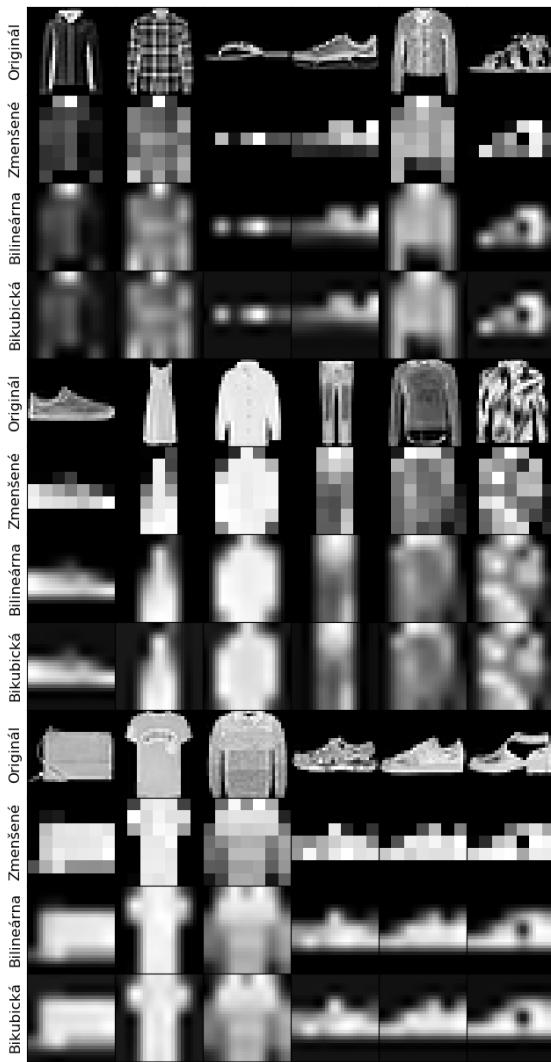
Príklad zväčšených obrázkov FMNIST generátorom GENERATOR_K4 pri oboch chybových funkciach



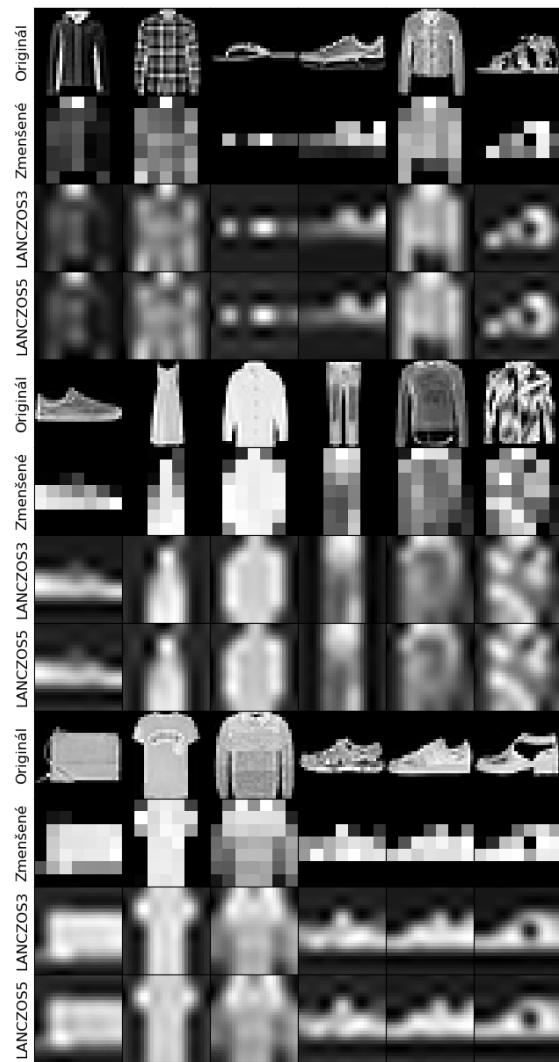
Obrázok 46 Porovnanie zväčšenia náhodne vybraných obrázkov z testovacej množiny FMNIST pre oba prípady chybových funkcií

V prípade výsledkov zobrazených na Obrázok 46 si môžeme všimnúť prítomnosť artefaktu bielych škvŕn, ktorý je najviac viditeľný na niektorých obrázkoch obuvi v prípade zväčšenia generátorom trénovaným na klasickej chybovej funkcií. V prípade použitia percepčnej chybovej funkcie tento artefakt v tomto experimente neregistrujeme.

Príklad zväčšených obrázkov FMNIST bilineárnu a bikubickou interpoláciou



Príklad zväčšených obrázkov FMNIST LANCZOS3 a LANCZOS5 interpoláciou



Obrázok 47 Porovnanie zväčšenia náhodne vybraných obrázkov z testovacej množiny FMNIST ostatnými zväčšovacími metódami

Z porovnania zväčšovania na Obrázok 47 vidíme, že metódy nevnášajú do obrázku žiadne nové detaľy, pretože ako sme spomínali v teoretickej časti práce, tieto metódy pristupujú rovnako k akémukoľvek obrázku, keďže sa jedná len o konštantný matematický predpis.

V prípade zväčšenia metódou najbližšieho suseda, nie je možné vidieť žiadny rozdiel v porovnaní so zmenšenými obrázkami, pretože metóda len kopíruje pôvodné pixely na

viacero pozícií a nevytvára tak žiadne iné pixely. Preto príklady zväčšenia metódou najbližšieho suseda neprezentujeme v tejto časti, k dispozícii sú ale v Príloha D.

Z predložených príkladov zväčšenia vidíme, že matematické predpisy zväčšovania rozlíšenia nevnášajú do obrázku dostatočné detaily, a tak sa nám obrázok zdá rozmazaný. Toto tvrdenie neplatí pre metódu najbližšieho suseda a takisto neplatí pre prezentované riešenia zväčšovania GANom, čo sme mohli vidieť z doteraz prezentovaných výsledkov.

5.4.3 ***GENERATOR_K4*** trénovaný na datasete ***Text_Set***

V tomto experimente realizujeme štvornásobné zväčšenie rozlíšenia na datasete *Text_Set*, teda realizujeme zväčšenie $28 \times 28 \rightarrow \text{GENERATOR_K2} \rightarrow 112 \times 112$. Postup je identický ako v experimente 5.3.4, teda výsledkom tréningu GANu je taký model generátora *GENERATOR_K4*, ktorý dosiahol najlepšiu hodnotu SSIM, ktorý je počítaný po každej epoche na základe celého testovacieho datasetu *Text_Set*. V tomto prípade takisto začíname tréning s 12 epochami. Následne takýto natrénovaný generátor porovnávame s ostatnými prezentovanými metódami zväčšovania rozlíšenia. Toto porovnanie rovnako realizujeme na základe počtu správne rozpoznaných textov zo zväčšeného testovacieho datasetu, ktorý po zväčšení prešiel rovnakým preprocessingom ako v experimente 5.3.4. Porovnanie výsledku OCR so skutočným textom je realizované taktiež rovnako.

Najskôr predkladáme hodnoty výsledných metrík medzi pôvodným testovacím datasetom a zväčšením datasetom

Tabuľka 20 Hodnoty metrík štvornásobného zväčšenia testovacieho datasetu *Text_Set*

Zväčšovacia metóda	Euklidovská vzdialenosť	Stredná kvadratická chyba (MSE)	SSIM index	PSNR
GENERATOR_K4	3432,42	943,38	0,92	18,42
Metóda najbližšieho suseda	3794,63	1149,65	0,88	17,54
LANCZOS3	3620,25	1047,34	0,86	17,95
LANCZOS5	3713,55	1101,96	0,83	17,73
Bikubická interpolácia	3459,12	956	0,88	18,35
Bilineárna interpolácia	3375,81	910,14	0,88	18,56

Môžeme si všimnúť, že hodnoty metrík sú podobné a taktiež vidíme, že SSIM index má veľmi dobré hodnoty, pretože ako bolo spomínané v 5.1.1, hodnota 1 sa pri SSIM indexe dá dosiahnuť len v prípade zhodných obrázkov. Preto výsledky metrík hovoria, že obrázky sú zväčšené relatívne dobre, pozrime sa ale na zopár príkladov

Príklad zväčšených obrázkov Test_Set generátorom #1 GENERATOR_K2 a metódou LANCZOS5

	NN 565 QM	HV 151 AC	DM 416 QD	RI 040 TQ	TR 667 JT	PP 982 VY	FB 902 ZR	RW 150 CW
Originál	NN 565 QM	HV 151 AC	DM 416 QD	RI 040 TQ	TR 667 JT	PP 982 VY	FB 902 ZR	RW 150 CW
Zmenšené	NN 565 QM	HV 151 AC	DM 416 QD	RI 040 TQ	TR 667 JT	PP 982 VY	FB 902 ZR	RW 150 CW
generátor	NN 565 QM	HV 151 AC	DM 416 QD	RI 040 TQ	TR 667 JT	PP 982 VY	FB 902 ZR	RW 150 CW
LANCZOS5	NN 565 QM	HV 151 AC	DM 416 QD	RI 040 TQ	TR 667 JT	PP 982 VY	FB 902 ZR	RW 150 CW
	DE 561 PN	MY 373 TG	DG 647 WE	QW 965 BD	CF 795 WJ	WR 858 KO	VA 881 KM	KA 258 HF
Originál	DE 561 PN	MY 373 TG	DG 647 WE	QW 965 BD	CF 795 WJ	WR 858 KO	VA 881 KM	KA 258 HF
Zmenšené	DE 561 PN	MY 373 TG	DG 647 WE	QW 965 BD	CF 795 WJ	WR 858 KO	VA 881 KM	KA 258 HF
generátor	DE 561 PN	MY 373 TG	DG 647 WE	QW 965 BD	CF 795 WJ	WR 858 KO	VA 881 KM	KA 258 HF
LANCZOS5	DE 561 PN	MY 373 TG	DG 647 WE	QW 965 BD	CF 795 WJ	WR 858 KO	VA 881 KM	KA 258 HF
	NI 935 MA	VB 101 CU	ZQ 069 UX	GD 398 RM	ZV 290 WJ	DP 310 BX	TR 973 SV	HW 058 SH
Originál	NI 935 MA	VB 101 CU	ZQ 069 UX	GD 398 RM	ZV 290 WJ	DP 310 BX	TR 973 SV	HW 058 SH
Zmenšené	NI 935 MA	VB 101 CU	ZQ 069 UX	GD 398 RM	ZV 290 WJ	DP 310 BX	TR 973 SV	HW 058 SH
generátor	NI 935 MA	VB 101 CU	ZQ 069 UX	GD 398 RM	ZV 290 WJ	DP 310 BX	TR 973 SV	HW 058 SH
LANCZOS5	NI 935 MA	VB 101 CU	ZQ 069 UX	GD 398 RM	ZV 290 WJ	DP 310 BX	TR 973 SV	HW 058 SH
	NI 935 MA	VB 101 CU	ZQ 069 UX	GD 398 RM	ZV 290 WJ	DP 310 BX	TR 973 SV	HW 058 SH
Originál	NI 935 MA	VB 101 CU	ZQ 069 UX	GD 398 RM	ZV 290 WJ	DP 310 BX	TR 973 SV	HW 058 SH
Zmenšené	NI 935 MA	VB 101 CU	ZQ 069 UX	GD 398 RM	ZV 290 WJ	DP 310 BX	TR 973 SV	HW 058 SH
generátor	NI 935 MA	VB 101 CU	ZQ 069 UX	GD 398 RM	ZV 290 WJ	DP 310 BX	TR 973 SV	HW 058 SH
LANCZOS5	NI 935 MA	VB 101 CU	ZQ 069 UX	GD 398 RM	ZV 290 WJ	DP 310 BX	TR 973 SV	HW 058 SH

Obrázok 48 Porovnanie zväčšenia pomocou GENERATOR_K4 a interpolácie LANCZOS5 na datasete Text_Set

Z príkladov zväčšení ale vidíme, že zväčšenie vôbec nie je prijateľné, keďže zväčšené texty sú nečitateľné čo ale nie je žiadne prekvapenie, keďže zmenšené obrázky neobsahujú dostatok pixelov na dostatočnú reprezentáciu textu. Z výsledkov je jasné, že algoritmus OCR nerozpoznal žiadny zväčšený text, teda presnosť OCR bola v prípade každej jednej metódy a generátora 0%. V tomto experimente sa preto ukazuje, že použité metriky nedokážu dobre určiť, či sa zväčšenie podarilo alebo nie, pretože tieto metriky nijako nesúvisia s konkrétnymi objektmi na obrázku, ktoré zväčšujeme. Toto sa netýka spomínanej metriky založenej na výsledku klasifikácie zväčšených obrázkov a spomínaného OCR, pretože tieto dve metódy posudzujú zväčšenie práve z pohľadu objektu, ktorý zväčšujeme.

V tomto prípade teda nebudem na experiment nadväzovať ďalším ako to bolo v prípade dvojnásobného zväčšovania datasetu *Text_Set* v experimentoch 5.3.3 a 5.3.4, keďže z prezentovaných výsledkov vyplýva, že to nemá zmysel, keďže zmenšené obrázky neobsahujú dostatok detailov na úspešné zväčšenie.

5.5 Zhrnutie experimentov

V tejto kapitole čitateľovi stručne prezentujeme zistené pozorovania a niektoré kľúčové hodnoty získané v experimentálnej časti práce.

V experimentoch dvojnásobného zväčšenia datasetu MNIST a FMNIST sa ukázalo, že percepčná chybová funkcia mala za následok lepšie hodnoty všetkých porovnávacích metrík okrem výsledku klasifikácie zväčšených obrázkoch, ktorej prikladáme najväčšiu váhu. Práve výsledok klasifikácie je ale metrikou, ktorá bola v prípade zväčšenia GANmi lepšia ako v prípade zväčšovania ostatnými metódami. Preto môžeme konštatovať, že dvojnásobné zväčšenie datasetov MNIST a FMNIST je z hľadiska zväčšovaných objektov, teda rukou písaných číslí, respektíve druhov oblečení, lepšie realizovať GANmi ako ostatnými prezentovanými metódami. Toto tvrdenie takisto platí aj pre klasickú chybovú funkciu generátora, keďže aj v tomto prípade boli dosiahnuté lepšie výsledky klasifikácií zväčšených testovacích datasetov.

V prípade experimentu dvojnásobného zväčšenia datasetu *Text_Set* sme ukázali, že pri dostatočne veľkom počte tréningových obrázkov a dostatočne dlhom tréningu, dosahuje lepšie výsledky takisto riešenie zväčšovania GANmi. V tomto experimente sme výsledky hodnotili na základe úspešne rozpoznaných textov algoritmom OCR na zväčšených obrázkoch, ktoré takisto ako metrika klasifikácie, posudzuje zväčšenie priamo z hľadiska

zväčšovaného objektu, ktorý bol v tomto prípade text na obrázku, ktorý algoritmus OCR rozpoznáva. V tomto experimente sme čitateľovi takisto ukázali dôležitosť dostatočného množstva tréningových dát a potrebu dlhšieho tréningu v prípade zväčšovania textov.

V prípade štvornásobného zväčšenia rozlíšenia datasetu MNIST a FMNIST sa pozorovanie z experimentov dvojnásobného zväčšenia, že percepčná chybová funkcia spôsobuje lepšie hodnoty porovnávacích metrík nepotvrdilo. V oboch prípadoch chybových funkcií pre oba datasety ale opäť platí zistenie, že zväčšovanie GANmi dosiahlo výrazne lepší výsledok porovnania na základe metriky klasifikácie na vopred natrénovanom klasifikátore, ktoréj prikladáme najväčšiu váhu.

V prípade štvornásobného zväčšenia datasetu *Text_Set* žiadna metóda nedosiahla priateľné výsledky, teda žiadnej metóde sa v tomto prípade nepodarilo zväčšiť žiadny z obrázkov tak, aby sa text stal rozpoznateľným algoritmom OCR.

Na záver prekladáme pre porovnanie dosiahnuté hodnoty metriky úspešnosti klasifikácie pre dvojnásobné a štvornásobné zväčšenie rozlíšenia datasetov MNIST a FMNIST

Tabuľka 21 Porovnanie úspešnosti klasifikácie dvojnásobného a štvornásobného zväčšenia rozlíšenia testovacieho datasetu MNIST

Zväčšovacia metóda	Výsledok klasifikácie na klasifikátore		
	MNIST (K-MNIST) (presnosť 99.21%)	Dvojnásobné zväčšenie	Štvornásobné zväčšenie
GAN, percepčná chybová funkcia	0,9874	0,8429	
GAN, klasická chybová funkcia	0,9867	0,8443	
Metóda najbližšieho suseda	0,9747	0,6629	
LANCZOS5	0,9729	0,5892	
LANCZOS3	0,9702	0,5812	
Bikubická interpolácia	0,9559	0,5253	
Bilineárna interpolácia	0,8567	0,3366	

Tabuľka 22 Porovnanie úspešnosti klasifikácie dvojnásobného a štvornásobného zväčšenia rozlíšenia testovacieho datasetu Fashion MNIST

Zväčšovacia metóda	Výsledok klasifikácie na klasifikátore FMNIST (K-FMNIST) (presnosť 90.06%)	
	Dvonásobné zväčšenie	Štvornásobné zväčšenie
GAN, percepčná chybová funkcia	0,8661	0,7466
GAN, klasická chybová funkcia	0,8682	0,7683
Metóda najbližšieho suseda	0,8619	0,7167
LANCZOS5	0,8542	0,7008
LANCZOS3	0,8554	0,7083
Bikubická interpolácia	0,8436	0,6861
Bilineárna interpolácia	0,7826	0,6062

Záver

Cieľom práce bolo preskúmať možnosti zväčšovania rozlíšenia založeného na Generatívnych konfrontačných sietiach (GAN) a porovnať ho s existujúcimi najpoužívanejšími metódami na to určenými. Toto porovnanie sme realizovali na základe metriky založenej na úspešnej klasifikácii zväčšených obrázkov a takisto na základe iných známych metrík.

V prvej časti práce sme čitateľa stručne uviedli do problému zväčšovania rozlíšenia. V kapitole 2 sa venujeme vysvetleniu princípu fungovania používaných metód zväčšovania, ktoré sme v experimentálnej časti porovnávali s riešením založeným na spomínaných GANoch. Prístup fungovania GANov a všeobecnú myšlienku neurónových sietí sme čitateľovi prezentovali v kapitole 3. V tejto kapitole sme takisto predstavili všetky použité vrstvy neurónových sietí, ktoré sme následne v kapitole 4 použili na tvorbu konkrétnych architektúr GANov. Tieto vytvorené architektúry sme v experimentálnej časti, ktorá sa nachádza v kapitole 5 porovnávali a takisto sme v nej predstavili percepčnú chybovú funkciu generátora, ktorej výsledky a vplyv na tréningový proces sme taktiež hodnotili a porovnávali. Toto porovnanie sme vykonávali na základe metrík, ktoré sme takisto predstavili v kapitole 5. Jednotlivé experimenty sme vykonávali na známych referenčných datasetoch MNIST a Fashion MNIST a taktiež na dataseite, ktorý vznikol ako výstup aplikácie na tvorbu datasetov obrázkov, v ktorých sa nachádza text. Táto aplikácia taktiež vznikla ako súčasť práce a bližšie sme ju predstavili v kapitole 4.4.2.

V experimentoch sa ukázalo, že pri hodnení zväčšených obrázkov na základe výsledku klasifikácie pre datasety MNIST a Fashion MNIST dosahuje lepšie výsledky riešenie zväčšovania GANmi a najviac sa mu približuje zväčšovanie pomocou metódy najbližšieho suseda. V prípade zväčšovania datasetu, ktorý simuluje formát EČV, sme na hodnenie výsledkov použili metriku založenú na počte správne rozpoznaných textov na zväčšených obrázkoch. Toto rozpoznávanie sme realizovali algoritmom OCR, konkrétnie jednou z najpoužívanejších implementácií s názvom Tesseract. V tomto prípade výsledky ukázali, že zväčšovanie rozlíšenia GANmi pri vhodnom počte tréningových dát a dostatočne dlhom tréningu výrazne prekonalo ostatné prezentované metódy zväčšovania, v najlepšom prípade bolo zlepšenie dosiahnuté GANmi až 14.81%, čo v prípade tohto datasetu znamenalo až o 1481 obrázkov viac na ktorých sa úspešne podaril rozpoznať text.

Zoznam použitej literatúry

- [Witt05] T. Wittman, *Mathematical Techniques for Image Interpolation*, 2015, strana 3
- [BB16] W. Burger a M.J. Burger, *Digital Image Processing: An Algorithmic Introduction Using Java Second Edition*, 2016, strana 539-555, ISBN: 978-1-4471-6684-9
- [YG+07] I.T. Young, J.J Gerbrands a L. J. van Vliet, *Fundamentals of Image Processing*, Delft University of Technology
- [Turk90] K.Turkowski, *Filters for Common Resampling Tasks*, strana 9
- [MP43] W.S.McCulloch a W.Pitts, *A Logical Calculus of Ideas Immanent in Nervous Activity*. v Bulletin of Mathematical Biophysics, 1943
- [Hebb49] D.Hebb, *The Organization of Behaviour*. ISBN: 978-0-471-36727-7
- [Rose58] F.Rosenblatt, *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain* v Cornell Aeronautical Laboratory, Psychological Review , 1964
- [MP69] M.Minsky a P.Papert: *Perceptrons: An Introduction to Computational Geometry*, The MIT Press . 1969. ISBN: 0-262-63022-2
- [BG+17] F.Bre,J.M.Gimenez a V.D.Fachinotti, *Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks*. 2017. Dostupné na: <https://www.researchgate.net/publication/321259051> (8.5.2020)
- [Chan18] A.L.Chandra, *McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron*. 2018. Dostupné na: <https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1> (8.5.2020)
- [Cyb89] G.Cybenko, *Approximations by superpositions of sigmoidal functions*, Mathematics of Control, Signals, and Systems. 1989.
- [LP+17] Z.Lu,H.Pu,F.Wang et al. *The Expressive Power of Neural Networks: A View from the Width*. Neural Information Processing Systems. 2017.

- [KH60] Kelley,J.Henry, *Gradient theory of optimal flight paths.* ARS Journal. 30. 1960
- [RHW86] D.E.Rumelhard, G.E.Hinton a R.J. Williams, *Learning representations by back-propagating errors.* Nature. 323. 1986
- [SH+14] N.Srivastava, G.Hinton et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting,* Journal of Machine Learning Research 15. 2014
- [RM16] A. Radford a L.Metz, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* 2016
- [KB15] D.P.Kingma a J.L.Ba, *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION.* ICLR. 2015
- [GA+14] I.J.Goodfellow, J.Pouget-Abadie et al. *Generative Adversarial Nets.* 2014
- [LT+17] C.Ledig, L.Theis et al Photo-Realistic *Single Image Super-Resolution Using a Generative Adversarial Network.* 2017.
- [WZ04] Wang,Zhou et al. *Image quality assessment: from error visibility to structural similarity.* IEEE Transactions on Image Processing. 13. 2004.
- [HZ10] A.Horé,D.Ziou *Image Quality Metrics: PSNR vs. SSIM,* International Conference on Pattern Recognition, Istanbul. 2010. doi: 10.1109/ICPR.2010.579

Zoznam príloh

Príloha A Priebeh tréningu klasifikátorov

Príloha B Príklady štvornásobného zväčšenia MNIST generátorom *GENERATOR_K2* a ostatnými zväčšovacími metódami

Príloha C Vývoj zväčšených vybraných obrázkov počas tréningu štvornásobného zväčšenia MNIST pre obe chybové funkcie generátora

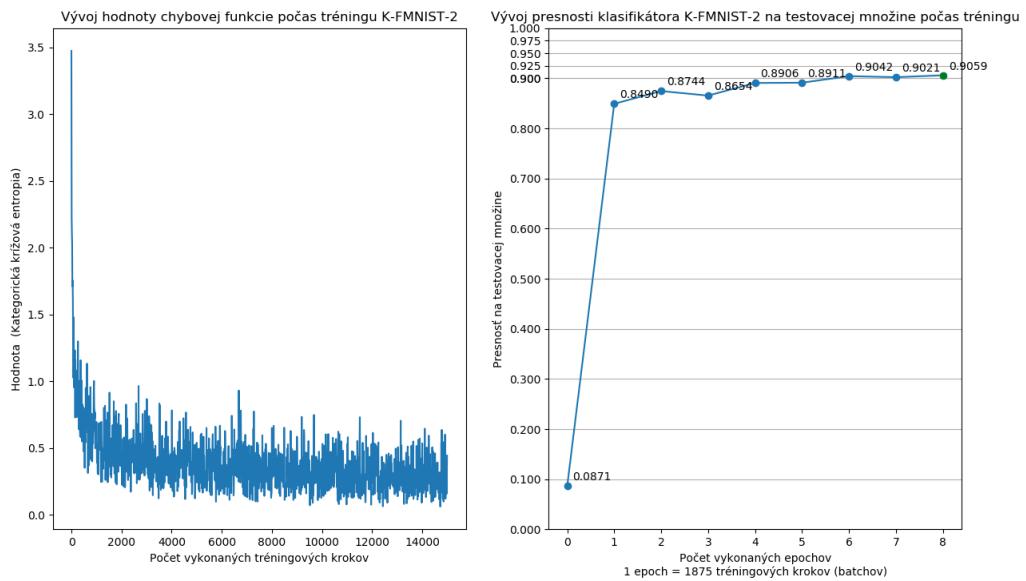
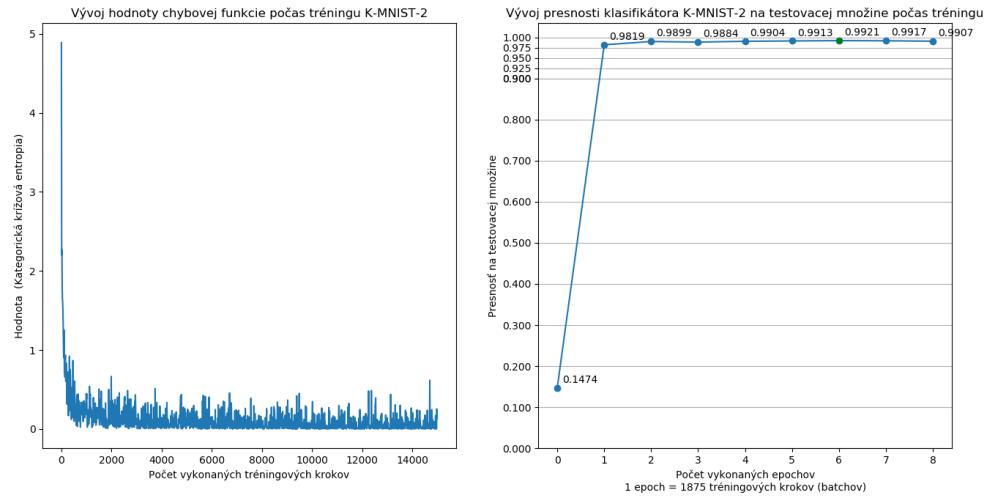
Príloha D Štvornásobné zväčšenie náhodných obrázkov datasetu FMNIST metódou najbližšieho suseda

Príloha E DVD médiu, na ktorom sa nachádza:

- Práca vo formáte .pdf
- Zdrojové kódy implementácií a natrénované modely sietí spolu s dátami použitými v tabuľkách a grafoch

Prílohy

Príloha A: Priebeh tréningu klasifikátorov



Príloha B: Príklady štvornásobného zväčšenia MNIST generátorom *GENERATOR_K2* a ostatnými zväčšovacími metódami

Príklad zväčšených obrázkov MNIST generátorom GENERATOR_K4 tréovanom na klasickej chybovej funkcií



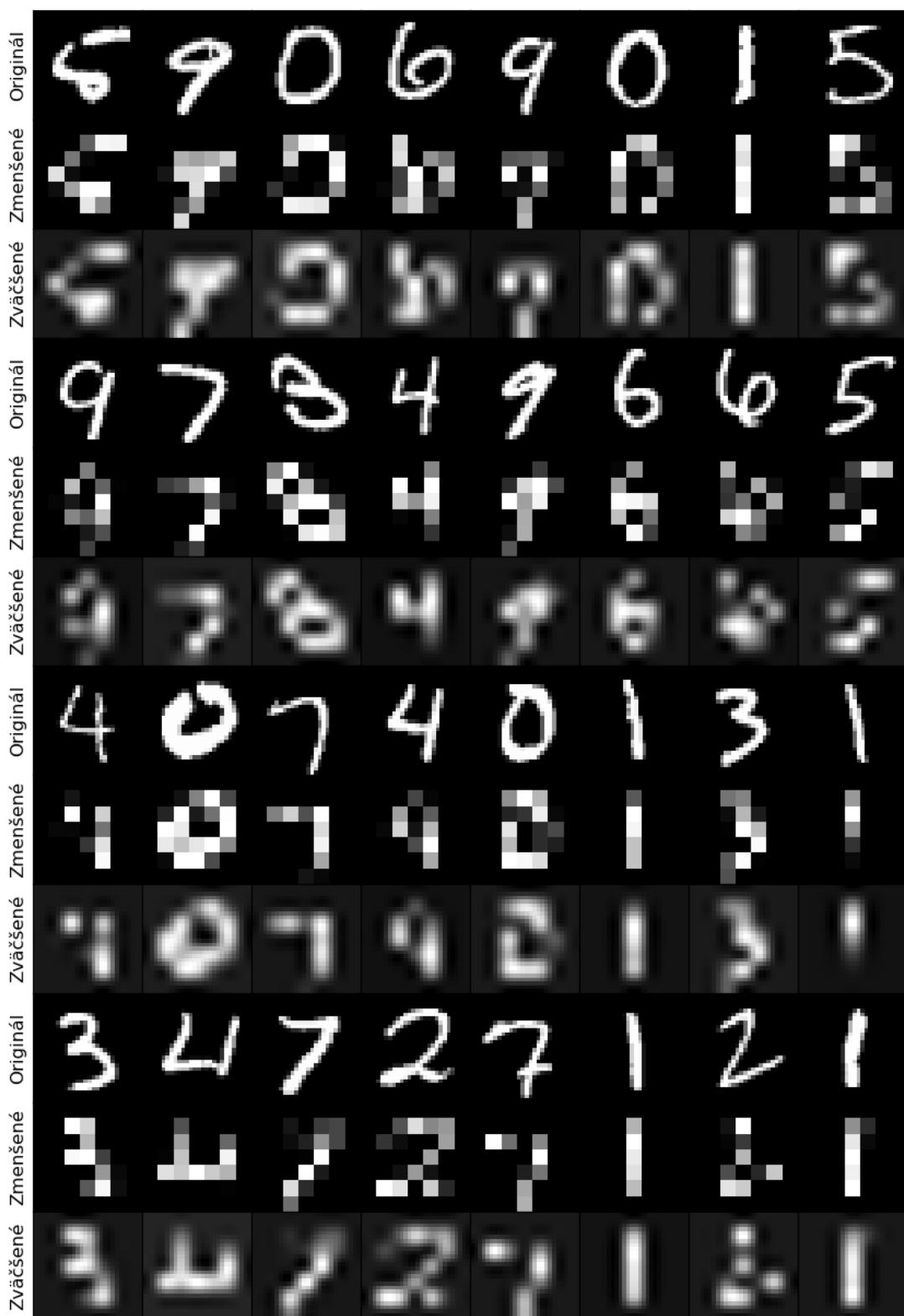
Príklad zväčšených obrázkov MNIST generátorom GENERATOR_K4 trénovanom na percepčnej chybovej funkcií.



Príklad zväčšených obrázkov MNIST bilineárной interpoláciou



Príklad zväčšených obrázkov MNIST bikubickou interpoláciou



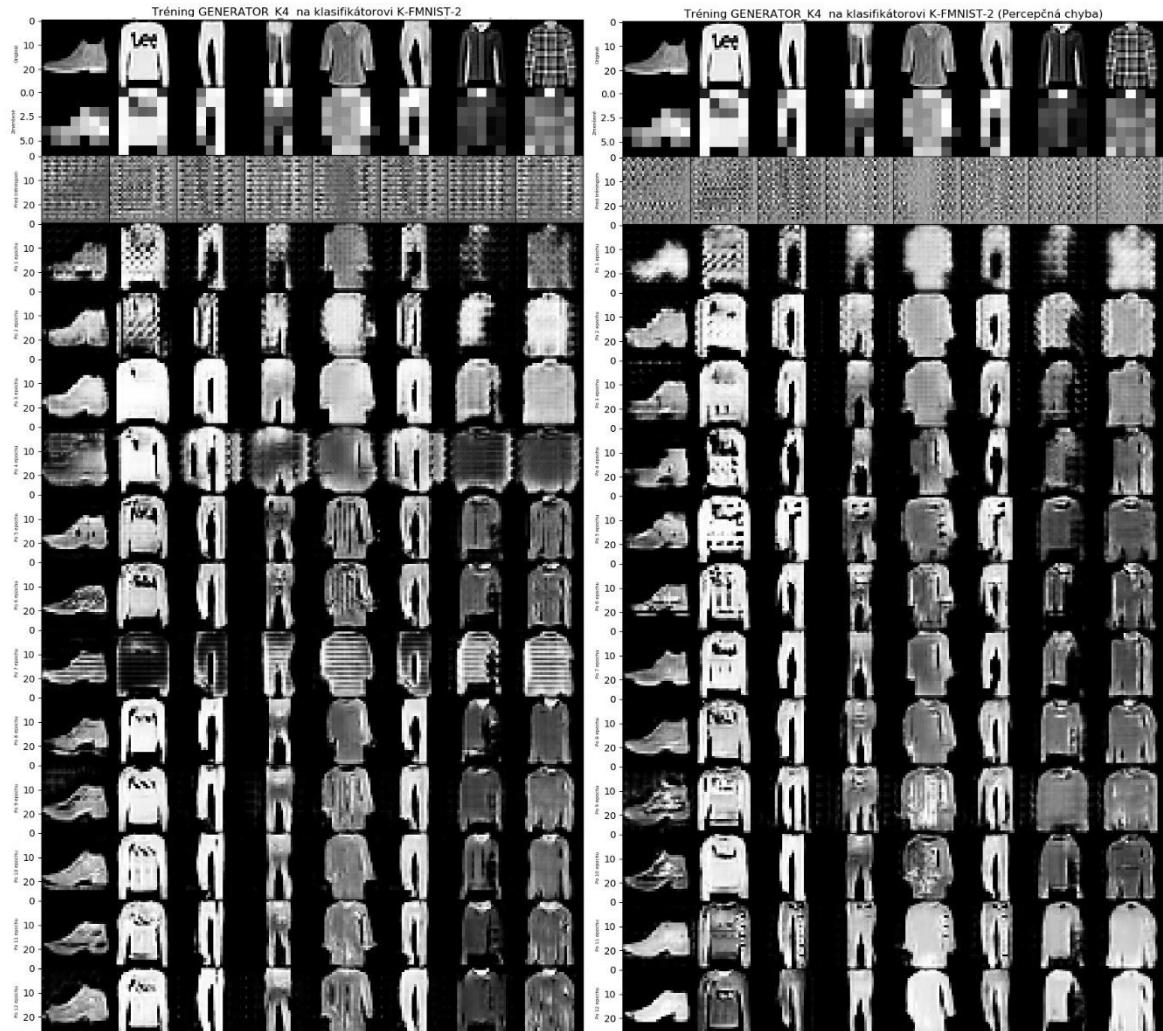
Príklad zväčšených obrázkov MNIST LANCZOS3 interpoláciou



Príklad zväčšených obrázkov MNIST LANCZOSS interpoláciou



Príloha C: Vývoj zväčšených vybraných obrázkov počas tréningu štvornásobného zväčšenia MNIST pre obe chybové funkcie generátora



Príloha D: Štvornásobné zväčšenie náhodných obrázkov datasetu FMNIST metódou najbližšieho suseda

Príklad zväčšených obrázkov FMNIST metódou najbližšieho suseda

