VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZVYŠOVÁNÍ KVALITY VIDEA POMOCÍ KONVOLUČNÍCH SÍTÍ

VIDEO ENHANCEMENT USING CONVOLUTIONAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DAVID SKÁCEL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL HRADIŠ, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Skácel David**
Obor: Informační technologie
Téma: **Zvyšování kvality videa pomocí konvolučních sítí**
Video Enhancement Using Convolutional Networks
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte základy konvolučních sítí.
2. Vytvořte si přehled o současném využití konvolučních sítí v počítačovém vidění a o metodách pro zvyšování kvality fotografií nebo videa.
3. Navrhněte konkrétní metodu pro zvyšování kvality obrazu využívající konvoluční sítě.
4. Připravte si databázi vhodnou pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty nad datovou sadou.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte krátké video prezentující vaši práci, její cíle a výsledky.

Literatura:

- HRADIŠ Michal et al.: Convolutional Neural Networks for Direct Text Deblurring. In: *Proceedings of BMVC 2015*, 2015.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Hradiš Michal, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Konvoluční neuronové sítě dnes představují v oblasti zpracování obrazu jeden z nejmodernějších přístupů k řešení problémů, jakými jsou například redukce kompresních artefaktů či zvyšování prostorového rozlišení obrazu. Některé výzkumné skupiny již dokazují, že lze tyto sítě adaptovat ke zpracování videa a využít tak přidané informace v čase ke zvětšení prostorového rozlišení videa či dosáhnout lepší úrovně komprese při zachování detailů. Otázkou, zdali je možné využít tento přístup také pro zvýšení časového rozlišení reálného videa, se zabývám v této práci. K tomu využívám konvolučních neuronových sítí, které, jak popisuji, dokáží do jisté míry interpolovat vstupní videosnímky ze skutečných videozáznamů, jsou-li dostatečně kvalitní, a napomoci tak zvýšení snímkové frekvence videa. Dosažené výsledky, ač pozitivní, jsou spíše mezikrokem na cestě za vhodnějším využitím těchto sítí k řešení daného problému.

Abstract

Convolutional neural networks (CNN) represent a state-of-the-art approach to non-trivial image processing tasks, including compression artifacts reduction and image super-resolution. As some research groups nowadays show, these networks can also be leveraged to perform such tasks on real-world video data, resulting in video spatial super-resolution and more. The main goal of this work is to determine whether these nets can be adjusted to perform temporal super-resolution of real-world video data. I utilize the aforementioned neural net architectures in this paper to do so. As I show, given that the input videos are of reasonable quality, these nets are capable of double-image interpolation up to a certain level, where the output image is usable for temporal upsampling. Although the presented results are promising, I encourage more research to be done on this topic.

Klíčová slova

Hluboké učení, strojové učení, hluboké neuronové sítě, konvoluční neuronové sítě, video, kvalita obrazu, restaurace obrazu, vylepšení, rozlišení v čase, snímková frekvence, interpolace snímků videa

Keywords

Deep learning, machine learning, deep neural networks, convolutional neural networks, video, image quality, image restoration, enhancement, temporal resolution, frames per second, fps, video image interpolation

Citace

SKÁCEL, David. *Zvyšování kvality videa pomocí konvolučních sítí*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Zvyšování kvality videa pomocí konvolučních sítí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Hradiše, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
David Skácel
18. května 2017

Poděkování

Tuto práci bych nebyl schopen vypracovat nebýt vedení Ing. Michala Hradiše, Ph.D., jemuž tímto děkuji za trpělivost ve vedení a snahu naučit nás vše podstatné skrze semináře a osobní konzultace.

Obsah

Kapitola 1

Úvod

Z hlediska šíření obrazových dat hraje podstatnou roli komprese za účelem snížení velikosti dat. Ať se jedná o jednotlivé fotografie či snímky svázané v čase, představuje většinou kódování jistou ztrátu entropie obrazové informace. Tu tradičně vnímáme jako snížení kvality obrazu formou různorodých artefaktů, jakými jsou například čtvercové struktury, rozmazání atp. Je tedy třeba stanovovat práh mezi úrovní komprese a ztrátou obrazové informace.

V současnosti představují konvoluční neuronové sítě jednu z nejmodernějších metod pro zpracování obrazu za rozličnými účely. Do této oblasti spadá například sémantická segmentace obrazu [19], rozpoznávání objektů v obraze [17], zvyšování kvality obrazu, jakým je například odstranění šumu [13, 25, 6], oprava rozmazání [11, 26], odstranění následků komprese [22, 4] či zvyšování rozlišení [5, 16, 1, 15] a jiné. V této práci se zabývám důsledky aplikace konvolučních neuronových sítí pro rekonstrukci obrazu na datové sadě vytvořené z videozáznamu.

Jednu z hojně využívaných standardizovaných kompresních metod představuje metoda JPEG[23]. Z existujících metod zaměřených na rekonstrukci obrazu komprimovaného zmíněnou metodou dosahují v posledních letech konvoluční neuronové sítě jedny z nejlepších výsledků [4, 22]. Pro účely této práce jsem tedy vycházel také z těchto sítí. Jelikož se však tato práce zaměřuje především na video a v době jejího vzniku se ve vědecké obci nevyskytovaly články zabývající se touto tematikou, bylo třeba navrhnout architekturu sítě, která by využila zmiňované metody najednou na více snímcích provázaných v čase. Tento problém představuje hlavní cíl této práce, kterým je zjistit, zdali tradiční konvoluční sítě pro rekonstrukci obrazu dokáží využít přidané hodnoty obrazových dat v čase a zvýšit tak některou z kvalit videozáznamu.

Jednou z těchto kvalit je právě snímková frekvence videa. Motivací této práce je tedy také vyzkoumat, jestli taková síť dokáže vytvářet obraz, který časově zapadá mezi vstupní snímky a jak dalece se tento obraz liší od skutečného snímku. Vycházím přitom z předpokladu, že vstupní video obsahuje reálný záznam videokamery s krátkou uzávěrkou zaznamenaný za dostatečných světelných podmínek proto, aby zaznamenané objekty nebyly rozmazány. Díky tomu nenahlížím na řešený problém z hlediska opravy rozmazání, což by vyžadovalo jiný přístup.

Mnou navrhované architektury se skládají ze tří základních bloků. První je blok interpolační, druhý je blok zaměřený na odstranění následků komprese a třetí je blok „residuální“. Jak ukazuje tato práce, lze použití všech zmiňovaných částí do jisté míry využít k řešení obou problémů jednou sítí.

Výsledkem práce je tak konvoluční neuronová síť, která dokáže najednou zpracovat dva vstupní snímky, na které byla aplikována JPEG komprese, zrekonstruovat tyto snímky a

zároveň do jisté míry vytvořit snímek třetí, který lze vzhledem k časové ose mezi původní snímky dosadit. Tímto způsobem je možné navýšit kvalitu obrazu videa, která byla snížena danou kompresí a zároveň zdvojnásobit jeho snímkovou frekvenci. Dalším produktem jsou také konvoluční sítě, jejichž jediným záměrem je interpolovat vstupní snímky či pouze odstraňovat blokovou strukturu po kompresi.

V následujících kapitolách se zabývám existujícími řešeními zmíněných problémů. Dále tím, které z těchto řešení se mi podařilo zopakovat, ze kterých vycházím a které metody se neukázaly býti příliš vhodné. Dále rozebírám kroky, kterými jsem postupoval při aplikaci těchto metod v praxi a jak mě tyto metody dovedly k mým závěrům.

Kapitola 2

Přehled současných možností zvýšení kvality obrazu

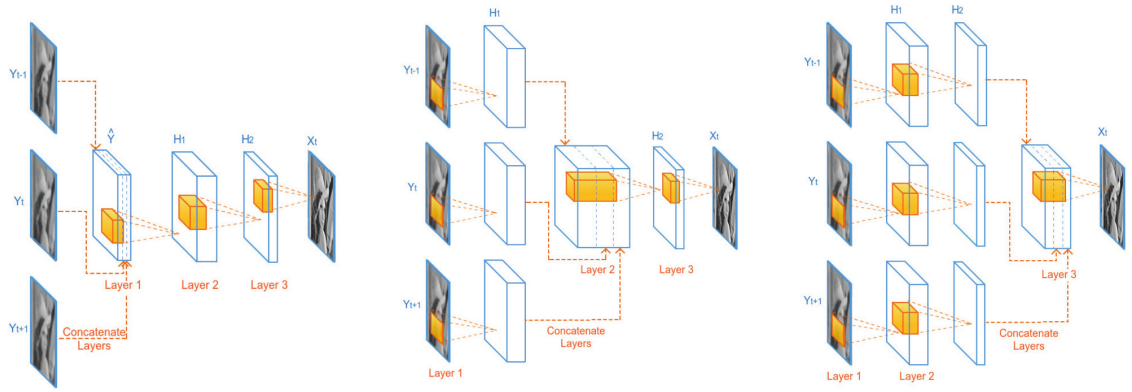
V této kapitole se věnuji teorii konvolučních neuronových sítí a existujícím řešením metod obnovení obrazu. Na závěr popisuji použité metriky pro objektivní vyhodnocení kvality zpracovaných dat.

2.1 Obnovení obrazu

Oprava následků komprese JPEG. Metoda JPEG[23] je dnes stále hojně využívanou kompresní metodou založenou na diskrétní cosinové transformaci (DCT) obrazového signálu a Huffmanově kódování. Tato metoda však zavádí nechtěné artefakty. Jedná se o čtvercové struktury a „zvonivý“ efekt (z angl. *ringing effect*), které touto kompresní metodou v obraze vznikají. Ty jsou následkem odstranění přebytečné informace pomocí DCT a následného kompresního kódování (Huffman, D.A., 1962). Z hlediska videa hraje tato metoda také roli. Dala vzniknout videoformátu MPEG[18], jehož novější formy jsou dnes stále používané. Dá se předpokládat, že dnes stále existuje nezanedbatelné množství dat kódovaných touto, či podobnou kompresí, a proto má smysl se jí stále zabývat. Dalším důvodem je možnost jejího využití v podobě odrazového můstku pro zkoumání konvolučních neuronových sítí.

Opravu JPEG komprese dnes zvládají konvoluční sítě velice dobře. Jak ukazuje skupina Dong *et al.*[4], zdánlivě jednoduchá architektura mělké sítě je schopna opravovat následky JPEG komprese na špičkové úrovni. Navrhovaná ARCNN vykazuje schopnost opravit následky komprese i při její vysoké úrovni (q10). Jedním z navrhovaných přístupů je předtrénovávání sítě na nižší úrovně komprese a poté převedení parametrů jedné vrstvy do vrstvy sítě druhé. Takové trénování však trvá déle už z podstaty vícefázového trénování a také protože síť konverguje k nízké ztrátě relativně pomalu. Jak sami zmiňují, již trénování pětivrstvé ARCNN bez předtrénovávání je nestabilní. Svoboda *et al.*[22] tento přístup dále prohlubuje a zavádí tzv. residuální síť[9] (z angl. *residual*) do problematiky opravy komprese konvoluční sítí. V této práci z obou sítí vycházím.

Prostorové rozlišení V době vzniku práce ještě nebyly dostupné výsledky prací zabývajících se tematikou konvolučních sítí pro zpracování dat tvořených videozáznamem. V průběhu této práce však bylo publikováno několik prací s podobnou tematikou. Mnou zde navrhované interpolační sítě nezávisle na Kappeler *et al.*[15] rovněž využívají konkatenace snímků již před první vrstvou, jak je vidět na obrázku (2.1) v architektuře *a*).



Obrázek 2.1: Konkatenace snímků videa podle Kappeler *et al.*. Obrázek převzat z [15].

Jejich síť však řeší pouze problematiku zvyšování prostorového rozlišení obrazu. Za stejným účelem Caballero *et al.* [1] dále rozvíjejí tento přístup o kompenzaci pohybu.

Časové rozlišení. Touto tematikou se zabývá Shahar *et al.*[21]. Jejich přístup ke zvyšování snímkové frekvence videa se však ještě nedotýká problematiky neuronových sítí a je zaměřen především na opakující se pohyb objektu v obraze. Zde uváděné řešení se od tohoto přístupu liší. Je totiž zaměřeno na generalizaci neuronové sítě na širší oblast reálných videí bez zaměření na opakovaný pohyb konkrétních objektů.

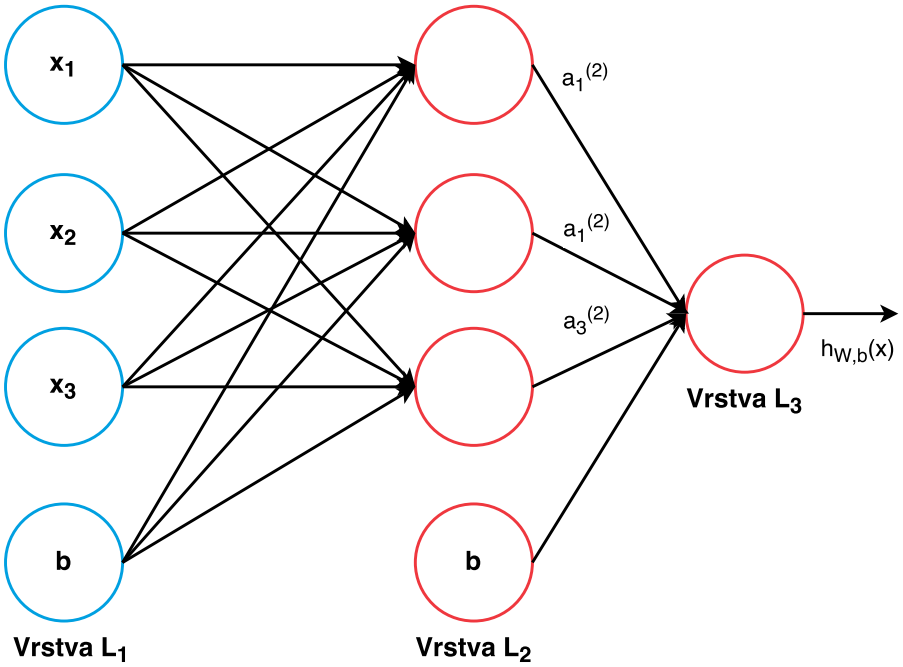
2.2 Neuronové sítě

Neuronové sítě a hluboké učení. Neuronové sítě v oblasti informatiky jsou inspirovány biologií. Jedná se o matematický model zpracování dat, který je založen na základní abstrakci fungování lidského mozku. Základní jednotkou takové sítě je model neuronu. Ten Ng *et al.*[20] definují jako

$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{n} W_i x_i + b) \ , \tag{2.1}$$

kde h představuje výstup neuronu a n je počet jeho vstupů x. Tento neuron je reprezentován svými parametry. Těmi jsou váhy W_i a vychýlení b (z ang. *weights* a *bias*). Tyto váhy se v neuronu násobí s patřičným vstupem neuronu. Na výstupu má pak zpravidla každý neuron aktivační funkci f, která vnáší do modelu nelinearitu. Její výstup je pak případně navázán na neurony v další vrstvě nebo na ztrátovou funkci. Jednotlivé neurony tedy mohou být různým způsobem propojeny ve vrstvách a tvořit tak potencionálně hlubokou síť (odtud *hluboké učení*). Jednou z variant propojení neuronů jsou sítě *plně propojené* (z ang. *fully connected*). Jejich schéma je vidět na obrázku 2.2. Aby však tato dopředná síť měla nějaký smysl, musí se její parametry nastavit tak, aby neprováděla pouze náhodné operace. Takovému nastavování parametrů se říká *učení*. Celá síť představuje nelineární funkci. Ta je modelem pomyslné funkce představující řešený problém, přičemž hledáním minima ztrátové funkce (resp. učením) se snažíme dosáhnout co nejlepší aproximace.

V případě této práce se jedná o tzv. **učení s učitelem** (z ang. *supervised learning*), což znamená, že kromě vstupních dat máme také data vzorová. Učení sítě se skládá ze dvou



Obrázek 2.2: Schéma jednoduché plně propojené neuronové sítě. Obrázek převzat z [20].

základních kroků. **Dopředného** a **zpětného** průchodu. Při průchodu dopředném síť propaguje vstupní data skrze vrstvy sítě na její konec, kde tvoří vstup tzv. **ztrátové funkce** (z ang. *loss function*). Tato ztrátová funkce má na výstupu chybu oproti vzorovým datům. Tato chyba se pak při průchodu zpětném propaguje skrze síť v opačném směru a nastavují se parametry sítě. A to tak, aby při příštím dopředném průchodu byla chyba menší. Hledání takových parametrů pro síť, která má řádově pár desítek neuronů, se dá teoreticky řešit i náhodným generováním parametrů. V praxi však mohou mít neuronové sítě řádově statisíce, až miliony parametrů. Proto se využívá matematické analýzy. Minimum ztrátové funkce lze hledat s pomocí derivací. K těm lze využít řetízkové pravidlo pro derivaci složených funkcí. Derivací funkce v daném bodě získáme směrnici. Blížíme-li se minimu funkce, směrnice se bude blížit nule. Tohoto principu využívá jeden z nejpoužívanějších algoritmů současnosti, tzv. „stochastic gradient descent“(SGD)(Orr, 1995; Schraudolph, 1999, 2002; Graepel and Schraudolph, 2002), což lze přeložit jako náhodný gradientní sestup. Tento algoritmus je založený na „gradient descent” algoritmu[2], který, jak již název napovídá, slouží k hledání minima pomocí gradientního sestupu [8]. Základního SGD algoritmu pro trénování využívám, až na jednu výjimku, také.

Konvoluční neuronové sítě. Konvoluční sítě tvoří podmnožinu neuronových sítí. Tyto sítě obsahují přinejmenším jednu skrytou vrstvu, což je zjednodušeně označení pro vrstvy mezi vstupem a výstupem sítě. A to vrstvu konvoluční. Jsou pak uzpůsobeny tak, aby mohly zpracovávat dvoudimenzionální data, jakými jsou například klasické obrázky. Konvoluční vrstva je reprezentována tzv. *filtry*, které se také nazývají *konvoluční jádra*. Konvoluční vrstva může mít n filtrů o velikosti $kernel_c \times kernel_w \times kernel_h$, kde $kernel_w$ je šířka jádra, $kernel_h$ je jeho výška a $kernel_c$ je hloubka, která je rovna počtu kanálů ve vstupu, tedy počtu filtrů předchozí vrstvy. Pro oba platí $kernel_w, kernel_h \leq w_i, h_i$ kde w_i, h_i je šířka a výška vstupu konvoluční vrstvy. Tyto filtry se procesem učení učí detekovat speci-

fické vlastnosti ve vstupním obraze, jakými jsou například hrany. Konvoluční jádra provádí matematickou operaci diskrétní konvoluce nad vstupním obrazem. Konvoluční vrstva jednotlivými filtry postupně prochází celý vstup tak, že jimi posouvá po vstupním obraze po osách výšky a šířky a provádí konvoluce nad těmito výřezy přes celou hloubku. Obvykle se používají konvoluční jádra se stejnou šířkou a výškou. Jsou-li jádra konvolucí posouvána vždy po 1 pixelu, jak bývá u konvolučních sítí s obrazovým výstupem zvykem, je výstupní obraz konvoluční vrstvy zmenšen následovně[1]:

$$h_{vystup} = \frac{h_i - kernel_h}{stride_h + 1} \qquad w_{vystup} = \frac{w_i - kernel_w}{stride_w + 1} \ , \tag{2.2}$$

kde h_{vystup} a w_{vystup} jsou výstupní výškou a šířkou. Dále $stride_w$ a $stride_h$ jsou délky posunutí konvolučního jádra po obou osách. Pakliže tedy posouváme konvoluční jádro o velikosti 3×3 se $stride_{h,w} = 1$, pak bude velikost výstupního obrazu snížena o $kernel_{h,w} - 1$ v obou osách. Důležité je také zmínit, že síť je invariantní vůči rozměru vstupního obrazu. Z hlediska učení i pouhého vyhodnocení je tak prakticky možná variace v rozměrech vstupních a vzorových dat. V praxi lze však vídat využití spíše konzistentních rozměrů.

Aktivační funkce V posledních letech se jako aktivační funkce konvolučních sítí hojně využívá tzv. ReLU(rectified linear unit)[10]. Tuto funkci Caffe[2] definuje jako:

$$y(x) = \max(0, x) \ , \tag{2.3}$$

kde x je vstup této funkce y, který je tvořený výstupem předešlé vrstvy. V této práci využívám ReLU pro zavedení nelinearit do navrhovaných sítí jako aktivační funkci mezi všemi konvolučními vrstvami. Je-li tedy nějaká konvoluční vrstva následována například konkatenací nebo ztrátovou vrstvou, její výstup není usměrňován pomocí ReLU.

2.3 Metriky pro hodnocení kvality obrazu

Ke zhodnocení zřetelné kvality obrazu využívám dvou standardizovaných metrik, které výše zmiňované práce také většinově využívají pro objektivní zhodnocení kvality obrazu.

PSNR. „Peak signal-to-noise ratio" je metrika používaná k měření kvality signálů. Její využití k měření kvality obrazu a videí je stále vídané. Alespoň tehdy, nedochází-li u videí ke změně kodeku, jak píší Huynh-Thu a Ghanbari[12]. Tento fakt je pro tuto práci důležitý a ke změně kodeku mezi srovnávaným obrazem zde nedochází. Tato metrika je logaritmická, její jednotka je v decibelech (dB) a vychází z chyby MSE(Mean squared error). MSE [8] je nad testovací sadou definována jako

$$MSE_{test} \quad = \quad \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(test)} - y^{(test)})_i^2 \ , \tag{2.4}$$

kde $\hat{y}$ je výstupní obraz sítě, y je obraz vzorový a m je velikost testované minisady. MSE tedy představuje aritmetický průměr chyby dvojic vzorový – výstupní obraz.

[1] Takto specifikuje konvoluční vrstvu nástroj Caffe [14], http://caffe.berkeleyvision.org/tutorial/layers/convolution.html

[2] http://caffe.berkeleyvision.org/doxygen/classcaffe_1_1ReLULayer.html

PSNR je pak podle Huynh-Thu a Ghanbari[12] pro měření kvality videa definováno jako:

$$PSNR = \frac{1}{N}\sum_{i=1}^{N} 10log\Big(\frac{255^2}{\varepsilon^2(i)}\Big) \ , \tag{2.5}$$

kde N je počet snímků ve videu. V mém případě vyhodnocuji dvojice snímků zvlášť[3], takže N představuje počet snímků v testovací sadě. Dále $255^2/\varepsilon^2(i)$ popisují jako MSE nad svítivostí pixelů snímku i. V mém případě se jedná o MSE nad pixely barevných kanálů r,g,b snímku i. Dělenec 255^2 představuje MAX^2 - kvadrát maximální hodnoty datového typu obrazu, nad kterým je metrika počítána.

SSIM. „Structural similarity index“[24] je druhou z metrik, která představuje jiný přístup k porovnávání strukturální podobnosti dvou obrázků. Využívám implementaci, kde šířka okénka pro porovnávání je odvozena z Gaussovy váhovací funkce se standardní odchylkou $\sigma = 1,5$. Níže uváděné hodnoty *SSIM* ve skutečnosti představují *MSSIM*, tedy průměrnou hodnotu *SSIM* nad celým snímkem. Tyto dva pojmy uváděné práce často zaměňují.

[3] Neporovnávám výsledná videa, jelikož interpolovaný snímek má nesrovnatelně nižší kvalitu oproti snímkům původním.

Kapitola 3

Postup při navrhování architektur konvolučních sítí pro interpolaci videosnímků.

V této kapitole hovořím o navrhovaných architekturách hlubokých konvolučních sítí. Popisuji způsob jejich učení, význam jejich částí a nakonec vysvětluji datové sady, které jsem vytvořil, a jejich význam.

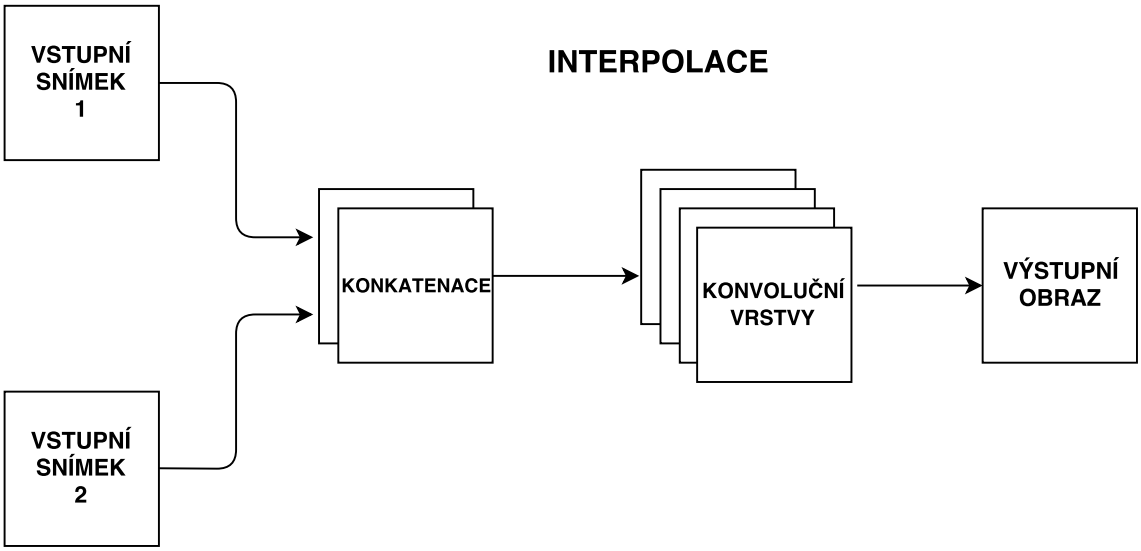
3.1 Navrhované sítě

Mnou navrhované sítě se skládají z několika základních bloků. Jedním z nich je blok pro opravu komprese, který je představován čtyřvrstvou konvoluční sítí. Dále pak blok pro interpolaci vstupních snímků, který je ve všech případech reprezentován konkatenací před první vrstvou a dále následován několika konvolučními vrstvami. Za samostatný blok zde považuji i odhad výstupu jednotlivých bloků, jelikož se v tomto případě bloky učí počítat pouze rozdíl. Takové sítě zde označuji jako „residuální" (z angl *residual networks*). Z podstaty konvolučních vrstev je nutné vyřešit zmenšování výstupů v závislosti na velikosti konvolučních jader. Toto u všech sítí kompenzuji vlastností *okrajů nul* (z angl. *zero-padding*), jak popisuji v sekci o konvolučních sítích (2.2). Aby byl na výstupu možný opět obrázek o stejném počtu kanálů, má poslední konvoluční vrstva na výstupu daný počet kanálů odpovídající vstupnímu obrázku. Schémata těchto sítí jsou součástí přílohy A. Všechny následující sítě byly, až na jednu výjimku, trénovány pomocí algoritmu Stochastic Gradient Descent, který popisuji v kapitole 2.2.

Způsob učení těchto sítí. Všechny zde trénované sítě se učí pomocí tzv. „Euklidovské ztrátové vrstvy", která se během trénování nachází za výstupní vrstvou všech v této sekci uváděných sítí. Tato funkce má stejný účel jako MSE(2.4) a sice spočítat chybu mezi dvěma stejně velkými vektory. Caffe ji definuje jako[1]

$$EuclideanLoss = \frac{1}{2N}\sum_{i=1}^{N}||(X_i^1 - X_i^2)||_2^2 \quad , \tag{3.1}$$

[1] http://caffe.berkeleyvision.org/tutorial/layers/euclideanloss.html



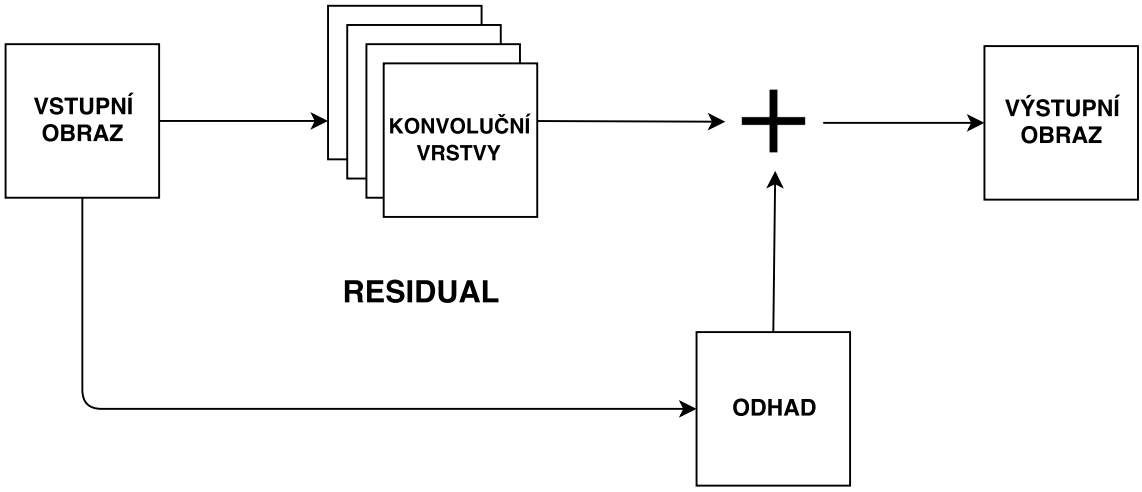
Obrázek 3.1: Schéma interpolačního bloku.

kde X^1 je výstupní obraz, X^2 obraz vzorový a i představuje pixel obrazu o N pixelech. Jedná se v podstatě o $\frac{1}{2}MSE$(2.4).

Oprava následků komprese. Pro tyto účely jsem implementoval čtyřvrstvou residuální síť, která je založená na ARCNN skupiny Dong *et al.*[4] a inspirovaná přístupem skupiny Svoboda *et al.*[22], který uvádí v praxi tzv. residuální síť pro opravu komprese. Dříve byla residuální architektura představena skupinou He *et al.*[9] za účelem rozpoznávání. Mnou implementovaná variace na ARCNN (dále také jako **4l-residual**) byla natrénována na datové sadě BSDS500[2] a jejích pod-sadách *train* a *val* a testována na pod-sadě *test*. Oproti odkazovaným však využívám barevnou verzi obrázků. Zde uváděné výsledky experimentů jsou na datové sadě *Ferrari* (3.2), ve které trénovací vstupní snímky byly uloženy ve formátu JPEG s kvalitou $q = 60$. Předtrénovaný blok této čtyřvrstvé residuální sítě dále slouží pro zpracování vstupních snímků sítě největší. Tou je níže popisovaná **vnet**. Schéma této sítě lze nalézt v příloze na obrázku A.3. Rovněž tuto síť popisuje schéma na obrázku 3.2. Oproti původní odkazované čtyřvrstvé *ARCNN*(9-7-1-5) se mnou implementovaná síť liší ve velikosti konvolučních jader *4l-residual*(11-3-3-5). Rovněž počty výstupů byly zvýšeny z *ARCNN*(64,32,16) na *4l-residual*(48,64,64). Jak je však patrné z výsledků práce, zvýšení počtu parametrů sítě zmíněným způsobem se zdá pro tento účel zbytečné. Této síti slouží jako odhad výstupu vstupní snímek.

Interpolace vstupních snímků pro doplnění obrazu v čase. Tento blok zde nazývám interpolačním, protože má za úkol ze dvou snímků vytvořit snímek nový tak, aby časově zapadal mezi ně. Tento blok jsem podobně jako již zmiňované práce založil na 4 až 5 konvolučních vrstvách následujících konkatenaci vstupních snímků. Toto znázorňuji schématem na obrázku 3.1. Stejně tak to řeší i Kappeler *et al.* se svojí architekturou *a*) na obrázku 2.1.

[2] https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html



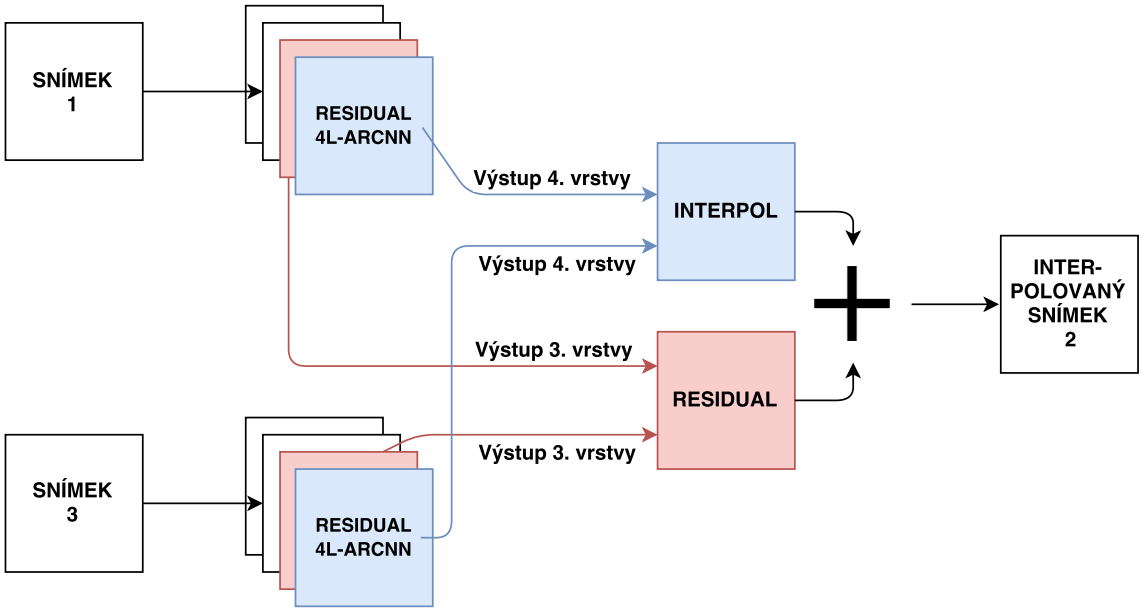
Obrázek 3.2: Schéma residuálního bloku.

Residuální sítě. Z časového hlediska trénování se projevily jako výhodné sítě residuální, které nejrychleji konvergují k přijatelným výsledkům a umožňují vyšší koeficient učení (z ang. „learning rate") [9, 22]. Tohoto principu v této práci využívám opakovaně.
Většina bloků uváděných v této podkapitole je navržená tak, že se učí počítat pouze rozdíl mezi odhadem výstupu a samotným výstupem. Podstatou tohoto bloku je, že se před ztrátovou vrstvou sečte odhad s výstupem poslední vrstvy sítě a dohromady dají výsledný obraz.

Síť 4l-res-iNET. Tato čtyřvrstvá interpolační residuální konvoluční síť tvoří druhou polovinu níže uváděné sítě **vnet**, zaměřenou na interpolaci chybějícího snímku. Je založena na brzké konkatenaci vstupních snímků, jež jsou následně zpracovány konvolučními vrstvami. Ty jsou zde čtyři o velikostech jader *4l-res-iNET*(11-3-3-5). Stejně jako ostatní sítě využívá zero-padding. Této síti slouží jako odhad výstupu průměr vstupních snímků (níže uváděno jako *merge*). Schéma interpolace je znázorněno na obrázku 3.1

Síť vnet. Tato síť využívá všechny výše zmíněné bloky. Na vstupu jsou oba snímky zpracovány čtyřvrstvou residuální konvoluční částí předtrénovanou pro opravu komprese. V rámci učení této sítě jsou tyto bloky zafixovány nastavením jejich *learning rate* na $lr = 0$. Na rozdíl od pouhých interpolačních sítí tvoří vstup její interpolační části konkatenace nikoliv vstupních snímků, ale výstupů třetích vrstev předešlé části, jak lze vidět na obrázku 3.3. Dále má rovněž čtyři konvoluční vrstvy, které se učí počítat rozdíl mezi skutečným vzorem a odhadem. Jako odhad residuální části pro interpolaci však slouží již zrekonstruované snímky, které jsou na výstupu čtvrtých vrstev první části zprůměrovány. Síť vnet má tedy jako jediná **tři výstupy**, namísto jednoho. Velikosti konvolučních jader jsou zachovány tak, jak popisuji v předešlých odstavcích. Kompletní schéma této sítě lze vidět v příloze na obrázku A.4.

Síť 5l-res/std-iNET. Tyto dvě pětivrstvé interpolační konvoluční sítě využívají již zmiňovaného principu brzké konkatenace vstupních snímků. Obě jsou velice podobné síti předešlé, přičemž jsou o jednu konvoluční vrstvu hlubší. Zatímco *5l-res-iNET* je residuální, *5l-std-iNET* nikoliv. Residuální síti slouží jako odhad výstupu *merge* vstupních snímků.



Obrázek 3.3: Schéma sítě vnet složené z popisovaných bloků. Blok *residual* zde představuje pouhý merge.

Residuální verzi této sítě jsem trénoval pomocí učícího algoritmu ADADELTA [27], který je založený na algoritmu Stochastic Gradient Descent. Tento algoritmus se během trénování adaptuje a sám stanovuje *learning rate*. Jak popisuji níže, s pomocí tohoto algoritmu se podařilo tuto síť natrénovat v rozumném čase, po nezdárném hledání optimálního nastavení hyperparametrů sítě. Sítě jsou koncipovány následovně: velikosti konvolučních jader jsou *5l-std-inet*(5-3-5-3-3) a *5l-res-inet*(7-5-5-3-3), počty výstupů jsou pak *5l-std-inet*(64,64,32,32) a *5l-res-inet*(32,48,64,96). Schéma residuální sítě je popsáno v příloze na obrázku A.1 a standardní verze je popisována obrázkem A.2.

3.2 Trénovací a testovací sada vytvořená z videí

Datová sada, na které se síť učí zpracovat obraz je přinejmenším stejně důležitá, jako architektura sítě samotné. Videa pro tyto datové sady byla vybrána z reálných záznamů o rozlišení 1920×1080 při 60 snímcích za vteřinu. Tyto záznamy byly pořízeny převážně s pomocí menších kamer, které se používají pro létající drony, záznamy z palubové desky atp. Trénovací sada (**YT_train**) obsahuje 59 těchto videí a testovací sada (**YT_test**) 8 videí se stejným zaměřením. Tyto sady obsahují videa s následujícím obsahem: Záznamy z jedoucích aut a motocyklů, létajících dronů, horských drah. Dále záznamy používání středověkých a moderních sečných a střelných zbraní. Pak také záznamy z adrenalinových sportů, jízd na kolech, plavání. Vytváření databáze popisuji v kapitole 4.4. Z trénovací sady **YT_train** dále vznikla její podmnožina, sada **YT_trainS**, která byla ručně protříděna s pomocí *optical flow* algoritmu [28]. Tímto způsobem vznikla mnohem menší sada, která by však měla obsahovat méně nevhodných snímků, které buďto neobsahují pohyb nebo obsahují pohyb příliš velký. Pro výjimečné případy byla dále použita **testovací** sada **Ferrari**, která představuje kvalitní záznam vybočující z řady videí trénovací sady. Tato byla dále využita pro testování sítě *4l-residual* se vstupem komprimovaným metodou *JPEG(q60)*.

Pro zkoumání interpolace pro různá posunutí jsem vytvořil sadu trojic snímků, které pocházejí ze čtyř různých mnou natočených videí kvality odpovídající předešlým sadám. Tyto snímky jsem posouval do osmi směrů tak, že každý snímek byl posunut čtyřikrát po jedné ose a čtyřikrát diagonálně. Diagonální posunutí představuje větší vzdálenost. Tyto snímky byly takto posunuty o $2 - 16.97$ pixelů. Je důležité upozornit na fakt, že je posunut celý obraz. Tedy všechny pixely stejně. Toto představuje jakýsi ideální stav, který lze velice jednoduše interpolovat i bez užití konvolučních sítí. Zde však slouží pouze k testování.

Kapitola 4

Implementace navržených sítí

V následující kapitole vysvětluji, jakým způsobem jsem implementoval popisované architektury. Dále hovořím o použitých nástrojích pro implementaci konvolučních neuronových sítí, jejich trénování, vyhodnocení, využití ke zpracování obrazu a videa a také pro práci s velkým množstvím těchto dat.

4.1 Použité nástroje

Caffe framework [14]. Pro implementaci a trénování neuronových sítí existuje celá řada nástrojů. Jmenovitě například Theano, Torch, Keras, Tensorflow či Caffe[1]. Jelikož se jedná o výpočetně náročnou operaci, využívá se hardwarové akcelerace s pomocí grafických čipů, které jsou pro paralelní výpočty nad vektory vhodnější. Pro účely této práce posloužil Deep learning framework Caffe, vyvíjený v Berkeley Artificial Intelligence Research. Nutno podotknout, že dnes již existuje jeho druhá verze, Caffe 2, která je pro tuto činost v mnoha ohledech vhodnější.

Hardwarová akcelerace. Výše zmiňovaný framework umožňuje využít HW akceleraci napsanou pro rozhraní CUDA, které je platformou společnosti NVIDIA a umožňuje tak akceleraci skrze jejich grafické čipy. Dále využívám optimalizačních knihoven pro tuto platformu určených pro hluboké učení - cuDNN.[2]

FFmpeg.[3]. Pro zpracování videa využívám nástroje FFmpeg, který jakožto multiplatformní nástroj pro nahrávání, kódování a „streamování“ audia a videa posloužil účelům níže uváděným v kapitole 4.4.

OpenCV.[4]. Pro načítání, zobrazování, ukládání a počítání posunu v obraze používám nástroj OpenCV ve verzi 3.1. Tento multiplatformní nástroj je vyvíjen pod BSD licencí a pro zdejší účely využívám jeho rozhraní pro jazyk Python. Z jeho možností využívám především funkcí pro základní práci s obrazem, případně při zpracování videa sítí. Ale i v tomto případě tento nástroj využívá výše zmíněného nástroje FFmpeg.

[1] Nástroje pro hluboké učení: http://deeplearning.net/software/theano/, http://torch.ch/, https://keras.io/, https://www.tensorflow.org/, http://caffe.berkeleyvision.org, https://caffe2.ai/.

[2] NVIDIA CUDA a cuDNN: https://developer.nvidia.com/cudnn, http://www.nvidia.com/object/cuda_home_new.html

[3] https://ffmpeg.org/

[4] http://opencv.org/

Youtube-dl.[5]**.** Pro obstarávání reálných videozáznamů jsem využil především webového serveru YouTube[6], který je dnes jedním z nejrožšířenějších volně dostupných zdrojů těchto dat. Nástroj youtube-dl mi umožnil skrze své jednoduché rozhraní v příkazové řádce výběr kvality stahovaných videí, včetně podpory seznamů, atp.

Skriptovací jazyky. Práce s datovou sadou vyžaduje mnohdy operace s velkým počtem obrazových souborů najednou. Pro tyto účely využívám skriptovacích jazyků GNU Bash[7] a Python 2.7.10[8] s knihovnou Numpy[9]. Podpůrné skripty v jazyce Bash mi umožňují práci až s miliony souborů najednou. Skripty v jazyce Python zahrnují masové ořezávání snímků na pod-snímky, vytváření LMDB databáze pro trénování, atp. Ty nejdůležitější z nich jsou v příloze na DVD.

4.2 Trénování

Před začátkem trénování je třeba inicializovat váhy. K tomu slouží celá řada metod, mezi které spadá například Gaussovo náhodné rozložení či metoda inicializace Xavier podle Glorot a Bengio [7]. Druhé zmiňované metody využívám k nastavení vah u všech zde navrhovaných sítí.

Hyperparametry trénování. Caffe definuje veškeré hyperparametry[10] (parametry trénovacího procesu, nikoliv sítě samotné) v souboru formátu .prototxt[11]. Mezi tyto hyperparametry se řadí například `learning rate`, který stanovuje koeficient učení (rychlost trénování vzhledem k učícímu algoritmu). Je-li rychlost příliš velká, chyba aproximační funkce může velice rychle dosáhnout hodnot v nekonečnu. V opačném případě může trénovací proces konvergovat nepoužitelně pomalu. Pro sítě využívající SGD tento hyperparametr nastavuji manuálně. K tomuto hyperparametru se váže `stepsize`, který při `lr_policy = step` určuje, jak často se `learning rate` vynásobí hyperparametrem `gamma`. Tímto způsobem je možné nastavit postupné snižování learning rate.

Nejdůležitějším hyperparametrem tohoto souboru je však cesta k samotné definici sítě. Ta je rovněž definována v souboru typu `.prototxt`. Jeho příklad lze také najít v příloze C u natrénovaných modelů sítí. Ty mají obvykle v názvu `train_val`[12] či `deploy`[13] a příponu `.prototxt`. Tento trend následuji. V tomto souboru vzhledem k využívaným databázím rovněž definuji velikost minisady, která se najednou zpracovává sítí. S tímto pojmem je také spjatý pojem *epocha*, který v mém případě značí moment, kdy sítí prošly všechny obrázky datové sady právě jednou.

Implementace trénování v Pycaffe. Caffe obsahuje rozhraní pro jazyk Python zvané Pycaffe. Pro problematiku trénování a používání sítí tohoto rozhraní využívám. Po impor-

[5] https://rg3.github.io/youtube-dl/
[6] https://www.youtube.com
[7] https://www.gnu.org/software/bash/
[8] https://www.python.org/
[9] http://www.numpy.org/
[10] http://caffe.berkeleyvision.org/tutorial/solver.html
[11] V příloze C lze tyto soubory najít u natrénovaných modelů sítí (mají v názvu `solver` a příponu `.prototxt`).
[12] Tento soubor slouží k trénování.
[13] Tento soubor slouží k použití sítě po natrénování, neobsahuje ztrátovou vrstvu a definuje rozměr vstupních dat.

tování knihovny `import caffe` lze kombinací dvou funkcí `caffe.set_mode_gpu()` a `caffe.set_device([DeviceID])` caffe jednoduše přepnout pro trénování na vybraném GPU. Poté stačí incializovat síť nastavením cesty k souboru `solver.prototxt` takto

```
solver = caffe.SGDSolver([solver.prototxt])
```

. Proměnná `solver` pak obsahuje veškerá data této sítě, kterou lze tímto způsobem inicializovat. K trénovací a testovací části sítě lze pak přistupovat následovně:

```
train_net = solver.net
test_net = solver.test_nets[i]
```

. Několikarozměrným parametrům sítě reprezentujícím jednotlivé vrstvy se potom říká `blobs` a lze k ním přistupovat skrze

```
train_net.blobs['nazev_vystupu_vrstvy'].data
```

. Samotné trénování lze potom kontrolovat voláním `solver.step(c)`, kde `c` značí kolik dopředných a zpětných průchodů sítí bude provedeno. Implementace tohoto trénování je k nalezení v příloze C v souboru `skripty/my_solve.py`

4.3 Vyhodnocení

Zde popisuji implementaci vyhodnocování sítí a jejich využití ke zpracování dat.

Zpracování videa natrénovanou sítí. Přístup k natrénované síti z hlediska jejího využití skrze rozhraní jazyka Python je obdobný, jako v předchozí sekci (4.2). V tomto případě stačí síť nainicializovat následovně:

```
net = caffe.Net(DEPLOY,CAFFEMODEL,caffe.TEST)
```

, kde `DEPLOY` je cesta k souboru obsahujícímu definici sítě `deploy.protoxt`, `CAFFEMODEL` je cesta k souboru natrénované sítě `.caffemodel` a `caffe.TEST` značí, že se má inicializovat pouze testovací část. Poté stačí ručně přistoupit k datovým vrstvám sítě, jak popisuji v implementaci trénování a uložit do nich potřebná data. Nad sítí pak stačí zavolat `net.forward()` a síť provede dopředný průchod. Ve výsledné vrstvě se nyní nachází požadovaný výstup, který lze opět přímým přístupem k `blobs` přečíst. Nahrávání dat do sítě, její následný průchod a čtení dat ze sítě obvykle provádím ve funkcích `forward()`, které se nachází ve všech přikládaných kódech `*.py`, které operují se sítí.

Pro zpracování videa využívám výše uvedené knihovny OpenCV, která s pomocí nástroje FFmpeg dokáže parsovat stream vstupního videa po snímcích. Jedná se především o funkce `cv2.VideoCapture.get` a `cv2.VideoWriter.write`. Snímky pak ručně ořezávám s pomocí indexací datových struktur `list` jazyka Python a knihovny Numpy. Tyto pod-snímky jsou poté zpracovávány sítí a následně opět skládány do snímku výsledného, který opět s pomocí knihovny OpenCV ukládám do výstupního videa. Implementace je uvedena v přikládaném souboru `skripty/processVideo-cv2.py` v příloze C. Návod k použití natrénovaných sítí je také uveden v příloze B.

Vyhodnocení. Vyhodnocení jednotlivých sítí provádím obdobně, jako v předchozím paragrafu popisujícím zpracování videa. Opět se jedná o iterativní volání `net.forward()`, ale v tomto případě jsou vstupní data obrázky a nad výstupem jsou volány funkce knihovny `skimage.measure` pro počítání PSNR (`compare_psnr()`) a SSIM (`compare_ssim()`). Odhadovaný interpolační snímek je pak implementován následovně

```
merge = cv2.addWeighted(in1, 0.5, in3, 0.5, 0)
```

, kde `in1` a `in3` jsou vstupní obrázky. Toto provede vážený průměr těchto dvou snímků. Popisované struktury lze nalézt v přikládaném souboru `skripty/eval_net_on_dataset.py` v příloze C.

4.4 Příprava video-dat pro trénování

Příprava dat pro trénování zahrnuje výběr vhodných videozáznamů (popsaných v kapitole 3.2), následné extrahování n-tic snímků, případné komprese snímků vstupních a vytvoření databáze LMDB.

Trénovací videa do databáze. Z hlediska velikosti trénovací datové sady je vhodné implementovat ji jako databázi formátu LMDB[14]. Z videí pomocí nástroje `FFmpeg` a skriptovacího jazyka `Bash` hromadně exportuji trojice snímků jdoucích ve videu za sebou. A to s pevně daným rozestupem. Ve výsledku toto tvoří zhruba 4000 trojic snímků, které jsou poté rozřezány na pod-obrázky o velikosti 80×80, kterých je z těchto videí ve výsledku kolem $1,25$ milionu trojic. Tento postup není ideální, protože takto vytvořená sada obsahuje spousty výřezů z jednoho snímku. Nejlepším způsobem by bylo síti předávat v odlehlém vlákně vyřezávané náhodné n-tice pod-obrázků přímo ze vstupních videí tak, aby se vůbec nepracovalo s celými snímky. Takto lze teoreticky obsáhnout mnohem více náhodných informací, což by mohlo pomoct síti lépe generalizovat.

Pro extrahování trojic snímků používám iterativně následující kus kódu:

```
ffmpeg -ss $ZAC -i $VIDEO -to $DELKA -vsync 0 -vf fps=$FPS $VYSTUP.
```

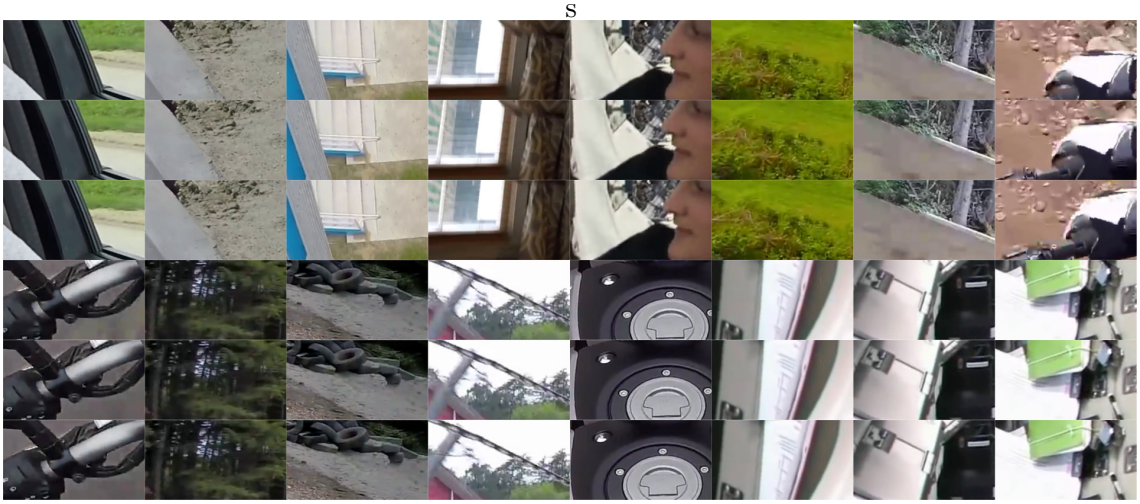
Parametru `-to` zde využívám pro stanovení délky trvání trojic, která se odvíjí od snímkové frekvence videa. Pro zdrojové video o frekvenci 60 snímků za vteřinu lze tedy odhadovat čas kolem `DÉLKA` $= 3 \cdot (1/60) = 0,05$ vteřiny. Dále pak odděluji jednotlivé trojice do tří složek tak, aby měly první, druhé a třetí snímky trojic umístění zvlášť. Následně seznam souborů náhodně zamíchám nástrojem `sed` a s pomocí skriptů přiložených ke Caffe vytvořím LMDB databázi. Popisovanou implementaci lze nalézt v příloze C ve složce `skripty/` v souborech `create_listOfFiles.sh`, `VideoToFrames*.sh` a `splitframes-batch.sh`.

Tyto postupy daly vzniknout datové sadě **YT_train**. Datovou sadu **YT_trainS** jsem optimalizoval s pomocí *optical flow* algoritmu, který je implementován ve skriptu pro vyhodnocení sítě. Ukázka trojic této sady je vidět a obrázku 4.1. Tato sada obsahuje pouhých 17 tisíc trojic, nicméně neobsahuje snímky nevhodné, jako sada předchozí. Toto malé číslo je dáno také tím, že výřezy této sady mají velikost 270×135, namísto původních 80×80. Na této sadě byly trénovány sítě **5l-res/std-iNET**. Bez dalších experimentů nad původní datovou sadou lze jen stěží usuzovat, jakou výhodu, či nevýhodu toto představuje.

Testovací videa do obrázků. Zde jsou trojice ze zdrojového videa brány s rozestupem tak, aby byl poměr trojic na video zhruba stejný. Tento způsob je vhodný i pro trénovací sadu. S ohledem na výpočetní a prostorovou náročnost trénovací sady jsem ale již tuto optimalizaci neprovedl. Způsob je stejný jako v předešlém odstavci s tím rozdílem, že již neprovádím krok k vytváření databáze. U testovacích sad dále nezmiňuji velikost výřezů, jelikož ty provádím až během procesu samotného vyhodnocování a protože je síť invariantní vůči velikosti vstupu.

[14] http://www.lmdb.tech/doc/

s



Obrázek 4.1: Trojice trénovacích obrázků z datové sady **YT_trainS**. Prostřední obrázky (2. a 5. řádek) jsou ze vstupu vynechány a fungují jako „label“.

Kapitola 5

Provedené experimenty

V této sekci se zabývám provedenými experimenty, jejichž výsledkem jsou popisované architektury konvolučních sítí. Dále uvádím vyhodnocení těchto sítí nad testovacími sadami výše popisovanými metrikami.

5.1 Trénování navržených sítí

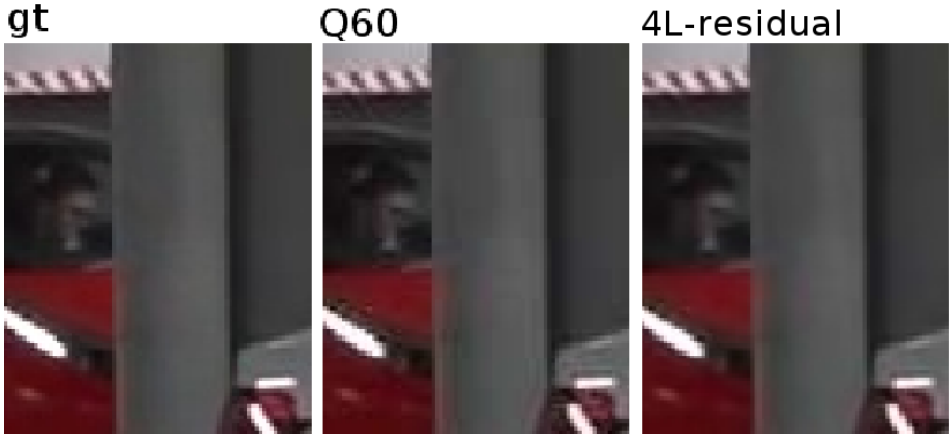
Pro zopakování uvádím jednotlivé sítě ve spojitosti s použitou datovou sadou pro jejich trénování. Trénování sítě *4l-res-iNET*[1] bylo provedeno na datové sadě **YT_train**. Sítě *5l-res/std-iNET* byly trénovány na sadě **YT_trainS**. Síť *vnet* měla na vstupu obrázky sady **YT_train** komprimované metodou JPEG(q60). A síť *4l-residual* byla trénována na datové sadě **BSDS500** a dotrénována na datové sadě **YT_train** komprimované metodou JPEG(q60). Datové sady jsou popisovány v sekci 3.2. Velikosti minisad byly následující: *4l-residual*(60,80×80), *5l-res-iNET*(32,240×135), *5l-std-iNET*(32,240×135) a *vnet*(32,80×80). Trénování bylo obvykle nastaveno tak, aby bylo provedeno několik desítek epoch, v závislosti na velikosti datové sady, s výjimkou využití datové sady **YT_train**.

Trénování bylo prováděno na osobním počítači s procesorem intel i7 4.generace a akcelerováno s pomocí grafických jednotek. Síť *vnet* a sítě ze kterých vychází, tedy *4l-residual* a *4l-res-iNET*, byly trénovány s pomocí grafické karty Nvidia GTX řady Maxwell s 4 GB paměti. Trénování probíhalo v řádech hodin. Všechny zmiňované sítě se natrénovaly v rozmezí 5 až 6 hodin. U sítě *vnet* je však třeba brát v potaz, že využívá parametrů předtrénované sítě *4l-residual*, a tedy její trénování prakticky trvalo 12 hodin.

Oproti tomu novější sítě pětivrstvé interpolační, jmenovitě *5l-res-iNET* a *5l-std-iNET*, byly trénovány s pomocí grafické karty Nvidia GTX řady Pascal s 11 GB paměti. Trénování *5l-res-iNET* trvalo zhruba 4 hodiny. *5L-std-iNET* se pak trénovala 2,5 hodiny, nicméně již po 35 minutách přestala konvergovat. Vzhledem k chybě v akceleračních knihovnách cuDNN verzí 5.1+ jsem byl nucen některé hyperparametry sítě residuální proti standardní pětivrstvé upravit tak, aby trénování mohlo probíhat při stejné velikosti minisady[2].

[1] Je součástí sítě vnet, samotná byla natrénována pro porovnání.

[2] Tato chyba neumožnila spustit trénování s více než pár sty MB paměti či konkrétním nastavením hyperparametrů. Podařilo se ji, krom zmiňované změny sítě, obejít až v závěru práce s pomocí knihovny cuDNN verze 5.0



Obrázek 5.1: Redukce artefaktů metody JPEG q60.

Výsledky sady Ferrari(q60)	PSNR	SSIM
4L-residual	34.28	0.948
JPEG-Q60	33.52	0.936

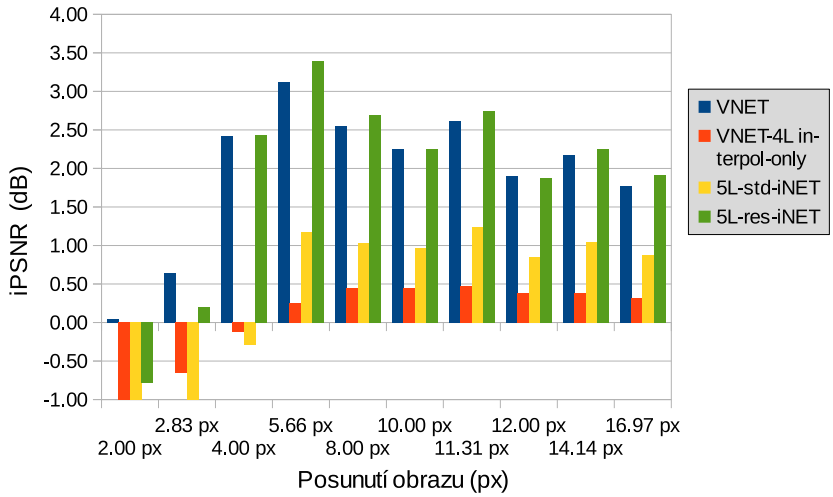
Tabulka 5.1: Výsledné PSNR opravy komprese.

5.2 Vyhodnocení experimentů

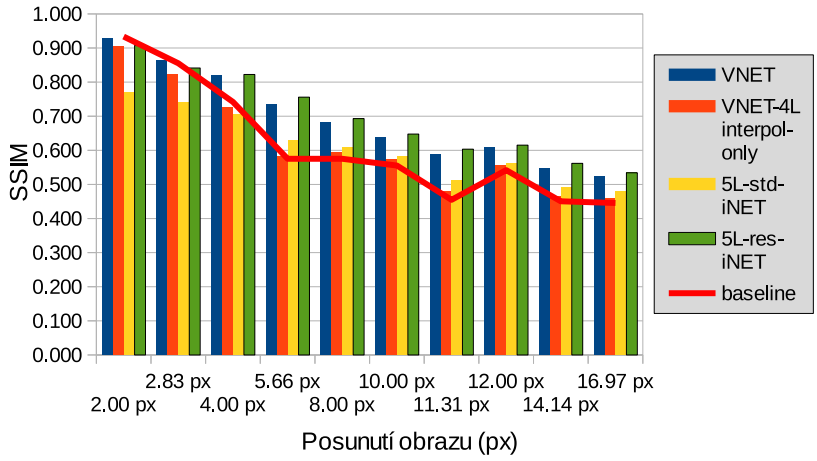
Všechny interpolační sítě byly testovány na datové sadě `YT_test` a sadě `umělých posunutí`. Obrázky obrazových výstupů jsou popsány názvem sítě, komprese, nebo zkratkou `gt`(z angl. ground truth), což je označení pro vzorový obrázek. Síť pro opravu komprese byla testována na datové sadě Ferrari.

Výsledky opravy komprese. Síť *4l-residual* byla testována na datové sadě `Ferrari`, jejíž vstupní obrázky byly komprimovány metodou JPEG(q60). Jak lze pozorovat na obrázku 5.1, síť je schopna obraz komprimovaný touto metodou zbavit blokových artefaktů a přiblížit jeho kvalitu o něco blíže původnímu snímku. Jelikož tato síť není primárním cílem této práce, nebylo jí věnováno tolik času. Dosažené výsledky jsou tedy dostačující pro její další využití sítí *vnet*.

Umělé posunutí. Na obrázcích zobrazujících sloupcový graf lze pozorovat, jak si jednotlivé interpolační sítě poradí s uměle posunutými daty, které popisuji v kapitole 3.2. Jak lze vidět na obrázku 5.2 a 5.3, u velkých posunů vykazují sítě zpravidla lepší výsledky, protože průměrovanému snímku prudce klesá podobnost s velikostí posunutí. SSIM obdobně kopíruje průběh PSNR a pro průměrné posunutí dosahují sítě nejvyšších hodnot. Ač se zprvu zdálo, že sítě selhávají v interpolaci obrazu již při minimálních posunech, není tomu tak. Jak lze pozorovat na obrázcích 5.5 a 5.4, obě navrhované sítě úspěšně interpolují uměle posunutý obraz.

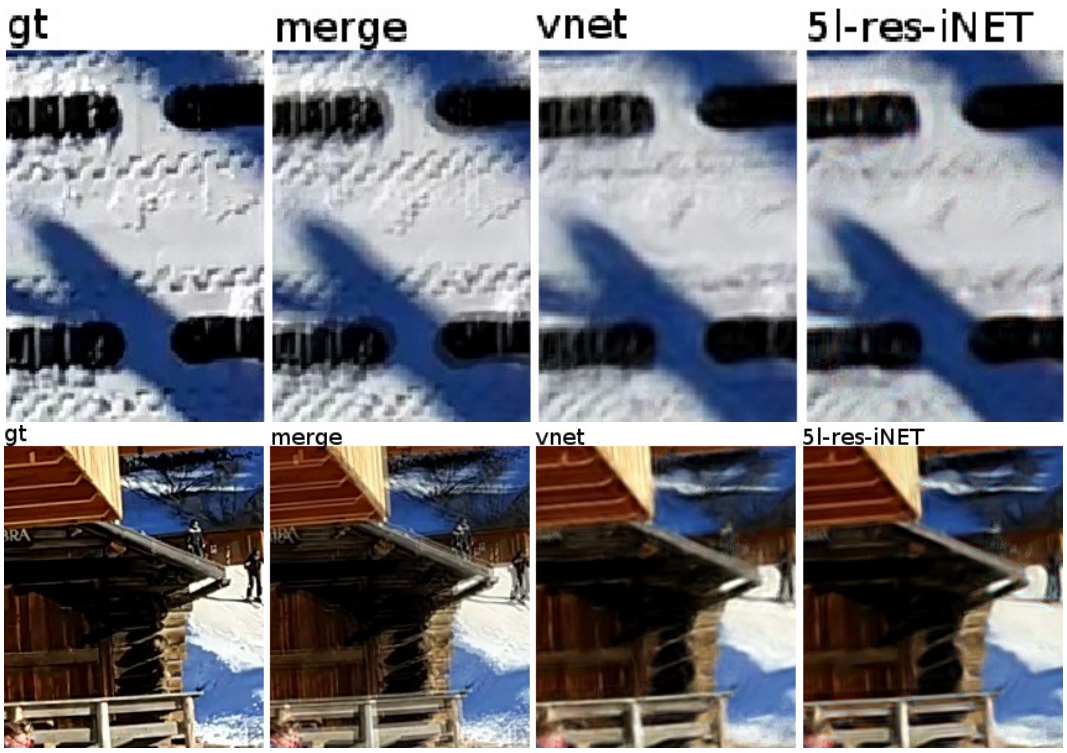


Obrázek 5.2: Interpolace uměle posunutých dat. Zobrazený graf popisuje iPSNR (rozdíl PSNR mezi výstupem sítě a odhadovaným snímkem).

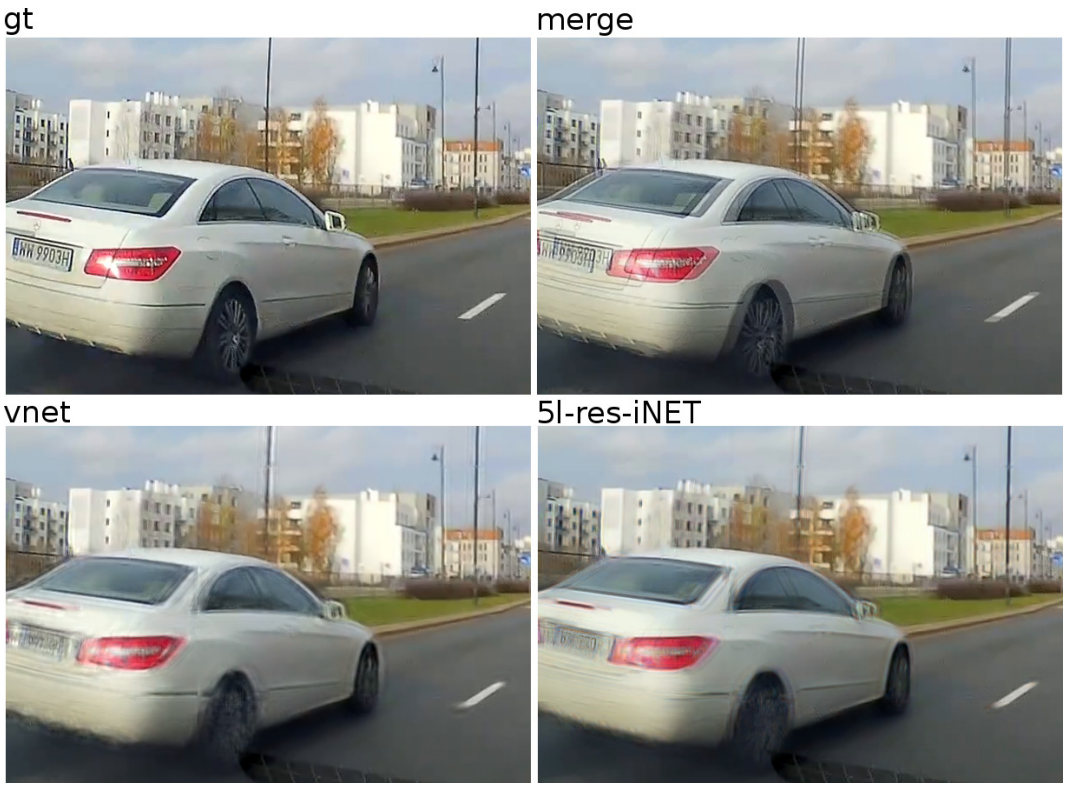


Obrázek 5.3: Interpolace uměle posunutých dat. Zobrazený graf popisuje SSIM. Červená křivka zobrazuje odhad.

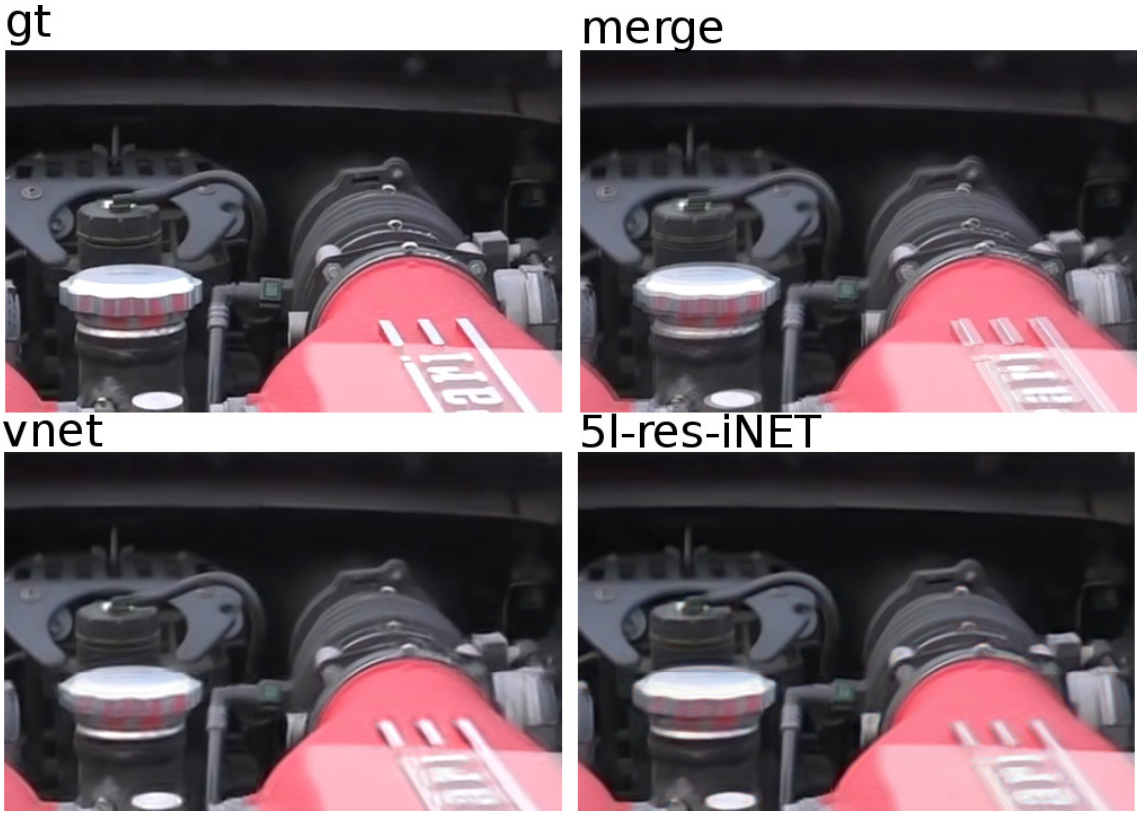
Obrázek 5.4: Interpolace uměle posunutých dat. Zde je názorně vidět, jak se síť vypořádá s posunutím **8 px** po jedné ose.



Obrázek 5.5: Interpolace uměle posunutých dat. Zde je názorně vidět, jak se síť vypořádá s posunutím **5.66 px** po úhlopříčce.



Obrázek 5.6: Ukázka výsledků na testovací sadě YT.



Obrázek 5.7: Ukázka výsledků na testovací sadě Ferrari.

Výsledky interpolace reálných dat. Interpolační sítě byly dále podrobeny testům na dvou datových sadách z reálných záznamů. V nich se objekty hýbou různou rychlostí, podle umístění vůči kameře, atp. K tomu posloužily datové sady Ferrari a YT_test (4.4). Výsledky jsou znázorněny tabulkami 5.2. Jelikož první zmíněná sada je tvořena pouze jedním videem a její snímky nebyly podrobeny třídění, není zcela vhodná. To dokazují výsledky experimentů nad ní. Navíc se jedná o profesionální záznam, na rozdíl od amatérských záběrů trénovacích datových sad. Jeho kvalita je tedy mnohem vyšší. Přesto se ji zde věnuji pro porovnání s druhou datovou sadou. Je však použita k vyhodnocení pouze dvou nejlépe natrénovaných sítí.

Dále se domnívám, že *vnet* zanechává v obraze více detailů na úkor slabší interpolace, kdežto *5l-res-iNET* dosahuje lepší SSIM asi také proto, že se jí ve většině případů daří lépe interpolovat jasně posunuté kontury. I když je to na úkor některých detailů. Toto lze pozorovat na obrázcích 5.7 a 5.6. Důležité je také zmínit fakt, že natrénovaný model *vnet* má bezmála `7MB`, zatím co model sítě *5l-res-iNET* má pouhých `385kB`.

Je poměrně nesnadné určit, která z těchto dvou sítí je vhodnější pro daný účel. Z mého pohledu je síť *vnet* vhodná už jen proto, že se zároveň může starat o kompresi a zachovat tak třeba více detailů, jak mnohdy dokazuje i na výstupních snímcích. Do budoucna by bylo vhodné provést experiment nad touto sítí upravenou tak, aby její interpolační část obsahovala 5 vrstev, namísto 4, jako je tomu u zmiňované pětivrstvé residuální sítě *5l-res-iNET*. Na druhou stranu síť *5l-res-iNET* je řádově menší a přesto dosahuje obdobných výsledků.

Výsledky sady YT_test	PSNR	SSIM
VNET	**29.76**	0.880
4L-res-iNET	28.62	0.860
5L-std-iNET	27.21	0.838
5L-res-iNET	29.40	**0.882**
merge	29.12	0.860

Výsledky sady Ferrari	PSNR	SSIM
VNET	35.90	0.957
5L-res-iNET	34.55	**0.957**
merge	**36.88**	0.953

Tabulka 5.2: Vlevo jsou zobrazeny výsledky sítí nad testovací datovou sadou YT. Průměrované snímky dosahují hodnot $PSNR = 29.12$ a $SSIM = 0.860$. Vpravo jsou zobrazeny výsledky sítí nad testovací sadou Ferrari. Průměrované snímky dosahují hodnot $PSNR = 36.88$ a $SSIM = 0.9525$.

Bylo by vhodné prozkoumat složitější architektury a porovnat, jak tyto budou soupeřit s mnou navrženými. K tomu by bylo vhodné navázat na přístup Caballero et.al.[1] a obdobně pracovat s širším časovým okénkem pro vstupní snímky s využitím postupné konkatenace, případně kompenzací pohybu. Zajímavé by také byly experimenty s kvalitněji vypracovanou datovou sadou tak, aby se mohla síť učit interpolovat různě velká posunutí, ovšem diskretizovaná. Nakonec by bylo vhodné zaměnit zmiňovanou kompresi JPEG za kompresi skutečně odpovídající používaným záznamům. K tomu navrhuji vycházet z práce skupiny Dai *et al.*[3], která se zaměřuje na moderní High Efficiency Video Coding (HEVC) standard.

Kapitola 6

Závěr

V této práci kombinuji nejmodernější postupy pro rekonstrukci obrazu s pomocí konvolučních neuronových sítí. Mnou navržené sítě zpracovávají dvojice snímků a interpolují snímek prostřední. Umožňují tak zvýšení snímkové frekvence vstupního videa dvojnásobně při vložení snímku mezi každou dvojici snímků videozáznamu. Ukazují také, že zároveň mohou odstraňovat jisté následky komprese, pro kterou byly trénovány.

Navrhované sítě byly složeny z několika základních bloků. A to z „residuální“ části, „interpolační” části, případně části pro zpracování komprese. Síť, která využívá všech zmiňovaných bloků, dosahuje v inteprolaci nejvyšší PSNR. Navrhované sítě dosahují v průměru lepších výsledků, než pouhý průměr vstupních snímků. Lze tak nadále polemizovat o tom, zdali jsou navrhované sítě k řešení problému zvýšení časového rozlišení videa vhodné, či nikoliv.

Experimenty byly provedeny nad hlavní datovou sadou vytvořenou z reálných full-hd záznamů se snímkovou frekvencí 60 snímků za vteřinu. Testovací datová sada byla vytvořena obdobným způsobem. Experimenty nad datovou sadou, které měly za účel ji optimalizovat pomocí „optical-flow“ a zjistit, jestli je příčinou slabších výsledků, neprokázaly dostatečný vliv na výslednou kvalitu rekonstruovaných snímků, avšak tyto experimenty nebyly provedeny do konce. Výsledky ukazují rovnoměrný pokles kvality v závislosti na posunu snímků.

Z hlediska zpracování videa obecně v době vzniku mé práce prakticky nebyly dostupné dokumenty se stejným zaměřením využívající konvoluční sítě. Nyní se již objevují práce, které podobné problémy řeší. Po stránce opravy komprese ve videu by bylo vhodné přejít z metody JPEG na moderní kompresi využívanou čistě pro video. K tomu navrhuji vycházet z práce skupiny Dai *et al.*[3], která se zaměřuje na High Efficiency Video Coding (HEVC) standard. Po stránce architektury dále navrhuji navázat na přístup Caballero et.al.[1] a obdobně pracovat s širším časovým okénkem pro vstupní snímky s využitím postupné konkatenace, případně kompenzaci pohybu. Dále pak navrhuji ručně vytvořenou datovou sadu úzce zaměřených dat a předtrénovávání sítí pouze pro určité posunutí nad uměle vytvořenými daty.

Literatura

[1] Caballero, J.; Ledig, C.; Aitken, A.; aj.: Real-Time Video Super-Resolution with Spatio-Temporal Networks and Motion Compensation. 2016, 1611.05250.

[2] Cauchy, A.: Méthode générale pour la résolution de systèmes d'équations simultanées. *Comptes Rendus des Séances de l'Académie des Sciences. Paris*, , č. 25, 1847: s. 536–538.

[3] Dai, Y.; Liu, D.; Wu, F.: A Convolutional Neural Network Approach for Post-Processing in HEVC Intra Coding. 2016, 1608.06690.

[4] Dong, C.; Deng, Y.; Loy, C. C.; aj.: Compression artifacts reduction by a deep convolutional network. 2016, doi:10.1109/ICCV.2015.73, 1504.06993.

[5] Dong, C.; Loy, C. C.; He, K.; aj.: Learning a deep convolutional network for image super-resolution. *European Conference on Computer Vision*, ročník 8689, 2014: s. 184–199, ISSN 15505499, doi:10.1007/978-3-319-10593-2_13.

[6] Eigen, D.; Krishnan, D.; Fergus, R.: Restoring an image taken through a window covered with dirt or rain. *Proceedings of the IEEE International Conference on Computer Vision*, 2013: s. 633–640, ISSN 1550-5499, doi:10.1109/ICCV.2013.84.

[7] Glorot, X.; Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[8] Goodfellow, I.; Bengio, Y.; Courville, A.: Deep Learning, 2016, book in preparation for MIT Press.
URL http://www.deeplearningbook.org

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

[10] He, K.; Zhang, X.; Ren, S.; aj.: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*, ročník abs/1502.01852, 2015.
URL http://arxiv.org/abs/1502.01852

[11] Hradiš, M.; Kotera, J.; Zemčík, P.; aj.: Convolutional Neural Networks for Direct Text Deblurring. *Procedings of the British Machine Vision Conference 2015*, , č. 1, 2015: s. 6.1—-6.13, ISSN 1-901725-53-7, doi:10.5244/C.29.6.

[12] Huynh-Thu, Q.; Ghanbari, M.: Scope of validity of PSNR in image/video quality assessment. *Electronics Letters*, ročník 44, June 2008: s. 800–801(1), ISSN 0013-5194.

[13] Jain, V.; Seung, S.: Natural Image Denoising with Convolutional Networks. ročník 21, 2009: s. 769–776.

[14] Jia, Y.; Shelhamer, E.; Donahue, J.; aj.: Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[15] Kappeler, A.; Yoo, S.; Dai, Q.; aj.: Video Super-Resolution with Convolutional Neural Networks. *IEEE Transactions on Computational Imaging*, ročník PP, č. 99, 2016: s. 1–1, ISSN 2333-9403, doi:10.1109/TCI.2016.2532323.

[16] Kim, J.; Lee, J. K.; Lee, K. M.: Accurate Image Super-Resolution Using Very Deep Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, ročník 38, č. 2, 2015: s. 295–307, ISSN 01628828, doi:10.1109/TPAMI.2015.2439281, 1511.04587.

[17] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, 2012: s. 1–9, ISSN 10495258, doi:http://dx.doi.org/10.1016/j.protcy.2014.09.007, 1102.0183.

[18] Le Gall, D.: MPEG: A Video Compression Standard for Multimedia Applications. *Commun. ACM*, ročník 34, č. 4, Duben 1991: s. 46–58, ISSN 0001-0782, doi:10.1145/103085.103090.

[19] Long, J.; Shelhamer, E.; Darrell, T.: Fully Convolutional Networks for Semantic Segmentation ppt. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015: s. 3431–3440, ISSN 10636919, doi:10.1109/CVPR.2015.7298965, 1411.4038.

[20] Ng, A. Y.; Ngiam, J.; Foo, C. Y.; aj.: UFLDL's Multi-Layer Neural Network,Convolutional Neural Network and Feature Extraction Using Convolution tutorial articles. May 2017.
URL http://ufldl.stanford.edu/tutorial/

[21] Shahar, O.; Faktor, A.; Irani, M.: Space-time super-resolution from a single video. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2011: s. 3353–3360, ISSN 10636919, doi:10.1109/CVPR.2011.5995360.

[22] Svoboda, P.; Hradis, M.; Barina, D.; aj.: Compression artifacts removal using convolutional neural networks. *Journal of WSCG*, ročník 24, č. 2, 2016: s. 63–72, ISSN 12136964, 1605.00366.

[23] Wallace, G. K.: The JPEG Still Picture Compression Standard. *IEEE Trans. on Consum. Electron.*, ročník 38, č. 1, Únor 1992: s. xviii–xxxiv, ISSN 0098-3063, doi:10.1109/30.125072.

[24] Wang, Z.; Bovik, A. C.; Sheikh, H. R.; aj.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, ročník 13, č. 4, April 2004: s. 600–612, ISSN 1057-7149, doi:10.1109/TIP.2003.819861.

[25] Xie, J.; Xu, L.; Chen, E.: Image Denoising and Inpainting with Deep Neural Networks. *Nips*, 2012: s. 1–9, ISSN 10495258.

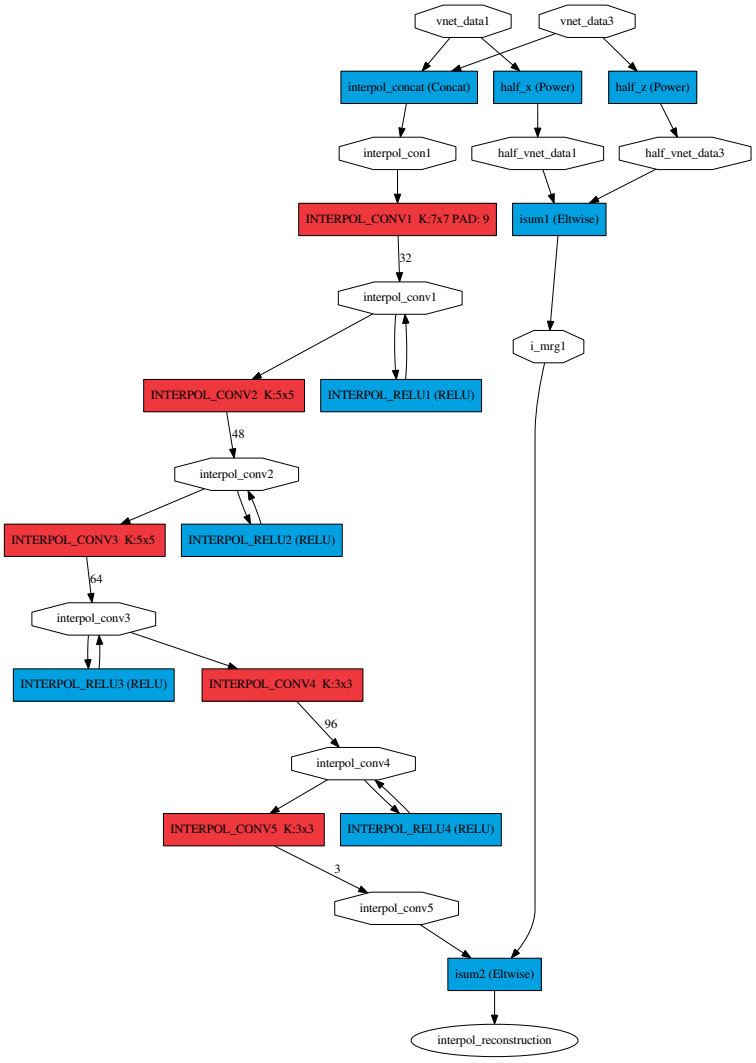
[26] Xu, L.; Ren, J. S.; Liu, C.; aj.: Deep Convolutional Neural Network for Image Deconvolution. *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, , č. 413113, 2014: s. 1–9, ISSN 10495258.

[27] Zeiler, M. D.: ADADELTA: An Adaptive Learning Rate Method. *CoRR*, ročník abs/1212.5701, 2012.

[28] Zhao, M.; Zhang, J.; Figueiredo, R.: Distributed file system support for Virtual Machines in Grid computing. *IEEE International Symposium on High Performance Distributed Computing, Proceedings*, ročník 130, č. x, 2004: s. 202–211, ISSN 10828907, doi:10.1109/HPDC.2004.1323531, 3629719.

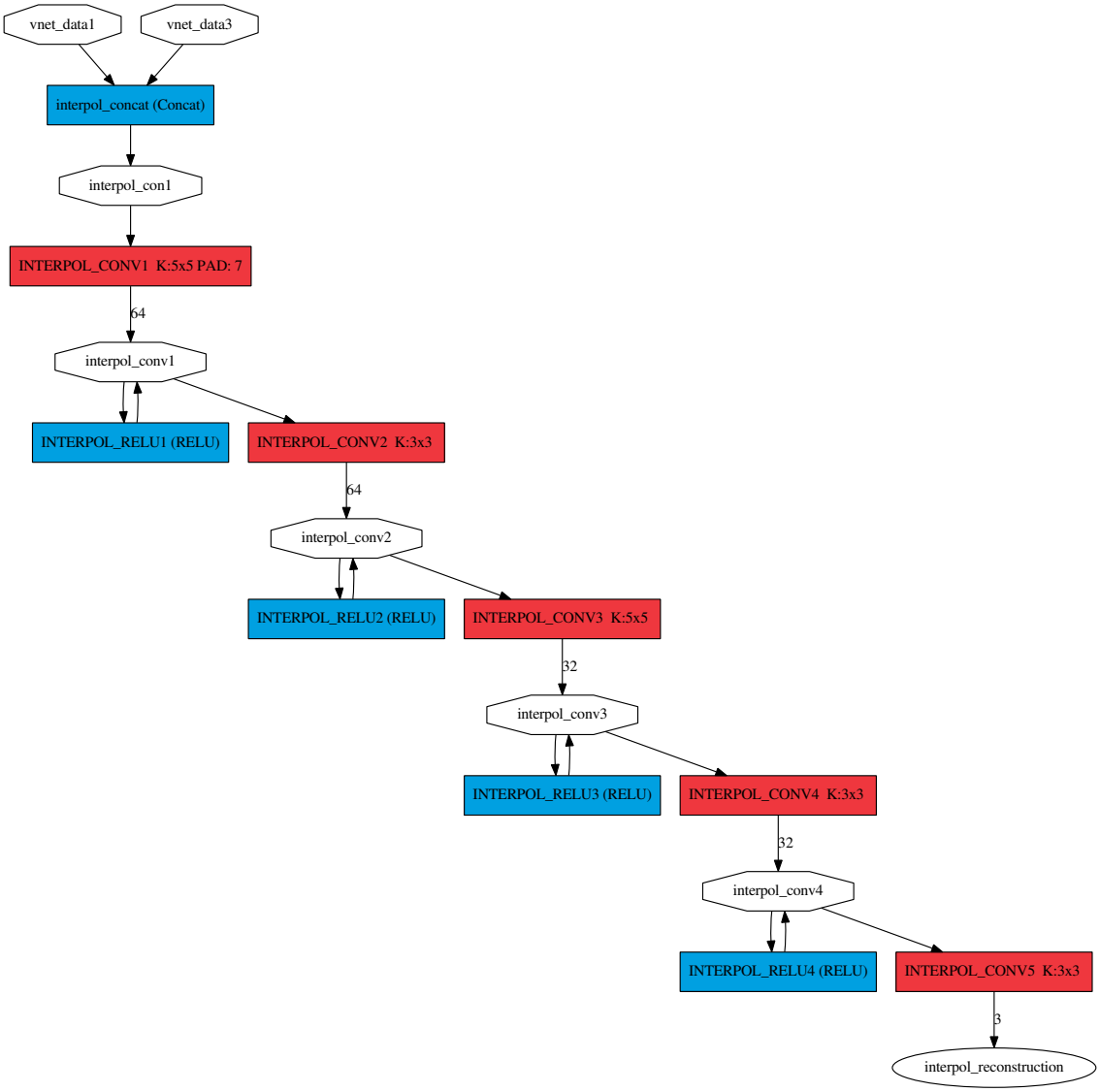
Příloha A

Schémata sítí

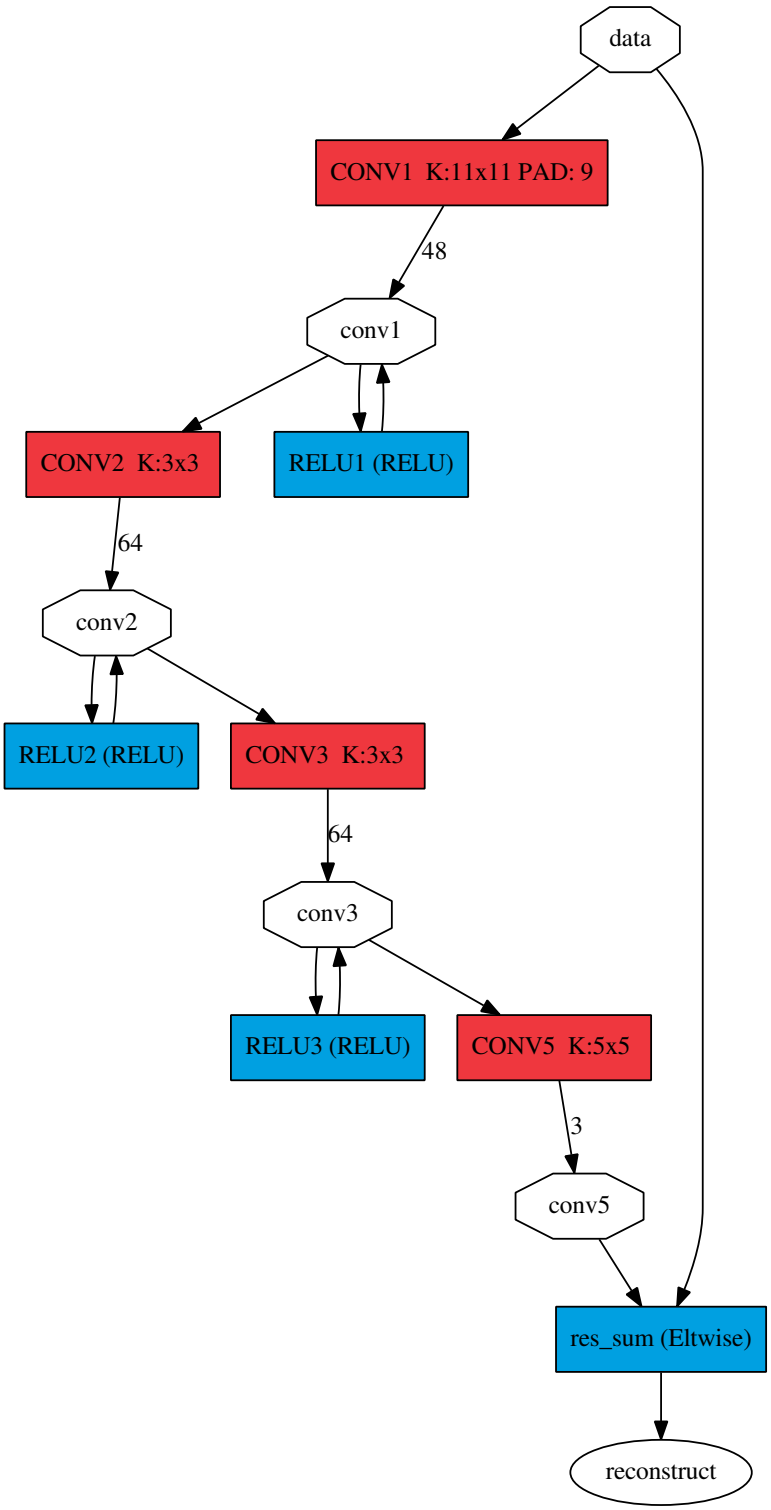


Obrázek A.1: Zde je schéma architektury 5l-res-iNET.

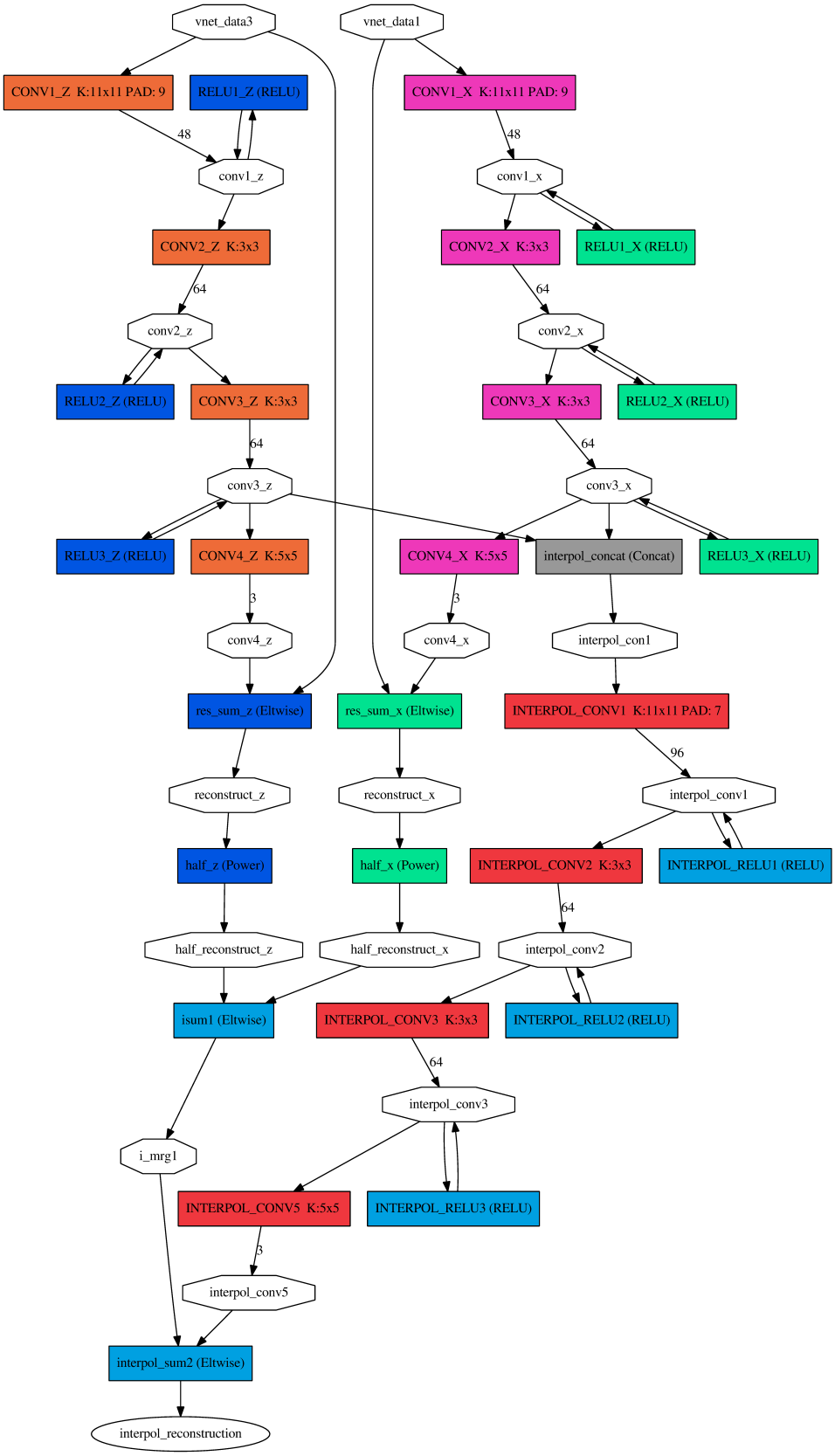
Schéma zmiňované sítě 4l-res-iNET zde není, jelikož je pouze součástí sítě **vnet**.



Obrázek A.2: Zde je schéma architektury 5l-std-iNET.



Obrázek A.3: Zde je schéma modifikované architektury 4l-res-ARCNN. Tuto síť lze nalézt 2x v síti vnet.



Obrázek A.4: Zde je schéma architektury VNET, která spojuje dva residuální bloky pro opravu komprese, které jsou barevně odlišeny, s čtyřvrstvým residuálním interpolačním blokem.

Příloha B

Návod k použití sítí ke zpracování videa

Zde uvádím postup pro využití natrénovaných sítí. Postup pro jejich vyhodnocování je uveden v příloze C v souboru `readme.md`.

B.1 Prerekvizity

- Nainstalované Caffe a knihovny Numpy a OpenCV 3.*(zkompilované s FFmpeg), které lze importovat v intepretu jazyka Python 2.*. Caffe je třeba přeložit příkazem make pycaffe.
- Interpret jazyka Python 2.*.

B.2 Použití

1. V souboru ./skripty/processVideo-cv2.py:
   - upravím cestu ke vstupnímu a výstupnímu souboru.
   - nastavím velikosti vstupních výřezů tak, aby odpovídaly vstupům v souboru sítě site/[síť]/*deploy*.prototxt.
   - nastavím parametry videa a výstupní FPS v hlavičce skriptu.
   - nastavím cestu k modelu natrénované sítě site/[síť]/*.caffemodel a souboru s její definicí site/[síť]/*deploy*.prototxt.
2. Pustím skript interpretem jazyka Python verze 2.*

Příloha C

Obsah přiloženého paměťového média

Jelikož datové sady pro trénování přesahují velikostí rámec klasického DVD, je zde z trénovacích sad přiložená pouze sada YT_trainS ve formátu obrázků s přiloženými seznamy souborů. Z těch lze využitím skriptu pro vytváření LMDB databází, který je součástí Caffe[1], vytvořit databázi LMDB.

- `datove_sady/testovaci_sady/` – Tato složka obsahuje testovací datovou sadu s umělým posunutím, datovou sadu YT_test a testovací datovou sadu Ferrari.
- `datove_sady/trenovaci_sady/` – Tato složka obsahuje trénovací datovou sadu YT_trainS.
- `site/` – Tato složka obsahuje definice trénovaných sítí, soubory potřebné k trénování, používání a natrénované modely sítí.
- `skripty/` – Tato složka obsahuje skripty používané k vytváření datových sad z videí, trénování sítí, vyhodnocení sítí a použití sítí ke zpracování videa.
- `technicka_zprava/` – Tato složka obsahuje zdrojový kód technické zprávy včetně přeložené zprávy ve formátu .pdf.
- `video/` – Tato složka obsahuje výstupní video popisující tuto práci.
- `README.md` – Tento soubor obsahuje postupy pro trénování, vyhodnocování a využívání natrénovaných sítí.

[1] https://github.com/intel/caffe/wiki/How-to-create-LMDB