VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

# Generování nočních obrazů z denních

Generating Night Image from Day Image

Petr Dihel

Diplomová práce

Vedoucí práce: Ing. Jan Gaura, Ph.D.

Ostrava, 2023

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/22/078936
vsboee22026024

# Zadání diplomové práce

Student: **Bc. Petr Dihel**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Generování nočních obrazů z denních
Generating Night Image from Day Image

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je generovat noční obrazy na základě poskytnutého denního obrazu. Takto generovaný obraz je možno použít pro testování jiných algoritmů počítačového vidění, např. v automobilových systémech. Obrazy budou vytvářeny pomocí generativních adversariálních sítí (Generative Adversarial Nets, GANs), které jsou poté použity pro trénování hlubokých neuronových sítí. Z dostupných dat vytvořte takový dataset, který je vhodný pro trénování takto zadané úlohy.

Ve své práci proveďte:
1. Nastudujte techniky generování obrazů pomocí generativních adversariálních sítí.
2. Sestavte adekvátní dataset pro zadanou trénovací úlohu z dodaných obrazů scén ve dne a noci.
3. Naimplementujte nebo rozšiřte existující algoritmy, které budou schopny natrénovat model strojového učení podle Vašeho datasetu.
4. Svou práci náležitě otestujte a výsledky zdokumentujte v závěru práce.

Seznam doporučené odborné literatury:

[1] Jun-Yan Zhu*, Taesung Park*, Phillip Isola, and Alexei A. Efros. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", in IEEE International Conference on Computer Vision (ICCV), 2017.
[2] Tero Karras and Samuli Laine and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks",
arXiv: 1812.04948, 2019.

17. listopadu 2172/15
708 00 Ostrava-Poruba
Česká republika

spojovatelka: +420 597 321 111
epodatelna: epodatelna@vsb.cz
ID datové schránky: d3kj88v

IČ: 61989100
DIČ: CZ61989100

email: studijni.fei@vsb.cz
www.fei.vsb.cz

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Gaura, Ph.D.**

Datum zadání: 01.09.2022

Datum odevzdání: 30.04.2023

Garant studijního oboru: prof. RNDr. Václav Snášel, CSc.

V IS EDISON zadáno: 07.11.2022 11:59:22

## Abstrakt

Tato práce se zabývá tématem generativních adversariálních sítí a jejich využití pro generování nočních snímků z denních. V teoretické části jsou nejprve představeny architektury, techniky a vlastnosti hlubokých neuronových sítí, které jsou později aplikovány. Následně je popsána příprava a charakteristika trénovacího a testovacího datasetu z dodaných materiálů. Poté práce popisuje implementaci vhodných metod a architektur, vybraných na základě teoretického rozboru, a detailně popisuje průběh experimentů. V závěrečné části je porovnání výsledků různých experimentů a jejich zhodnocení.

## Klíčová slova

GAN; neuronové sítě, učení bez dozoru

## Abstract

This paper deals with the topic of generative adversarial networks and their use for the generation of the nigh-time images from the day-time ones. Initially, the theoretical part introduces the architectures, techniques and properties of deep neural networks, which are later applied. Next, the preparation and characterization of the training and test dataset from the provided materials is described. Then, the paper describes the implementation of the appropriate methods and architectures selected based on the theoretical analysis and details the experiments. The final section compares the results of the different experiments and evaluates them.

## Keywords

GAN, neuron networks, unsupervised learning

## Poděkování

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli a kteří mě po celou dobu podporovali, protože bez nich by tato práce nevznikla.

# Obsah

# Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| GAN | – | Generativní adversariální sítě |
| GPU | – | Graphics processing unit |
| CNN | – | Convolutional neural network |
| OCR | – | Optical character recognition |
| VAE | – | Variační autoenkódér |
| TTUR | – | Two Time-scale Update Rule |
| MLP | – | Multi layer perceptron |
| AMP | – | Automatic mixed precision |

# Seznam obrázků

# Seznam tabulek

# Kapitola 1

# Úvod

Strojové učení a umělá inteligence se v dnešním světě stávají stále důležitějšími nástroji v automobilovém průmyslu. Tyto technologie zde nalézají hojné využití. Příkladem využití strojového učení je zlepšení bezpečnosti, efektivity a zlepšení komfortnosti jízdy. Například od detekce různých kritických situací a jejich zabránění, až po celkové autonomní řízení vozidla. V některých případech, si hluboké neuronové sítě, dokonce vedou lépe než lidé. Jeden takový příklad lze uvést z medicíny. Například autoři práce *Development and validation of a deep learning algorithm for improving Gleason scoring of prostate cancer*[1] využili hluboké neuronové sítě pro určení *Gleason score*. Na validačních datech si tak síť vedla v průměru lépe, než vybraní experti z oboru. Lze tak předpokládat, že bezpečností systémy v autech, založené na strojovém učení, budou nedílnou součástí budoucích vozidel. Strojové učení, a tím hluboké neuronové sítě, se tak staly jedním z nejrychleji se rozvíjejících oborů s velkými skoky v rostoucí kvalitě a praktických využití výstupů, za krátkou dobu.

Hlavní úkol této práce je generovat noční obrazy na základě poskytnutého denního obrazu z kamery auta. Generované obrázky mohou být využity místo reálných snímků, například pro trénovaní jiných hlubokých neuronových sítí, které řeší již zmíněné úkoly. Získání kvalitních vstupních trénovacích dat pro neuronové sítě je stále jeden z největších problémů v oblasti strojového učení. Velkou množinu trénovacích dat vyžaduje například klasifikování objektů při různých světelných podmínkách, jiného zabarvení a různého prostředí. Takovou trénovací sadu je obtížné a časově náročné získat. V některých situacích může být nemožné získat tak velké množství reálných dat. Například stejný stav křižovatky, se stejnými účastníky vozovky, ale za jiných podmínek jako jsou roční období, počasí a čas. Jako řešení pro získání takové množiny vstupních dat se nabízí využití neuronových sítí, které dokáží vstupní obrázek modifikovat nebo vygenerovat úplně nový unikátní obrázek, dle zadaného referenčního obrázku.

V oboru zpracování obrazu je přenos stylu z referenčního obrázku na jiný obrázek dlouho řešený problém. S příchodem strojového učení, konkrétně hlubokého učení neuronových síti a takzvaných velmi hlubokých neuronových sítí, nastal zásadní obrat. Již nebyla potřeba ruční analýza a implementace různých metod pro získání příznaků z obrázku, které jsou zásadní pro modifikaci či

generování obrázků. Tento proces se dokáží hluboké neuronové sítě naučit samy. Jeden takový model neuronové sítě se tak dokáže natrénovat pro řešení různých obdobných problémů. Vzniklo tak obecnější řešení, které může být aplikováno na různé problémy.

Generativní adversariálních sítě (GANs), se prokázaly jako účinné řešení pro generování realistických obrazů. Jedna z aplikací GANs je také přenos natrénovaného stylu na vstupní obrázek, při zachování jeho obsahu. [2]

Tato práce se bude soustředit na problém přenesení stylu jednoho obrázku na druhý pomocí využití GANs. Nejdříve se věnuje důkladnému prostudování a popisu různých architektur GANs. Dále se bude zabývat vytvořením kvalitního trénovacího datasetu, který bude sloužit jako vstup pro trénovaní různých modelů založených na architektuře GAN. Na modelech pak budou spouštěny různé experimenty a ty budou důkladně popsány včetně různých problémů, které se během implementací a testování vyskytnou. V závěru této práce je srovnání experimentů a jejich výsledků pomocí vybraných metrik.

# Kapitola 2

# Modely

GANs jsou příkladem neuronových sítí využívajícího učení bez dozoru (Unsupervised Learning). To znamená, že nepotřebují anotovaná data k učení, které by jednoznačně určila, správné a nesprávné odpovědi. Jinými slovy, tyto sítě by měly dokázat řešit problémy, které nelze jednoduše vyjádřit jako pravda, nepravda a tím je správně či nesprávně klasifikovat. Například, u anotovaných trénovacích dat, kde je instance datasetu obrázek s objektem auta, by bylo snadné určit správnost odpovědi. U výstupu sítě stačí určit, zda se síť mýlí, když objekt klasifikuje jako auto či nikoliv. U problému generování obrázku však není tak jednoduché určit, zda je výsledek správný či nesprávný. U architektur založených na GAN je přidána druhá neuronová síť, která se učí hodnotit správnost výsledku. U konkrétního problému přenosu stylu z jednoho obrázku na druhý se objevuje pojem "doména obrázku". [3, 4, 5]

Například při řešení problému převodu obrázku ze dne na noc, jsou data rozděleny do domén den a noc. U některých architektur, vycházejících z GAN a využívající různé domény, je lze označit, jako sítě využívající učení s částečným dozorem.

Hlavní myšlenkou GANs jsou dvě neuronové sítě, a to generátor a diskriminátor, které mezi sebou soupeří. Ze základní architektury GANs však vzniklo více architektur, které se liší například počtem neuronových sítí, ztrátovou funkcí a z toho vyplývající výhody či nevýhody. Kromě řešení problému syntézy obrazů lze některé GANs využít také ke generování zvuku a textu. Vstupní data v takové neuronové sítí projdou řadou neuronových vrstev, které se učí postupně najít v datech různé vzory. Tyto informace se postupně komprimují až na pomyslný bod v takzvaném latentním prostoru. Generátor dokáže z jakéhokoliv bodu z latentního prostoru vygenerovat výsledek, který se statisticky obtížně rozeznává od reality. Na podobném principu fungují například předchůdci GANs a to variační autoenkódéry. Na rozdíl od variačních autoenkodérů z publikace *Auto-Encoding Variational Bayes od Kinga a Wellinga* [6] nedokáží generovat spojitě se měnící se výsledek. [6, 5]

Variační autoenkodér, který vstupní obraz kóduje do latentního prostoru pomocí průměru a rozptylu, dokáže postupným vzorkováním vygenerovat například postupně se měnící psané číslice, nebo postupně se měnící výraz ve tváři člověka. Typicky u GAN by každý takový vzorek nemusel být

vždy smysluplný a takový prostor nemusí být u ní spojitý. Na rozdíl od VAE, by se každý další vzorek v GAN architektuře mohl výrazně odlišovat od předchozího vzorku. [2]

Při výběru architektury, bylo také potřeba zvážit velikost výsledného modelu a tím jeho náročnost na čas a technické nároky na paměť při trénování. Tedy model se muselo podařit spustit alespoň na jedné z testovacích konfigurací. GANs jsou náročné hlavně na paměť, kde záleží primárně na velikosti dávky pro iteraci trénování takzvanou *batchsize* a velikosti modelu. Dále je v této sekci popsána teorie a metody k architekturám neuronových sítí, které jsou dále použity v experimentech. Teoretické poznatky pomohou pochopit závislosti vybrané architektury a ztrátové funkce pro sestavení modelu, který dokáže nejlépe splnit cíl práce, a to vygenerovat výstupní obrázek, který se bude co nejvíce držet vstupního obrázku. Vstupní obrázek by také měl být co nejblíže reálnému snímku, tedy kupříkladu by měl co nejlépe dodržovat základní zákony optiky a šíření světla po scéně.

## 2.1 Generativní adversariální síť

GAN sítě se skládají minimálně ze dvou neuronových sítí nazvaných jako generátor a diskriminátor. Myšlenka generátoru je taková, že existuje nějaký prostor bodů, kde může být jakýkoliv bod mapován na realisticky vypadající syntetický obraz. Takový prostor může představovat nízko dimenzionální tenzor (vektor) příznaků. Poté generátor dokáže takový bod dekomprimovat na výstup. Druhá síť, diskriminátor, se snaží rozeznat, zda vstupní obrázek byl reálný, nebo zda byl vygenerován generátorem. [5]

Práce se tedy bude zabývat takovou ztrátovou funkcí, aby se výsledné obrázky co nejvíce blížily obsahem původním vstupním obrázkům. Pro dosažení takového výsledku se v experimentech bude tedy dbát na správně nastavenou ztrátovou funkci, která bude síť vést k zachování rysů původního obrázku a zároveň přenesení chtěného stylu dle referenčního obrázku.

První architekturou je základní GAN z publikace [5]. Hlavní myšlenku lze předvést na dvou neuronových sítích, které mezi sebou soupeří a navzájem se snaží překonat tu druhou. Jedna síť se snaží generovat nějaký výstup a druhá síť se snaží rozeznat, zda byl výstup vygenerovaný, nebo skutečný. První síť se tedy snaží vygenerovat takový výstup, aby dokázala přesvědčit druhou síť, že se jedná o reálný vstup. Druhá síť se zase snaží zlepšovat v rozpoznávání vygenerovaného a reálného vstupu. Je tak vhodná pro generování dat, které nemusí existovat, ale mají stejnou distribuci jako skutečná vstupní data. Výsledek je pak obtížné statisticky rozeznat od skutečných dat. První síť je zvaná generátor a druhá síť diskriminátor (někdy také klasifikátor). [5, 2]

Generátor má jako vstup náhodný vektor, který je označen jako vstupní šum $p_z(z)$. Ten se učí pomocí nastavení vah neuronů distribuci $p_g$ na vstupních datech $x$. Generátor tak implementuje funkci $G(z, \theta)$, kde $G$ je funkce reprezentována neuronovou sítí s parametry $\theta$. Funkce $G$ tedy vrací vygenerovaný snímek dle vstupního šumu a nastavených parametrů $\theta$. [5]

Druhou neuronovou síť, zvanou diskriminátor, je definována jako $D(x, \theta)$. Vstupem je $x$ a neuronová síť představuje parametry $\theta$. Výstupem je skalární hodnota pravděpodobnosti, která předsta-

vuje odhadovanou hodnotu, zda vstup $x$ pochází ze skutečných dat, nebo jestli byl vstup vygenerován pomocí distribuce $p_g$ generátoru. Diskriminátor je trénován k minimalizování chyby odhadu predikce pomocí binární klasifikace na trénovacích datech a výstupu z generátoru $G$. Současně je trénován generátor pro maximalizování pravděpodobnosti, že jeho výstup diskriminátor určí jako skutečná data. To lze vyjádřit jako $log(1 - D(G(z)))$. Tedy generátor se snaží maximalizovat chybu diskriminátoru. [5]

Ztrátovou funkci celé sítě, zvanou $L_{gan}$, tak lze definovat na množině dat $pdata$ a náhodného vektoru $z$ jako následující funkci:

$$L_{gan}(G, D) = E_{x\sim pdata}[\log(D(x))] + E_z[\log(1 - D(G(z)))] \tag{2.1}$$

Termín $E_{x\sim pdata}[\log(D(x))]$ představuje střední hodnotu logaritmu výstupu diskriminátoru z reálných dat a termín $E_z[\log(1 - D(G(z)))]$ výstup diskriminátoru se vstupem z generátoru.

Diskriminátor se tedy učí tuto ztrátovou funkci minimalizovat a generátor maximalizovat. Dá se říci, že při optimalizaci s každým krokem gradientu se mění ta funkce, pro kterou hledáme minimum. Jedná se tedy o optimalizaci rovnováhy mezi generátorem a diskriminátorem.

Z toho lze určit ideální generátor, kde by diskriminátor nebyl schopný určit z jaké množiny dat výstupní obrázek pochází, tedy $D(x) = \frac{1}{2}$. Distribuce trénovacích vstupních dat $p_data$, by se tak rovnala distribuci výstupu generátoru $p_g$. Ideální stav generátoru $G*$ lze popsat tedy následovně:

$$G* = arg \min_G \max_D L_{gan}(G, D) \tag{2.2}$$

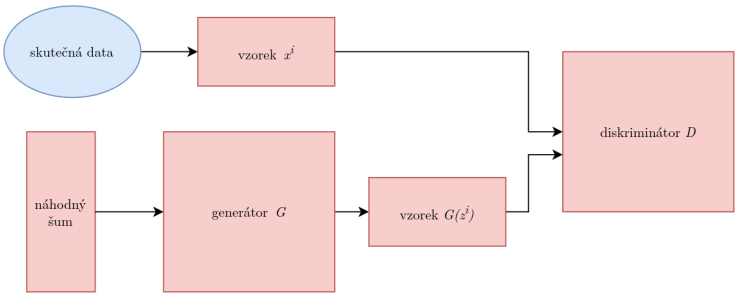
Při experimentování může nastat problém, když generátor $G$ generuje příliš nereálné výsledky a diskriminátor tak dokáže predikovat jeho výstupy s velkou pravděpodobností, tak funkce $\log(1 - D(G(z)))$ saturuje. Proto se využívá místo minimalizování funkce $\log(1 - D(G(z)))$ maximalizování funkce $\log(D(G(z)))$. V prvních fázích učení poskytuje větší gradient a síť tak rychleji konverguje ke správnému výsledku.

Na náčrtku 2.1 architektury GAN, kde lze vidět generátor G, který má jako vstup náhodný šum. Náhodný šum můžeme chápat obdobně jako u modelu enkodéru za latentní prostor. Diskriminátor má jako vstup data, která se snaží klasifikovat jako skutečná, nebo vygenerovaná generátorem.

Postup pro natrénování sítě je tedy iterovat trénovací množinu a pro každý prvek v iteraci spočítat stoupající stochastický gradient jako:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [logD(x^i) + log(1 - D(G(z^i)))], \tag{2.3}$$

kde $z$ je z množiny $\{z^1, ..., z^m\}$ o délce $m$ a $x$ je z dávky množiny vygenerovaných vzorků $\{x^1, ..., x^m\}$ také o délce m. Tímto gradientem změníme diskriminátor, takže je tento postup je použit při zpětné šíření chyby v neuronové síti.




Obrázek 2.1: Náčrt schématu GAN

Generátor změníme jeho klesajícím stochaistický gradientem:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [log(1 - D(G(z^i)))] \tag{2.4}$$

GAN sítě nejsou stabilní. Diskriminátor a generátor se učí současně. Když diskriminátor nedokáže dostatečně správně klasifikovat, výstupní generovaná data mohou obsahovat, například v případě obrázků, nechtěné artefakty. Když diskriminátor naopak dominuje, tak nastává také nechtěný výsledek sítě. Výstup generátoru totiž zkolabuje na pár podobných řešení, které nejvíce vyhovují diskriminátoru. Toto se děje u všech sítí založené na této architektuře. Například Pix2Pix by tak zkolabovala na pár řešení, které by se významně neměnila i dle vstupního referenčního obrázku.

Konvergence původních nemodifikovaných GANs nebyla stále v době psaní této práce dokázána. Publikace *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium* [7] navrhuje používat během procesu trénování pravidlo aktualizace ve dvou časových škálách, které zahrnuje aktualizaci generátoru a diskriminátoru různou rychlostí. Na základě experimentů a teoretické analýzy ukazují, že tato úprava vede ke stabilnějším GAN s lepším výsledkem. [7]

Sítě postavené na této architektuře se značí tím, že se učí transformaci vstupních dat na výstupní, ale také, jak již bylo naznačeno, se učí ztrátovou funkci, které se tím mění. Konstrukce je tedy výjimečnější o proti jiným tím, že okolí gradientu, po které se síť optimalizuje, se mění. Tedy je to obecnější řešení problému a nemusíme vždy řešit konkrétně ztrátovou funkci. [2]

Takové implementované sítě, například při transformací obrázků, se typicky jeví výsledné obrázky jako méně rozmazané, než jako při řešení pomocí jednodušších konvolučních neuronových sítí s klasickými ztrátovými funkcemi.

Generátoru a diskriminátoru můžeme dodat informaci z množiny $Y$, podle kterého jsou omezeny. Informace může být například název domény vstupních dat. Tuto informaci předáme generátoru a diskriminátoru jako další vstupní vrstvu, toto rozšíření se nazývá architekturou CGAN. [8]

Na technice GAN vzniklo mnoho různých modelů řešící různé typy problémů. GAN architektura se hojně využívá při problému transformace obrázku na obrázek. Má tedy zastupovat obecnější řešení mapování nějakých pixelů na jiné, místo specifického algoritmu, který umí například převést

barevný obrázek na obrázek šedi či naopak. Problém transformace vstupního obrázku na jiný lze řešit pomocí CNN sítě, které se snaží minimalizovat ztrátovou funkci, například Euklidovskou vzdálenost mezi vstupem a výstupem. Takováto ztrátová funkce je minimalizována pomocí průměrování všech možných výsledků a to má za příčinu to, že výsledky se jeví jako rozmazané. Místo toho síť má vlastně za úkol minimalizovat ztrátovou funkci určenou tím, že výsledek nemá být rozeznatelný od skutečného výstupu, místo nějakých specifických ztrátových funkcí.

Rozdíl mezi cGAN a GAN si také lze ukázat na příkladu natrénovaných sítí na trénovacím setu ručně psaných čísel. Jako vstup GAN je vždy pouze náhodný šum, nelze předem určit jaké číslo se má vygenerovat. U cGAN je jako vstup přidána podmínka. Ve zmíněném případě je to příznak představující číslo, které má být generováno. Je tak možné předem určit, jaké číslo se má vygenerovat. V případě úkolu transformace obrázku na obrázek je tato podmínka představována vektorem příznaků vstupního obrázku. První část generátoru tak dokáže ze vstupu vytáhnout důležité příznaky z obrázku a komprimovat ho na tento latentní prostor bodů. Generátor lze tak podobně jako u VAE rozdělit na pomyslné dvě části, a to enkódér a dekódér.

Dále se bude práce zabývat hlavně modely, které mají jako vstup obrázek z jedné domény a učí se ho transformovat na obrázek do druhé. Tedy konkrétně architektury, které vychází z CGAN.

## 2.2 Pix2Pix

Pix2Pix dle původní publikace *Image-to-Image Translation with Conditional Adversarial Networks* [3] je typ sítě, která implementuje již zmíněnou GAN, konkrétně cGAN z publikace *Conditional Generative Adversarial Nets* viz [8, 3].

Síť byla navržena k účelu řešení mapování obrázků z jedné domény do druhé. Jedná se o GAN síť s přidanou podmínkou jako vstup pro generátor a diskriminátor. Zatímco tedy GAN má jako vstup pouze náhodný šum, Pix2Pix má navíc jako vstup referenční obrázek. Díky tomu má síť více informací a většinou generuje lepší výsledky než GAN. Je vhodný například pro rekonstrukci map z obrysů, vybarvování obrázků v odstínu šedi, fúze obrázků ve viditelném světle a infračerveném záření a mnoho další problémů. Při použití kvalitních spárovaných datasetů dosahuje tato síť kvalitních výsledků, které jsou těžko vizuálně rozeznatelné od skutečných obrázků. [3]

Jak již bylo zmíněno, generátor má na rozdíl od základní GAN architektury jako vstup ještě navíc data $x$. Mapuje tedy vektor náhodného šumu $z$ a navíc data $x$ na výstup $G : \{x, y\} - > Y$. Umí se tedy naučit transformaci vstupního obrázku na výstupní. Architektura tedy řeší obecněji problém transformace jednoho obrázku na druhý bez nutnosti specifikovat ztrátové funkci pro různé řešení problému transformací obrázků. [3]

Definovaná ztrátová funkce se moc neliší od základní GAN 2.1 a je definovaná jako:

$$L_{cGAN}(G, D) = E_{x,y}[log(D(x, y))] + E_{x,z}[log(1 - D(G(x, z)))] \tag{2.5}$$

Změna je tedy ve vstupu generátoru $G$, který místo náhodného šumu přijímá jako vstup obrázek z množiny $X$, který se má transformovat na obrázek z množiny $Y$. Mírně se liší od schématu znázorněného na obrázku 2.1. Dle zdrojové práce viz [3] je dosaženo lepších výsledků v kombinaci s tradičnější ztrátovou funkcí, například $L1$ vzdáleností (Manhattanská metrika):

$$L_{L1}(G) = E_{x,y,z}[||y - G(x,z)||] \tag{2.6}$$

Ideální stav generátoru $G^*$ pak lze vyjádřit jako:

$$G^* = arg \min_G \max_D L_{cGAN}(G,D) + \lambda_{L1} L_{L1}(G) \tag{2.7}$$

Jedná se tedy o snahu minimalizovat ztrátu generátoru, maximalizovat ztrátu diskriminátoru a minimalizování ztráty $L_{L1}(G)$. Dle publikace [3] ztráta $L_{L1}$ dokáže velmi dobře zachytit nízké frekvence. GAN diskriminátor tak může být zaměřen na vysoké frekvence. Lze tak použít diskriminátor, který se soustředí místo celého obrázku na nějakého výřezy. Konkrétně je použit *PatchGAN* z publikace *Patch-Based Image Inpainting with Generative Adversarial Networks*, který z výřezů $N * N$ rozeznává, zda je výřez vygenerovaný, nebo skutečný. Generátor při trénování postupně snižuje váhu vlivu náhodného šumu $z$ na výsledku. Při trénování může náhodné vypadávání neuronů z různých vrstev tento šum nahradit. Při testování pak tento náhodný šum je spíše nechtěný, a ze vstupního obrázku je chtěný jeden možný výsledek. [3, 9]

Za předpokladu, že by síť měla kvalitní spárovaný dataset, by tato architektura měla dosahovat vyhovujících výsledků. Což je problém, protože jak bude dále vysvětleno viz 3.1.1, dostupný spárovaný dataset pravděpodobně není dostatečně vhodný jako vstup do této sítě. V tomto případě tak Pix2Pix architektura bude využita spíše k porovnávání s jinými architekturami a jako teoretický základ pro pochopení dalších a komplexnějších architektur.

## 2.3 CycleGAN

K hlavnímu úkolu této práce je potřeba vybrat takový model, který ve výsledku dokáže zachovávat prvky z reálného vstupu. Tento závěr bude v této práci ověřen a popsán z výsledků experimentů sítě Pix2Pix. Jedním z takových modelů je model založený na architektuře CycleGAN.

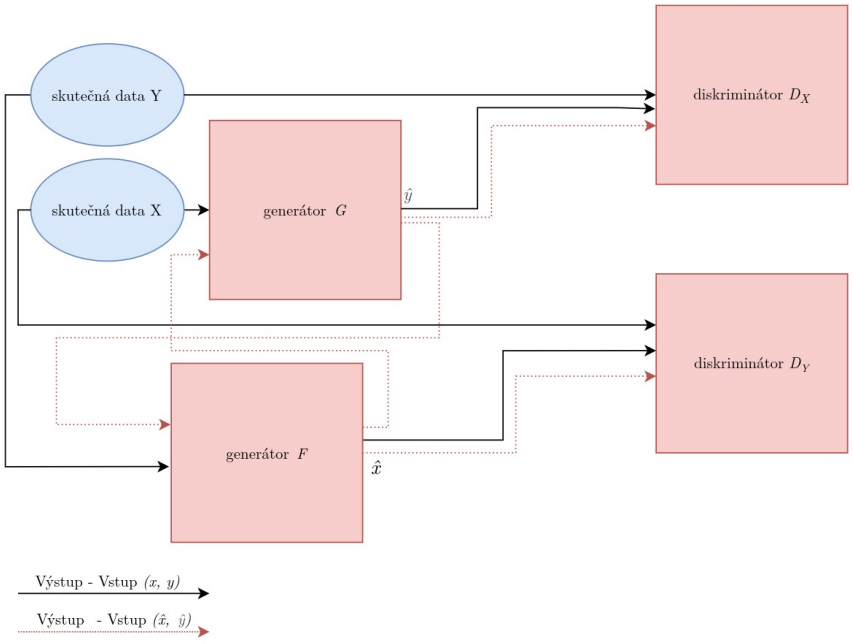
Hlavní rozdíl mezi Pix2Pix a CycleGAN je totiž takzvaná cyklická ztráta, díky ní, by CycleGAN experimenty mohly výsledky experimentů Pix2Pix výrazně překonat. CycleGAN totiž převádí obrázek z jedné domény do druhé a poté zase zpět. Výsledný obrázek by tak měl být v ideálním případě stejný jako vstupní. Síť tak udržuje cyklickou konzistenci, která je nejdůležitějším prvkem úspěchu této sítě. [4, 3]

CycleGAN z původní publikace *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks* [4] je architektura neuronové sítě, která řeší problém mapování obrázků z jedné

domény do druhé bez nutnosti mít jakkoliv spárované trénovací data. Její vhodné použíti je tak například přenos stylu z jednoho obrázku na druhý, kde na obrázcích jsou různé objekty. [4]

Jednoduchý příklad lze předvést na úkolu převodu satelitní mapy na mapu typickou pro navigaci, kde jsou zobrazeny jen cesty. Pro model založený na Pix2Pix architektuře by jako vhodný vstupní trénovací dataset měl pár satelitního snímku a navigační mapy. Tento pár by se měl nejlépe překrývat a každý pixel z jednoho obrázku na druhý by měl být takový, jaký by byl chtěný výsledný pixel. V případě sítě založené na CycleGAN architektuře by teoreticky měl vystačit takový dataset, kde by byl pár satelitního snímku a navigační mapy takový, že by se nepřekrývaly, a stále by jako výstup byl požadovaný výsledek. Tento model by tak měl být vhodnější a měl by mít lepší výsledky pro úkol transformace obrázku ze dne na noc ze snímků z auta. [4]

Síť CycleGAN je založená na GAN architektuře. V síti se tak nachází generátor $G$ a klasifikátor $D_Y$, navíc se v ní také nachází druhý generátor $F$ a druhý klasifikátor $D_X$. Nechť jsou dvě domény dat $X$ a $Y$. Dopředná propagace sítě tedy probíhá jako vstup obrázku z domény $X$ do generátoru $G$, který zobrazuje do domény $Y$. Výsledek generátoru $G$ je vstup pro generátor $F$, který zobrazuje do domény $Y$. V případě této práce je doména $X$ doména obrázků dne a doména $Y$ noci. Jinak řečeno, funkce $G$ zobrazuje množinu $X$ na množinu $Y$, což lze matematicky zapsat jako $G : X -> Y$. Naopak, funkce $F$ zobrazuje množinu $Y$ na množinu $X$, což lze zapsat jako $F : Y -> X$. Klasifikátor $D_Y$ zajišťuje, že funkce $G$ se učí mapovat $X$ do $Y$, tak aby výsledek klasifikátor $D_Y$ nerozeznal od $Y$. To samé platí pro klasifikátor $D_X$, který zajišťuje, že generátor $F$ správně mapuje opačný směrem. Na schématu 2.2 lze vidět popisované schéma sítě, která se liší od původní GAN 2.1. [4]




Obrázek 2.2: Náčrt schématu *CycleGAN*

Ztrátová funkce celé sítě se obdobně jako u tradičních GAN skládá z několika ztrátových funkcí. První část je ztrátová funkce mapování $G : X- > Y$. Tato ztrátová funkce vyjadřuje adversariální hru mezi generátorem $G$ a diskriminátorem $D_Y$. [4, 5]

Ztrátovou funkci lze zapsat jako:

$$L_{GAN}(G, D_Y, X, Y) = E_{y \sim pdata(y)}[log(D_Y(y))] + E_x[log(1 - D_Y(G(X)))], \tag{2.8}$$

Druhou částí celé ztrátové funkce je také ztrátová funkce $L_{GAN}$. V tomto případě jsou ale změněny parametry funkce na $L_{GAN}(F, D_X, Y, X)$. [4]

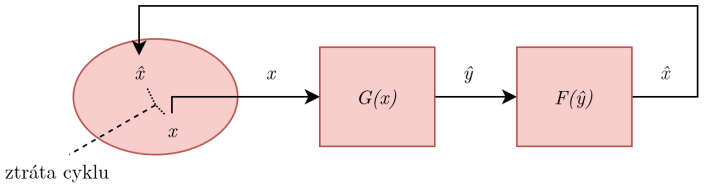
Další částí celkové ztrátové funkce je ztrátová funkce konzistence cyklu. Jak již bylo naznačeno, pro dosažení lepšího výsledku zobrazení se používá ztráta ze zobrazení, což lze zapsat jako

$$x- > G(x)- > F(G(x))- > \approx x$$

a obráceně pro zobrazení $Y$ do $X$ a zpět do $Y$. Ztrátu cyklu lze zapsat jako $||F(G(X)) - x||$ a je zobrazena na schématu 2.3. Tato ztrátová funkce se tak skládá ze dvou částí, tedy části kdy se obrázek mapuje z domény $X$ do domény $Y$ a obráceně. [4]

Celou ztrátu cyklu tak lze definovat jako:

$$L_{cyc}(G, F) = E_{x \sim pdata(x)}[||F(G(X)) - x||] + E_{y \sim pdata(y)}[||G(F(y)) - y||] \tag{2.9}$$




Obrázek 2.3: Ztráty cyklu

Ztrátová funkce celé sítě je tak spojení všech již zmíněných ztrátových funkcí a lze ji zaspat následovně:

$$L_{cycleGAN}(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda_{L_{cyc}} L_{cyc}(G, F) \tag{2.10}$$

Ideální stav generátorů je tedy výsledek hry maximalizování ztrátové funkce pro klasifikátory a minimalizování ztrátové funkce pro generátory, což lze zapsat jako:

$$G^*, F^* = arg \min_{G,F} \max_{D_X, D_Y} L_{cycleGAN}(G, F, D_X, D_Y) \tag{2.11}$$

U některých experimentů, kde barva je důležitou částí pro výsledný obrázek, bylo dosaženo lepších výsledků, když celková ztrátová funkce sítě byla rozšířena takzvanou ztrátu identity. Ztráta identity je definována jako zobrazení z jedné množiny do té stejné množiny. V tomto případě tak generátor $G$ přijme jako vstup $y$ z domény $Y$. Ztráta se pak vypočte jako vzdálenost mezi $y$ a vygenerovaným $\hat{y}$, což lze zapsat jako $||\hat{y} - y||$. Obdobně tak generátor $F$ přijme jako vstup $y$ z domény $Y$, zapsáno jako $||G(y) - y||$. [4]

Výsledná ztrátová funkce identity je tak definována jako:

$$L_{identity}(G, F) = E_{y \sim pdata(y)}[||G(y) - y||] + E_{x \sim pdata(x)}[||F(x) - x||] \tag{2.12}$$

Celkovou ztrátovou funkce navíc s přidanou ztrátovou funkcí identity tak lze zapsat jako:

$$\begin{aligned} L_{cycleGAN}(G, F, D_X, D_Y) = & L_{GAN}(G, D_Y, X, Y) \\ & + L_{GAN}(F, D_X, Y, X) \\ & + \lambda_{L_{cyc}} L_{cyc}(G, F) \\ & + \lambda_{L_{identity}} L_{identity}(G, F) \end{aligned} \tag{2.13}$$

Na základě teoretických předpokladů diskutovaných v této kapitole, lze předpokládat, že v případě nedostatečně kvalitně spárovaného datasetu, bude CycleGAN dosahovat lepších výsledků, než Pix2Pix. Při řešení úkolu transformace denního obrazu na noční obraz v kontextu pozemní komunikace je získání vhodného spárovaného trénovacího datasetu velmi obtížné. Problém přípravy datasetu je detailněji diskutován v kapitole 3.1.1. Vzhledem k dostupným testovacím konfiguracím 3.2.5, má ale CycleGAN značnou nevýhodou o proti Pix2Pix ve velikosti celkové sítě a tím také také delší trénovací proces. V případě experimentálnímu zkoušení změn koeficientů a jiných modifikací modelu je tak vhodné zvážit experimentování na datasetu s menším rozlišení obrázku, i za cenu možné ztráty informací a tím horšího výsledku.

## 2.4 Contrastive Learning for Unpaired Image-to-Image Translation

Další architektura, která se snaží řešit problém mapování nespárovaných obrázků z jedné domény do druhé je CUT, z původní publikace *Contrastive Learning for Unpaired Image-to-Image Translation* [10]. Tenhle problém řeší více architektur jako například DualGAN, DiscoGAN a již zmíněná CycleGAN. Všechny ale využívají nějakou formu ztrátové cyklické funkce. [11, 4, 12]

Generátor v takových to sítí pak připomíná bijekci, kde jeden vstup z domény $X$ má vždy jeden výstup v doméně $Y$. V publikaci CUT [10] nabízejí možnou alternativu. Hlavní myšlenkou je, že jakýkoliv výřez obrázku by měl být podobný výřezu ze stejné pozice stejného obrázku z jakékoliv jiné domény. Tedy například výřez, na kterém se nachází výřez auta ze dne, by se měl podobat stejným výřezům auta, jen v noci, či například za jiného počasí. Takové vzorky můžeme nazvat jako

pozitivní. Naopak, výřez silnice by se měl hodně lišit od prvního výřezu auta. Výřez silnice je tak označen jako negativní vzorek. [10]

Několikrát v této práci již bylo zmíněno, že generátor lze teoreticky rozdělit na dvě části, a to enkódér a dekódér. Tedy enkódér vytahuje z obrázku různé příznaky a mapuje ho na latentní prostor. Generátor pak tento prostor mapuje zpět na obrázek. Což lze zapsat jako 2.14:

$$\hat{y} = G(z) = G_{dec}(G_{enc}(x)) \tag{2.14}$$

Každý výřez z části enkódéru by měl tak vlastně odpovídat části výřezu obrázku. Přesněji řečeno, každá vrstva enkódéru a pozice v tomto souboru příznaků reprezentují výřez obrázku, kde hlubší vrstvy představují větší výřezy vstupního obrázku. [10]

Můžeme tak vybrat několik výřezů z různých vrstev enkódéru a porovnat je s výřezy z odpovídajících vrstev enkódéru jiného obrázku z druhé domény. Tyto výřezy se následně využijí jako vstup do *multi layer perceptron* (MLP) sítě, která se učí projektovat tyto výřezy do společného latentního prostoru. Následné porovnání výřezů můžeme provést pomocí kontrastivního učení, které následně vede k aktualizaci vah generátoru, takže se stává schopným provádět přesnější mapování mezi doménami. [10]

Kontrastivní ztráta, která se používá v CUT architektuře, je založena na Noise Contrastive Estimation. Jednotlivé vektory příznaků z výstupu sítě MLP jsou normalizované na jednotkovou kružnici. Pro každý pozitivní pár výřezů $x_i$ a $y_j$ z různých domén, tedy $x_i \in X$ a $y_j \in Y$, a pro $N$ negativních vzorků $z \in X$, je ztráta definována následovně:

$$l(x_i, y_j, z) = -\log \frac{\exp(x_i \cdot y_j)/\tau)}{\exp(x_i \cdot y_j)/\tau) + \sum_{n=1}^{N} \exp(x_i \cdot z_n/\tau)} \tag{2.15}$$

Suma v jmenovateli se provádí přes všechny negativní vzorky $z_n$. Parametr $\tau = 0.07$ představuje teplotní škálu, který škáluje vzdálenost mezi vzorky. Enkódér se tak učí nacházet a rozpoznávat vzory jako například silniční lampy, značky, billboardy a auta. Dekódér se učí mapovat různé styly patřící dané doméně. Například silničními osvětlení v doméně noci přidá zdroj světla.

Ztráta 2.15 je rozšířena pro více vrstev, pro různé výřezy na různých místech. Taková ztráta je nazvaná jako *LPatchNCE* a je definována jako:

$$L_{LPatchNCE}(G, H, X) = E_{x \sim X} \sum_{l=1}^{L} \sum_{s=1}^{Sl} l(\hat{z}_l^s, z_l^s, z_l^{\mathrm{S}\setminus\mathrm{s}}) \tag{2.16}$$

- $L_{LPatchNCE}(G, H, X)$ je funkce *LPatchNCE* s parametry na vstupu, kde $G$ je generátor, $H$ je MLP síť a $X$ je množina dat z jedné domény.
- $E_{x \sim X}$ je očekávaná hodnota (průměrná ztráta) přes všechny vzorky $x$ z množiny dat $X$.
- $l = 1, L$ je index vrstvy enkodéru. Zde $L$ značí celkový počet vrstev.

- $s = 1, Sl$ je index pozice ve vrstvě enkodéru. Zde $Sl$ značí celkový počet pozic ve vrstvě.
- $\hat{z}_l^s$ je výstup MLP sítě $H$ pro vrstvu $l$ a pozici $s$ z enkódéru $G_{enc}$ při zpracování generovaného obrázku $\hat{y} = G(x)$,lze tedy zapsat jinak jako $\{z_l\}_L = \{H_l(G_l^{enc}(G(x)))\}_L$.
- $z_l^s$ je výstup MLP sítě $H$ pro vrstvu $l$ a pozici $s$ z enkódéru $G_{enc}$ při zpracování skutečného obrázku $x$ z domény $X$.
- $z_l^{S\setminus s}$ je výstup sítě MLP $H$ pro vrstvu $l$ a pozici $s$ z enkódéru $G_{enc}$ při zpracování negativního vzorku ze vstupního obrázku $x$.

Celý tento postup lze vidět i na diagramu 2.4. Nejprve je vstup do generátoru $G$ obrázek z domény $X$ značený jako $x$. Výstupem z generátoru $G$ je pak vygenerovaný obrázek z domény $Y$ značený jako $\hat{y}$. Vstupem do sítě $H$ jsou různé tenzory příznaků z různých vrstev části generátoru $G_{enc}$. Na diagramu je uveden výřez z obrázku $\hat{y}$, který se porovnává s jedním výřezem $x$, který je pozitivní vzorek a s třemi výřezy, které zastupují negativní vzorky. Další součástí ztrátové funkce celé sítě je, jako u obdobných GAN, také diskriminátor $D$.

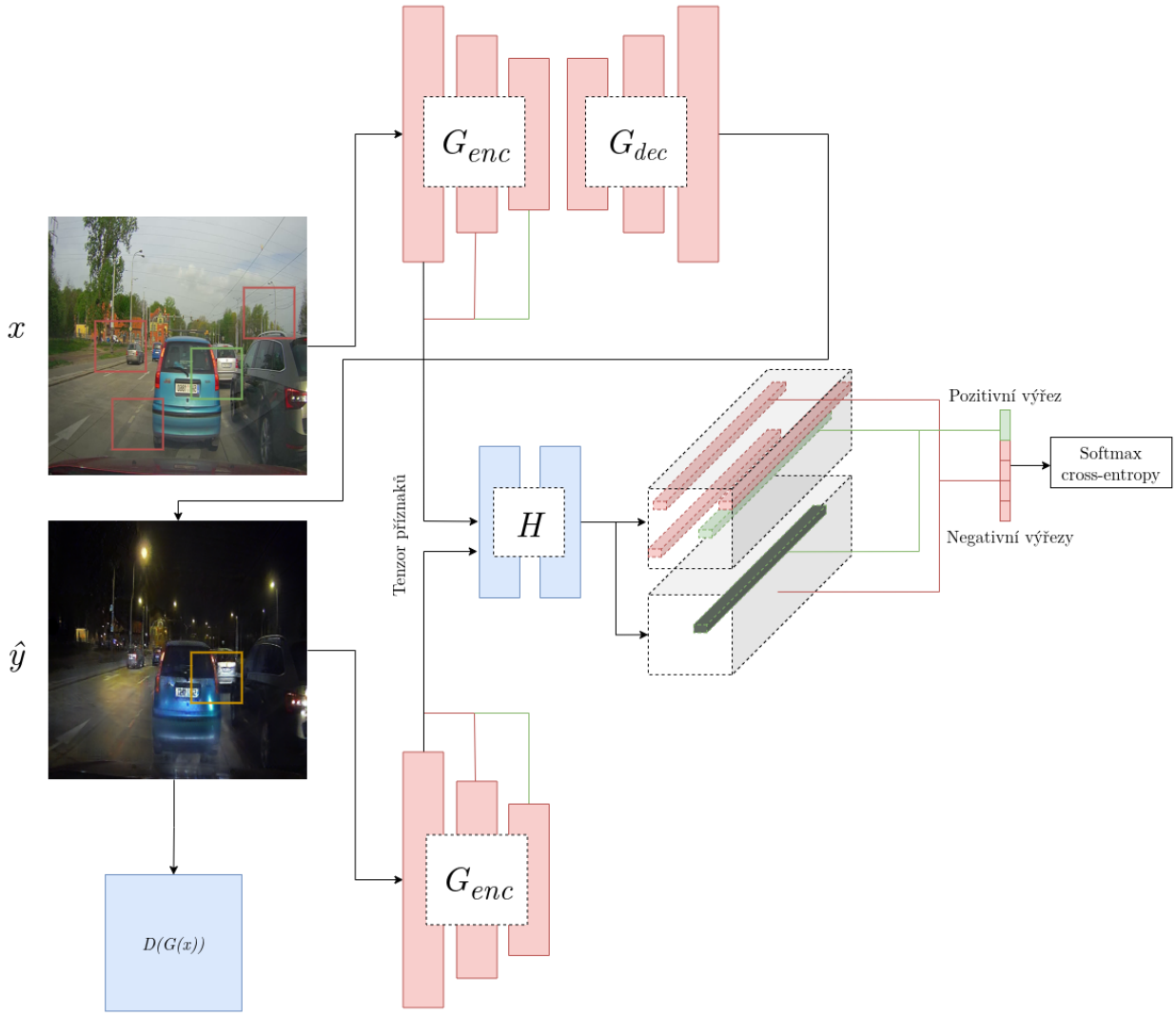
Negativní vzorky se teoreticky mohou brát i z jiných obrázků, tedy z celé množiny $X$. Dle publikace [10] však experimenty s obrázky, které toho využívaly, měly na výstupu horší výsledky. V této práci tak experimenty této sítě budou využívat pouze vzorky z konkrétního vstupního $x$. [10]

Jako u předchozích sítí, lze i k této přidat ke ztrátové funkci složku, která představuje ztrátu identity. Konkrétně lze zde použít $LPatchNCE(G, H, Y)$, což představuje již probíranou ztrátu výřezů z více vrstev s menší změnou. Tato ztrátová funkce totiž má jako parametr doménu $Y$. Generátor tak má jako vstup obrázek z domény $Y$, a ten se zase snaží projektovat do domény $Y$. Díky této ztrátové funkci by se tak síť měla naučit nedělat zbytečné a nechtěné změny na vygenerovaném obrázku $\hat{y}$. Ztrátovou funkci celé sítě CUT tak lze zapsat následovně jako:

$$L_{CUT}(G, D, X, Y) + \lambda_X L_{LPatchNCE}(G, H, X) + \lambda_Y L_{LPatchNCE}(G, H, Y) \tag{2.17}$$

Dle původní publikace [10] má síť CUT lepší výsledky než CycleGAN. Při doplnění $\lambda_Y = 0$ do ztrátové funkce 2.17 lze dosáhnout výsledků, které jsou vizuálně více podobné CycleGAN. Model postavený na takové ztrátové funkci je v publikaci nazván jako FastCUT a je to tedy odlehčená a rychlejší alternativa k síti postavené dle CycleGAN. [10]

V této práci tak je vhodné využít obou verzí této architektury a porovnat je s výsledky CycleGAN. CUT se tak skládá pouze z jednoho generátoru, diskriminátoru a malé MPL sítě. Oproti CycleGAN, které má dva generátory a dva diskriminátory je tak méně náročná na paměť a trénování. Kvalita vygenerovaných obrázků by se měla přibližovat a překonávat ty z experimentů ze CycleGAN.




Obrázek 2.4: Schéma *CUT* a *Patchwise contrastive Loss*

# Kapitola 3

# Implementace experimentů

## 3.1 Příprava vstupních dat

Jako u každé jiné neuronové sítě, je pro trénování GANs potřeba připravit dostatečně kvalitní dataset. Kvalita datasetu přímo úměrně ovlivňuje přesnost a robustnost modelu. Výběr zdrojových dat a následné generování datasetu je tak velmi důležitou části k vytvoření úspěšného modelu.

V oblasti počítačového vidění se často využívají veřejné databáze jako zdroj datasetu. V případě této práce však byla zadána zdrojová data, ze kterých se má dataset vytvořit. Pro tvorbu datasetu ze zdrojových dat existují různé nástroje. V tomhle případě ale byl nakonec napsán vlastní nástroj, který dokáže ze zdrojových dat vytáhnout potřebné informace k rozřazení do dvou domén, a to konkrétně dne a noci. Poté z těchto anotovaných dat vytvoří konkrétní datasety.

Vytvoření vlastního datasetu není jednoduché, a většinou tak vznikají modely, které sice dokáži plnit zadaný úkol, ale nejsou dostatečně robustní. Hodně záleží na konkrétní úloze. V tomto případě, kde se řeší scéna na pozemní komunikaci je velmi pravděpodobné, že zdroj dat z jednoho vozidla v jedné oblasti nebude dostatečný. Model natrénovaný na takovém datasetu si tak pravděpodobně na reálných datech nepovede příliš dobře. Nicméně výsledek takového modelu by mohl předurčit, zda má smysl se mu dále věnovat. Získání vlastního kvalitního a robustního datasetu totiž může stát hodně prostředků. V tomto případě by zdrojová data měly být z kamer různých typů vozidel a různých destinací.

Kvalitní dataset by měl splňovat hned několik kritérií. Aby se model naučil rozpoznávat a zobecňovat různé vzorce dat, instance dat v datasetu by měla být různorodá. Dále by měl být dataset dostatečně velký, aby bylo možné natrénovat komplexní problémy jako je transformace obrázků. Zde je bohužel možné narazit na omezení a to velikost dostupné paměti GPU. Dataset by měl být také očištěn od očividných chyb a anomálií, které by mohli zhoršovat kvalitu výsledku. I když bylo zmíněno, že jsou GANs považovány jako učení bez dozoru, je v našem případě potřeba rozdělit obrázky na dvě domény pro sítě vzniklé na myšlence cGAN, kde jsou vstupní obrázky, které mají přebrat styl z druhého obrázku. Pro trénování *GAN*, která má řešit problém transformace nespárovaných

| | Uložená Informace |
|---|---|
| 1 | Unikátní haš získaný z informací 2-7. |
| 2 | Den nebo noc. Hodnoty: D, N |
| 3 | Přední nebo zadní kamera. Hodnoty: *True*, *False* |
| 4 | GPS lokace. |
| 5 | Index na rozpoznání duplicitní GPS Lokace. |
| 6 | Název souboru videa. |
| 7 | Index snímku ve videu. |

Tabulka 3.1: Uložené informace o snímcích

obrázků je vhodné si připravit dataset minimálně o velikosti 1000 instancí z jedné domény. Taková velikost datasetu je daleko od ideálního, ale měl by dostačovat ke konvergenci sítě k nějakým uspokojivým výsledkům ze kterých by již bylo možné dělat nějaké závěry. Architektury pro spárované datasety, jako je Pix2Pix, by obvykle pravděpodobně mohly dosahovat požadovaných výsledků už při menších datasetech, ale zase je těžší získat.

Zdroj vstupních dat je z videí, kde je potřeba dále data vybrat náhodně, aby se omezily velmi podobné instance datasetu. Vhodné je také zvážit využití různých technik pro vylepšení trénovacích dat jako je prokládání šumem a geometrické transformace. V tomto případě, kdy je dataset použit pro více architektur, je těmihle technikami dataset vylepšen v implementacích při jeho načítání. Je to kvůli tomu, že některé sítě používají techniky, které samy o sobě využívají například náhodné oříznutí obrázku.

Zdroje videí jsou záznamy z videokamer umístěných na předním a zadním skle auta. V obraze jsou napsány čas a GPS souřadnice. Videa jsou ve formátu MP4 v celkové velikosti 768 GB a přibližně 130 hodin záznamu. Ze zadaných videí je potřeba si připravit obrázky pro datasety. Vstupní data jsou zpracována skriptem, který projde všechna zadaná videa z daných složek a uloží ke každému snímku z videa informace do souboru.

Uložené informace v souboru lze vidět v tabulce 3.1.

Příklad již konkrétních uložených informací do souboru lze vidět v příkladu 3.1.

```
a79934b7fb35607327ea860af40b293f:D:True:49.823477778:18.21990277776:0:2018
    _1106_062930_015F.MP4:3
a698893b429e7726724b815732ef4d7d:D:True:49.823477778:18.21990277776:1:2018
    _1106_062930_015F.MP4:5
85b99a6d0c00e94ef2b072501c91eb29:D:True:49.823477778:18.21990277776:2:2018
    _1106_062930_015F.MP4:2
```

Kód 3.1: Příklad z uložených informací ze souboru

GPS souřadnice jsou získány pomocí optického rozpoznávání znaků, konkrétně je použita Pytesseract OCR. Skript nejprve ořeže část snímku, kde se text nachází. Daný výřez se pomocí techniky

prahování binárně rozdělí na černé a bíle pixely. Skript poté spustí OCR s argumenty nastavení čtečky a obrázkem. Při volání OCR je skript paralelní pro rychlejší zpracování dat.

### 3.1.1 Spárovaná data

Spárovaná data jsou připravena jako vstup do sítě Pix2Pix viz kapitola 2.2, a také pro pro testovací set dat, na kterém jsou měřeny metriky. Data jsou nejprve spárovaná dle lokace souřadnic GPS a orientace auta. Skript načte data ze souboru viz 3.1 a vypočte pro každý řádek *hash* pro vyhledání druhého do páru. *Hash* se poskládá viz 3.1, vyjma informaci o "Den nebo noc", kde se použije opačná hodnota. Obrázek se pak získá ze snímku videa, dle informací získaných ze záznamu s konkrétní hodnotou *hash*. Samotné vyhledávání tak má asymptotická složitost $O(N)$. Pro lepší výsledky kontrastu páru, se ještě kontroluje, zda je rozdíl průměru hodnot ořezaných snímků větší než daná konstanta. Daný výpočet je implementován dle vzorce 3.1 ,

$$diff_{mean} = \left| \frac{\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_1(m,n)}{M * N} - \frac{\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_2(m,n)}{M * N} \right| \tag{3.1}$$

Spárovaná data jsou tedy spárovaná pouze dle polohy GPS a objekty na scéně obrázku souhlasit nemusí. Je tedy trochu v otázce, zda se jedná doopravdy o kvalitní spárovaný dataset. Správný spárovaný dataset by měl být takový, že na každém páru je přesně v referenčním obrázku chtěný výsledný pixel. Tento problém je dále řešen v závěru experimentů, které tento dataset využívají. Příklad fotek z párovaného datasetu lze vidět na obrázku 3.1.

Obrázek 3.1: Příklad spárovaných dat

### 3.1.2 Nespárovaná data

Nespárovaná data se generují obdobně jako spárovaná. Na rozdíl od spárovaných dat se pár nespojuje dle lokace. Jedná se tedy o dvojice den a noc, kde nelze určit nějaký klíčový bod, který by šlo

namapovat na druhý obrázek z páru. Z těchto dat je vygenerován dataset, kde bylo náhodně ze všech videí pro každou doménu vybráno 2000 snímků určené k natrénování sítě a 125 k testování. Pro správné otestování modelu je důležité vybrat testovací data mimo trénovací množinu. Dataset nebyl plně ručně probírán. Objevily se zde obrázky, které byly rozmazané, nebo byly vytvořeny za zhoršených viditelných podmínek. V ideálním případě by dataset měl mít zdroje dat z více vozidel a prostředí. Nejlepší modely mají datasety velikosti mnohem větší například řádově i v milionech. Jako příklad často využívaného zdroje takového datasetu je ImageNet [13]. V tomto případě by však dataset o velikosti tisíců měl pro experimenty vyhovovat a síť by měla generovat dostatečné výsledky, ze kterých by šlo různé experimenty porovnávat a dělat různé závěry. Příklad nespárovaných dat lze vidět na obrázku 3.2.

Obrázek 3.2: Příklad nespárovaných dat

## 3.2 Použité metriky pro srovnání výsledků

Další nelehký úkol při generování realistických obrázků pomocí GANs je kvantifikování správnosti výsledků. Pro porovnávání úspěšnosti výsledků byly vybrány čtyři různé metriky. Kombinací více různých metrik se dosáhlo toho, že celkové hodnocení se přibližuje subjektivnímu seřazení výsledků dle člověka. Tyto metriky jsou vyhodnocovány na všech experimentech na stejném spárovaném testovacím datasetu obrázků ve všech provedených experimentech. Jak už bylo zmíněno, spárovaný dataset nemá na scéně stejné objekty, tedy skóre je o něco nižší než jakého by dosahovali na některých veřejně dostupných datasetech. Je tedy důležité brát v úvahu fakt, že se výsledky poskytují relativní srovnání mezi uvedenými experimenty uvedené v této práci, avšak pravděpodobně nejsou vhodné na přímé srovnání s jinými veřejně dostupnými experimenty a jinými publikacemi s jinými datasety.

### 3.2.1 Fréchet Inception Distance

FID z publikace *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium* [14] je metrika pro měření podobnosti dvou datasetů. Podle studií [14] se ukázalo, že je silná korelace mezi touhle metrikou a lidským posouzení obrazu. Často se využívá právě pro evaluaci výsledků *GAN*. Je tedy vhodným kandidátem pro porovnání výsledků vygenerovaných snímků, kde by měl rozdíl mezi výslednými obrázky různých experimentů dosahovat hodnocením člověka. Konkrétně je použita varianta využívající implementace knihovny Pytorch [15]. FID se vypočítává výpočtem Fréchetovy vzdálenosti mezi dvěma Gaussovy křivkami přizpůsobenými k reprezentaci prvků neuronové sítě *Inception* sítě, viz publikace *Going Deeper with Convolutions* [16]. [4, 14]

### 3.2.2 L1 loss

L1 loss metrika počítá průměrnou absolutní chybou mezi $X$ a $Y$ podle rovnice

$$L1_{loss}(X,Y) = \frac{\sum_{i=1}^{m} |X_i - Y_i|}{m}, \tag{3.2}$$

kde m je velikost datového setu.

### 3.2.3 L2 loss

L2 loss metrika zobrazuje průměrný rozdíl na druhou mezi $X$ a $Y$ podle rovnice

$$L2_{loss}(X,Y) = \frac{\sum_{i=1}^{m} (X_i - Y_i)^2}{m}, \tag{3.3}$$

kde m je velikost datového setu. L2 je citlivější na rozdíly a je tak méně odolný proti šumu než L1 3.2.2.

Tabulka 3.2: Testovací konfigurace 1

| Parametr | Hodnota |
|---|---|
| Procesor | AMD Ryzen 5 2600 Six-Core Processor |
| RAM | 32 GB |
| GPU | NVIDIA GeForce RTX 3070 8 GB |

Tabulka 3.3: Testovací konfigurace 2

| Parametr | Hodnota |
|---|---|
| Procesor | AMD Ryzen 7 5800H |
| RAM | 32 GB |
| GPU | NVIDIA GeForce RTX 3060 6 GB |

### 3.2.4 Structural similarity index

Další vhodnou metrikou pro porovnání výsledků GAN sítě je *Structural similarity index* (zkráceně SSIM) z původní publikace *Image quality assessment: from error visibility to structural similarity* [17]. SSIM je definována jako poměr tří složek: luminance (jasu), kontrastu a strukturální podobnost. Luminance se používá k vyjádření jasu každého pixelu, kontrast ukazuje, jak se jednotlivé pixely liší v rámci obrazu a strukturální podobnost se snaží popsat, jak se jednotlivé obrazové bloky vzájemně podobají. Index je tedy vypočítán dle vzorce 3.4, kde $\alpha > 0$, $\beta > 0$ a $\gamma > 0$.

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = [l(\mathbf{x}, \mathbf{y})]^{\alpha} \cdot [c(\mathbf{x}, \mathbf{y})]^{\beta} \cdot [s(\mathbf{x}, \mathbf{y})]^{\gamma} \tag{3.4}$$

Tato metrika je hlavně v této práci využívána k porovnání strukturální podobnosti dvou obrázků a v některých experimentech je implementována jako část ztrátové funkce.

### 3.2.5 Testovací konfigurace

Pro trénování a testování byli použity dvě konfigurace, testovací konfigurace 1 (*desktop*), kterou lze vidět v tabulce 3.2 a testovací konfigurace 2 (*notebook*) v tabulce 3.3.

## 3.3 Průběžné sledování výsledků

Trénování GAN sítí je obtížné a časově náročné. Zároveň GANs jsou nestabilní a jejich implementace není jednoduchá. Experimenty tak často skončí jako neúspěch a lze to vidět již v prvních pár epochách. Například při špatné implementaci modifikace sítě v experimentu došlo k tomu, že gradienty se postupně snižovaly až zanikly. Na výstupu pak během trénování nevznikal žádný výstupní obrázek. Je tak vhodné vybrat i kvalitní nástroje pro sledování průběhu trénování a ověřování kvality

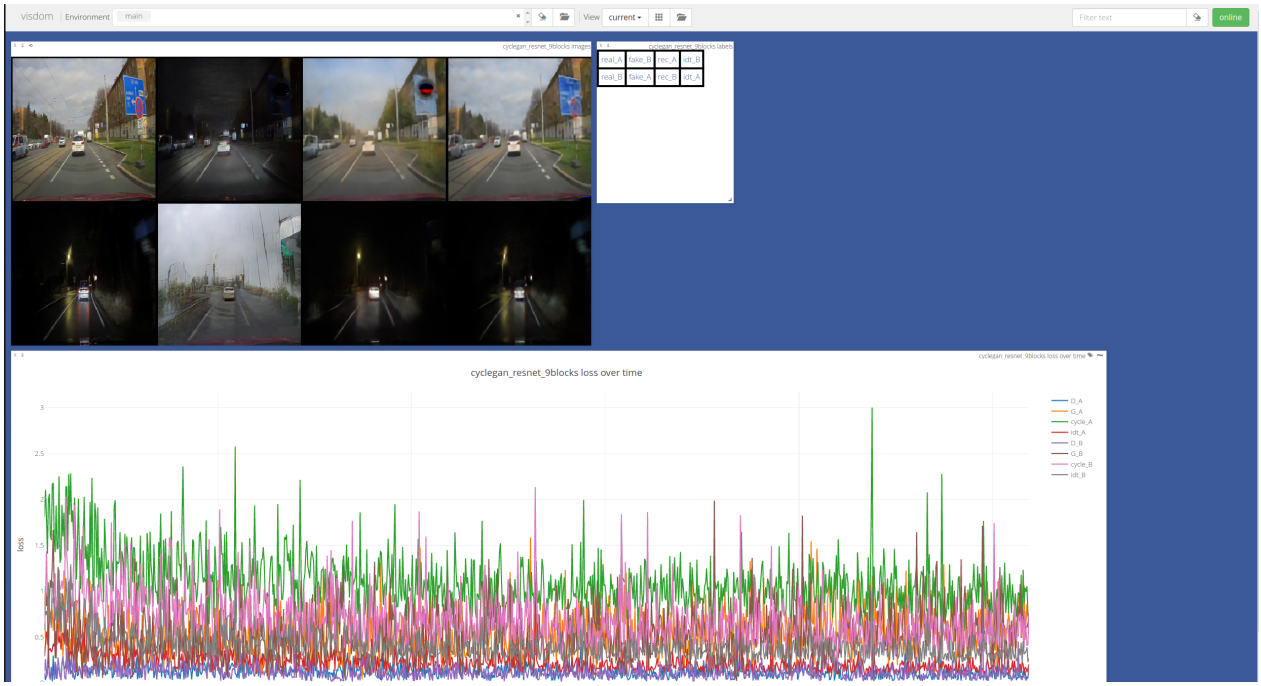
výsledků. Část nepovedených experimentů tak lze zachytit dříve, než skončí celý trénovací proces sítě.

### 3.3.1 Visdom

Prvním využitým nástrojem byl Visdom. Visdom je open-source projekt [18], který umožňuje průběžné sledování průběhu trénování neuronových sítí. Lze zde tak sledovat grafy ztrátových funkcí všech částí sítě, přesnosti a dalších metrik. Také lze vizualizovat výstupy sítě, tedy v tomhle případě lze vidět vygenerované obrázky.

Visdom lze přidat k projektu jako Python knihovnu. Díky tomu ho lze ho snadno integrovat do implementace experimentů pomocí jeho Python API. Visdom funguje jako lokální server, který běží na výchozí lokální adrese, například http://localhost:8097. Poté stačí jednoduše přistupovat k vizualizacím a metrikám prostřednictvím webového prohlížeče.

Na obrázku 3.3 lze vidět příklad GUI Visdom při trénování jednoho experimentu modelu postaveného na CycleGAN.




Obrázek 3.3: Prostředí Visdom

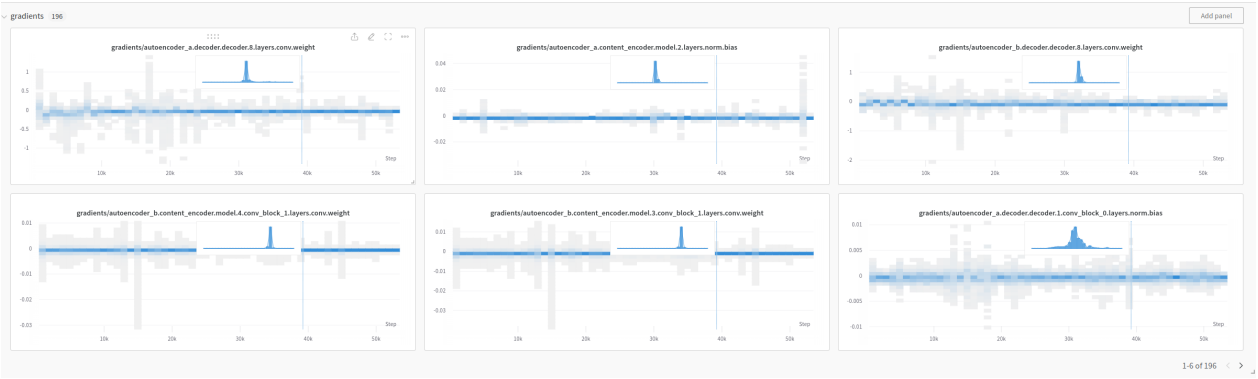
### 3.3.2 Wandb

Weights & Biases (Wandb) ze zdroje [19], je velmi užitečný nástroj pro sledování různých metrik sítě v průběhu trénování. Wandb poskytuje snadno použitelné API pro Python, který umožňuje

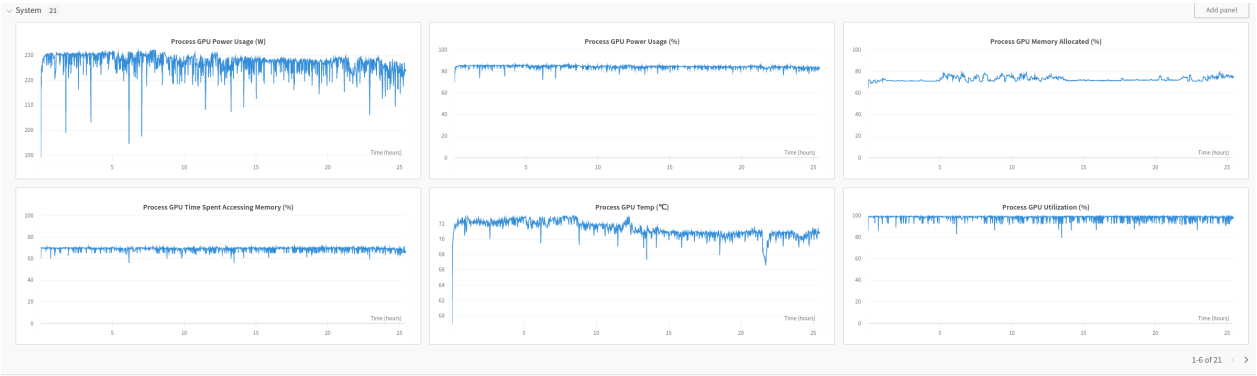
zaznamenávat metriky, hyperparametry, výstupy modelů, generované obrázky a další informace související s tréninkem.

Wandb poskytuje i webovou platformu `https://wandb.ai/`, kde si lze založit účet. Při spuštění experimentu pak stačí zadat autorizační *token* poskytnutý pomocí webové platformy a experiment se tak propojí s uživatelským účtem. Na webové platformě poté lze sledovat i již spuštěné a ukončené experimenty.

V drtivé většině experimentů se tak v této práci využil nástroj Wandb pro sledování průběhu trénování sítě. Nespornou výhodou je fakt, že je velmi jednoduché zakomponovat tento nástroj do implementace experimentu. Další výhodou je jednoduché sdílení přes webovou platformu, tedy k výsledkům lze jednoduše přihlížet i z jiných zařízení. Poslední výhodou je velmi široký výběr z různých grafů a vizualizací. Například na obrázku 3.4 lze vidět vizualizaci gradientů při jednom z experimentů sítě UNIT. Další užitečnou vizualizaci lze vidět na obrázku 3.5, kde je vizualizace systémových informací testovacího zařízení, konkrétně 3.2, v průběhu trénovacího procesu CycleGAN experimentu.




Obrázek 3.4: Vizualizaci gradientů v prostředí Wandb




Obrázek 3.5: Vizualizace informací o testovacím prostředí v prostředí Wandb

## 3.4 Pix2Pix

Pix2Pix, jak již bylo zmíněno, není dle vlastností této architektury nejvhodnější volbou pro řešení hlavní úlohy této práce. Vstupní trénovací spárovaný set je spárován jen díky poloze a objekty ve scéně se různí a není tak perfektní pro učení s dozorem. Výsledek experimentu je použit pro porovnání s ostatními experimenty.

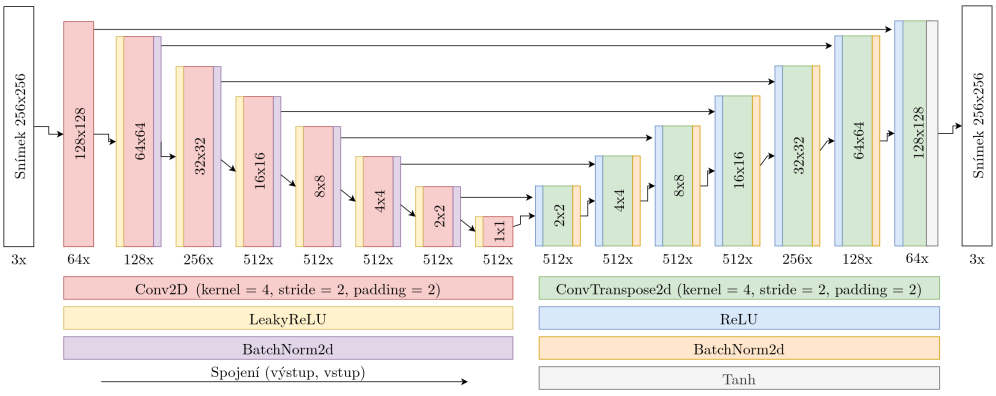
### 3.4.1 Implementace

Implementace se skládá tedy z generátoru $G$ a diskriminátoru $D$. Konkrétně je řešení implementováno v programovacím jazyce *Python* a ve *frameworku PyTorch*. Pro experimenty byl využit kód z repozitáře [20], který byl modifikován a dále rozvíjen.

### 3.4.2 Architektura generátoru

Generátor je U-Net z *U-Net: Convolutional Networks for Biomedical Image Segmentation* [21] síť, která, jak již název publikace napovídá, byla původně vyvinuta jako nástroj pro biomedicínskou segmentaci obrazu. Síť překonává v době vydání publikace nejlepší techniku posouvaného okna konvoluční sítě. [21]

U-Net lze také pomyslně rozdělit na *enkodér* (kontrakční část) a *dekodér* (expanzní část). Enkodér vzorkuje a komprimuje vstup až na vektor, který dekodér vzorkuje a dekomprimuje na výstupní obrázek. Navíc má přeskoky mezi bloky vzorkování komprimace a dekomprimace stejné velikosti. Expanzní vrstva tak má globálnější informace rysů a příznaků z nižší vrstvy a také více informací z přeskoku. Na obrázku 3.6 lze vidět podrobně jednotlivé vrstvy architektury. Ve schématu jsou uvedeny názvy vrstev z *PyTorch framework*. [21]

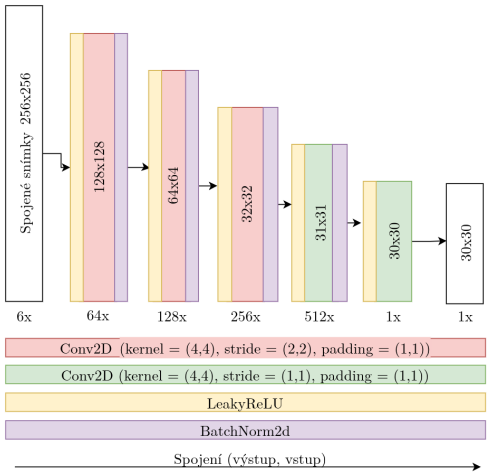



Obrázek 3.6: Schéma U-Net 256

### 3.4.3 Architektura diskriminátoru

Diskriminátor využívá místo tradiční konvoluční neuronové sítě architektura PatchGAN, jak již bylo zmíněno v kapitole modelu Pix2Pix 2.2. V implementaci jsou vstupem pro diskriminátor spojené dva obrázky do jednoho tenzoru, a to buď pár reálný den a vygenerovaná noc, nebo pár reálný den a reálná noc. [3, 9]

Na obrázku 3.7 je schéma sítě PatchGAN.




Obrázek 3.7: Schéma PatchGAN

Ve schématu jsou uvedeny názvy vrstev z *PyTorch framework*

### 3.4.4 Experimenty

Síť *Pix2pix* má výhodu v tom, že o proti následujícím modelům je jednodušší, méně náročná na paměť a lze ji tak rychleji trénovat na dostupných testovacích prostředí. Například lze zvětšovat *batchsize*, nebo rozlišení snímku. Tato volitelná nastavení tak pomohli zlepšit konvergenci sítě k požadovanému výsledku. První experiment byl co nejjednodušší a spuštěn na základní Pix2Pix síti bez nějakých větších úprav. Díky kratší době trénování je tato metoda nejvhodnější pro experimentování s různými datasety nebo změnami ztrátové funkce sítě, i když v porovnání s ostatními nemusí dosahovat nejlepších výsledků. Tyto poznatky lze nicméně přenést i na další experimenty s jinými modely.

Při průběhu trénování byly zapisovány informace o průběhu ztrátových funkcí a průběžně byly ukládány výsledky sítě. Tyto informace byly zobrazeny pomocí vizualizačního nástroje *Visdom*. Díky tomu bylo možné průběžně důkladně sledovat výsledky sítě a tím identifikovat různé artefakty a nedostatky experimentu i v brzkých epochách trénování sítě.

Při trénování se objevilo několik problémů, které negativně ovlivňovaly výsledný obrázek. Konkrétně se na výsledcích na spodní části obrázku objevovaly artefakty, připomínající původní text

dokumentujících GPS lokaci. Vzhledem k tomu, že spodní část obrázku neměla pro výsledný výstup žádný význam, bylo rozhodnuto o řešení tohoto problému oříznutím těchto nežádoucích částí obrázků. Po provedení této úpravy se artefakty již v dalších výstupech neobjevovaly a výsledky se tak dále přiblížily chtěnému výsledku. Tento postup byl proveden na všechny datasety a experimenty.

Během sledování průběhu dalších experimentů s původním datasetem, byl zpozorován dalším problém ve výstupu sítě. Výstupem sítě byly často snímky, kde osvětlení vozidel neodpovídaly reálným situacím. Projevovalo se to tak, že vozidla v proti jedoucím pruhu měla často místo předních světelných reflektorů světla, která připomínala výstražná světla obvykle umístěna na zadní straně vozidel. Naopak auta na výsledném obrázku jedoucí ve stejném směru před kamerou tak měli přiřazené osvětlení, které spíše připomínalo přední světelné reflektory vozidel, místo obvyklých výstražné barvy. Síť tedy v tomto experimentu nebyla schopna rozpoznat korelaci mezi přední a zadní stranou vozidla a vhodným dosazením osvětlení.

Pro řešení tohoto problému bylo navrženo několik variant. Nastavení více epoch pro trénování by mohlo pomoci. V tomto případě však při delším trénování více než 400 epoch pro jeden dataset se začal objevovat problém *overfitting*, a tak toto řešení bylo zavrženo. Obecně mají GAN sítě problém zvaný "vanishing gradient", tedy skoky, které se každou iterací snaží přiblížit minimu funkce, se zmenšují. Tyhle dva problémy se objevují za předpokladu, že síť nemá žádné další modifikace, které by tomuhle mohli zabránit, které budou probrány dále. Další možností bylo zvětšení velikosti instancí datasetu. Toto řešení však narazilo na další problém a to omezení testovacího prostředí. V dalších experimentech by tak bylo možné, že i při menších velikostí *batchsize* a menších velikostí vstupů by nebylo možné experiment spustit.

Další alternativní řešení bylo vylepšení datasetu nebo jeho zvětšení či zvolení komplexnější architektury sítě. Obvyklé vylepšení datasetu se využívá například různé ořezy, šum a další podobné techniky.

V případě řešení transformace obrazu na noční scénu je vhodné brát v úvahu fyzikální principy přenosu světla ve scéně. Například odraz světelného paprsku, vychází z předpokladu, že se světelný paprsek odráží od povrchů podle zákona odrazu a úhel odrazu je tak roven úhlu dopadu. Ve scéně tak záleží na pozici zdrojů světla. Některé zdroje světla, jako jsou lampy silničního osvětlení, se ve snímcích vyskytují a tak tato informace ve snímku je. Síť by si korelaci mezi pozicí takového zdroje světla a osvícení materiálu dokázala naučit. V případě ořezaných snímků by se však ztratila informace ohledně pozice hlavního zdroje světla v noční jízdě vozidla, a to pozice předních reflektorů vozidla. Přední reflektory vozidla nejsou na snímcích vidět a jejich odvození polohy je vždy závislé na pozici kamery. Problém může být u vozidel, které využívají technologii systému adaptabilních světlometů automaticky regulujících osvětlení v závislosti na aktuálních podmínkách. Pro reálnější simulaci scény by tak bylo potřeba více informací, než jen obraz z kamery, nebo mít dataset a testovací data takový, kde by na těchto dalších informacích nezáleželo. Nakonec bylo zvoleno řešení vyloučení snímků ze zadní kamery auta a využití pouze snímků zachycených přední kamerou. Dále by bylo vhodné nevyužívat vylepšení datasetu pomocí náhodného oříznutí.

V tabulce 3.4 jsou uvedeny důležité vstupní parametry použité při spuštění experimentů. První experiment byl tedy spuštěn s výchozími parametry typické pro Pix2Pix síť. Další experiment byl spuštěn s předem natrénovaným generátorem, tím byla počáteční epocha 400 a byla trénována dalších 300 epoch. Generátor byl předem natrénován na podobném datasetu převodu snímku ze dne na noc, kde se ale jednalo o jiné scenérie než jízda autem. Předen natrénovaná síť byla trénována na datasetu *day2night* a je získán z odkazu viz [4], kde byl pouze dostupný natrénovaný model generátoru. To už nebyl předem natrénovaný, diskriminátor se tak negativně projevuje na výsledku a celkovém průběhu trénování sítě. Sítě nedosahovaly požadovaných výsledků. Konkrétně scéna se změnila v noční, ale problémem bylo hlavně zachování původních objektů. Dle předpokladu by měl být hlavní problém u objektů jako jsou auta, které se ve scénách trénovacího páru lišily. Ale navíc sítě produkovaly výsledky, kde vozovka a okolní budovy ze vstupního reálného výsledku chyběly, nebo nebyly dostatečně podobné.

Na tento problém se naskytovalo řešení úpravy koeficientů ve ztrátové funkci čí přidání dalšího kritéria do ztrátové funkce. Cílem je upravit ztrátovou funkci tak, aby výstup ze sítě více zachovával původní obrysy objektů. Pro zachování struktury objektů byla nakonec vybrána metrika SSIM a přidána do ztrátové funkce. Experimenty, které využívají rozšířenou ztrátovou funkci o SSIM mají v názvu sufixem _ssim. V implementaci byla využita funkce structural_similarity, dále jako funkce *ssim*, z knihovny *skimage.metrics*. Kód pro definování vlastní ztrátové funkce v Pytorch lze vidět v ukázce kódu 3.2. Funkce *ssim* má jako vstup dva tenzory, kde první tenzor byl reálný obrázek ze vstupu sítě, a druhým parametrem byl vygenerovaný obrázek z generátoru.

```
class SSIMLoss(torch.nn.Module):
    def __init__(self):
        super(SSIMLoss, self).__init__()

    def forward(self, realA, fakeB):
        return ssim(realA, fakeB, data_range=1, multichannel=True)
```

Kód 3.2: Definice SSIM Pytorch ztrátové funkce

Funkce *ssim* vrací hodnoty v rozmezí $(0, 1)$, kde 0 znamená že se vstupy velmi liší a 1, že jsou vstupy identické. Výstup funkce *ssim* tak bylo nutné ještě upravit dle jednoduchého vzorečku $1 - ssim$. Celková ztrátová funkce sítě tak lze zapsat jako vzorec 3.5.

$$\begin{aligned} L_{Pix2Pix}(G, D) = & E_{x,y}[\log D(x, y)] \\ & + E_{x,y,z}[\log(1 - D(x, G(x, z)))] \\ & + \lambda_{L1} * E_{x,y,z}[|y - G(x, z)|] \\ & + \lambda_{SSIM} * E_{x,y,z}[1 - SSIM(y, Gx, z)] \end{aligned} \tag{3.5}$$

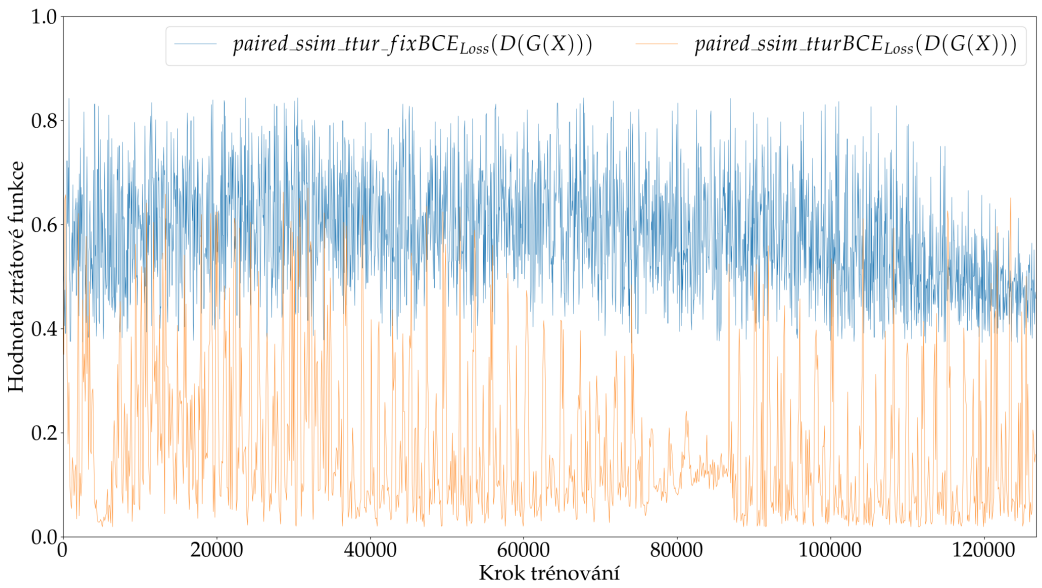
Tabulka 3.4: Vstupní parametry Pix2Pix

| experiment | dataset | epochs | batchSize | $lamda_{ssim}$ | $lamda_{l1}$ | $lr_D$ | $lr_G$ |
|---|---|---|---|---|---|---|---|
| pix2pix_1 | paired_830 | 300 | 40 | 0.0 | 50.0 | 0.00020 | 0.00020 |
| pix2pix_pr | paired_830 | 700 | 40 | 0.0 | 50.0 | 0.00020 | 0.00020 |
| pix2pix_2 | unpaired_1375 | 300 | 40 | 0.0 | 50.0 | 0.00020 | 0.00020 |
| pix2pix_3 | paired_830 | 300 | 40 | 1.0 | 50.0 | 0.00020 | 0.00020 |
| pix2pix_4 | unpaired_1375 | 300 | 40 | 1.0 | 50.0 | 0.00020 | 0.00020 |
| pix2pix_5 | paired_830 | 300 | 40 | 1.0 | 50.0 | 0.00002 | 0.00020 |
| pix2pix_6 | paired_830 | 300 | 40 | 5.0 | 50.0 | 0.00002 | 0.00020 |
| pix2pix_7 | paired_830 | 300 | 57 | 20.0 | 50.0 | 0.00002 | 0.00020 |
| pix2pix_8 | paired_830 | 400 | 57 | 20.0 | 50.0 | 0.00010 | 0.00020 |
| pix2pix_9 | paired_830 | 300 | 50 | 5.0 | 50.0 | 0.00040 | 0.00020 |

Další vylepšení výsledku bylo provedeno za pomocí již zmíněné úpravy *Two Time-scale Update Rule*, zkráceně TTUR, kde generátor a diskriminátor mají rozdílné koeficienty učení. V publikaci bylo experimentálně dokázáno, že při takové změně implementace lze dosáhnout lepších výsledků. V některých experimentech lze zkusit i změnit frekvenci učení diskriminátoru a generátoru. V dalších experimentech je tedy provedena další úprava kódu, kde je zvýšen počet celkových epoch, a diskriminátor se učí s jiným koeficientem učení než generátor. [7]

Generátor a diskriminátor provádí zpětnou propagaci sítě pro každou *minibatch* zvlášť. Konkrétně pix2pix_8má zvýšený počet epoch a nastavenou konstantu učení pro diskriminátor jako $ttur\_lr = 0.0001$, kde každou pátou epochou je prováděna zpětná propagace sítě na generátor a diskriminátor, v jiných případech prováděna pouze na diskriminátor. První experiment pix2pix_5 nedopadl dle očekávání a výsledky byly mnohem horší než jak bylo předpokládáno. Experiment si vedl i hůře něž ostatní experimenty, a tak byla raději překontrolována implementace a nalezena chyba. Experiment byl tedy spuštěn znovu jako pix2pix_6 a již dopadl v relativním porovnání s ostatními experiment, jak se očekávalo. Důsledek špatného nastavení TTUR lze také vidět na ztrátové funkci diskriminátoru a generátoru během trénování. Konkrétně tedy na ztrátové funkci diskriminátoru, který, na rozdíl od ostatních experimentů, dominuje nad generátorem. Na grafu viz 3.8 je zobrazen průběh ztrátové funkce diskriminátorů, kdy je vstup vygenerovaný obrázek. Experiment s názvem pix2pix_6 má tak hodnoty ztrátové funkce mnohem blíže k ideální hodnotě 0.5. Experiment pix2pix_5 má ale diskriminátor, který má velice malou ztrátu, tedy v takovém případě, jak již bylo zmíněno, generátor produkuje velmi podobné výsledky nehledě na referenční obrázek. Generátor tak zkolaboval na pár řešení, které mají nejblíže ke splnění podmínek diskriminátoru. V publikaci [7] je spíše u experimentů zvýšen koeficient učení u diskriminátoru při stejné frekvenci učení, takový experiment jménem pix2pix_9 si však vedl hůře, než předchozí experimenty.

Na každý natrénovaný model byl poté spuštěn stejný testovací dataset a výstupné obrázky byly uloženy. Poté byl vytvořen Python skript, který iteroval přes výsledky experimentů. Skript




Obrázek 3.8: Porovnání ztrátových funkcí $D_{fake}$

v každé iteraci spočítal pro každý experiment všechny metriky viz kapitola 3.2. Pro relativní seřazení výsledků experimentů je vhodné si definovat další metriku, která dokáže na základě ostatních metrik výsledky ohodnotit a relativně seřadit. Je tak definována další metrika *score*, která je váženou sumou všech ostatních metrik. Metriku *score* tak lze zapsat jako vzorec:

$$\begin{aligned} score = w_1 * \frac{L1_{max} - L1}{L1_{max} - L1_{min}} \\ + w_2 * \frac{L2_{max} - L2}{L2_{max} - L2_{min}} \\ + w_3 * \frac{FID_{max} - FID}{FID_{max} - FID_{min}} \\ + w_4 * \frac{SSIM - SSIM_{min}}{SSIM_{max} - SSIM_{min}} \end{aligned} \tag{3.6}$$

Váhy ve vzorci pro jednotlivé složky jsou zvoleny jako $w_1 = 0.125, w_2 = 0.125, w_3 = 0.375, w_4 = 0.375$

Z vizuálních výsledků experimentů lze vidět, že síť nedosahuje požadovaných výsledků. Jedno z kritérií bylo zachování objektů ze scény původního obrázku. Výsledek tak podkládá teoretické předpoklady toho, že model založený na Pix2Pix nebude dosahovat chtěných výsledků na dostupném trénovacím datasetu. V tabulce 3.6, lze vidět relativní srovnání dle vypočítané metriky score. Seřazení v tabulce z většiny odpovídá subjektivnějšímu seřazení člověka dle vizuálního posouzení.

Nejlepšího výsledku dosáhl experiment paired_ssim_ttur_3. Ve výsledku tohohle experimentu,

Tabulka 3.5: Výsledky Pix2Pix

| Experiment | $L_1$ | $L_2$ | $L_{SSIM}$ | $L_{FID}$ |
|---|---|---|---|---|
| pix2pix_1 | 0.043481 | 0.007637 | 0.097193 | 152.04 |
| pix2pix_pretrained | 0.069405 | 0.014592 | 0.046725 | 175.70 |
| pix2pix_2 | 0.082159 | 0.016718 | 0.027338 | 142.50 |
| pix2pix_3 | 0.042569 | 0.007329 | 0.104666 | 142.93 |
| pix2pix_4 | 0.081299 | 0.016285 | 0.030724 | 135.80 |
| pix2pix_5 | 0.059059 | 0.011451 | 0.044726 | 403.22 |
| pix2pix_6 | 0.042406 | 0.007298 | 0.101080 | 162.32 |
| pix2pix_7 | 0.043612 | 0.007387 | 0.099231 | 148.87 |
| pix2pix_8 | 0.042046 | 0.007169 | 0.106371 | 136.58 |
| pix2pix_9 | 0.043670 | 0.007702 | 0.097436 | 185.00 |

Tabulka 3.6: Relativní srovnání experimentů Pix2Pix

| Experiment | score |
|---|---|
| pix2pix_8 | 0.998909 |
| pix2pix_3 | 0.978191 |
| pix2pix_7 | 0.940071 |
| pix2pix_6 | 0.934896 |
| pix2pix_1 | 0.923078 |
| pix2pix_9 | 0.876584 |
| pix2pix_pretrained | 0.478614 |
| pix2pix_4 | 0.399415 |
| pix2pix_2 | 0.365606 |
| pix2pix_5 | 0.223434 |

již lze vidět lepší zachování detailů vozovky a různých objektů kolem silnice. Experiment má však stále problém zachovávat ostatní objekty na vozovce z původního obrázku. Také absolutní hodnota metriky FID nedosahuje dobrých výsledků, měla by se více blížit aspoň absolutní hodnotě 100.


Obrázek 3.9: Výsledky Pix2Pix sítě

Mimo již použité techniky na vylepšení výsledku existují i další možnosti. Například zmenšení modelu, aby se odstranil problém s mizejícím gradientem. Další možnosti pro stabilizování GAN sítě je například takzvaná spektrální normalizace. Všechny experimenty založené na Pix2Pix mají však kritickou slabinu ve výsledku, a to naučit se zachování objektů z původního obrázku. Tuto slabinu pravděpodobně zmíněnými technikami nelze odstranit. Možným řešením je využití jiného modelu, který by dokázal například extrahovat různé objekty ze scény, a tyhle příznaky zakomponovat do Pix2Pix modelu. Taková modifikace modelu by již byla komplexnější, než modely založená na architektuře CycleGAN.

## 3.5 CycleGAN

Dle teoretického rozboru by měl CycleGAN generovat lepší výsledky než Pix2Pix. Jak již bylo zmíněno, CycleGAN využívá cyklickou ztrátu a lze ho tak trénovat pomocí datasetu, který obsahuje nespárované obrázky. Modely postavená na CycleGAN architektuře jsou ale komplexnější a je náročnější je trénovat. Většina experimentů je tak spuštěna s menším rozlišením, nebo s menším *batchsize*, tak aby se je povedlo spustit na testovacích konfiguracích.

### 3.5.1 Implementace

Model postavený na architektuře CycleGAN se skládá ze dvou generátorů $G$, $F$ a dvou diskriminátorů $D_X$ a $D_Y$. Implementace je provedena stejně jako u předchozí Pix2Pix za pomocí *PyTorch* frameworku. Pro experimenty byl využit kód z repozitáře [20] a [22], který byl modifikován a dále rozvíjen.

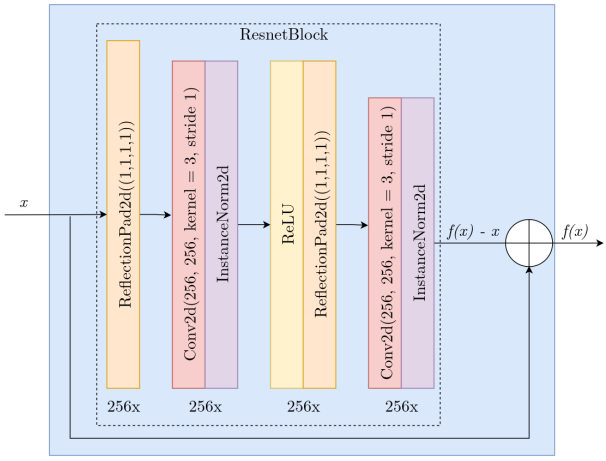
U hlubokých neuronových sítí bylo ukázáno, že jednotlivé vrstvy se učí rozeznávat příznaky obrázku, kde teoreticky přidání dalších vrstev dokáže síť extrahovat z obrázku více příznaků. Nejlépe si tak vedly při klasifikacích obrázků modely, které využívaly velmi hlubokých neuronových sítí. Problémem ale je, že při přidávání dalších vrstev se funkce, která sleduje přesnost sítě při učení, saturovala a poté snižovala. [23]

CycgleGAN generátor tak využívá sítě *ResNet* z publikace Deep Residual Learning for Image Recognition [23]. Tato síť obsahuje bloky takzvané ResNetBlocks, které jsou základními bloky této sítě. ResNetBlock byl navržen k tomu, aby řešil tento problém saturující funkce. Vnitřní blok na schématu 3.10 se musí naučit "zbytkovou funkci" $f(x) - x$, ze kterého tento blok dostal své jméno. Tento blok se tak rychleji učí, například mapování identity. Vnější spojení $x$ je zváno zkratkou. Podobně jako u U-Net, má tato zkratka za úkol řešit problém se ztrátou informací. Na výstupu je tak vstup s dalšími informacemi navíc, které se naučila konvoluční složka tohoto bloku. [23]

Architektura celé sítě se pak skládá prvních pár konvolučních vrstev, z 9 ResNetBlocks, a zpátky na požadované dimenze obrázku pomocí ConvTranpose2D vrstev. Jak již bylo naznačeno, koncept je tedy velmi podobný U-Net 3.6 použitou v Pix2Pix. Celou síť lze vidět na diagramu 3.11. Vstupem sítě je obrázek velikosti 256x256 pixelů a výstupem sítě je vygenerovaný obrázek 256x256 pixelů. Na diagramu lze vidět, že síť se skládá z následujících vrstev:

- **ReflectionPad2d((3,3,3,3))**: Tato vrstva provádí reflexi na okrajích vstupního obrazu, čímž rozšiřuje jeho rozměry. ReflectionPad2d((3,3,3,3)) znamená, že na každém okraji budou přidány 3 pixely ze sousedních oblastí, které jsou zrcadlově překlopeny.

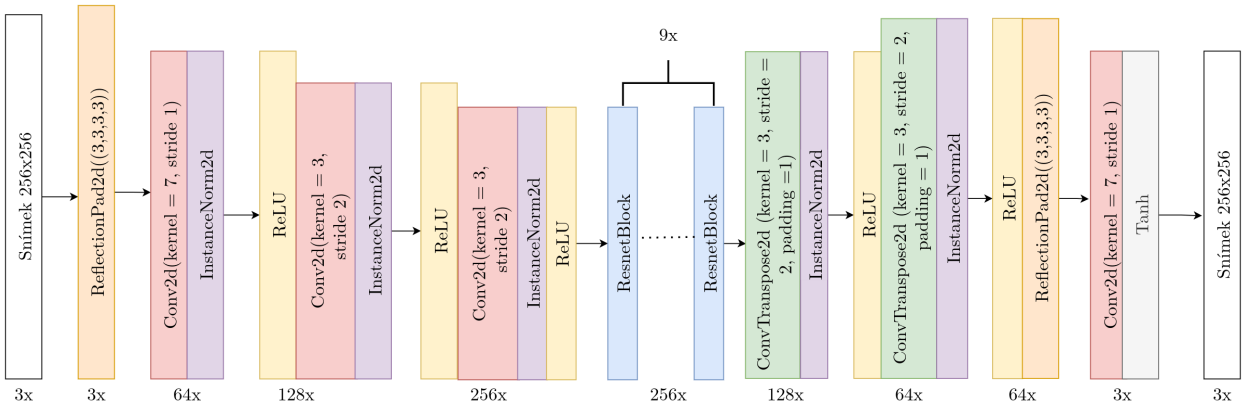
- **Conv2d(kernel = 7, stride 1)**: Konvoluční vrstva s jádrem o velikosti 7x7 a krokem (stridem) 1. Tato vrstva slouží k detekci různých vzorců ve vstupním obrazu a dá je jako výstup.




Obrázek 3.10: Schéma *ResNetBlocks*

- **InstanceNorm2d**: Tato vrstva normalizuje výstupy konvoluční vrstvy na úrovni jednotlivých instancí v dávce (batch). Pomáhá zlepšit stabilitu a rychlost trénování. Je tak přidaná za každou konvoluční vrstvou.
- **ReLU**: Aktivační funkce ReLU (Rectified Linear Unit) je nelineární vrstva, která zavádí nelinearitu do modelu. ReLU upravuje výstupy předchozí vrstvy tak, že všechny záporné hodnoty jsou nastaveny na 0.
- **ResnetBlock**: Blok reziduální sítě, který se skládá z několika konvolučních a normalizačních vrstev, jak lze vidět na obrázku 3.10. ResnetBlock umožňuje modelu efektivněji se učit složitější funkce a snižuje problém mizení gradientu.
- **ConvTranspose2d(kernel = 3, stride = 2, padding = 1)**: Dekonvoluční vrstva, nebo někdy také zvaná jako transponovaná konvoluční vrstva, s jádrem o velikosti 3x3, krokem 2 a výplní (padding) 1. Tato vrstva je použita k zvětšení výstupu a obnovení původních rozměrů obrázku.
- **Tanh**: Aktivační funkce hyperbolický tangens, která normalizuje výstupy do rozsahu (-1, 1). *Tanh* se často používá na konci generativních modelů pro obrazy, aby se zajistila správná normalizace pixelových hodnot.

V ukázce Python kódu 3.3 lze vidět trochu zjednodušenou implementovanou zpětnou propagaci generátoru. Pokud je tedy nastavená $\lambda_{L_{identity}} > 0$, tak se do generátoru $G$, v kódu pojmenován jako *netG_A*, jako vstup vloží reálný obrázek z domény $Y$. Jinak řečeno, vstupem do generátoru, který mapuje denní obrázek na noční, je noční obrázek. Ztrátová funkce se poté počítá pomocí *torch.nn.L1Loss()*, která má jako parametry vygenerovaný obrázek pomocí generátoru a reálný obrázek. Obdobně se to provede i pro druhý generátor.




Obrázek 3.11: Schéma *ResNet*

```
    def backward_G(self):
        if lambda_idt > 0:
            self.idt_A = self.netG_A(self.real_B)
            self.loss_idt_A = self.criterionIdt(self.idt_A, self.real_B) * lambda_B
                * lambda_idt
            self.idt_B = self.netG_B(self.real_A)
            self.loss_idt_B = self.criterionIdt(self.idt_B, self.real_A) * lambda_A
                * lambda_idt
        else:
            self.loss_idt_A = 0
            self.loss_idt_B = 0

        self.loss_G_A = self.criterionGAN(self.netD_A(self.fake_B), True)
        self.loss_G_B = self.criterionGAN(self.netD_B(self.fake_A), True)
        self.loss_cycle_A = self.criterionCycle(self.rec_A, self.real_A) *
            lambda_A
        self.loss_cycle_B = self.criterionCycle(self.rec_B, self.real_B) *
            lambda_B
        self.loss_G = self.loss_G_A + self.loss_G_B + self.loss_cycle_A + self.
            loss_cycle_B + self.loss_idt_A + self.loss_idt_B
        self.loss_G.backward()
```

Kód 3.3: Kód výpočtu ztrátové funkce generátoru

Úplně první experimenty byly provedeny na frameworku *Tensorflow*, kde se bohužel nepovedlo zpětně provést spuštění na testovacím datasetu, nejsou uvedené mezi výsledky. Nicméně jeden z nich

Tabulka 3.7: Vstupní parametry experimentů CycleGAN 1

| experiment | cycleGAN 1 | cycleGAN 2 | cycleGAN 3 | cycleGAN 4 |
|---|---|---|---|---|
| dataset | un_1375 | un_1375 | un_1375 | un_2000 |
| epochs | 100 | 200 | 200 | 800 |
| decay | 100 | 200 | 44 | 100 |
| imageSize | 256 x 256 | 256 x 256 | 128 x 128 (náhodné ořiznutí) | 128 x 128 |
| batchSize | 1 | 8 | 8 | 8 |
| dropout | False | True | False | False |
| $lr_D$ | 0.00020 | 0.00002 | 0.00002 | 0.00020 |
| $lr_G$ | 0.00020 | 0.00020 | 0.00002 | 0.00020 |
| antialias | False | True | False | False |
| gNet | resnet_9blocks | resnet_9blocks | resnet_9blocks | resnet_9blocks |

Tabulka 3.8: Vstupní parametry experimentů CycleGAN 2

| experiment | cycleGAN 5 | cycleGAN 6 | cycleGAN 7 | cycleGAN 8 |
|---|---|---|---|---|
| dataset | un_200 | un_200 | un_2000 | un_10000 |
| startEpoch | 0 | 0 | 0 | 150 (cycleGAN 7) |
| epochs | 200 | 200 | 150 | 200 |
| decay | 200 | 200 | 0 | 50 |
| imageSize | 128 x 128 | 256 x 256 | 256x 256 | 256x 256 |
| batchSize | 5 | 1 | 1 | 1 |
| dropout | True | True | True | True |
| $lr_D$ | 0.00020 | 0.00002 | 0.00002 | 0.00020 |
| $lr_G$ | 0.00020 | 0.00020 | 0.00002 | 0.00020 |
| antialias | True | True | True | True |
| gNet | resnet_6blocks | resnet_9blocks | resnet_9blocks | resnet_9blocks |

využíval spárovaný dataset a druhý nespárovaný, kde šlo vidět, že nespárovaný dataset měl lepší výsledky než spárovaný. Lze to odůvodnit tím, že model má tak více rozmanitý dataset, a díky ztrátové funkci cyklu, si síť vede dobře i na nespárovaném datasetu.

V tabulce 3.7 lze vidět vstupní parametry první sady experimentů provedených na architektuře CycleGAN.

Hned první výsledek dopadl na testovací sadě lépe než nejlepší experimenty Pix2Pix. Vozovka ve výsledných obrázcích se jeví jako zcela správná, není nějak pokroucená a nejsou zde viditelné nechtěné artefakty. Silnice je i správně osvětlená, jak od reflektorů tak i od silničního osvětlení. Objevují se zde nedokonalosti u některých vzdálenějších objektů. Například vzdálenější vozidla již mohou ztrácet původní tvar a nejsou správně osvětlená. Na dalším experimentu bylo provedeno vylepšení implementace. Bylo tak do implementace přidáno rozšíření pro PyTorch, a to rozšíření *NVIDIA apex* z repozitáře [24]. Cílem tohoto rozšíření pro PyTorch je zvýšení výkonu a efektivity

Tabulka 3.9: Vstupní parametry experimentů CycleGAN 3

| experiment | cycleGAN 9 | cycleGAN 10 | cycleGAN 11 | cycleGAN 12 |
|---|---|---|---|---|
| dataset | un_2000 | un_2000 | un_2000 | un_2000 |
| epochs | 150 | 150 | 150 | 150 |
| decay | 50 | 50 | 50 | 50 |
| imageSize | 256x256 | 256x256 | 256x256 | 256x256 |
| batchSize | 3 | 3 | 3 | 3 |
| dropout | False | False | False | False |
| $lr_D$ | 0.00002 | 0.00020 | 0.00020 | 0.00020 |
| $lr_G$ | 0.00020 | 0.00010 | 0.00010 | 0.00010 |
| antialias | True | True | True | True |
| gNet | resnet_9blocks | resnet_9blocks | resnet_9blocks | resnet_9blocks |

Tabulka 3.10: Vstupní parametry experimentů CycleGAN 4

| experiment | cycleGAN 13 | cycleGAN 14 | cycleGAN 15 |
|---|---|---|---|
| dataset | un_2000 | un_2000 | un_2000 |
| startEpoch | 0 | 200 | 0 |
| epochs | 150 | 100 | 150 |
| decay | 50 | 50 | 50 |
| imageSize | 256x256 | 486x486 | 300x300 |
| batchSize | 3 | 1 | 2 |
| dropout | False | False | False |
| $lr_D$ | 0.00020 | 0.00020 | 0.00002 |
| $lr_G$ | 0.00010 | 0.00002 | 0.00020 |
| antialias | True | True | True |
| gNet | resnet_9blocks | resnet_9blocks | resnet_9blocks |

pro hluboké trénování neuronové sítě. Rozšíření obsahuje i podporu pro takzvané "mixed-precision training", které má za následek rychlejší čas trénování a díky využívání poloviční přesnosti pro desetinná čísla i efektivnější využití paměti. Příklad implementace je například pro propagaci sítě, pomocí anotace u metody, jak lze vidět na příkladě kódu 3.4.

```
@autocast()
def forward(self, input):
```

Kód 3.4: Apex mixed-precision

Toto vylepšení se však nepovedlo implementovat na poprvé a některé experimenty tak selhaly. Chyba implementace se tak projevila například ve 115. epoše trénovacího procesu, kdy generátor na výstupu vyprodukoval pouze černý obrázek. Bylo tak zapotřebí upravit tuto modifikaci.

Pravděpodobná příčina toho problému byla taková, že síť při dopředném průchodu počítá s datovým typem *float16*. Díky tomu tak zpětná propagace bude počítat s velmi malými gradienty. Takové malé gradienty se v datovém typu *float16* mohou ztratit a narazí tak na známý problém "underflow", neboli podtečení. Tím se aktualizace pro parametry sítě ztratí.

Řešením předcházení problému "podtečení"je škálovaní těchto gradientů. Škálovaní gradientů je implementováno v PyTorch pomocí třídy *torch.cuda.amp.GradScale*, který násobí ztráty sítě škálovacím faktorem a předá je pro zpětnou propagaci sítě. Tato operace je rychlá a tak by neměla mít na rychlost sítě velký vliv.

Gradienty však mohou také mít opačný problém takzvaný "gradient clipping". Gradienty sítě se mohou rychle zvětšovat nebo se stát nekonečně velikými, což způsobí nestabilitu sítě a špatnou konvergenci modelu. Funkce *clip_grad_norm_* v PyTorch řeší tenhle problém tím, že gradienty normalizuje tak, aby nepřekračovaly zadanou maximální hodnotu. Tento problém se většinou projevuje u rekurentních neuronových sítí, přidat ho však do implementace GAN může mít za následek lepší stabilitu sítě. Tato operace by neměla mít na výkon sítě moc velký vliv, tedy je vhodné ji v implementaci využít a předejít tak neúspěšným modelům.

Spolu s použitím *torch.cuda.amp.GradScale* je však nutné nejprve gradienty před normalizací provést zpětnou operaci škálování. Při správné implementaci smíšené přesnosti a škálování gradientů by síť měla dosahovat podobných výsledků a přitom být rychlejší a efektivnější. CycleGAN model má celkem čtyři sítě a v implementaci jsou využity dva optimalizéry. Implementace je tak složitější, ale lze ji implementovat. V ukázce kódu 3.5, lze vidět zjednodušenou implementaci. Ve skutečné implementaci bylo potřeba navíc v kódu, ještě dle parametru, určit zda se má využít smíšená přesnost *floatu*.

```
def optimize_parameters(self, iteration):
    self.forward()
    self.set_requires_grad([self.netD_A, self.netD_B], False)
    self.optimizer_G.zero_grad()
```

```
self.backward_G()

self.scaler.unscale_(self.optimizer_G)
torch.nn.utils.clip_grad_norm_(self.netG_A.parameters(), self.
    max_grad_norm)
torch.nn.utils.clip_grad_norm_(self.netG_B.parameters(), self.
    max_grad_norm)
self.scaler.step(self.optimizer_G)

self.set_requires_grad([self.netD_A, self.netD_B], True)
self.optimizer_D.zero_grad()
self.backward_D_A()
self.backward_D_B()

self.scaler.unscale_(self.optimizer_D)
torch.nn.utils.clip_grad_norm_(self.netD_A.parameters(), self.
    max_grad_norm)
torch.nn.utils.clip_grad_norm_(self.netD_B.parameters(), self.
    max_grad_norm)
self.scaler.step(self.optimizer_D)

self.scaler.update()
```

Kód 3.5: Implementace GradScaler s AMP

Zmíněný problém, vyskytující se na předcházejícím experimentu, se tak u následujících experimentů neprojevil. Velmi pozitivní následek této úpravy je zrychlení celého trénovacího procesu. Experimenty, na testovací konfiguraci viz 3.2, měly čas trénování pro jednu iteraci průměrně $time\_per\_iter = 0.318ms$. Experiment spuštěný se zmíněnou modifikací, stejnými parametry a na stejné testovací konfiguraci byl na tom o dost lépe a to $time\_per\_iter = 0.172ms$. Navíc byl model ještě efektivnější s prací GPU paměti. Druhý model se tak podařilo spustit s trojnásobným *batch_size* a tím ještě více urychlit proces trénování. Průměrné naměřené časy trénování však nemusí být perfektní ukazatel, jak moc je model efektivnější, záleží zde totiž na více faktorech. Pro jeden takový příklad může být faktorem i okolní teplota v místnosti, kde se nachází testovací konfigurace. Daná modifikace také nemusí mít tak moc velký vliv na jiné architektury a modely, které využívají jiné operace, které nejsou optimalizované. S jistotou lze ale tvrdit, že energická spotřeba na jednu trénovací iteraci se v tomto případě zmenšila.

Vylepšení sítě o smíšenou přesnost datového typu float (AMP), je však dle mého nejzásadnější a nejvíce užitečná pro práci s tak velkými modely jako jsou sítě postavené na architektuře GAN.

V aktuální době psaní této práce nabývá na významu i energická náročnost trénování a spouštění takového modelu. Modifikace architektury sítě přispívá ke zlepšení energetické efektivity za skoro žádný negativní vliv na kvalitě výsledku. Modifikace však nemusí být tak prospěšné i na jiných modelech, v tomto případě však její implementace byla velmi přínosná.

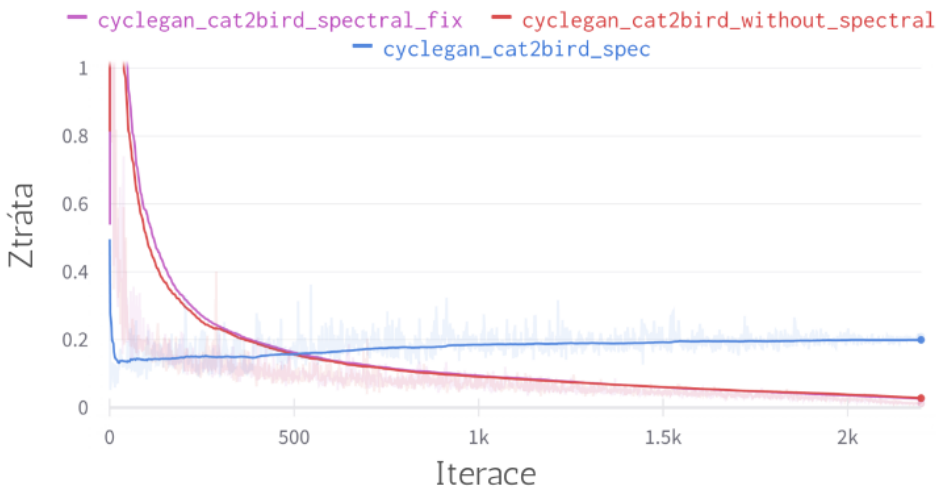
Dalším vylepšením bylo přidání takzvané spektrální normalizace. Jedná se o další prvek jak zlepšit stabilitu sítí GAN. Konkrétně jde o techniku normalizaci vah, zvanou spektrální normalizace. Jedním z hlavních problémů, které mohou nastat při trénování GANs, je skutečnost, že generované obrázky mohou mít jinou distribuci než reálná data. To může vést k přetrénování, kdy se GAN naučí reprodukovat jen určitou část trénovacích dat, ale selhává při generování nových a rozmanitých obrázků. [25]

Spektrální normalizace řeší ten problém tím způsobem, že pro každou vrstvu v diskriminátoru spočítá spektrální normalizaci vah. Tato technika byla navržena v publikaci *Spectral Normalization for Generative Adversarial Networks* [25]. Nová normalizovaná váha je vypočtená dle vzorce 3.7, kde $W$ je vstupní matice vah a $\sigma(W)$ představuje nejvyšší singulární hodnotu matice $W$. Tento postup umožňuje diskriminátoru pracovat s většími gradienty a zlepšuje jeho schopnost rozlišovat mezi reálnými a generovanými obrázky. [25]

$$\overline{W}_{SN}(W) = W/\sigma(W) \tag{3.7}$$

Publikace tedy radí využívat spektrální normalizaci pro každou vrstvu diskriminátoru. Případnou modifikaci lze však případně provést i na generátoru. V experimentech pro implementaci spektrální normalizace je využita funkce z knihovny PyTorch, konkrétně *torch.nn.utils.spectral_norm*. Implementace využívá metodu zvanou "Power iteration". Tato metoda je rychlím způsobem jak najít aproximaci největší singularní hodnoty matice vah. Poté tedy stačí touto funkcí "obalit"v implementaci chtěné vrstvy diskriminátoru. Tato modifikace však bude kolidovat s modifikací *AMP*. Pokud tedy experiment bude problémový, tak je možné v této části kódu udělat výjimku a *AMP* pro tuto operaci nevyužívat. Konkrétně knihovna využívá operaci *matmul*, která bude dle seznamu z dokumentace [26] ovlivněna. U experimentů transformace dne na noc bylo obtížné určit správnost implementace této úpravy a její vliv.

Pro otestování je tedy potřeba vymyslet jednodušší experiment, kde by délka trénování byla výrazně menší. Byl tak napsán skript, který zpracoval dataset ze zdroje *CIFAR-10* [27]. Z datasetu byly použity dvě třídy, konkrétně *cat* a *bird*. Experiment *cyclegan_cat2bird_without_spectral* nevyužíval implementovanou spektrální normalizaci a druhý experiment *cyclegan_cat2bird_spec* ji využíval. Všechny ostatní parametry byly pro experimenty nastaveny totožně. První experiment si však vedl lépe, a diskriminátor lépe rozeznával reálné obrázky od vygenerovaných. Výsledky tak naznačovali nejspíše chybu implementace, který však byla nalezena a opravena. Následoval tak třetí experiment s již opravenou implementací spektrální normalizace. Ztrátovou funkci diskriminátorů zmíněných experimentů lze vidět na grafu 3.12.




Obrázek 3.12: Ztráta diskriminátorů experimentů na datasetu *Cat2Bird*

Modifikované experimenty 2,3 a 4 dopadly mnohem hůře, než první spuštěný. Důvodem jsou tedy již popisované průběžné úpravy implementace a jinak zvolené vstupní parametry, jako například rozlišení obrázku.

V tabulce 3.7 lze vidět vstupní parametry experimentů provedených na architektuře CycleGAN.

Druhý a třetí experiment selhaly. Měly špatnou implementace spektrální normalizace a další experimenty selhaly y důvodu špatné implementaci AMP. Oproti Pix2Pix je celkový výsledek sítě lepší, když je ztráta diskriminátoru mnohem nižší. U Pix2Pix bylo lepší, když se ztráta diskriminátoru během trénování pohybovala okolo hodnoty $D_{loss} < 0.5$. V případech kdy byla ztráta menší, například $D_{loss} < 0.2$, tak generátor zkolaboval na pár podobných výsledků. U experimentů CycleGAN taková menší ztráta nevadila, což je pravděpodobně způsobenou komxplexnější ztrátovou funkcí.

Další experimenty lze vidět v tabulkách 3.8, 3.9 a 3.10. Vypočítané metriky z výsledků lze vidět v tabulce 3.11. Jejich srovnání je poté v tabulce 3.12. Z výsledků lze vidět, že jsou si výsledky kvalitou mnohem blíže, než jak tomu bylo u experimentů Pix2Pix.

Z výsledných obrázků 3.13 lze usoudit, že se nepovedlo udělat dokonalý model. Zobrazený testovací obrázek byl vybrán právě z důvodem, že se na něm nachází více objektů v jiných vzdálenostech. Lze tak snadno odhalit nedokonalosti naučené transformace. Mezi jednu takovou například je, že billboard na obrázku není osvětlen. S porovnání Pix2Pix je tato jediná věc horší. Jinak si každý experiment vedl dobře v zachování struktury silnice a jejím osvělením. Zajímavými experimenty jsou CycleGAN 8, který pokračoval z experimentu CycleGAN 7 a byl dále trénován na větším datasetu.

Srovnání výsledků těchto experimentů již není tak jednoduché jako u výsledků Pix2Pix a jak celkové *score*, tak i jednotlivé metriky nedokáží zachytit subjektivní seřazení dle člověka. Například experiment CycleGAN 13 má *score* nižší, než co bych já sám očekával.

Závěrem této kapitoly lze říci, že výsledky již nebyly špatné a blížily se realitě. Pravděpodobně

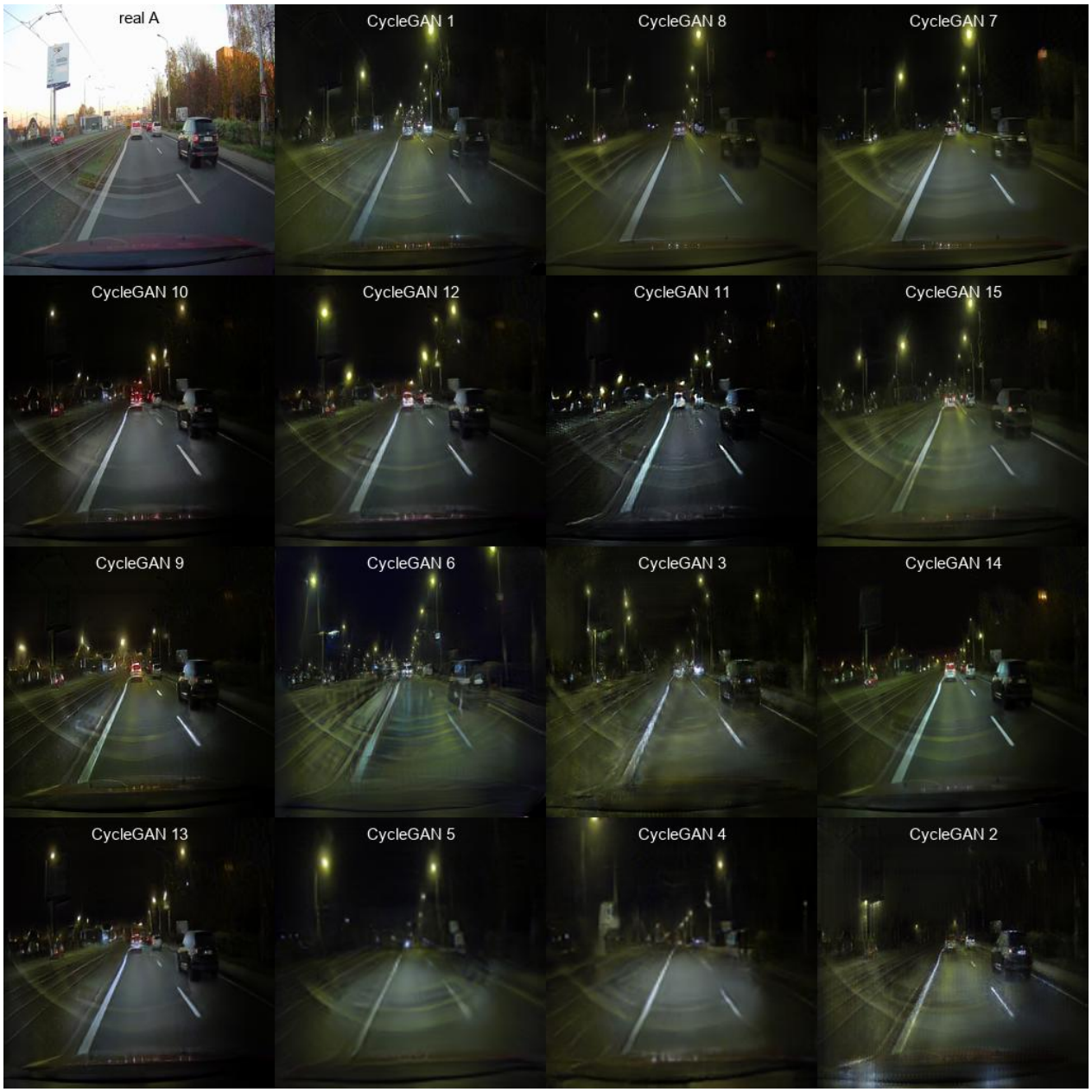
Tabulka 3.11: Výsledky CycleGAN

| Experiment | $L_1$ | $L_2$ | $L_{SSIM}$ | $L_{FID}$ |
|---|---|---|---|---|
| CycleGAN 1 | 0.075683 | 0.014421 | 0.066841 | 80.45 |
| CycleGAN 2 | 0.295069 | 0.155471 | 0.174108 | 214.37 |
| CycleGAN 3 | 0.080461 | 0.016125 | 0.044960 | 110.57 |
| CycleGAN 4 | 0.089210 | 0.018031 | 0.042258 | 133.66 |
| CycleGAN 5 | 0.098217 | 0.014390 | 0.009147 | 102.08 |
| CycleGAN 6 | 0.087098 | 0.017939 | 0.052601 | 112.04 |
| CycleGAN 7 | 0.080135 | 0.016384 | 0.063544 | 89.25 |
| CycleGAN 8 | 0.073193 | 0.013770 | 0.070743 | 86.36 |
| CycleGAN 9 | 0.082755 | 0.017070 | 0.054432 | 102.54 |
| CycleGAN 10 | 0.079666 | 0.016047 | 0.059338 | 93.59 |
| CycleGAN 11 | 0.076980 | 0.014946 | 0.059761 | 102.45 |
| CycleGAN 12 | 0.081248 | 0.016410 | 0.058570 | 95.51 |
| CycleGAN 13 | 0.085101 | 0.018349 | 0.044097 | 124.50 |
| CycleGAN 14 | 0.080804 | 0.016118 | 0.035113 | 105.39 |
| CycleGAN 15 | 0.078078 | 0.014941 | 0.042518 | 89.20 |

Tabulka 3.12: Relativní srovnání experimentů CycleGAN

| Experiment | Score |
|---|---|
| CycleGAN 1 | 0.754177 |
| CycleGAN 8 | 0.748482 |
| CycleGAN 7 | 0.717811 |
| CycleGAN 10 | 0.696661 |
| CycleGAN 12 | 0.688322 |
| CycleGAN 11 | 0.675290 |
| CycleGAN 15 | 0.672588 |
| CycleGAN 9 | 0.657801 |
| CycleGAN 6 | 0.623808 |
| CycleGAN 3 | 0.615907 |
| CycleGAN 14 | 0.607847 |
| CycleGAN 13 | 0.570350 |
| CycleGAN 5 | 0.549782 |
| CycleGAN 4 | 0.538504 |
| CycleGAN 2 | 0.375000 |

by šlo implementovat další experimenty, které by si vedly lépe. Problémem je však komplexnost nastavení všech hyperparametrů sítě a ne vždy je jednoduché předpovědět výsledek. Ze srovnání lze taky říci, že takové výsledky není ani jednoduché srovnat. Díky tomu, že je ztrátová funkce max - min hra, tak nelze ani jednoduše sledovat průběžné výsledky trénování, protože hodnoty rychle klesají ale i rostou. Také lze říci, že správnost výsledku a nastavení hyperparametrů záleží na každé konkrétní úloze. Nelze tedy říci, že by se jednalo o perfektní obecné řešení.




Obrázek 3.13: Srovnané výsledky CycleGAN

## 3.6 Contrastive unpaired translation

Další testovanou architekturou je tedy CUT. Dle teoretických předpokladů probraných v kapitole o CUT viz 2.4, by tato architektura měla mít kvalitu výstupu podobnou těm postavených na CycleGAN. Implementace využívá již implementované bloky sítě z modelu CycleGAN. V implementaci je tak použit již zmíněný PatchGAN jako diskriminátor, a ResnetNet jako generátor. Navíc je zde MPL, která vytahuje příznaky z generátoru. Přední průchod generátoru je tedy implementován tak, že dokáže vracet jednotlivé vrstvy. Implementace lze vidět na ukázce kódu 3.6. V implementaci CUT bylo také postupně implementováno AMP a spektrální normalizace.

```
def forward(self, input, layers=[], encode_only=False):
        if -1 in layers:
            layers.append(len(self.model))
        if len(layers) > 0:
            feat = input
            feats = []
            for layer_id, layer in enumerate(self.model):
                feat = layer(feat)
                if layer_id in layers:
                    feats.append(feat)
                else:
                    pass
                if layer_id == layers[-1] and encode_only:
                    return feats

            return feat, feats
        else:
            fake = self.model(input)
            return fake
```

Kód 3.6: Implementace předného průchodu CUT generátoru

### 3.6.1 FastCUT

Jako první byl experiment s FastCUT. Jak už bylo řečeno, je to zjednodušená verze CUT, a liší se například v tom, že ve své ztrátové funkci nemá ztrátu identity. Navíc jako vylepšení datasetu využívá zrcadlové obrácení. V původní publikaci CUT je doporučeno pro FastCUT zvolit a $lambda_{NCE}$, aby se nahradil chybějící "regulátor". [10]

Tabulka 3.13: Vstupní parametry FastCUT

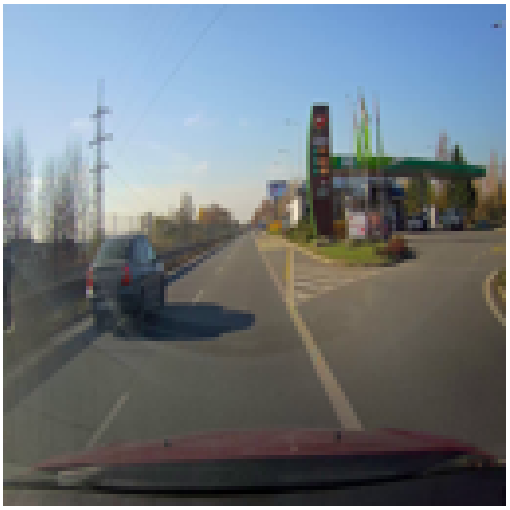
| Experiment | Dataset | Epochy | decay | batchSize | $lamda_{NCE}$ | velikost obrázku |
|---|---|---|---|---|---|---|
| FastCUT 1 | un_1375 | 150 | 50 | 1 | 2.0 | 256x256 pixelů |
| FastCUT 2 | un_1375 | 150 | 50 | 1 | 5.0 | 256x256 pixelů |
| FastCUT 3 | un_1375 | 150 | 50 | 1 | 1.0 | 256x256 pixelů |
| FastCUT 4 | un_1375 | 300 | 100 | 1 | 1.0 | 256x256 pixelů |
| FastCUT 5 | un_2000 | 200 | 50 | 3 | 1.0 | 128x128 pixelů |

Tabulka 3.14: Výsledky FastCUT

| Experiment | $L_1$ | $L_2$ | $L_{SSIM}$ | $L_{FID}$ |
|---|---|---|---|---|
| FastCUT 1 | 0.080983 | 0.016773 | 0.057746 | 86.65 |
| FastCUT 2 | 0.127429 | 0.028327 | 0.021013 | 214.41 |
| FastCUT 3 | 0.083649 | 0.016721 | 0.055294 | 132.06 |
| FastCUT 4 | 0.092699 | 0.018896 | 0.043357 | 123.72 |
| FastCUT 5 | 0.082016 | 0.015903 | 0.082137 | 159.03 |

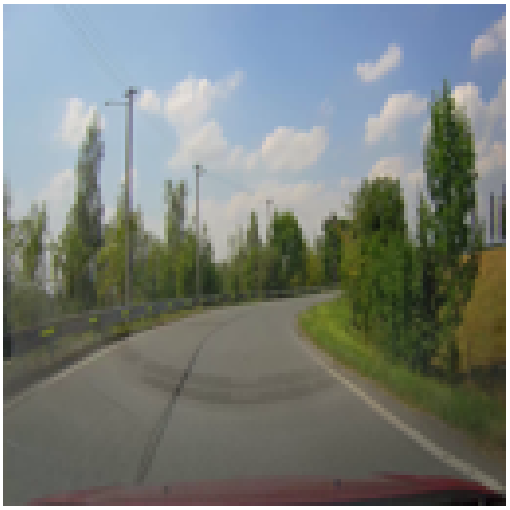
Hned u prvních iterací bylo poznat, že se síť dostatečně nedrží původního obrázku. Byla zde totiž nastavena $lambda_{NCE} = 10.0$. Tato konstanta přidává na váze výstupu MPL sítě, která porovnává výřezy původního obrázku s vygenerovaným. Tato váha se však zdála moc velká pro řešení zadaného problému. Projevovalo se to tak, že se generátor co nejvíce blížil původnímu obrázku a vůbec se nedržel referenčního. U dalšího experimentu, kde byla nastavena $lambda_{NCE} = 1.0$, výsledek vypadal přijatelně již ve 13. epoše. Na obrázcích 3.14 lze vidět vstupy a výstupy obou popisovaných experimentů když již probíhala 13. epocha.

Vybrané vstupní parametry experimentů lze vidět v tabulce 3.13. Výsledky z naměřených metrik pak lze vidět v tabulce 3.14 a jejich srovnání v tabulce 3.15. Výsledky jsou stále lepší než Pix2Pix, avšak většina jich je horší, než CycleGAN. Jsou zde stejné problémy jako u CycleGAN. Například některé plochy, které by měly být osvětlené, osvětlené nejsou. Dalším problémem jsou více rozmazané výsledky, jak lze vidět ve výsledných obrázcích 3.15. Tenhle problém by se ale, dle mého názoru, dal vyřešit správně zvolenými parametry při trénování sítě. Například experiment FastCUT 1 si v tomhle ohledu vedl mnohem lépe a vzhledem k implementovaným modifikacím sítě by výsledek šel určitě dotrénovat k lepší kvalitě. Nicméně jako ukázka schopnosti modelu to stačí, protože by pravděpodobně tenhle "fine tunning"nepomohl s klíčovými nedostatky. Vzhledem k velikosti sítě oproti CycleGAN, je však tento výsledek působivý.

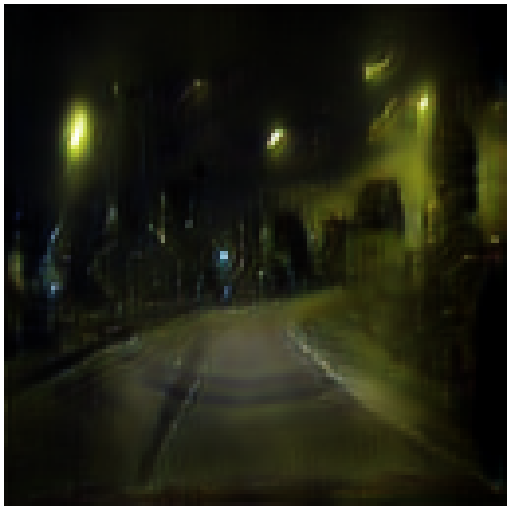


(a) vstupní reálný obrázek

(b) vygenerovaný obrázek $lambda_{NCE} = 10$



(c) vstupní reálný obrázek



(d) vstupní vygenerovaný $lambda_{NCE} = 1$

Obrázek 3.14: Výsledky experimentů FastCUT 13. epocha

Tabulka 3.15: Relativní srovnání experimentů FastCUT

| Experiment | score |
|---|---|
| FastCUT 1 | 0.841601 |
| FastCUT 5 | 0.784771 |
| FastCUT 3 | 0.686623 |
| FastCUT 4 | 0.591625 |
| FastCUT 2 | 0.000000 |


Obrázek 3.15: Výsledky FastCUT sítě

Tabulka 3.16: Vstupní parametry experimentů CUT 1

| experiment | CUT 1 | CUT 2 | CUT 3 | CUT 4 | CUT 5 |
|---|---|---|---|---|---|
| dataset | un_1375 | un_1375 | un_1375 | un_1375 | un_1375_with_back |
| startEpoch | 0 | 400 (CUT 1) | 0 | 0 | 0 |
| epochs | 200 | 400 | 200 | 200 | 200 |
| decay | 200 | 200 | 200 | 200 | 200 |
| imageSize | 256x256 | 256x256 | 256x256 | 256x256 | 256x256 |
| batchSize | 1 | 1 | 1 | 1 | 1 |
| $lamda_{NCE}$ | 1.5 | 1.5 | 1.0 | 1.0 | 1.0 |
| numPatches | 512 | 512 | 256 | 256 | 256 |
| NGF | 128 | 128 | 64 | 64 | 64 |
| NDF | 128 | 128 | 64 | 64 | 64 |
| nLayersD | 3 | 3 | 3 | 3 | 3 |
| AMP | False | False | False | False | False |

### 3.6.2 CUT

Při experimentech CUT byla oproti FastCUT využita ztráta identity. V ukázce kódu 3.7 lze vidět, že do funkce *calculate_NCE_loss* je vložen jako vstup reálný obrázek z domény B a dle něj vygenerovaný obrázek také z domény B.

```
if self.opt.nce_idt and self.opt.lambda_NCE > 0.0:
   self.loss_NCE_Y = self.calculate_NCE_loss(self.real_B, self.idt_B)
   loss_NCE_both = (self.loss_NCE + self.loss_NCE_Y) * 0.5
```

Kód 3.7: Ztráta indetity

Vstupní parametry experimentů lze vidět v tabulce 3.16 a v tabulce 3.17. Výsledné naměřené metriky lze pak vidět v tabulce 3.18 a jejich srovnání v tabulce 3.19. U experimentů CUT 1, CUT 2 byl zvolen dvojnásobný počet NGF a NDF, tedy počet filtrů v poslední vrstvě generátoru a počet filtrů v první vrstvě diskriminátoru. Dále byl u těchto experimentů navíc změněn ještě parametr *num_patches*, který určuje počet porovnaných výřezů z generátoru pro jednu vrstvu. Všechny experimenty byly odzkoušeny předáním těchto výřezů z vrstev 0,4,8,12,16. Experiment CUT 5 byl spuštěn na datasetu, který měl v sobě i obrázky, pocházejících ze zadní strany auta.

V druhé tabulce 3.17 jsou již experimenty se spektrální normalizací a s jinými datasety. Experimenty CUT 9 a CUT 10 měly navíc zvětšený počet vrstev diskriminátoru (nLayersD). Experimenty CUT 7, CUT 8, CUT 9 a CUT 10 již měly modifikaci AMP, takže bylo možné experimentovat i s větším batchSize či větší velikostí obrázku i na druhé testovací konfiguraci.

U výsledků lze vidět, jako u CycleGAN, že síť neměla problém se zanecháním správné silnice a jejím osvětlením. Problémem však stále zůstává správné osvětlení některých vozidel a objektů. Na rozdíl od CycleGAN ale vypadá, že si umí lépe poradit s nalezením a správným osvětlením pouličního

Tabulka 3.17: Vstupní parametry experimentů CUT 2

| experiment | CUT 6 | CUT 7 | CUT 8 | CUT 9 | CUT 10 |
|---|---|---|---|---|---|
| dataset | un_1375 | un_2000 | un_2000 | un_10000 | un_10000 |
| startEpoch | 0 | 0 | 100 (CUT 7) | 0 | 0 |
| epochs | 200 | 50 | 100 | 150 | 200 |
| decay | 200 | 50 | 100 | 50 | 200 |
| imageSize | 256x256 | 256x256 | 300x300 | 256x256 | 500x500 |
| batchSize | 1 | 3 | 1 | 5 | 1 |
| $lamda_{NCE}$ | 1.5 | 1.0 | 1.0 | 2.0 | 1.0 |
| numPatches | 256 | 256 | 256 | 512 | 256 |
| NGF | 64 | 64 | 64 | 64 | 64 |
| NDF | 64 | 64 | 64 | 64 | 64 |
| nLayersD | 3 | 3 | 3 | 5 | 6 |
| AMP | False | True | True | True | True |

Tabulka 3.18: Výsledky CUT

| Experiment | $L_1$ | $L_2$ | $L_{SSIM}$ | $L_{FID}$ |
|---|---|---|---|---|
| CUT 1 | 0.086121 | 0.017668 | 0.04795 | 84.14588737 |
| CUT 2 | 0.081507 | 0.016366 | 0.052594 | 95.02932508 |
| CUT 3 | 0.080203 | 0.016168 | 0.059033 | 91.52973728 |
| CUT 4 | 0.079892 | 0.0158 | 0.060457 | 88.23848167 |
| CUT 5 | 0.084107 | 0.0182 | 0.054645 | 85.45706929 |
| CUT 6 | 0.077087 | 0.01531 | 0.062615 | 88.556269 |
| CUT 7 | 0.085588 | 0.019092 | 0.048161 | 96.87840394 |
| CUT 8 | 0.080926 | 0.01823 | 0.041819 | 98.4845252 |
| CUT 9 | 0.081588 | 0.019236 | 0.057121 | 86.98851063 |
| CUT 10 | 0.066978 | 0.016152 | 0.016927 | 102.5347291 |

Tabulka 3.19: Relativní srovnání experimentů CUT

| Experiment | Score |
|---|---|
| CUT 5 | 0.679556 |
| CUT 7 | 0.567312 |
| CUT 3 | 0.706348 |
| CUT 2 | 0.798901 |
| CUT 4 | 0.703981 |
| CUT 1 | 0.844050 |
| CUT 8 | 0.379777 |
| CUT 9 | 0.352857 |
| CUT 6 | 0.676537 |
| CUT 10 | 0.223192 |

osvětlení. Sít se také zdá být stabilnější a model je celkově menší a jednoduší než CycleGAN. Stabilnější výsledek má i oproti FastCUT. Podle výsledků experimentů se zdá, že větší rozlišení nemá velký vliv. V případě potřeby však lze model případně dotrénovat na větší rozlišení.


Obrázek 3.16: Výsledky CUT experimentů

## 3.7 Výsledné srovnání a závěr

Z výsledků se velice těžce vybírá nejlepší experiment. Každá síť si vedla na některých scénách lépe. Například na diagramu 2.4 lze vidět výsledek, vygenerovaný experimentem CUT 5, který se velmi těžko rozeznává od skutečného. Tento model si však na srovnávacím testovacím obrázku tak dobře nevedl.

Počáteční myšlenka, že GAN sítě dokážou řešit zadaný problém bez zásadnějšího zásahu, jako je obecný problém mapování transformace vstupního obrázku na výstupní, nebyla plně potvrzena. Modely realistickou transformaci dokázali, ale závisí na velkém množství faktorů, od kvality datasetu až po citlivé nastavování správných parametrů, kde často záleží ještě na konkrétní úloze. Během trénování různých experimentů a na základě jejich průběžného sledování GAN architektur lze říci, že i po všech modifikacích pro lepší stabilitu, je pořád velmi snadné narazit na různé problémy, které vedou ke kolapsu sítě a tím ke špatným výsledkům.

U některých výsledků a jejich relativnímu srovnání lze vidět, že zvolená metrika stále nemusí odpovídat seřazení člověka. I seřazení dle aktuálně hojně využívané metriky GAN, a to FID, nedokáže seřadit výsledky nejlépe dle řešeného úkolu.

Experimenty byly vyzkoušeny i na jiných architekturách, jako jsou DualGAN, DiscoGAN či UNIT (Unsupervised Image-to-Image Translation). DualGAN a DiscoGAN má však tak blízko k implementaci CycleGAN, že po prvních ne moc úspěšných pokusech experimentů mělo smysl spíše dále pokračovat v CycleGAN. Experimenty by se totiž pravděpodobně moc neliší od nastavování jiných parametrů u CycleGAN. [12, 4, 11]

U UNIT nebyl experiment dokončen, protože na testovací konfiguraci 2 trval příliš dlouho a průběžné výsledky byly horší než na CycleGAN. Experiment byl u UNIT spuštěn na optimalizované implementaci s AMP od NVIDIA z repozitáře [28] a byl po 30 hodinách běhu na testovací konfiguraci 2 zastaven. Průměrná doba trénování pro CycleGAN s AMP modifikací spuštěná na testovací konfiguraci 1 byla 1 den a 18 hodin pro 400 epoch se vstupním datasetem o velikosti 1375 obrázků a rozlišením 256 x 256 pixelů.

V oblasti syntézy realistických obrazů GAN již aktuálně nejsou *State of the art*. Napřed jsou aktuálně takzvané difúzní modely. Nicméně vznikají i hybridní architektury jako *Diffusion-GAN: Training GANs with Diffusion* [29], kde se využívají obě architektury a výsledné obrázky jsou od reálných těžko rozeznatelné.

# Kapitola 4

# Závěr

V této práci bylo v první kapitole, o modelech, důkladně teoreticky rozebrána problematika adversárních neuronových sítí a konrétně řešeného problému. Bylo tak zmíněno více architektur postavených na myšlence GAN. Dále bylo rozebráno jejich možné využití, různé výhody a nevýhody na různých datech a možná vylepšení. Z těchto myšlenek vznikl pak odhad na možný výsledek experimentů na datasetu z dodaných dat.

V další kapitole byla následně popsána příprava dat. Z dodaných zdrojových dat se podařilo vytvořit datasety, které byly následně využity v experimentech. Kapitola popisuje získání jednotlivých snímků z dodaných videí, a jejich případné spárování pomocí polohy. Zde se tak povedlo vytvořit datasety vhodné pro trénování vybraných neuronových sítí.

Nakonec se povedlo využít existující implementace a rozšířit je o různé modifikace. Na těchto vytvořených modelech pak byly spuštěné různé experimenty. Výsledky těchto experimentů byly důkladně zdokumentovány a odůvodněny. Z výsledků a různých problémů byly navrženy modifikace implementací a nakonec byly úspěšně implementovány. Průběh trénování byl pečlivě sledován a dle toho se odvíjely další experimenty. Během realizace jsem narazil na mnoho potíží a jejich vyřešení.

Na základě experimentů lze doporučit modely postavené na CUT, případně CycleGAN architektuře. Na základě zkušeností získaných z implementací a spouštění experimentů je dobré vždy klást důraz na průběžné sledování experimentů. V nejlepším případě je vhodné vybrat podobný jednodušší experiment a na něm zkoušet různé nastavení hyperparametrů sítě. Dále se také ukázalo, že vybrané metriky nejsou dostačující pro správné zhodnocení správnosti přenosu stylu z referenčního obrázku na vstupní, a stále bylo nutné lidské posouzení.

Z výsledku experimentů bylo částečně ukázáno, že GAN lze využít na zadaný úkol a to převodu denní scény obrázku na noční. Dále bylo ukázáno, že na vybraných obrázcích lze stále vidět značné nedostatky.

Závěrem lze tedy říci, že v této práci byly provedeny úspěšně experimenty pro zadaný úkol transformace obrázků ze dne na noc a byly zde uvedeny důležité poznatky, které jsou užitečné pro řešení podobných úkolů a syntézy obrazu pomocí neuronových síti.

# Literatura


1. NAGPAL, Kavya; FOOTE, Daniel; LIU, Yue; CHEN, Peter C; WULCZYN, Eric; TAN, Fei; OLSON, Nathan; SMITH, Jason L; MOHTASHAMIAN, Amir; WREN, Jonathan H; CORRADO, Greg S; MACDONALD, Ryan; PENG, Lily H; AMIN, Mahul B; EVANS, Andrew J; SANGOI, Ankur R; MERMEL, Craig H; HIPP, Jason D; STUMPE, Martin C. Development and validation of a deep learning algorithm for improving Gleason scoring of prostate cancer. *NPJ digital medicine*. 2019, roč. 2, s. 48. Dostupné z DOI: 10.1038/s41746-019-0112-2.
2. CHOLLET, François. *Deep Learning v jazyku Python: knihovny Keras, Tensorflow*. Přel. PECINOVSKÝ, Rudolf. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.
3. ISOLA, Phillip; ZHU, Jun-Yan; ZHOU, Tinghui; EFROS, Alexei A. *Image-to-Image Translation with Conditional Adversarial Networks*. arXiv, 2016. Dostupné z DOI: 10.48550/ARXIV.1611.07004. Přístup: 17. září 2022.
4. ZHU, Jun-Yan; PARK, Taesung; ISOLA, Phillip; EFROS, Alexei A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017. Přístup: 25. srpna 2022.
5. GOODFELLOW, Ian J.; POUGET-ABADIE, Jean; MIRZA, Mehdi; XU, Bing; WARDE-FARLEY, David; OZAIR, Sherjil; COURVILLE, Aaron; BENGIO, Yoshua. *Generative Adversarial Networks*. 2014. Dostupné z arXiv: 1406.2661 [stat.ML]. Přístup: 1. května 2022.
6. KINGMA, Diederik P; WELLING, Max. *Auto-Encoding Variational Bayes*. arXiv, 2013. Dostupné z DOI: 10.48550/ARXIV.1312.6114. Přístup: 25. května 2022.
7. HEUSEL, Martin; RAMSAUER, Hubert; UNTERTHINER, Thomas; NESSLER, Bernhard; HOCHREITER, Sepp. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. arXiv, 2017. Dostupné z DOI: 10.48550/ARXIV.1706.08500. Přístup: 12. srpna 2022.
8. MIRZA, Mehdi; OSINDERO, Simon. *Conditional Generative Adversarial Nets*. arXiv, 2014. Dostupné z DOI: 10.48550/ARXIV.1411.1784. Přístup: 23. července 2022.

9. DEMIR, Ugur; UNAL, Gozde. *Patch-Based Image Inpainting with Generative Adversarial Networks*. arXiv, 2018. Dostupné z DOI: 10.48550/ARXIV.1803.07422. Přístup: 30. července 2022.

10. PARK, Taesung; EFROS, Alexei A.; ZHANG, Richard; ZHU, Jun-Yan. *Contrastive Learning for Unpaired Image-to-Image Translation*. arXiv, 2020. Dostupné z DOI: 10.48550/ARXIV.2007.15651. Přístup: 12. října 2022.

11. YI, Zili; ZHANG, Hao; TAN, Ping; GONG, Mingyang. DualGAN: Unsupervised Dual Learning for Image-to-Image Translation. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, s. 2868–2876.

12. KIM, Taeksoo; CHA, Moonsu; KIM, Hyunsoo; LEE, Jung Kwon; KIM, Jiwon. Learning to Discover Cross-Domain Relations with Generative Adversarial Networks. In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. 2017, sv. 70, s. 1857–1865.

13. DENG, Jia; DONG, Wei; SOCHER, Richard; LI, Li-Jia; LI, Kai; FEI-FEI, Li. Imagenet: A large-scale hierarchical image database. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, s. 248–255. Dostupné z DOI: 10.1109/CVPR.2009.5206848.

14. HEUSEL, Martin; RAMSAUER, Hubert; UNTERTHINER, Thomas; NESSLER, Bernhard; HOCHREITER, Sepp. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. arXiv, 2017. Dostupné z DOI: 10.48550/ARXIV.1706.08500. Přístup: 19. října 2022.

15. SEITZER, Maximilian. *pytorch-fid: FID Score for PyTorch* [https://github.com/mseitzer/pytorch-fid]. 2020-08. Version 0.2.1, Přístup: 16. července 2022.

16. SZEGEDY, Christian; LIU, Wei; JIA, Yangqing; SERMANET, Pierre; REED, Scott; ANGUELOV, Dragomir; ERHAN, Dumitru; VANHOUCKE, Vincent; RABINOVICH, Andrew. *Going Deeper with Convolutions*. arXiv, 2014. Dostupné z DOI: 10.48550/ARXIV.1409.4842. Přístup: 28. září 2022.

17. WANG, Zhou; BOVIK, A.C.; SHEIKH, H.R.; SIMONCELLI, E.P. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*. 2004, roč. 13, č. 4, s. 600–612. Dostupné z DOI: 10.1109/TIP.2003.819861.

18. FOSSASIA. *Visdom* [https://github.com/fossasia/visdom]. GitHub, 2023. Přístup: 21. listopadu 2022.

19. BIEWALD, Lukas; TEAM. *Weights & Biases* [https://www.wandb.com/]. 2020. Přístup: 2. února 2023.

20. ZHU, Jun-Yan; PARK, Taesung; ISOLA, Phillip; EFROS, Alexei A. *PyTorch-CycleGAN-and-pix2pix: Image-to-Image Translation in PyTorch* [https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix]. [B.r.]. [Python].

21. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv, 2015. Dostupné z DOI: 10.48550/ARXIV.1505.04597. Přístup: 14. června 2022.

22. PARK, Taesung; LIU, Ming-Yu; WANG, Ting-Chun; ZHU, Jun-Yan. *Contrastive Unpaired Translation (CUT): PyTorch implementation* [https://github.com/taesungp/contrastive-unpaired-translation]. [B.r.]. [Python].

23. HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. *Deep Residual Learning for Image Recognition*. arXiv, 2015. Dostupné z DOI: 10.48550/ARXIV.1512.03385. Přístup: 5. října 2022.

24. NVIDIA. *NVIDIA Apex* [https://github.com/NVIDIA/apex]. 2021. Přístup: 20. ledna 2023.

25. MIYATO, Takeru; KATAOKA, Toshiki; KOYAMA, Masanori; YOSHIDA, Yuichi. *Spectral Normalization for Generative Adversarial Networks*. 2018. Dostupné z arXiv: 1802.05957 [cs.LG]. Přístup: 9. září 2022.

26. PYTORCH. *Autocasting (Automatic Mixed Precision) - CUDA ops that can autocast to float32*. 2021. Dostupné také z: https://pytorch.org/docs/stable/amp.html#cuda-ops-that-can-autocast-to-float32. Přístup: 25. ledna 2023.

27. KRIZHEVSKY, Alex; HINTON, Geoffrey. *CIFAR-10 and CIFAR-100 datasets*. 2009. Dostupné také z: https://www.cs.toronto.edu/~kriz/cifar.html. Přístup: 11. října 2022.

28. CORPORATION, NVIDIA. *Imaginaire: A Universal PyTorch Library for Image and Video Synthesis* [https://github.com/NVlabs/imaginaire]. [B.r.]. [online].

29. WANG, Zhendong; HUANGJIE, Zheng; PENGCHENG, He; WEIZHU, Chen; MINGYUAN, Zhou. *Diffusion-GAN: Training GANs with Diffusion* [arXiv preprint arXiv:2206.02262]. 2022. Dostupné také z: https://arxiv.org/pdf/2206.02262.pdf.