VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ OPTIMALIZACE KONVOLUČNÍCH NEURONOVÝCH SÍTÍ

EVOLUTIONARY OPTIMIZATION OF CONVOLUTIONAL NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. VOJTĚCH ČOUPEK**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

148487

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Čoupek Vojtěch, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Strojové učení
Název:	**Evoluční optimalizace konvolučních neuronových sítí**
Kategorie:	Umělá inteligence
Akademický rok:	2022/23

Zadání:

1. Seznamte se s konvolučními neuronovými sítěmi (CNN), evolučními algoritmy (EA) a využitím EA pro návrh a optimalizaci CNN.
2. Na základě získaných znalostí navrhněte způsob použití evolučního algoritmu pro vylepšení zvolených vlastností referenční implementace CNN na problému, který bude doporučen vedoucím. Zaměřte se na optimalizaci výpočetní náročnosti CNN.
3. Implementujte EA a podpůrné programy pro automatizovanou optimalizaci CNN.
4. Porovnejte vlastnosti původní CNN a optimalizovaných CNN získaných pomocí EA.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Sekanina Lukáš, prof. Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	31.10.2022

Abstrakt

Práce se zabývá problematikou komprese vah neuronové sítě pomocí techniky Weight-Sharing a optimalizací parametrů této techniky pomocí nekonvenčních optimalizačních algoritmů. Důvodem optimalizace je snížení paměťové, respektive energetické náročnosti výpočtu odezvy neuronové sítě. Cílem je navrhnout systém, který dokáže přijmout neuronovou síť a snížit její paměťovou náročnost, a ověřit použitelnost této metody na několika případových studiích. Práce prozkoumává využití různých optimalizačních algoritmů, dodatečnou kompresi pomocí kvantizace nad technikou Weight-Sharing a navrhuje metodu ladění výsledků kvantizace pro zlepšení přesnosti. Tyto postupy jsou nejdříve vyzkoušeny na síti Le-Net-5 a následně aplikovány na kompresi sítě MobileNet_v2.

Abstract

This thesis deals with the problem of neural network weights compression using the technique of Weight-Sharing and parameter optimization of this technique by unconventional optimization algorithms. The reason for the optimization is decreasing the memory or energy demands of the neural network response calculation. The aim is to design a system that accepts a neural network and reduces its memory demands. Its functionality is demonstrated with the help of several experiments. The thesis investigates the use of various optimization algorithms, additional compression using the quantization above the Weight-Sharing technique, and proposes the quantization results tuning method to improve accuracy. These procedures are first tested on the Le-Net-5 network and then applied for the MobileNet_v2. network compression.

Klíčová slova

Komprese, neuronové sítě, konvoluční neuronové sítě, sdílení vah, optimalizační algoritmy, genetický algoritmus, optimalizace hejnem částic, algoritmus černé díry, shlukování, K-means, Le-Net-5, Mobilenet_v2

Keywords

Compression, neural networks, convolutional neural networks, weight-sharing, optimization algorithms, genetic algorithm, particle swarm optimization, blackhole algorithm, clustering, K-Means, Le-Net-5, Mobilenet_v2

Citace

ČOUPEK, Vojtěch. *Evoluční optimalizace konvolučních neuronových sítí.* Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Evoluční optimalizace konvolučních neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením prof. Ing. Lukáše Sekaniny, Ph.D. Další informace mi poskytl Ing. Vojtěch Mrázek, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vojtěch Čoupek
10. května 2023

Poděkování

Chtěl bych moc poděkovat prof. Ing. Lukáši Sekaninovi, Ph.D za vedení práce a věcné rady. Také děkuji Ing. Vojtěchu Mrázkovi, Ph.D. za rady a poskytnutí výpočetních zdrojů k měření spojeného s prací. Dík patří také mé rodině a blízkým za podporu při studiu a při tvoření této práce. Tato práce byla podpořena Ministerstvem školství, mládeže a tělovýchovy České republiky prostřednictvím e-INFRA CZ (ID:90140) a GAČR projektu číslo 21-13001S.

Obsah

Seznam obrázků

Seznam zkratek

ACC	Z anglického „Accuracy“ – přesnost. 50, 53, 55, 57
AL	Z anglického „Accuracy Loss“ – ztráta přesnosti. 42, 46, 47, 50, 57
BH	Z anglického „Blackhole algorithm“, v práci často označuje upravený PSO algoritmus. 17, 33, 38, 42, 46, 47
CNN	Konvoluční neuronová síť z anglického „Convolutionl Neural Network“. 5, 11, 20
CR	Z anglického „Compression rate“ – kompresní poměr. 26, 37, 42, 46, 47, 50, 57
GA	Genetický algoritmus. 2, 15–17, 38, 42, 43, 46, 47
MAC	Operace násobení a sčítání v neuronové síti z anglického „Multiply and ACumulate“. 11
PSO	Paricle Swarm Optimization. 2, 16, 17, 28, 33, 38, 42, 43, 46, 47, 51
RND	Náhodné prohledávání. 38, 42

Kapitola 1

Úvod

Počítače jsou v dnešní době velkými pomocníky v každodenním životě nejen v práci, ale i ve volném čase. Jsou nám neustále po ruce, ať už ve formě mobilního telefonu, notebooku, jsou také součástí automobilů, domácích spotřebičů a podobně. Jejich užitečnost je v současnosti prohlubována velkými pokroky v oblasti strojového učení, ve kterém se daří využívat principy ze světa biologie a uplatnit je při řešení komplexních úloh. Řešení takovým způsobem bývá v současné době často kvalitnější než konvenční přístupy. Příkladem zmíněného řešení je neuronová síť, jejíž atomické prvky – neurony – jsou inspirovány nervovými buňkami. Speciálním případem takovéto sítě je pak konvoluční neuronová síť (dále pouze CNN z anglického „Convolutional neural network“). CNN jsou užitečnými a moderními nástroji v informatice v oblasti klasifikace či segmentace obrazu a dalších odvětvích. Dosahují v těchto úlohách velmi dobrých výsledků a jsou považovány za špičku. Mohou sloužit například při autonomním řízení vozidel [20], poštovních službách [4] nebo klasifikaci či segmentaci dat z oblasti medicíny [22].

Jejich nasazení ovšem vyžaduje velké množství výpočetní síly. Nejsou proto příliš vhodné k použití v přenosných zařízeních napájených na baterii. Jak bylo zjištěno v [35], nejvíce energie při inferenci (tj. výpočtu odezvy) neuronové sítě je využito pro přístup do paměti. Neuronové sítě dnešní doby mohou mít několik desítek milionů parametrů (např. ResNet-50 využívá přes 23 milionů parametrů). Z tohoto důvodu je jedním z klíčů k energetické optimalizaci sítě snížení její paměťové náročnosti, neboli komprese vah získaných po natrénování sítě. S rostoucí kompresí ovšem obecně klesá přesnost výstupu inference. Díky struktuře neuronových sítí je ovšem možné dosáhnout poměrně vysoké komprese za cenu nízké ztráty přesnosti [35].

Cílem práce je seznámení se s problematikou komprese neuronových sítí a jejími různými technikami. Dále bude implementována komprese vah technikou sdílení vah (dále použito anglické „Weight-Sharing“) a bude demonstrována komprese jednoduché i složité CNN pro klasifikaci obrazu. Nad vytvořenou implementací budou provedeny experimenty, jejichž cílem bude vyzkoušet a porovnat různé optimalizační a shlukovací algoritmy pro dosažení co nejlepšího poměru komprese a ztráty přesnosti. Vyzkoušeny budou také různé úrovně nasazení techniky Weight-Sharing. Nakonec bude také navržena a vyzkoušena technika zlepšující přesnost klasifikace při zachování stejné komprese.

Práce se v kapitole 2 zabývá současným stavem v oblasti CNN, jejich kompresí a prostředky, které jsou ke kompresi potřebné, hlavně optimalizačními algoritmy a shlukovacími algoritmy. Popsány jsou také neuronové sítě využité v této práci.

V kapitole 3 je navržena implementace komprese typu Weight-Sharing, dodatečná kvantizace nad jmenovanou technikou a technika fine-tuning pro zlepšení výsledků získaných

kompresí. Tato kapitola popisuje také návrh optimalizace parametrů těchto metod pomocí optimalizačních algoritmů a účelové funkce. Samotná implementace tohoto návrhu je obsahem kapitoly 4. Z implementace řešení vychází kapitola 5, která představí a vyhodnotí sadu experimentů cílených na efektivní řešení problematiky komprese neuronových sítí. Práce je uzavřena shrnující kapitolou 6.

Kapitola 2

Neuronové sítě a jejich optimalizace

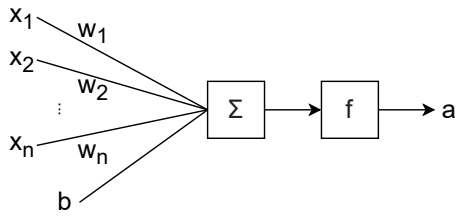
Tato kapitola shrne současný stav technologie neuronových sítí, jejich optimalizace a technik k tomu potřebných. Sekce 2.2 bude věnována samotným neuronových sítím, konvolučním neuronovým sítím a jejich efektivnímu zpracovávání. Ke konci této sekce budou představeny sítě Le-Net-5 a MobileNet_v2, které jsou dále v práci využity. Následně v sekci 2.3 budou shrnuty optimalizační algoritmy, které budou dále použity při řešení a implementaci této práce. Ke kompresi je využito shlukování, proto mu bude věnována sekce 2.4. Na závěr kapitoly budou v sekci 2.5 shrnuty různé techniky komprese neuronových sítí.

2.1 Neuronové sítě

Oblast strojového učení, jak již název napovídá, nabízí algoritmy se schopností učení, tedy přizpůsobení se konkrétnímu problému místo přímého naprogramování řešení problému. Tato schopnost v současné době pomáhá zvládat úkoly, které jsou konvenčními algoritmy těžko řešitelné. Hlavní výhodou schopnosti učení se je, že není třeba algoritmy specificky programovat pro řešení daného úkolu. Díky tomu je možné využít stejný algoritmus pro řešení několika různých problémů bez nutnosti změny kódu, pouze například natrénováním na jiné datové sadě (příkladem může být síť Le-Net-5 [31], která je schopná klasifikovat data jak z datasetu MNIST [7], tak z datasetu FashionMNIST [38]). Mezi takové úlohy jsou řazeny například klasifikace nebo regrese, které v současné době nacházejí velkého využití v různých odvětvích průmyslu a výzkumu. Velká část těchto přístupů je inspirována jevy z oblasti biologie, od chování kolonií živočichů (například kolonie včelstva snažící se najít optimální cestu k potravě – Bees Algorithm [15]) až po komplexní procesy probíhající v evolučním vývoji organismů (evoluční algoritmus, který bude detailněji popsán v sekci 2.3). Jednou z takových inspirací byla nervová buňka, jež vedla v roce 1943 k vytvoření modelu umělého neuronu Warrenem McCulochem a Walterem Pittsem [23], který pracoval s binárními vstupy. Později se tento model vyvinul v aktuálně používanou reprezentaci umělého neuronu definovanou následovně:

$$a = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

kde výstup a označuje výstup neuronu – aktivaci, w značí vektor vah příslušného neuronu, x značí vektor vstupu neuronu a b značí bias neuronu (někdy se v české literatuře používá



Obrázek 2.1: Výpočetní model neuronu – značení je stejné jako v rovnici 2.1

.

označení práh). Neuron vždy provede součet součinů všech n vstupů s patřičnými vahami a přičte k nim svůj bias, přičemž váhy a bias jsou parametry neuronu, které jsou měněny v průběhu učení. Tento výpočet je znázorněný i na obrázku 2.1. Ve vzorci je prezentována lineární bázová funkce, lze se ale setkat i s radiální bázovou funkcí [5], ovšem v rámci této práce se budeme zabývat pouze sítěmi s lineárními bázovými funkcemi. Funkce f představuje aktivační funkce, které se dále dělí na spojité, nespojité a po částech spojité. Mezi nejčastější představitele aktivačních funkcí patří:

$$\text{Sigmoida (spojitá): } f(u) = \frac{1}{1 + e^{(-u)}} \tag{2.2}$$

$$\text{Hyperbolický tangens (spojitá): } f(u) = tanh(u) \tag{2.3}$$

$$\text{ReLu (po částech spojitá): } f(u) = \begin{cases} u, & \text{pro } u \geq 0 \\ 0, & \text{pro } u < 0 \end{cases}, \tag{2.4}$$

$$\text{kde } u = \sum_{i=1}^{n} w_i x_i + b \tag{2.5}$$

$$\tag{2.6}$$

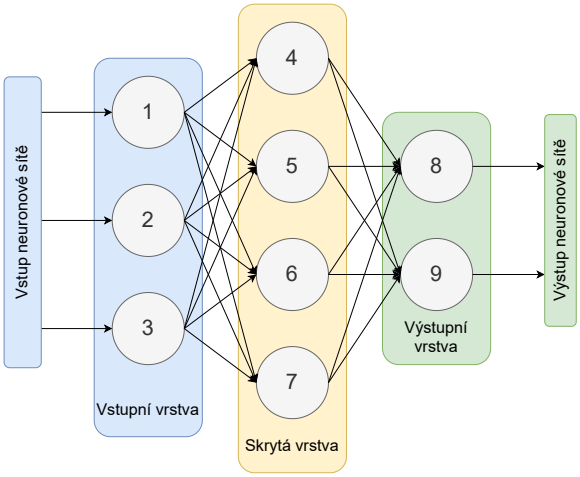
Pro efektivnější počítání aktivace neuronu je možné převést výpočet z rovnice 2.1 na maticové násobení následujícím způsobem:

$$a = f(\mathbf{wx}) \tag{2.7}$$

kde stejně jako v původním výrazu a značí aktivaci neuronu a f značí aktivační funkci. V tomto případě ale $\mathbf{w}$ značí matici s jedním řádkem a $n + 1$ sloupci, kde ve sloupcích $[1, 2, \ldots, n]$ je váha pro příslušný vstup x_n a na pozici $\mathbf{w}_0$ je uložen bias. Následně $\mathbf{x}$ značí matici vstupu s jedním sloupcem a $n + 1$ řádky, kde na pozicích $[1, 2, \ldots, n]$ jsou vstupy neuronu a na pozici $\mathbf{x}_0$ je hodnota 1. Je tedy zřejmé, že maticové násobení těchto dvou matic je ekvivalentní k původnímu výpočtu pomocí sumy. Navíc, pokud matici vah rozšíříme tak, že přidáme řádky různých neuronů ve stejném formátu, můžeme naráz vypočítat aktivaci celé sady neuronů zvaných vrstva.

Jak již inspirace nervovými buňkami naznačuje, neurony mohou být propojeny, aby tvořily komplikovanější strukturu – neuronovou síť. Existuje několik různých architektur takových sítí. Děleny mohou být následovně:

1. Plně propojená síť – každý neuron je propojen se všemi ostatnímu neurony v této síti. Váhy mezi dvěma neurony mohou být různé.



Obrázek 2.2: Příklad schématu jednoduché dopředné neuronové sítě se skrytou vrstvou neuronů.

2. Plně propojená symetrická síť – opět je každý neuron propojen se všemi ostatními neurony v síti, ovšem tentokrát musí platit, že váhy mezi dvěma neurony jsou stejné (tedy pokud máme neuron a a neuron b, potom w_{ab} označující váhu spoje z neuronu b k neuronu a musí být stejná jako váha w_{ba}).

3. Vrstvová (nebo také vrstevnatá) síť – neurony jsou uspořádány do vrstev, přičemž neurony vrstvy n nemohou ovlivnit vrstvy menší než n (tedy v této síti mohou existovat konexe neuronů v rámci vrstvy).

4. Acyklická síť – stejně jako vrstvová síť, ovšem nejsou povoleny konexe neuronů v rámci vrstev.

5. Dopředná síť – Acyklická síť, kde neurony vrstvy n mohou ovlivnit pouze neurony vrstvy $n + 1$.

Pokud se v architektuře nachází neurony, které mají nějakou přímou i nepřímou vazbu samy na sebe, označujeme síť jako rekurentní. Ve zmíněném seznamu jsou první tři typy rekurentními sítěmi. Pro takové sítě je nutné využít jim specifických přístupů učení.

Pro účely této práce je nejdůležitější dopředná neuronová síť. Neurony jsou v takové síti uspořádány do plně propojených vrstev. První, neboli vstupní vrstva přijímá vstup neuronové sítě. Na opačném konci se nachází vrstva výstupní, která udává výstup celé sítě. Mezi nimi se poté může nacházet libovolný počet skrytých vrstev. Pravidlem v dopředných neuronových sítích je, že každý neuron skryté vrstvy je připojen pouze na neurony předešlé vrstvy a jeho výstupy jsou připojeny na neurony následující vrstvy. Pokud je každý jeden neuron z jedné vrstvy propojen s každým neuronem z předešlé vrstvy, jedná se o plně propojenou vrstvu. Příklad takové plně propojené sítě je znázorněn na obrázku 2.2. Síť, která má velké množství vrstev, se nazývá hluboká neuronová síť (zkráceně DNN z anglického „Deep neural network“).

Dále můžeme sítě dělit podle jejich aplikace, jako jsou: asociace, klasifikace, shlukování, predikce, optimalizace a podobně. V práci budou využity pouze sítě pro klasifikaci.

Již bylo zmíněno, že důležitou vlastností neuronových sítí je schopnost se učit. V rámci této práce bude přestavena typická technika pro klasifikační sítě – učení s učitelem, tedy

v datové sadě budou dostupné dvojice trénovacího vstupu a jeho požadovaného výstupu. Učení probíhá díky úpravě vah a hodnot biasů, které se využívají při výpočtu aktivace neuronů. Nejčastějším způsobem úprav těchto vah je algoritmus Backpropagation (česky algoritmus zpětného šíření chyby) [32]. Pro použití algoritmu je nutné, aby aktivační funkce všech neuronů byly diferencovatelné, a je třeba dodat účelovou funkci (označována také jako chybová funkce nebo anglicky „Loss function“). Ta je v průběhu učení minimalizována, tedy měla by vyjadřovat odchylku odezvy sítě od požadované hodnoty výstupu při přiložení daného vzoru na vstup. Minimalizace probíhá pomocí metody gradientního sestupu, která určuje změny vah jako zápornou hodnotu aktuálního gradientu. Samotný gradientní sestup lze realizovat následujícími způsoby:

- Stochastický přístup, Stochastic Gradiend Descent (SGD): Prvky se z trénovací množiny vybírají náhodně, změna vah se provádí po každém prvku.
- Dávkový přístup, Batch Gradiend Descent (BGD): Spočte se suma chyb pro všechny prvky, změna vah se tedy provádí až po vyhodnocení všech prvků.
- Mini-Batch Gradiend Descent (MBGD): Spočte se suma chyb pro zvolený počet prvků, které jsou náhodně zvoleny z trénovací množiny, následně se provede úprava vah (kombinace předchozích dvou přístupů).

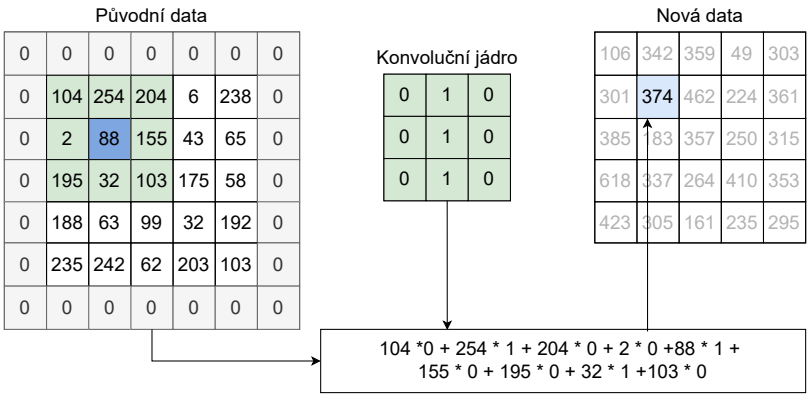
2.2 Konvoluční neuronové sítě

Důležitým pokrokem bylo vytvoření konvolučních neuronových sítí. Ty k současně popsanému modelu přidávají konvoluční vrstvy, aktivační vrstvy a seskupující filtry. Operace konvoluce bude pro lepší znázornění představena pro dvourozměrná data, kde je zadána následujícím vztahem:

$$I_{new}(u,v) = \sum_{i=-a}^{b} \sum_{j=-c}^{d} h(i,j) I(u+i, v+j) \tag{2.8}$$

kde $I_{new}(u,v)$ je hodnota nových dat na pozici u, v, která je vypočítána jako součet součinů hodnot původních dat I s konvolučním filtrem h o rozměrech $a+b$ a $c+d$. V praxi často platí $a = b = c = d$, tedy rozměry masky se dají vyjádřit jako $(2a+1)\cdot(2a+1)$. Operace konvoluce je znázorněna na obrázku 2.3. Do operace konvoluce vstupuje ještě parametr „Stride“, který určuje počet polí, o které se posune konvoluční jádro při každé iteraci výpočtu, a parametr „Padding“, který popisuje šířku doplnění okrajů originálních dat zadanou hodnotou (opět znázorněn na obrázku 2.3). Konvoluční vrstva typicky obsahuje více konvolučních filtrů a přijímá více než jeden vstupní kanál – představuje hloubkovou dimenzi vstupních dat dané vrstvy. Příklad více vstupujících kanálů do konvoluční vrstvy je znázorněn na obrázku 2.5, kde je řešena problematika klasifikace obrazu. S konvolučními vrstvami úzce souvisí aktivační vrstvy, které na každý prvek nově vytvořených dat aplikují některou z aktivačních funkcí, tak jako u klasického neuronu, který byl popsaný výše.

Seskupující (anglicky „Pooling“) filtry jsou určeny k redukci rozměrů dat. Pro jejich činnost je třeba definovat velikost masky a typ seskupení. Mezi časté typy se řadí například operace maxima – vybere maximální hodnotu z masky, minima – vybere minimální hodnotu z masky a průměru – vypočítá průměr v masce. Činnost takových filtrů je znázorněna na obrázku 2.4.

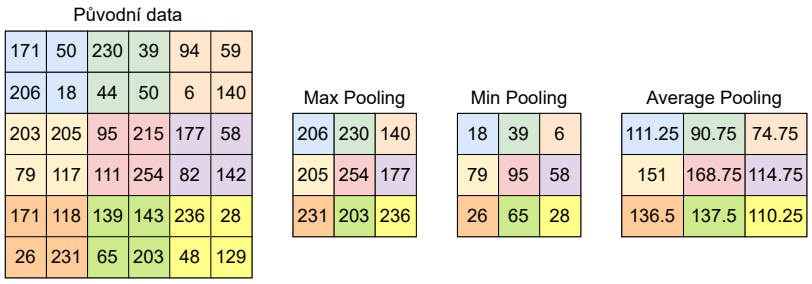


Obrázek 2.3: Příklad principu operace konvoluce nad 2D daty s parametry „Stride“=1 a „Padding“=1. Šedě vyznačená pole v původních datech jsou vytvořena díky volbě parametru „Padding“.

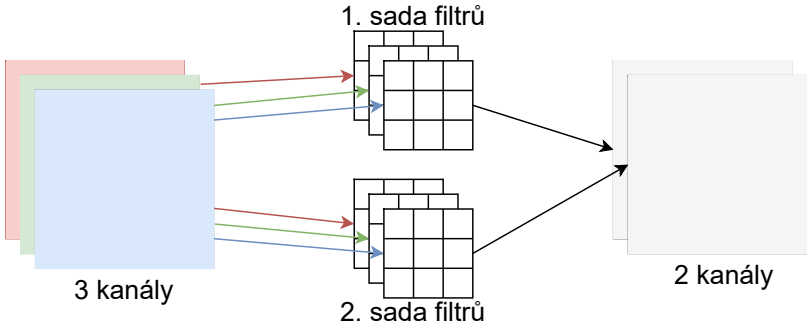
Konvoluční neuronové sítě jsou v současné době velice úspěšné především v oblasti počítačového vidění. Za první konvoluční neuronovou síť je považován Le-Net5 [31], který byl navržený pro klasifikaci ručně psaných číslic (bude podrobněji rozebrán v sekci 3.1). Za průlomovou CNN by se dala označit Alex-Net [18], která zvítězila v roce 2012 v soutěži „ImageNet Large Scale Visual Recognition Challenge“ (ILSVRC) [6]. Soutěž se zaměřuje na detekci a klasifikaci objektů v obrázku. Od tohoto ročníku byla každoročně vítězná síť založena na nějaké variantě konvoluční neuronové sítě.

Jak je z rovnic 2.7 a 2.8 patrné, při výpočtu inference konvoluční neuronové sítě jsou nejčastějšími operacemi násobení a sčítání, které se označují jako MAC operace (z anglického „Multiply-and-accumulate“). Ty dle [39] zastupují více než 99% operací, a proto jistě výrazně přispívají k časové a energetické náročnosti řešení. Konvoluční neuronové sítě se od průlomového Alex-Netu vylepšovaly hlavně v oblasti kvality výsledku. Například v samotné soutěži ILSVRC, kde dosáhl Alex-Net Top-1 přesnosti kolem 63%, přesahují dnešní řešení hranici 90% přesnosti [37]. Ovšem tento vývoj se velmi podepsal na složitosti struktur a počtu parametrů sítí, s čímž také roste výpočetní náročnost trénování a inference. Například Alex-Net provede při výpočtu inference přibližně 371 milionů MAC operací, kdežto novější ResNet-50 [12] s 25,6 miliony parametry (Top-1 přesnost na ImageNetu 80,9%) provede 3,9 miliard MAC operací dle [35]. Existují i sítě zaměřené na efektivní výpočet, například MobileNet_v2 s 3,5 miliony parametrů, který provede 585 milionů MAC operací s Top-1 přesností na ImageNet 71,8% (dále popsán v sekci 2.2.2).

Tyto náročné operace by nemohly být rozumně provedeny bez použití hardwarové akcelerace pomocí grafických karet, nebo specializovaných TPU [17] (z anglického „Tensor Processing Unit“) či jiného specializovaného hardware. Ty je sice možné využít při trénování na výkonných strojích, kde energetická spotřeba není tak velkým faktorem, ovšem pro cílová zařízení by energetická náročnost neoptimalizovaných řešení byla nepřípustná (například v mobilním telefonu, autonomním vozidle, IoT zařízení, apod.). Tato skutečnost je prokázána v práci [39], podle které není možné na nejnovějším chytrém mobilním telefonu provádět inference sítě Alex-Net v reálném čase déle než hodinu, než dojde energie v baterii. Proto je klíčem k efektivnější inferenci CNN optimalizovat počet MAC operací. Energetická náročnost MAC operací plyne především z přístupu do paměti pro hodnoty vah či prvků konvolučních jader. Z tohoto důvodu je komprese těchto prvků klíčem k efektivnímu vy-



Obrázek 2.4: Příklad seskupujícího (anglicky „Pooling“) filtru s maskou velikosti 2x2. Barevné označení znázorňuje oblast seskupení.



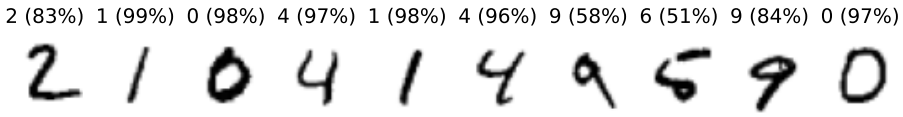
Obrázek 2.5: Příklad práce s kanály ve vrstvě konvoluční neuronové sítě. Vstupem jsou tři kanály představující RGB složky obrázku. Dvě sady tří konvolučních filtrů poté podle vstupu vytváří dva výstupní kanály.

hodnocení konvoluční neuronové sítě. Dále budou podrobněji popsány konkrétní neuronové sítě důležité pro tuto práci.

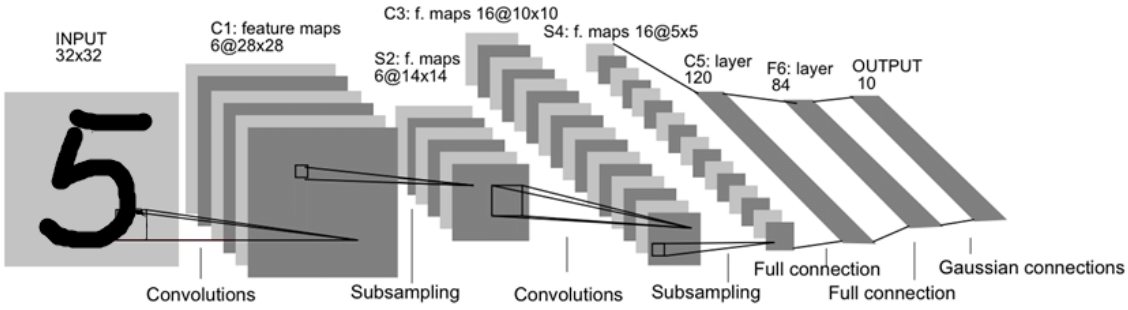
2.2.1 Le-Net-5

Jedná se o jednoduchou dopřednou konvoluční neuronovou síť, která byla původně navržena pro klasifikaci ručně psaných číslic pro poštovní službu Spojených států amerických pro strojové třídění zásilek podle poštovního směrovacího čísla. Jedná se o jednu z prvních konvolučních neuronových sítí, která byla navržena Yannem LeCunem v roce 1998 [31]. Skládá se z pěti vrstev, které jsou znázorněny v tabulce 2.1 nebo také na obrázku 2.7.

Vstupními daty pro tuto síť, jak je patrné z tabulky, jsou šedotónové obrázky o rozměru 32×32 pixelů. Přímo psané číslice v tomto formátu poskytuje dataset MNIST [7], ten obsahuje 60 tisíc popsaných trénovacích vstupů a 10 tisíc popsaných validačních vstupů. Příklad těchto dat je znázorněn na obrázku 2.6. Je však možné natrénovat síť i pro klasifikaci jiných cílů, jako například na datasetu Fashion MNIST [38], kde je cílem určovat různé oděvy. V současné době se tato síť často využívá k demonstračním či testovacím účelům pro techniky nad konvolučními neuronovými sítěmi, pro její nízkou výpočetní náročnost.



Obrázek 2.6: Příklad vstupů z datasetu MNIST společně s pravděpodobnostmi klasifikace do dané třídy sítě Le-Net-5



Obrázek 2.7: Schéma struktury neuronové sítě Le-Net-5 pro klasifikaci číslic z datasetu MNIST[1].

Vrstva	Kanálů	Výstup	Jádro	Stride	Aktivační funkce
Vstup	1	32×32	–	–	–
Konvoluční	6	28×28	5×5	1	Tanh
Avg Pooling	6	14×14	2×2	2	–
Konvoluční	16	10×10	5×5	1	Tanh
Avg Pooling	16	5×5	2×2	2	–
Konvoluční	120	1×1	5×5	1	Tanh
Plně propojená	84	–	–	–	Tanh
Plně propojená	10	–	–	–	Softmax

Tabulka 2.1: Struktura sítě Le-Net

2.2.2 MobileNet_v2

MobileNet_v2 [33] je neuronová síť patřící do třídy sítí MobileNets zaměřené na efektivní zpracování vstupních dat na mobilních zařízeních při současném zachování rozumné přesnosti. Pracuje s daty, která poskytuje dataset ImageNet [6]. Ten obsahuje obrázky, které lze dle obsahu zařadit do jedné z tisíce tříd (příkladem mohou být pneumatika, hrad, koště, panda apod.). Lze jej vyhodnocovat i na subdatasetech, jako například Imagenette [14], který obsahuje reprezentanty z 10 skupin (či jeho rozšířenou variantu Imagewang). Přesnosti se často určují podle metrik Top-N, kde N značí, v kolika nejpravděpodobnějších třídách určených sítí se může správná odpověď nacházet. Zjednodušeně řečeno, pokud je

[1]Obrázek převzat z https://www.researchgate.net/figure/The-architecture-of-LeNet-5-23-a-CNN-used-for-digits-recognition-for-the-MNIST-dataset_fig2_321665783.

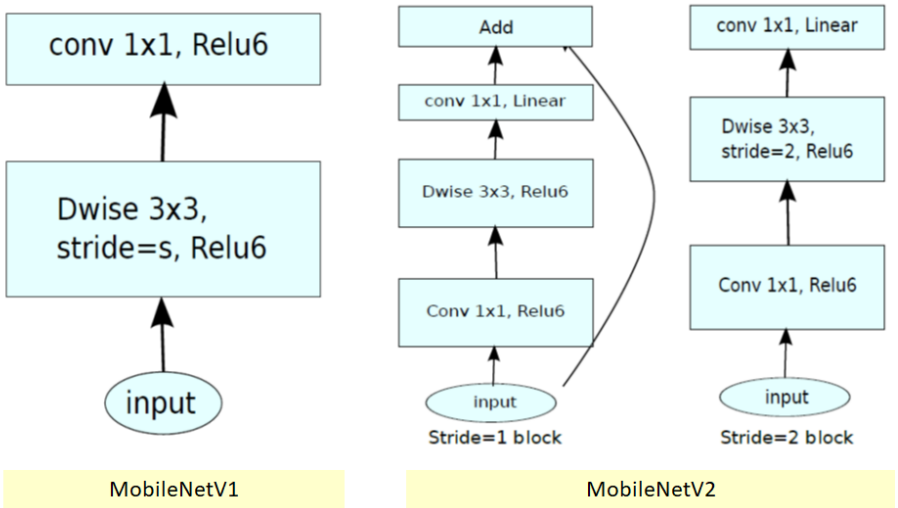
zvolena metrika Top-5, pak se odpověď považuje za správnou, je-li správná třída mezi pěti nejpravděpodobnějšími třídami dle výstupu neuronové sítě. Na ImageNetu má tato síť Top-1 přesnost klasifikace 71,88% a Top-5 přesnost 90,29%.

Sítě architektury MobileNet využívají Depthwise Separable Convolutions – rozdělení operace konvoluce na dvě vrstvy. První se nazývá Depthwise convolution, která využívá jeden filtr pro každý vstupní kanál. Na vzniklé tensory je poté aplikována Pointwise 1×1 konvoluce. Díky tomuto přístupu je dosaženo velké úspory parametrů. Detailněji je tento přistup popsán v [13] a je znázorněn na obrázku 2.8.

Dále byla architektura rozšířena prací [33], která přidává bloky vrstvy zvané Bottleneck znázorněné na obrázku 2.8, které mohou obsahovat reziduální propojení. Výsledná síť je charakterizována v tabulce 2.2.

Vrstva	Vstup	Výstupních kanálů	Opakování	Stride
conv2d	$224^2 \times 3$	32	1	2
bottleneck	$112^2 \times 32$	16	1	1
bottleneck	$112^2 \times 16$	24	2	2
bottleneck	$56^2 \times 24$	32	3	2
bottleneck	$28^2 \times 32$	64	4	2
bottleneck	$14^2 \times 64$	96	3	1
bottleneck	$14^2 \times 96$	160	3	2
bottleneck	$7^2 \times 160$	320	1	1
conv2d 1×1	$7^2 \times 320$	1280	1	1
avgpool 7×7	$7^2 \times 1280$	–	1	–
conv2d 1×1	$1 \times 1 \times 1280$	1000	–	

Tabulka 2.2: Struktura sítě MobileNet_v2 – opakování značí, kolikrát za sebou se daný blok s identickým nastavením nachází



Obrázek 2.8: Bloky použité v sítích typu MobileNet[2] – vlevo Depthwise separable convolution, vpravo Bottleneck bloky.

[1]Obrázek převzat z https://developer.ridgerun.com/wiki/index.php?title=GstInference/Supported_architectures/MobileNet. Originály obrázků jsou zdrojem převzaty z [33].

2.3 Optimalizační algoritmy

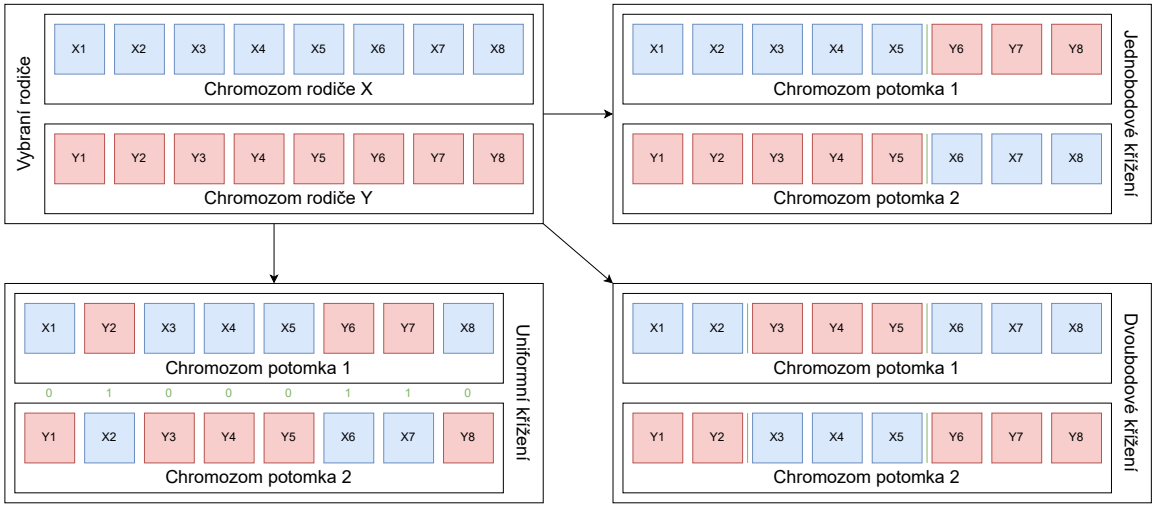
Pro účely implementované komprese neuronové sítě, která bude popsána dále v sekci 2.5, je třeba využít optimalizačního algoritmu. Cílem takových algoritmů je snažit se vyhledat takové hodnoty proměnných, pro které daná účelová funkce nabývá maximální či minimální hodnoty. V praxi pak tato funkce reprezentuje určitý problém, který je třeba optimalizovat, například návrh obvodu [24], návrh antén [21], nebo v případě této práce určení parametrů pro kompresi neuronové sítě. Optimalizační algoritmy můžeme dělit do dvou kategorií: jednoúčelové – optimalizuje jedno kritérium v zadaném systému (například pouze správnost obvodu) a víceúčelové – optimalizuje více kritérií najednou (například správnost obvodu a jeho cenu). Při využití vícekriteriální optimalizace je třeba správně vyvážit vliv jednotlivých aspektů, aby v průběhu optimalizace nedošlo k upřednostnění jednoho kritéria před ostatními (například v příkladu s optimalizací obvodu vznikne nefunkční obvod s minimem komponent). Samotný optimalizační algoritmus poté určuje, jaké hodnoty se do účelové funkce pro vyhodnocení vloží.

Nejjednodušší z optimalizačních algoritmů je náhodné vyhledávání (anglicky „Random Search“), které se využívá jako reference pro vyhodnocování efektivity ostatních optimalizačních algoritmů. Operuje na principu náhodného vyhledávání – tedy do každé optimalizované proměnné dosadí náhodnou hodnotu, a poté provede vyhodnocení pomocí účelové funkce. Jeho výstupem v čase je posloupnost hodnot účelové funkce, přičemž kandidátní řešení vedoucí na nejlepší hodnotu účelové funkce považujeme za řešení optimalizačního problému. Nevýhodou náhodného prohledávání je, že algoritmus nedokáže předat zjištěnou informaci o vhodnosti nebo nevhodnosti kandidátních řešení do dalších iterací. Pokud by některý z porovnávaných optimalizačních algoritmů dosahoval statisticky horších výsledků než náhodné vyhledávání, pak není efektivní, je chybný (chybně implementovaný nebo chybně nastavené parametry), nebo řešení problému má zvláštní charakter (problém hledání jehly v kupce sena).

Dalším představitelem optimalizačních algoritmů je genetický algoritmus [25] (zkráceně GA). Inspirací tohoto přístupu je evoluční vývoj živočišných druhů. Algoritmus pracuje se skupinou řešení, která se označují jako populace. Konkrétní řešení je označeno jako jednotlivec a jeho seznam hodnot pro účelovou funkci se nazývá jako chromozom (jednotlivé hodnoty jsou pak označovány jako geny). Místo účelové funkce se používá pojem fitness funkce, neboť určuje fitness hodnotu – kvalitu jedince z hlediska evoluce. GA pracuje iterativně následujícím způsobem:

1. Náhodně vygeneruje počáteční populaci o daném počtu jedinců.
2. Ohodnotí všechny jedince pomocí fitness funkce.
3. Vybere rodiče ke křížení.
4. Provede křížení jedinců.
5. Provede mutace jedinců.
6. Připraví novou populaci o stejném počtu jedinců, jaký byl v generaci předchozí, a pokud nebylo splněno ukončovací pravidlo, pokračuje od bodu 2.

Populace v takovéto jedné iteraci GA je označována za generaci. Jak bylo zmíněno, prvotní populace je vygenerována náhodně, obdobně jako u algoritmu Random Search. Selekce rodičů se řídí pravidlem, že jedinci s vyšší fitness mají větší šanci podílet se na



Obrázek 2.9: Příklad seskupujícího druhů křížení v rámci GA

tvorbě další generace, přičemž jedinec může být vybrán jako rodič vícekrát. Takový výběr může být deterministický, tedy vybere se potřebný počet nejlepších jedinců, náhodný výběr seřazené generace, kde pravděpodobnost výběru odpovídá fitness hodnotě jedince, nebo turnajem jedinců.

Jakmile jsou vybrány páry rodičů pro vytvoření nových potomků, přechází se ke křížení. Mezi metody křížený patří následující:

- Jednobodové – náhodně se vygeneruje bod křížení a jedinci si za tímto bodem prohodí části svých chromozomů.
- Vícebodové – náhodně se vygeneruje více bodů křížení, mezi kterými se části chromozomů prohazují.
- Uniformní – náhodně se pro každý gen vybere, ze kterého rodiče se do potomka převezme.

Metody jsou znázorněny na obrázku 2.9. Je nutné zdůraznit, že při jedné takové operaci křížení jsou vytvořeni dva komplementární potomci, jak je ukázáno i na obrázku. Po vytvoření nových jedinců jsou provedeny mutace, tedy náhodné změny některých genů nově vytvořených potomků s určitou pravděpodobností. Následně se z nově vytvořených potomků vytvoří nová generace populace a cyklus se opakuje. Při vytváření nové generace je možné uplatnit elitismus, tedy princip, kdy n nejlepších jedinců z minulé generace je zařazeno i do generace nové.

GA tedy oproti metodě náhodného prohledávání využívá znalosti optimalizovaného prostoru během průzkumu a snaží se prohledávat nadějná místa – křížení jedinců s dobrou fitness hodnotou. Mutace zajišťují schopnost algoritmu prohledávat nová řešení, je ale třeba je dobře vyvážit. Při příliš vysoké pravděpodobnosti mutace se z GA stává náhodné prohledávání a při příliš nízké pravděpodobnosti hrozí uváznutí algoritmu v lokálním optimu.

Particle swarm optimization [28] (zkráceně PSO, česky optimalizace hejnem částic) je také jedním z představitelů optimalizačních algoritmů. Inspirací pro tento algoritmus byl pohyb ptačího hejna při hledání potravy. Podobně jako GA pracuje se skupinou řešení, která se zde nazývá hejno (anglicky „swarm“). Konkrétní řešení je určeno částicí v n-rozměrném prostoru. Částice je komplexní struktura, která ukládá následující informace:

- Polohu $\mathbf{x}$ - n-rozměrný vektor reprezentující řešení.
- Rychlost $\mathbf{v}$ - n-rozměrný vektor, kde každý prvek reprezentuje rychlost v dané dimenzi.
- Vlastní nejlepší pozice $\mathbf{x_{my\_best}}$ - n-rozměrný vektor reprezentující nejlepší řešení, na kterém se daná částice v průběhu běhu algoritmu nacházela.

Algoritmus nejprve vygeneruje nové hejno, kde každé částici přiřadí v každém rozměru i náhodnou pozici x_i z intervalu $< x_{ilow}, x_{imax} >$, kde x_{ilow} značí minimální a x_{imax} maximální hodnotu v daném rozměru i. Obdobně je přiřazena počáteční rychlost v_i z intervalu $< -v_{imax}, v_{imax} >$, kde v_{imax} značí maximální rychlost v dané dimenzi. Následně je nastaven počáteční vektor $\mathbf{x_{my\_best}}$ všech částic na aktuální pozice $\mathbf{x}$. Poté se provede ohodnocení každé částice a určí se nejlepší ohodnocení v celém hejnu. Pro každou částici se v dalším kroku vypočítá její nová rychlost a pozice pomocí následujících výrazů:

$$\mathbf{v_n} = \omega\mathbf{v} + c_p r_p(\mathbf{x_{my\_best}} - \mathbf{x}) + c_g r_g(\mathbf{x_{best}} - \mathbf{x}) \tag{2.9}$$

$$\mathbf{x_n} = \mathbf{x} + \mathbf{v} \tag{2.10}$$

kde $\mathbf{v_n}$ a $\mathbf{x_n}$ značí novou rychlost, respektive polohu částice, ω značí koeficient setrvačnosti, který se doporučuje volit z intervalu $< 0,4; 0,9 >$, $\mathbf{v}$, $\mathbf{x_{my\_best}}$, $\mathbf{x}$ jsou symboly ze struktury dané částice, c_p a c_g jsou kognitivní a sociální koeficienty, které se doporučují volit stejně, obvykle 2.05, r_p a r_g jsou náhodná čísla z intervalu $< 0, 1 >$, $\mathbf{x_{best}}$ představuje nejlepší pozici hejna. Pokud ještě nebyl dosažen požadovaný počet iterací, tak algoritmus pokračuje opětovným ohodnocením částic a posunem. Algoritmus po skončení vrátí nejlepší nalezenou pozici v prostoru.

Oproti GA a náhodnému prohledávání vyžaduje algoritmus PSO pro svůj výpočet spojitý prohledávací prostor pro částice, kdežto ostatní zmíněné algoritmy dokáží pracovat i s diskrétním prostorem. Pokud je ale pozice částic diskretizována do potřebných hodnot, lze PSO využít i pro prohledávání diskrétního prostoru. Problémem PSO by mohlo být ustálení řešení - tedy situace, kde se všechny částice dostanou na pozici, která je v aktuálním běhu nejlépe hodnocená, prohledávání přestává a hrozí uvíznutí v lokálním optimu.

Posledním představeným optimalizačním algoritmem je BH („Black hole“) [19]. Ten je inspirovaný fyzikou černých děr, jak z názvu vyplývá. Podobně jako PSO pracuje s body, které se v prohledávaném prostoru nazývají hvězdy. Na začátku svého běhu tedy vygeneruje N hvězd a vyhodnotí jejich účelovou funkci. Nejlepší nalezené řešení se označí jako červí díra. Ostatní body jsou poté atraktované k této díře, čímž mění pozici v prostoru. Pohyb je popsán následujícím výrazem:

$$\mathbf{x_i}(t+1) = \mathbf{x_i}(t) + r * (\mathbf{x_{BH}} - \mathbf{x_i}(\mathbf{t})) \qquad i = 1 \ldots m \tag{2.11}$$

kde $\mathbf{x_i}(t)$, $\mathbf{x_i}(t+1)$ značí pozici hvězdy $\mathbf{x_i}$ v čase t respektive $t+1$, r značí náhodnou hodnotu z intervalu $< 0, 1 >$ a $\mathbf{x_{BH}}$ značí pozici aktuální červí díry. Index i poté značí posouvanou hvězdu, přičemž platí, že $m = N - 1$ (samotná pozice červí díry se nemění). Po změně pozic přitahovaných částic se opět vyhodnocuje účelová funkce a případně se určí nová červí díra. Následně se určí poloměr horizontu událostí následujícím výrazem:

$$R = \frac{f(\mathbf{x_{BH}})}{\sum_{i=1}^{m} f(\mathbf{x_i})} \tag{2.12}$$

kde R značí daný poloměr horizontu událostí a $f()$ značí účelovou funkci (ostatní značení je stejné jako ve výrazu 2.11). Pokud se některá hvězda dostane pod horizont událostí (vzdálenost konkrétní hvězdy a černé díry je menší než R), pak tato hvězda zaniká a vzniká místo ní nová s náhodnou pozicí v prohledávaném prostoru. Díky tomuto mechanismu nikdy nedojde k ustálení algoritmu v nějakém z lokálních maxim a prohledávání bude neustále pokračovat.

2.4 Shlukování

Shlukování je úloha, která má za cíl roztřídění vstupních dat do skupin, které jsou si v zadaných metrikách podobné. Trénovací data v tomto případě nemají žádné popisy, dá se tak hovořit o úloze strojového učení bez učitele. Prostor, ve kterém shlukování probíhá, se dá jednoduše představit jako n-dimenzionální, kde n je rovno počtu zvolených metrik pro porovnávání. Shlukovací algoritmy poté v tomto prostoru tvoří skupiny prvků podle jejich vzájemných vzdáleností. Z tohoto důvodu je třeba dát důraz na to, aby vzdálenosti jednotlivých metrik byly porovnatelné – častým řešením je normalizace.

Příkladem takového algoritmu je K-Means [16]. Toto shlukování probíhá následujícím způsobem:

1. V prostoru se náhodně určí k centroidů.
2. Každému bodu ze zadané množiny se přiřadí nejbližší z centroidů.
3. Vypočítá se nová pozice centroidů jako těžiště podle přiřazených dat.
4. Opakuje se od kroku 2, dokud nedojde k ustálení.

Výstupem je pak přiřazení všech vstupních bodů do příslušných shluků a centroid pro každý shluk. Důležité pro tento algoritmus je mít normalizované rozměry ve všech dimenzích, aby vzdálenosti navzájem korespondovaly a bylo možné správně určovat přiřazení bodů.

Pro podobné účely by bylo možné využít i Gaussian Mixture Model (GMM) trénovaný pomocí EM algoritmu na vstupních datech (EM algoritmus definovaný dále). Princip spočívá ve vytvoření modelu daného počtu gaussovských rozložení, která co nejlépe popisují daný prostor. Gaussovská neboli normální rozložení jsou definovaná následovně:

$$p(x) = \mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.13}$$

$$\mu_{ML} = \underset{\eta}{\operatorname{argmax}}\ p(\mathbf{x}|\mu, \sigma^2) = \frac{1}{N}\sum_n x_n \tag{2.14}$$

$$\sigma^2_{ML} = \underset{\sigma^2}{\operatorname{argmax}}\ p(\mathbf{x}|\mu, \sigma^2) = \frac{1}{N}\sum_n (x_n - \mu_{ML})^2 \tag{2.15}$$

kde $\mathbf{x}$ značí sadu trénovacích dat, μ a σ^2 značí střední hodnotu, respektive rozptyl, přičemž μ_{ML} a σ^2_{ML} značí odhady s maximální věrohodností střední hodnoty, respektive rozptylu vzhledem k trénovacím datům $\mathbf{x}$.

Samotný GMM je definovaný jako:

$$p(x) = \sum_z p(x|z)P(z) = \sum_c \mathcal{N}(x|\mu_c, \sigma_c^2)\mathrm{Cat}(z = c|\pi) \tag{2.16}$$

kde z značí skrytou proměnnou (zjednodušeně řečeno určuje, jakou komponentou je daný bod vygenerovaný), c značí index gaussovské komponenty, Cat značí diskrétní rozdělení.

EM algoritmus (z anglického „Expectation Maximization“) je iterační algoritmus pro odhad parametrů modelu s maximální aposteriorní pravděpodobností (MAP). Cílem je maximalizovat následující pravděpodobnost:

$$\ln p(\mathbf{X}|\eta) = \mathcal{Q}(q(\mathbf{Z}), \eta) + H(q(\mathbf{Z})) + D_{KL}(q(\mathbf{Z})||p(\mathbf{Z}|\mathbf{X}, \eta)) \tag{2.17}$$

$$\text{Pomocná funkce: } \mathcal{Q}(q(\mathbf{Z}), \eta) = \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln p(\mathbf{X}, \mathbf{Z}|\eta) \tag{2.18}$$

$$\text{Entropie rozdělení: } H(q(\mathbf{Z})) = -\sum_{\mathbf{Z}} (q(\mathbf{Z}) \ln q(\mathbf{Z})) \tag{2.19}$$

$$\text{K–L divergence: } D_{KL}(q(\mathbf{Z})||p(\mathbf{Z}|\mathbf{X}, \eta)) = -\sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \frac{p(\mathbf{Z}|\mathbf{X}, \eta)}{q(\mathbf{Z})} \tag{2.20}$$

kde $q(\mathbf{Z})$ je pravděpodobnostní rozdělení přes skrytou proměnnou, D_{KL} označuje Kullbackova–Leiblerovu divergenci, která udává rozdílnost dvou rozdělení, $\mathbf{X}$ označuje trénovací množinu dat a η označuje parametry trénovaného modelu (v případě GMM kovariančních matic gaussovských komponent, tedy $\eta = (\mu_c, \sigma_c^2, \pi_c)$). EM algoritmus maximalizuje tuto pravděpodobnost ve dvou krocích:

1. Krok E: $q(\mathbf{Z}) := P(\mathbf{Z}|\mathbf{X}, \eta)$ – Pro GMM se dá zjednodušeně říct, že se body měkce přiřadí k jednotlivým gaussovským komponentám (Určí se pravděpodobnosti, pro každou z nich, že do nich daný bod patří. Součet všech pravděpodobností náležící k jednomu budu je roven jedné). Výpočet této pravděpodobnosti popisuje následující rovnice:
$$q(z_n = c) = \gamma_{nc} = \frac{\mathcal{N}(x_n|\mu_c, \sigma_c^2)\pi_c}{\sum_k \mathcal{N}(x_k|\mu_k, \sigma_k^2)\pi_k} \tag{2.21}$$

2. Krok M: $\eta_{i+1} = \underset{\eta}{\text{argmax}}\ \mathcal{Q}(q(\mathbf{Z}), \eta_\mathbf{i})$ – Opět pro GMM se dá říci, že se vypočítají nové hodnoty μ_c, σ_c^2 a π_c s maximální věrohodností vzhledem k přiřazeným datům. Lze popsat následujícími rovnicemi:
$$\mu_c = \frac{\sum_n \gamma_{nc} x_n}{\sum_n \gamma_{nc}} \tag{2.22}$$
$$\sigma_c^2 = \frac{\sum_n (x_n - \mu_c)^2}{\sum_n \gamma_{nc}} \tag{2.23}$$
$$\pi_c = \frac{\sum_n \gamma_{nc}}{\sum_c \sum_n \gamma_{nc}} \tag{2.24}$$

Pokud budou tyto dva kroky EM algoritmu iterativně opakovány, bude natrénován model. Ten poté bude možné použít k rozřazení bodů jako jejich přiřazení k jednotlivým gaussovským komponentám. Z definice normálního rozdělení je také možné získat střední hodnotu.

2.5 Komprese neuronové sítě

Jak již bylo naznačeno v sekci 2.2, komprese neuronové sítě obvykle vede k nižší energetické náročnosti jejího výpočtu. Práce zkoumá pouze náročnost inference – vyhodnocení neuronové sítě na vstupu (nesnaží se optimalizovat proces učení). To je možné provést díky tomu, že neuronové sítě mají z podstaty své architektury velké množství redundance v rámci struktury a parametrů, přičemž není vždy potřebná pro přesnou inferenci [11, 35]. Bohužel ale s rostoucí kompresí zpravidla roste i chyba sítě, a je tedy nutné ke kompresi přistupovat systematicky. Pro uložení neuronové sítě je třeba zapsat hlavně strukturu a potřebné parametry (váhy a biasy). Komprese lze dosáhnout třemi základními cestami:

- změnou výpočetní struktury sítě, která vede k menší síti, tedy menší náročnosti jak na výpočet, tak na paměťovou náročnost pro uložení vah [34]. Do této kategorie lze zařadit techniku prořezávání [2] (anglicky „Pruning“), která se zaměřuje na odstranění nejméně důležitých vah (pokud je váha 0, lze odstranit příslušný propoj);
- úpravou dat modifikací vah sítě, jejíž hlavním zástupcem je kvantizace [3] se záměrem uspořit prostor změnou přesnosti dat;
- aproximací aritmetických operací, což je zajímavé hlavně při využití akcelerátorů [10].

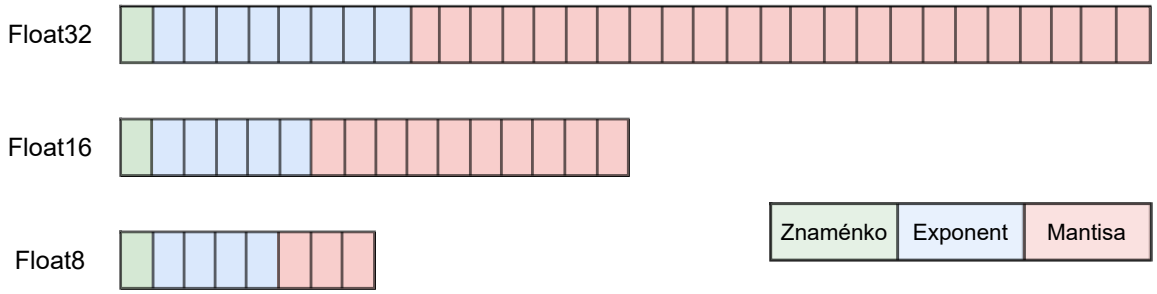
Tato práce se bude věnovat kompresi pomocí úpravy a modifikace vah sítě. Proto budou techniky této cesty popsány podrobněji.

Myšlenka techniky Prunning [2] spočívá v tom, že komplexní konvoluční neuronové sítě využívají více parametrů, než je třeba k přesné inferenci, a je tak tedy možné tyto váhy identifikovat a ze systému odstranit. Není tak pouze redukována kapacita paměti vah, ale je i přímo redukován počet operací, které je nutné pro inferenci neuronové sítě provést. Plně propojené vrstvy v CNN využívají poměrově největší počet vah a dovolují tak vysokou kompresi pomocí této techniky. Na druhou stranu konvoluční vrstvy CNN sice koncentrují poměrově méně vah, ovšem jsou původem většiny operací při výpočtu. Proto i nízká komprese v těchto vrstvách vede k vysoké úspoře energie, neboť jsou váhy využívány opakovaně. Prořezávání lze implementovat dvěma způsoby.

(a) Takzvané řídké prořezávání (anglicky „sparse Prunning“) posoudí každou váhu zvlášť, zdali je užitečná k výpočtu, či nikoliv. Tento postup typicky vede k nižší ztrátě přesnosti, ovšem nedokáže systematicky vylepšit energetickou náročnost výpočtu, neboť je těžké akcelerovat řídké tensory. Je proto využíván především pro snížení zabrané kapacity úložiště. Za účelem zjednodušení výpočtu se využívá sofistikovanějších metod prořezávání, které odstraňují méně potřebné váhy ve vzorech. Díky tomu je umožněná akcelerace výpočtu.

(b) Dalším způsobem, jak modifikovat váhy, je modifikovat jejich reprezentace. V počítačích jsou čísla v paměti reprezentovaná binárně, přičemž obecně platí, že čím více bitů číslo zabírá, tím přesnější je, případně jeho reprezentace může mít vyšší rozsah. Pokud tedy ve vahách sítě není třeba taková míra přesnosti či tak velký rozsah, je možné čísla zakódovat pomocí reprezentací s nižším počtem bitů a dosáhnout tak požadované komprese. Toto je myšlenka kvantizace vah. Pro hlubší pochopení je nejprve třeba představit reprezentace čísel, se kterými se při výpočtech pracuje.

Nejčastěji se při programování pracuje s typy `int` a `float`. První zmíněný popisuje celá čísla (odsud plyne jeho název z anglického „integer“, což znamená celé číslo). Dále se pak hodnoty `int` rozdělují podle počtu bitů využitých k reprezentaci a podle toho, zdali



Obrázek 2.10: Různé typy reprezentace `float`. `float32` je definován podle IEEE 754 [1]. `float16` byl do jmenovaného standardu přidán v revizi z roku 2008. `float8` byl převzat z práce [26].

jsou se znaménkem či bez něj („unsigned“). Hodnota čísla v desítkové soustavě se poté z reprezentace o n bitech vypočítá tak, že se pro každý bit na pozici i v rozsahu 0 až $n-1$ vypočítá 2^i a tyto hodnoty se sečtou. Pokud je číslo se znaménkem, jeden bit je vyhrazený pro znaménko a rozsah pozic i je snížený na 0 až $n-2$. Přehled různých používaných reprezentací typu `int` je znázorněn v tabulce 2.3. Pokud uvažujeme pevnou řádovou čárku (tj. několik bitů je vyhrazeno pro celou část a další bity pro desetinnou část), výpočet rozsahů se provede obdobně.

Typ	Bitů	Znaménko	Rozsah
`int8`	8	signed	-128 až 127
		unsigned	0 až 255
`int16`	16	signed	-32 768 až 32 767
		unsigned	0 až 65 535
`int32`	32	signed	-2^{31} až $2^{31}-1$
		unsigned	0 až $2^{32}-1$
`int64`	64	signed	-2^{63} až $2^{63}-1$
		unsigned	0 až $2^{64}-1$

Tabulka 2.3: Nejčastější reprezentace čísel typu `int`

Druhým zmíněným způsobem reprezentace je typ `float` který je popsaný standardem IEEE 754 [1]. Jak již název napovídá, jedná se o způsob uložení čísel s pohyblivou čárkou (z anglického „floating point number“). Zde se pole n bitů dělí do tří skupin: znaménko s, exponent e a mantisu m. Toto dělení je znázorněné i na obrázku 2.10. Výpočet hodnoty z reprezentace je poté proveden pomocí výrazu 2.25. Jak je i ve vzorci znázorněno, reprezentace float šetří jeden bit mantisy implicitní jedničkou (jelikož se jedná o reprezentaci, kde se mantisa násobí exponentem, je zaručené, že na tomto místě vždy jednička bude). Dále specifikace obsahuje popis speciálních hodnot, pokud jsou jmenované tři části ve specifických hodnotách (mezi speciální hodnoty se řadí například nula, nekonečno, nebo hodnota NaN – Not a Number). V zápise

$$(-1)^s \cdot 1,(m) \cdot z^e \tag{2.25}$$

je z základ číselné soustavy, což je v tomto případě 2 (ostatní označení popsána výše). Z tohoto výrazu a daného typu uložené hodnoty float lze poté zjistit reprezentované intervaly, které jsou znázorněny v tabulce 2.4.

Typ	Bitů	E	M	Největší číslo	Nejmenší kladné číslo
`float32`	32	8	23	$3,40 \times 10^{38}$	$1,17 \times 10^{-38}$
`float16`	16	5	10	65540	6.10×10^{-6}
`float8`	8	4	3	15360	0.125

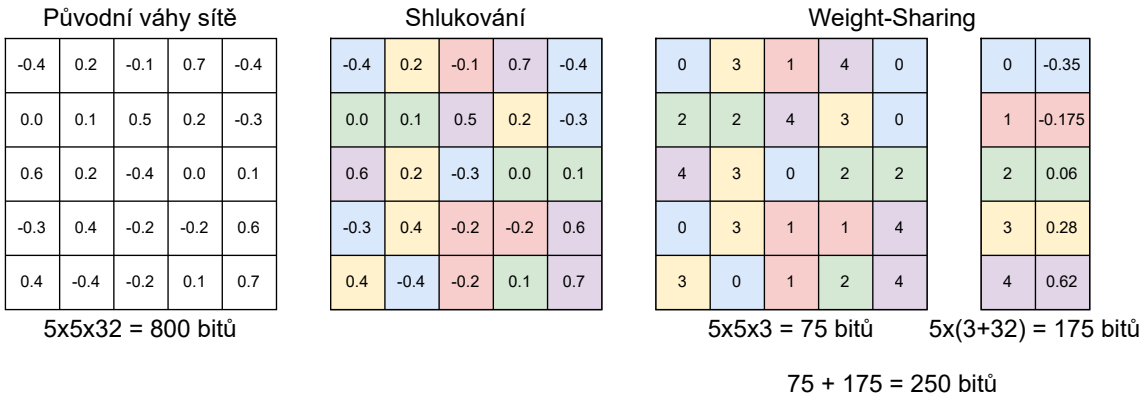
Tabulka 2.4: Různé druhy typů `float`. `float32` je definován podle IEEE 754 [1]. `float16` byl do jmenovaného standardu přidán v revizi z roku 2008. `float8` byl převzat z práce [26].

Pokud porovnáme tyto dva zmíněné typy, je patrné, že práce s typem `int` je daleko jednodušší než výpočty s typem `float`, přičemž jednodušší výpočty vedou k jednoduššímu hardware a nižší energetické náročnost výpočtu. Na druhou stranu čísla s plovoucí řádovou čárkou poskytují širší rozsah čísel. Nejčastěji se u neuronových sítí využívá výpočet s aritmetikou `float32`. Kvantizace tedy ke kompresi přistupuje tak, že nahradí hodnoty v jedné reprezentaci za podobné hodnoty v reprezentaci jiné. Klasickým příkladem může být převedení neuronové sítě z výpočtu nad `float32` do `float16`, díky čemuž je dosaženo dvojnásobné komprese, neboť nová reprezentace vyžaduje poloviční počet bitů. Změna reprezentace ovšem může vést ke snížení přesnosti sítě.

Využívají se i přístupy, kde je kvantizace uplatněna již během učení a síť se tak na metodu může přizpůsobit a dosáhnout tak lepších výsledků, než kdyby se trénovala na původní reprezentaci a poté převedla na jinou. Tímto způsobem lze síť natrénovat tak, že její váhy jsou reprezentovány typem `int`, což s sebou přináší benefity efektivního výpočtu na jednodušším hardware.

Tato práce se bude věnovat především technice Weight-Sharing, kterou lze zařadit jako podkategorii kvantizace, neboť pracuje se stejnou myšlenkou – tedy nahrazením čísel jinou reprezentací. Tato technika se ale neupíná na existující reprezentace čísel, ale místo toho vytváří své vlastní. Pracuje následujícím způsobem: nejprve vezme zadanou skupinu vah. Pro tuto skupinu provede operaci shlukování na předem zadaný počet skupin, typicky pomocí algoritmu K-Means popsaného v sekci 2.4. Následně je vytvořena tabulka, která obsahuje dvojice klíč–hodnota, přičemž klíčem je myšlen identifikátor shluku a hodnotou zástupce daného shluku, často pozice přiřazeného centroidu. Poté jsou váhy ve struktuře nahrazeny odkazy do této tabulky podle jejich přiřazeného shluku. Tento postup je znázorněn na obrázku 2.11.

Nejčastěji se komprese provádí pro skupiny vah ve vrstvách – tento typ bude nejvíce využívaný v této práci. Pro úplnost bude ale v kapitole 5 porovnán s postupem, kdy se jako daná skupina vezmou veškeré váhy v celé síti. Existují ovšem i jiné přístupy [27]. Zjednodušeně řečeno tedy technika Weight-Sharing využívá robustnosti neuronové sítě tak, že podobné hodnoty nahradí jejich zástupcem a do struktury tuto hodnotu uloží úsporně jejím klíčem, přičemž platí, že čím nižší je počet klíčů, tím vyšší komprese lze docílit. Konkrétně pokud by se podařilo veškeré váhy vrstvy reprezentovat pouze dvěma hodnotami, pak je možné jejich klíče zakódovat do jednoho bitu, a teoreticky bychom tak dosáhli 32-násobné komprese (pokud není započítána paměťová náročnost tabulky). Teoreticky by mohlo být možné nahradit celou vrstvu pouze jednou hodnotou a uložit tak váhy jedné vrstvy jako jedno číslo, ovšem v praxi toto nahrazení vede k velké ztrátě přesnosti sítě, a pokud by nevedlo, nejspíše by bylo možno optimalizovat samotnou strukturu sítě. Tento přístup vede k velké úspoře paměti, jak bude demonstrováno v kapitole 5, ovšem je poměrně náročný na správné určení parametrů. Je totiž nutné pro každou skupinu vah určit, na kolik shluků se bude daná skupina redukovat, přičemž je třeba podotknout, že se vrstvy v síti



Obrázek 2.11: Příklad techniky Weight-Sharing. V levé matici jsou znázorněny původní váhy sítě. V prostřední matici jsou váhy rozděleny pomocí shlukovacího algoritmu do 5 skupin. Nakonec jsou v pravé matici nahrazeny váhy indexy skupin a je vytvořena překladová tabulka, kde pro každou skupinu je vybrán centroid jako zástupce. Pod maticí původních vah a maticí s komprimovanými vahami je znázorněn potřebný počet bitů pro uložení.

navzájem ovlivňují, a je tak potřeba nalézt globálně optimální parametry. Návrh a řešení této problematiky budou uvedeny v následující kapitole.

Kapitola 3

Návrh řešení

V této kapitole budou představeny nosné myšlenky, které budou tvořit spojení mezi teorií popsanou v kapitole 2 a implementací, která bude rozvedena v kapitole 4. Nejprve bude v sekci 3.1 popsán návrh samotné komprese pomocí techniky Weight-Sharing. Následně bude rozvržen návrh použitých optimalizačních algoritmů, které budou využity v sekci 3.2.

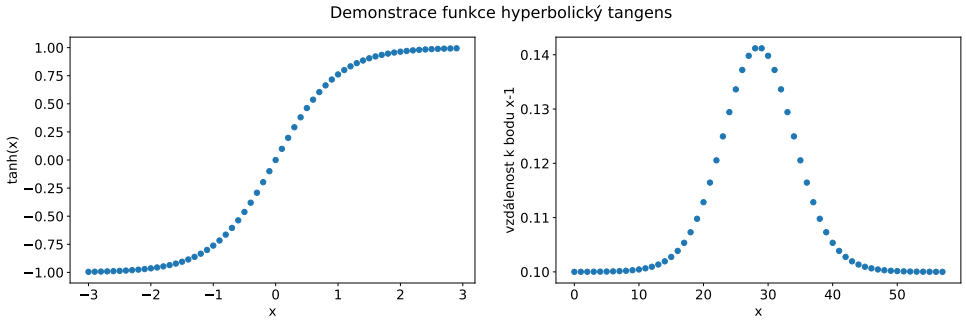
3.1 Návrh algoritmu komprese

Cílem práce je použití komprese Weight-Sharing a provedení experimentů s touto technikou. Je třeba podotknout, že pro vyhodnocení efektu komprese není nutné řešit uložení vah, a práce se tedy nebude zabývat implementací efektivního uložení. Zaměří se na vyhodnocení dopadu komprese vah na výkon sítě a návrh efektivního způsobu určování správných parametrů pro techniku Weight-Sharing. Dále se poté pokusíme prohloubit kompresi pomocí dodatečné kvantizace nad zástupci techniky Weight-Sharing, a v neposlední řadě se pokusíme navrhnout ladící techniku (dále „fine-tuning“), jejímž úkolem bude zvýšení přesnosti sítě při zachování dané úrovně komprese.

Jako cíl komprese byly vybrány dvě neuronové sítě. První z nich je Le-Net-5 popsaná v sekci 2.2.1, která byla zvolena pro své jednoduché vyhodnocení. Tato síť v experimentech klasifikuje vzory z datasetu MNIST [7] – tedy úkolem je klasifikace ručně psaných číslic zadaných jako šedotónový vstupní obraz o rozměrech 32×32 pixelů. Struktura sítě je vyznačená na obrázku 2.7. Podstatné je, že síť se skládá z pěti vrstev pracujících s vahami – třemi konvolučními vrstvami a dvěma plně propojenými vrstvami. Tyto vrstvy, konkrétně jejich váhy, budou cílem komprese algoritmu, buď jako celek a nebo jednotlivě po vrstvách. Komprese po vrstvách pro techniku Weight-Sharing bude zakódována jako seznam pěti hodnot $[x_1, x_2, x_3, x_4, x_5]$ tak, že každé x_n určuje počet shluků nad vahami v dané vrstvě n. Vrstvy budou v seznamu zastoupeny od vstupu sítě k výstupu, tedy první konvoluční vrstva je v seznamu označena jako x_1 až po poslední plně propojenou vrstvu označenou jako x_5.

Druhou zvolenou sítí je Mobilnet_v2 pro porovnání úspěšnosti techniky s prací [8], která se zabývá také metodou Weight-Sharing, ovšem nezavádí dodatečnou kvantizaci ani metodu ladění výsledné sítě. Síť v experimentech klasifikuje vstupní obrázky z datasetu Imagenette. Jak síť tak dataset jsou popsány v sekci 2.2.2. Pro kompresi je podstatné, že se síť skládá z 53 vrstev obsahujících váhy, které je možné zpracovat. Jejich reprezentace je obdobná jako u sítě Le-Net-5. Přesnost sítě je měřena na datasetu Imagenette.

Nejprve bude popsán způsob komprese přes celý model. Při aplikaci této metody se vytvoří skupina vah z celého modelu, zvolí se počet shluků a kvantizační parametr. Zpracování



Obrázek 3.1: Ukázka vzdáleností mezi body funkce hyperbolický tangens

vah poté funguje, jak bylo nastíněno v sekci 2.5, tedy váhy se shromáždí do jednoho pole a provede se nad ním shlukování pomocí algoritmu K-Means popsaného v sekci 2.4 s využitím požadovaného počtu shluků (jedno-dimenzionální shlukování). Výsledné centroidy se poté pomocí techniky kvantizace a daného parametru převedou do požadované přesnosti. Hodnoty původních vah se nahradí přiřazenými upravenými hodnotami centroidů clusterů. Nyní je možné změřit dopad komprese na přesnost sítě vyhodnocením na požadovaném datasetu.

Obdobně pracuje i komprese přes vrstvy. Zvolí se pořadí $[p_1, p_2, \ldots p_n]$, které popisuje, v jakém pořadí se mají jednotlivé vrstvy zpracovávat. Zpracování vrstvy poté funguje, jak bylo nastíněno v sekci 2.5, tedy všechny váhy vrstvy se shromáždí do jednoho pole, provede se nad ním shlukování pomocí algoritmu K-Means s využitím přiřazeného počtu shluků k dané vrstvě. Výsledné centroidy také převedou do požadované přesnosti a nahradí se jimi váhy v dané vrstvě a uzamknou se proti trénování. Nyní je oproti předchozímu přístupu možné zbylé neuzamčené váhy sítě dotrénovat a zvýšit tak přesnost. Jedná se ale o vysoce výpočetně náročnou operaci, která nebude v experimentech prováděna.

Jako parametry kvantizace je možné zvolit tři následující kvantizační přesnosti (jsou také znázorněny na obrázku 2.10 a v tabulce 2.4):

1. `f4` – `float32` – původní základní přesnost hodnot vah v síti dle IEEE 754.
2. `f2` – `float16` – reprezentace plovoucí řádové čárky na 16 bitech dle IEEE 754.
3. `f1` – `float8` – reprezentace plovoucí řádové čárky na 8 bitech inspirované podle práce [26].

Do kompresí zasahuje již zmíněná technika fine-tuning. Ta pracuje díky převedení prostoru shlukování technikou K-Means do druhé dimenze tak, aby shluky byly orientované více kolem průměrné hodnoty ze seznamu vah. Každé váze v seznamu je přidána druhá dimenze pomocí následujícího výrazu:

$$y = spread \cdot (max(\mathbf{w}) - min(\mathbf{w})) \cdot tanh(focus \cdot (x - mean(\mathbf{w})) \tag{3.1}$$

kde $\mathbf{w}$ je daná skupina vah, $spread$ a $focus$ jsou parametry techniky fine-tuning. Zjednodušeně řečeno se souřadnice druhé dimenze vypočítá dosazením váhy do funkce hyperbolického tangentu, přičemž jako nulový bod tangentu je zvolen průměr vah v dané skupině. Důvodem zvolení funkce hyperbolického tangentu je to, že je tak dosaženo zvětšení vzdáleností kolem nulové hodnoty a naopak relativního snížení vzdáleností dál od nulové hodnoty. Tento fakt je znázorněn na grafu 3.1. To následně vede shlukovací algoritmus K-Means k vytvoření více

shluků právě kolem průměru vah, kde se nachází nejvíce hodnot, a tím pádem bude tento rozsah přesněji reprezentován a teoreticky by mohla být dosažena vyšší přesnost. Tento prostor lze modifikovat zmíněnými parametry *spread* a $focus$, přičemž parametr $focus$ zaměřuje působení kolem průměru a *spread* ovlivňuje účinnost techniky. Efekt na konkrétních vahách bude znázorněn v sekci 5.5. Samotný proces fine-tuningu probíhá tak, že je zvolen seznam hodnot, na které jsou jednotlivé vrstvy komprimovány, a hledají se takové hodnoty parametrů $focus$ a *spread*, které zvýší přesnost sítě na datasetu.

Po provedení libovolné z kompresí se uloží informace o počtu shluků v dané vrstvě či celém modelu, kvantizačním parametru, původním počtu vah sítě a jejich přesnosti. Tabulka techniky Weight-Sharing není přímo vytvářena, pouze je simulován dopad této techniky. Tyto hodnoty budou dále sloužit k vyhodnocení úspěšnosti komprese.

První metrikou – ukazatelem účinnosti komprese – je kompresní poměr (dále zkráceno na CR podle anglického „Compression Rate“). Tato metrika popisuje, kolikrát je nové řešení zmenšeno oproti původnímu stavu. Počítá se podle následujícího vzorce:

$$CR = \frac{p \cdot bits_v}{p \cdot bits_k + k \cdot (bits_{rv} + bits_k)} \tag{3.2}$$

kde p značí počet vah ve vrstvě, $bits_v$ značí přesnosti původní hodnoty (kolik bitů je třeba pro uložení), $bits_k$ značí, kolik bitů je třeba pro uložení klíče do tabulky hodnot, k značí počet řádků tabulky hodnot a $bits_{rv}$ značí uloženou přesnost v tabulce (kolik bitů je třeba pro uložení sdílené hodnoty). Neformálně řečeno se počítá, kolikrát se nový model včetně své tabulky (jmenovatel zlomku) vejde do stejného prostoru jako původní model (čitatel zlomku).

Pro účely práce bude model s komprimovanými vahami vyhodnocován na testovací sadě datasetu a vznikne tak metrika přesnosti. Pro vizualizaci však bude převedena na ztrátu přesnosti sítě po kompresi pomocí následujícího výrazu:

$$AL = ACC_{orig} - ACC_{current} \tag{3.3}$$

kde AL značí ztrátu přesnosti (z anglického „Accuracy Loss“), ACC_{orig} značí přesnost originální nezměněné sítě a $ACC_{current}$ značí přesnost aktuálně měřeného modelu. Metrika je použita především pro vizualizaci, ve výpočtu se využívá $ACC_{current}$.

Pro vizualizace bude také využito znázornění Pareto dominantních bodů. Bod A dominuje bod B, pokud je A alespoň stejně dobré podle všech kritérií a lepší alespoň v jednom kritériu než je B. Množina bodů, které nejsou dominovány jiným bodem, tvoří Pareto linii. Příkladem může být znázornění Pareto linie na obrázku 5.8. V projektu bude vždy kritériem komprese (čím vyšší, tím lepší) a ztráta přesnosti (čím nižší tím lepší) při znázorňování Pareto linií.

3.2 Optimalizační algoritmy pro techniku Weight-Sharing

V této sekci budou popsány optimalizační algoritmy použité k určení počtu shluků pro každou vrstvu sítě pro techniku Weight-Sharing. Důvod použití optimalizačních algoritmů oproti prohledávání hrubou silou spočívá v množství možných řešení a ve výpočetní náročnosti účelové funkce. Velikost prostoru lze obecně vyjádřit jako rozsah shluků umocněný na počet vrstev. Je-li uvažován jednoduchý příklad, kde pro každou vrstvu sítě Le-Net-5 bude možné volit počet shluků z hodnot $[1, 2, \ldots, 50]$, prohledávací prostor nabízí 50^5 různých možností. Jelikož na sobě hodnoty vzájemně závisí, není možné tento prostor příliš zjednodušit. Pokud bychom uvažovali, že vyhodnocení každého prvku trvá jednu milisekundu,

zabralo by vyhodnocení celého prostoru tři a půl dne, reálně ovšem vyhodnocení jednoho řešení trvá kolem jeden a čtvrt sekundy na procesoru AMD Ryzen 4700U, tedy touto metodou by vyhodnocení trvalo zhruba 12 let.

Prostor je možné redukovat pomocí techniky popsané v [9]. Redukce probíhá tak, že postupně jsou pro vrstvy separátně zkoušeny jednotlivé počty shluků s jinak nemodifikovanou sítí. Pokud přesnost klesne pod určitou mez, tak je konkrétní počet shluků pro konkrétní vrstvu z prohledávaného prostoru vyřazen. Předpokládá se, že již není možné dosáhnout předchozí přesnosti ani s úpravami ostatních vrstev, když nedosahuje požadované přesnosti ani s původními váhami ve zbytku sítě. Výhodou je, že náročnost tohoto výpočtu není exponenciální, ale je určená rozsahem shluků krát počet vrstev, přičemž díky exponenciální složitosti prostoru prohledávání může mít i vyřazení malého počtu možných shluků ve vrstvě velký vliv na velikost prohledávaného prostoru.

Optimalizační algoritmy ve většině případech očekávají účelovou (také fitness) funkci pro hodnocení řešení. Tato funkce popisuje konkrétní řešení, přičemž je možné buď funkci maximalizovat (maximum funkce znamená nejlepší řešení) či minimalizovat (minimum funkce znamená nejlepší řešení). Pro účely Weight-Sharingu proto byla v rámci této práce vytvořena následující účelová funkce, která vychází z metrik popsaných v minulé sekci následujícím způsobem (tuto funkci se algoritmy snaží maximalizovat):

$$Fit = \frac{1}{\sqrt{(1 - \frac{ACC_{current}}{ACC_{target}})^2 + (1 - \frac{CR_{current}}{CR_{target}})^2}} \tag{3.4}$$

kde $ACC_{current}$ a $CR_{current}$ značí přesnost, respektive kompresi aktuálně hodnoceného řešení, ACC_{target} a CR_{target} značí cílový ideální bod v těchto veličinách. Cíl (neboli „target“) je pro optimalizaci nastaven buď pevně, anebo jej lze dynamicky upravovat. Důvod upravování je podložen experimenty v sekci 5.3, kde bylo zjištěno, že je náročné tyto cíle správně pro neuronovou síť odhadnout, přičemž správně nastavené cílové hodnoty jsou klíčové pro prohledávání. Z tohoto důvodu byl navržen dynamický systém úpravy cílových hodnot během výpočtu. Každé řešení bude hodnoceno na základě metriky přesnosti a komprese. Nejprve je na začátku prohledávání zvolena počáteční cílová hodnota jak v ose komprese, tak v ose přesnosti. Dále jsou zvoleny hraniční hodnoty úpravy a rozdíl úpravy. Poté jsou při každém vyhodnocení optimalizačních algoritmů nejprve vyhodnoceny dílčí metriky pro hodnotící funkci. Ty jsou porovnány s cílovou hodnotou. Pokud nějaká kombinace těchto hodnot překoná v obou aspektech hraniční hodnoty a současně alespoň v jednom aspektu cílovou hodnotu, je cílová hodnota v daném aspektu upravena na tuto nově objevenou hodnotu s přidaným rozdílem úpravy (z důvodu zamezení dělení nulou).

Jak bylo v sekci 2.3 naznačeno, budou využity čtyři optimalizační algoritmy: náhodné prohledávání, Genetický algoritmus, Particle Swarm Optimization a Blackhole algorithm. V následující části textu bude popsáno, jak jsou využity pro optimalizaci parametrů komprese pomocí techniky Weight-Sharing, jejíž návrh je zmíněný v předchozí sekci. Platit tedy bude, že účelová funkce je definována rovnicí 3.4 a komprimovaná reprezentace je kódována jako seznam hodnot v předem daném rozmezí se stejnou délkou, jako je počet vrstev zadané neuronové sítě.

Technika náhodného prohledávání je návrhově velmi jednoduchá. Pro náhodné kombinace hodnot n-prvkového seznamu kódujícího kompresi, kde každá pozice popisuje počet shluků pro danou vrstvu, vyhodnotí jejích účelovou funkci. Jejím výstupem bude nalezená pozice s nejlépe ohodnocenou hodnotou účelové funkce.

U genetického algoritmu je návrh následující: za chromozom jedince bude zvolena reprezentace komprese, pro vyhodnocení jedince bude využita účelová funkce. Po vyhodnocení

fitness se přechází k vytváření párů pro křížení. Výběr je realizován pomocí algoritmu rulety, kde jsou náhodně vybraní jedinci s pravděpodobností výběru úměrnému své hodnotě účelové funkce (fitness). Vybere se takový počet párů, aby vytvořily novou populaci o stejné velikosti, jako je ta aktuální. Dostupné je buď jednobodové anebo uniformní křížení. Mutace jsou prováděny tak, že pro každý gen chromozomu se s danou malou pravděpodobností provede jeho změna na některou z možných hodnot. Do nové populace je počítán taky vybraný počet elitních jedinců, tedy jedinců s největší hodnotou fitness z minulé populace, kteří neprocházejí mutací. Výpočet trvá předem stanovený počet generací pro daný počet jedinců v generaci.

Částice algoritmu Particle Swarm Optimization se pohybují v n-dimenzionálním prostoru reprezentujícím počet shluků pro každou vrstvu sítě (kódování komprese). Problémem ovšem je, že tento algoritmus pracuje ve spojitém prostoru. Proto se při výpočtu účelové funkce pozice částic diskretizují. Rychlosti částic jsou také reprezentovány spojitě, ovšem je pro ně zavedeno omezení na maximální rychlost, aby se příliš neroztěkaly po prostoru. Tato úprava se osvědčila při experimentování. Všechny ostatní principy tohoto algoritmu zůstávají zachovány.

Následně byl pro prohledávání vytvořen algoritmus spojující prvky Particle Swarm Optimization a Blackhole algorithm. Základ algoritmu tvoří samotné PSO. Jak již ale bylo řečeno v samotném teoretickém popisu PSO, pokud se všechny částice sejdou v lokálním optimu, PSO nepokračuje v hledání a zůstane v tomto optimu. Oproti tomu algoritmus Blackhole částice opět vrací do prostoru a používá pro hledání, pokud se přiblíží příliš k aktuálně nejlepšímu nalezenému řešení. Toto chování tedy bylo přidáno k jádru PSO následovně: je pevně určen poloměr a maximální rychlost, při které může být částice restartována. Následně, stejně jako v Blackhole algoritmu, pokud se částice nachází ve vzdálenosti od nejlepší částice kratší než je poloměr a současně má nižší rychlost než stanovený práh, je náhodně umístěna do prostoru, její rychlost je náhodně určena a vynuluje se nejlepší nalezená pozice. To motivuje algoritmus více prohledávat prostor. Jelikož je pro potřeby Weight-Sharingu nutno pozice částic diskretizovat, je možné této vlastnosti využít. Namísto výběru částic k restartování porovnáním vzdálenosti s poloměrem je možné určit, zdali nereprezentují stejné řešení. Pokud ano a současně je jejich rychlost pod limitem, jsou restartovány. Rychlostní limit slouží k tomu, aby mohly částice kmitat okolo nejlepšího řešení a prohledávat i blízká sousední řešení.

V následující kapitole bude popsána implementace řešení a způsob spouštění projektu.

Kapitola 4

Implementace

V této kapitole bude stručně popsána implementace realizující návrh zmíněný v předchozí kapitole. Projekt je implementován v jazyce Pyhton (konkrétně vyvíjeno pro Python3.10). Tento jazyk podporuje různá programovací paradigmata, včetně objektově orientovaného, které je v implementaci projektu využito. Nejprve bude popsána implementace neuronových sítí a jejich komprese, poté implementace optimalizační algoritmů. Popisy implementací budou stručné a pro podrobnější informace se doporučuje nahlédnout do programové dokumentace, která je dostupná skrze soubor `site/index.html` (dokumentace generována nástrojem MkDocs[1]). Nakonec bude popsán postup přípravy prostředí a spuštění projektu.

4.1 Neuronové sítě a komprese

Tato sekce nejprve popíše implementaci datasetů, neuronových sítí a následně implementaci jejich komprese. Pro realizaci datasetů a neuronových sítí je využita knihovna PyTorch [29]. Implementace i data datasetů se nacházejí ve složce `data`. Implementace je vytvořena pro dva datasety – MNIST a Imagenette (technicky by tato implementace měla podporovat i kompletní ImageNet). Dataset MNIST je poskytnut samotnou knihovnou PyTorch (jeho stažení a vytvoření rozhraní pro načítání), přičemž nad implementací knihovny je vybudován objekt, který data rozdělí do trénovací, testovací a validační sady pro jednodušší použití (při tvorbě lze zvolit jejich poměry). Jeho implementace je realizována v souboru `data/mnist.py`. Dále jsou pro tento dataset vytvořeny pomocné funkce pro trénování sítě, ověření přesnosti a podobně v souboru `data/utils/mnist_utils.py`.

Stejnou formu má i implementace datasetu Imagenette. Ta ovšem nemá podporu v knihov-ně PyTorch, a proto jsou funkce jejího stahování a rozbalování implementovány zvlášť. Opět je ale dataset dostupný ve formě obalujícího objektu, který poskytuje trénovací, validační a testovací množiny datasetu (opět lze zvolit jejich poměry). Dataset nad daty také automaticky provádí transformace předepsané knihovnou PyTorch pro model MobileNet_v2 a zajistí převáděcí soubor popisující, který výstup sítě reprezentuje jakou ImageNet třídu. Podpůrné funkce pro ImageNet dataset jsou obdobně jako v předchozím případě dostupné v souboru `data/utils/imagenet_utils.py`.

Soubory neuronových sítí se nacházejí v adresáři `models`, přičemž pro každou síť je vytvořena složka s vlastní implementací a uloženými daty náležícími k síti. Implementace samotné neuronové sítě Le-Net-5 (popsaná v sekci 2.2.1) včetně načítání datasetu je in-

[1] dostupné na https://www.mkdocs.org/

spirována a z části převzata z veřejného zdroje[2], neboť se práce nezabývá implementací sítě, ale její kompresí. Implementace zahrnuje dvě varianty sítě – s aktivačními funkcemi hyperbolického tangentu a s aktivačními funkcemi ReLu.

Dále program automaticky zajišťuje natrénování zvolené neuronové sítě a uložení jejího stavu do podadresáře modelu `saves`, případně pokud je požadovaný soubor ukládající stav nalezen, je z něj síť načtena.

Síť MobileNet_v2 je poté poskytnuta samotnou knihovnou PyTorch z důvodů jednodušší replikovatelnosti experimentů a možného porovnání výsledků s ostatními implementacemi. Implementace je ale schopná přijímat jakoukoliv síť implementovanou v knihovně PyTorch. Pokud síť pracuje s daty ze sady ImageNet, je možné jí přímo propojit s Imagenette implementací popsanou výše, a pouze pomocí zadání názvu modelu provádět experimenty se zvoleným modelem.

Samotné nosné soubory techniky Weight-Sharing a implementace optimalizací se nachází v souboru `utils/weight_sharing.py`. Způsob implementace pracuje na bázi kontroléru, kterým je třída třída `WeightShare`. Ta kontroluje objekty třídy `Layer`. Druhá ze zmíněných tříd reprezentuje jednu z vrstev sítě (k ní má přiřazenou referenci), kterou je možné komprimovat za použití následujících metod:

- `share_weight` – váhy přiřazené vrstvy zkomprimuje metodou Weight-Sharing na daný počet zástupců. Pro kompresi jsou dostupné shlukovací algoritmy K-Means, Mini-Batch K-Means a Gaussian Mixture model implementovány knihovnou sklearn [30].
- `set_reset` a `reset`, které umožňují uložení, respektive načtení stavu sítě.
- `compression_rate`, která vypočítá kompresní poměr vrstvy dle rovnice 3.2.

Třída `WeightShare` přijímá celý model, vytváří si pole instancí tříd `Layer` pro všechny potřebné vrstvy a poskytuje následující metody:

- `share` – pro kompresi modelu technikou Weight-Sharing po vrstvách.
- `share_total` – pro kompresi modelu technikou Weight-Sharing přes celý model.
- `compression_rate` – pro výpočet aktuálního kompresního poměru, který reaguje na typ komprese, který byl proveden.
- `set_reset` a `reset` – pro uložení stavu, respektive navrácení na uložený stav.
- `get_optimized_layer_ranges` – pro určení přesností komprese jednotlivých vrstev zvlášť pro optimalizaci prohledávaného prostoru. Jejím parametrem je soubor, ze kterého jsou data načtena. Pokud soubor chybí, nebo v něm chybí potřebná data, jsou dopočítána.
- `finetuned_mod` – implementace techniky fine-tuning popsané v sekci 3.1. Přijímá seznam počtů shluků pro jednotlivé vrstvy, podle kterých je síť komprimována. Poté je snaha najít pro pevné nastavení parametru *spread* takový parametr $focus$, pro který dosáhne síť na testovacím datasetu nejvyšší přesnosti, přičemž výpočet probíhá po vrstvách od vstupní k výstupní vrstvě a iterativně přes parametr $focus$. Opět je možno data načítat a ukládat do souboru jako v předchozí funkci.

[2]Zdroj je zmíněn ve zdrojovém kódu – https://github.com/erykml/medium_articles/blob/master/Computer%20Vision/lenet5_pytorch.ipynb

Podporován je jak běh na CPU, tak akcelerovaný běh na GPU, ovšem shlukování probíhá vždy na CPU a výsledek je poté přesunut do zařízení, kde se daná síť nachází. Implementace v tomto ohledu přizpůsobí svůj výpočet podle toho, na kterém zařízení se neuronová síť používá.

Dalším prvkem techniky je dodatečná kvantizace, jejíž implementace se nachází v adresáři `utils/float_prec_reducer`. Zde je implementován objekt, který přijme pole hodnot a převede je na požadovaný typ, přičemž možné jsou `float32` (bez úprav), `float16` a `float8`. Na instanci této třídy má referenci objekt `WeightShare`.

4.2 Optimalizační algoritmy

Implementace optimalizačních algoritmů jsou vytvořeny velice podobně jako implementace již zmíněné komprese. Každý optimalizační algoritmus tvoří své jedince (ať už chromozomy, částice, hvězdy nebo náhodné výběry) jako objekty, které jsou následně manipulovány řídícím objektem realizujícím daný typ optimalizačního algoritmu. Podrobněji jsou implementace optimalizačních algoritmů popsány níže:

- Random search – implementace se nachází v souboru `utils/rnd/rnd.py`. Pro kompatibilitu s ostatními algoritmy byla i tato optimalizace implementována stylem kontroléru s využitím objektu `Individual`, který představuje aktuálně zkoumaný prvek v prostoru, a `RandomController`, který poskytuje funkcionalitu prohledávání.
- Genetic algorithm – implementován v adresáři `utils/genetic`. Prohledávání je poté implementováno v souboru `genetic.py`, kde kontrolérem je `GeneticController` a jedince zastupuje třída `Individual`. Dále složka obsahuje soubor `genetic_config.py`, který popisuje algoritmy párování, křížení, mutací a elitismu. Obsahuje i různé implementace těchto úkonů a je možné je zde přenastavit.
- Particle Swarm Optimization – implementován v souboru `utils/pso/pso.py`. Obdobně jako v předchozích případech je kontrolérem `PSOController` a částice zastupuje třída `Particle`, která obsahuje metody pro pohyb částic a ukládá k tomu potřebné informace. Jak bylo nastíněno dříve, částice se pohybují ve spojitém prostoru a následně se provádí konverze do diskrétního prostoru vyžadovaného technikou Weight-Sharing.
- Particle Swarm Optimization s Blackhole úpravou – její implementace je pouze rozšířením předchozí implementace dodáním logiky ohledně poloměru a restartováním částice na náhodnou pozici v prostoru.

Všechny kontroléry si při inicializaci vytvoří potřebnou populaci zkoumaných hodnot a metoda `run` následně realizuje dané prohledávání. Dále jsou také všechny algoritmy vytvářeny tak, aby byly schopné běhu v nestabilním prostředí, kde může kdykoliv dojít k pádu či odpojení od výpočetních prostředků (například prostředí Google Colab, nebo při vypršení času na výpočetním clusteru). Tedy po každém stanoveném počtu iterací je vytvořen soubor, kde je uložený průběh optimalizace (realizováno pomocí knihovny Pandas [36]). Z tohoto souboru je poté možné optimalizaci inicializovat a pokračovat tak ve výpočtu. Současně jsou tato data využita k vytváření vizualizací průběhu optimalizace.

Pro testování algoritmů je možné spouštět přímo jmenované soubory, kde se provádí demonstrace optimalizací jednoduché úlohy pro vyzkoušení funkcionality implementace.

Důležitým prvkem pro optimalizační algoritmy je fitness funkce. Pro již zmíněné dynamické vlastnosti je vytvořena implementace v souboru `utils/fitness_controller.py`.

Ta realizuje pouze dynamickou úpravu prvků výpočtu fitness funkce, samotný výpočet hodnoty účelové funkce je nutno definovat zvlášť. Implementace také podporuje načítání z uložených dat. Funkčně probíhá výpočet fitness tak, že se nejprve určí dílčí hodnoty, tedy přesnost a komprese dané sítě podle daného jedince. Následně se tyto hodnoty jako dvojice porovnají s hraničními hodnotami. Pokud obě dvě hodnoty nepřesáhnou danou hranici, nejsou dále brány v potaz. Tato bariéra je zde převážně pro odfiltrování vyloženě špatných řešení (například maximální komprese, ale nulová přesnost). Jestliže dvojice takto projdou filtrací, jsou jejich hodnoty jednotlivě porovnávány s cílem. Když je cílová hodnota v nějakém aspektu překonána, je za cíl určena nová hodnota s přidanou konstantou. Následně je po tomto zpracování vypočítána fitness hodnota pro všechny jedince z populace podle stejného cíle.

4.3 Spouštění a běh

Tato sekce popisuje postup vytvoření prostředí pro běh projektu, spouštění, prohlížení výsledků, a také nastiňuje průběh výpočtu. Jak již bylo zmíněno, systém je vytvořený v jazyce Python, je tedy třeba mít zajištěný interpret, nejlépe pro Python3.10. Potřebné knihovny jsou shrnuty v souboru `requirements.txt` a pomocí příkazu:

```
pip install -r requirements.txt
```

je prostředí stáhne a nainstaluje. Alternativně je možné pro běh vytvořit virtuální prostředí například pomocí systému Anaconda[3], ve kterém se vytvoří nové virtuální prostředí pomocí:

```
conda create -n [název] python=3.10
```

Poté se prostředí aktivuje přes příkaz:

```
conda activate [název]
```

a opět pomocí `pip install -r requirements.txt` se v tomto virtuálním prostředí nainstalují všechny potřebné knihovny (pro běh Python notebooků je třeba přidat ještě balík `Jupyter`). Pro deaktvaci prostředí je třeba použít

```
conda deactivate.
```

Před popisem spouštění však bude nejprve stručně vysvětleno, jak výpočet probíhá, a které prvky je třeba nastavit při spouštění programu. Soubory využívající CLI způsob spouštění výpočtu jsou implementovány v adresáři `compress_optim`. Pro konfiguraci běhu je především důležitý soubor `compressor_config.py`, který obsahuje nastavení optimalizačních algoritmů, cíle komprese, optimalizace prostoru, typ sítě a podobně. Konfigurace se ukládá a načítá ze souborů typu `.yaml`. Očekávaná práce s touto konfigurací spočívá v nastavení požadovaných hodnot pro daný experiment a uložení této konfigurace společně s daty experimentu. Poté je možné konfiguraci využít pro opakování měření či vyhodnocení výsledků.

Další důležité nastavení je třeba provést specificky pro genetický algoritmus v souboru `utils/genetic/genetic_config.py`. Toto nastavení je však uloženo ve stejném konfiguračním souboru jako v předchozím případně a je z něj také načítáno.

[3] Dostupné na https://www.anaconda.com/

Výpočet tedy nejprve načte tuto konfiguraci. Poté načte dataset (pokud není nalezen lokálně, je stažen a rozbalen do potřebné struktury), načte nebo natrénuje příslušnou neuronovou síť (v případě sítí pracujících s Imagenetem jsou natrénované sítě staženy pomocí knihovny PyTorch), a vytvoří potřebné relace mezi sítí a implementací techniky Weight-Sharing. Následně, pokud je zvolen přístup komprese přes celý model, je tato komprese provedena a výpočet končí (výsledky jsou uloženy). Jinak je provedena optimalizace prohledávaného prostoru, buď výpočtem anebo opět načtením z příslušného souboru (optimalizaci je možné zakázat v konfiguračním souboru). Poté je vytvořen objekt pro ovládání cíle fitness `FitnessController` a spustí se příslušná optimalizace.

Pro jednotlivé běhy jsou dostupné programy v souborech `lenet_compression.py` pro pokus se sítí Le-Net-5 a `net_compression.py` pro pokus na síti pracující s datasetem ImageNet/Imagenette. Oba dva programy obsahují následující parametry (lze také vypsat pomocí parametru `-h` nebo `-help`.):

- `-comp [alg]`, `-compressor [alg]` – výběr komprese / optimalizačního algoritmu z následujících možností:
  - `random` – náhodné prohledávání po vrstvách
  - `genetic` – genetické prohledávání po vrstvách
  - `pso` – PSO prohledávání po vrstvách
  - `blackhole` – PSO s BH úpravou prohledávání po vrstvách
  - `total` – prohledávání komprese přes celý model
- `-pop [N]`, `-num_population [N]` – velikost populace pro prohledávání
- `-its [N]`, `-num_iterations [N]` – počet iterací prohledávání
- `-up [N]`, `-upper_range [N]` – horní hranice počtu shluků pro každou vrstvu
- `-lo [N]`, `-lower_range [N]` – spodní hranice počtu shluků pro každou vrstvu
- `-hp`, `-hide` – nezobrazí graf prohledávání na konci výpočtu (příklad grafu na obrázku 5.8)
- `-sv`, `-save` – uloží graf prohledávání
- `-cfs [FILE]`, `-config_save [FILE]` – uloží konfiguraci
- `-cfl [FILE]`, `-config_load [FILE]` – načte konfiguraci
- `-sf [FOLDER]`, `-save_folder [FOLDER]` – zvolí složku, kam bude ukládán průběh výpočtu, a kde se bude nacházet výsledek

Pro spouštění více běhů je dostupný Bash skript `run_test.sh`. Na začátku tohoto souboru se specifikují typy optimalizací, složky experimentů a rozsahy prostoru. Skript poté pro všechny složky načte jejich dané konfigurace. Očekává se, že jsou ve složce. Pokud není konfigurace dostupná, vytvoří se s aktuálním nastavením.

Posledním způsobem jsou přiložené Python notebooky. Ty jsou však implementovány pouze pro experimenty s Le-Net-5 ve složce `notebooks/lenet_compression/lenet_[typ komprese]_compression.ipynb`. Veškeré nastavení se provádí přímo v noteboocích.

Kapitola 5

Vyhodnocení experimentů

V této kapitole budou v samostatných sekcích popsány a vyhodnoceny experimenty provedené nad implementací techniky Weight-Sharing nad sítí Le-Net-5 [31], pracující s datasetem MNIST [7] (síť popsána v sekci 2.2.1). Síť Le-Net-5 má v experimentech dva zástupce, jednoho s aktivačními funkcemi hyperbolického tangentu a druhého s aktivačními funkcemi Rectified Linear unit. Obě varianty byly natrénovány se ztrátovou funkcí křížové entropie a optimalizátorem Adam. Trénováno bylo na 100 epochách, přičemž byly vybrány takové váhy, které během trénování dosáhly nejnižší hodnoty ztrátové funkce na validačních datech. Výsledné sítě jsou znázorněny v tabulce 5.1. Více druhů této sítě bylo užito proto, že většina moderních sítí tuto aktivační funkci také používá. Proto byly pokusy provedeny i s touto sítí. Cílem kompresí bude získat co nejkomprimovanější síť, která bude mít maximálně jedno procento ztráty přesnosti.

Druh sítě	Základní přesnost [%]	Cílená minimální přesnost [%]
Le-Net Tanh	98,64	97,64
Le-Net ReLu	98,46	97,46

Tabulka 5.1: Sítě Le-Net pro pokusy

Pro pokusy se sítěmi většího rozsahu jsou zvoleny sítě, které jsou volně dostupné pomocí knihovny PyTorch[1] [29], kvůli své snadné reprodukovatelnosti výsledků a možnosti porovnání s ostatními výzkumy. Zvolenou sítí je MobileNet_v2 popsaná v sekci 2.2.2. Její výsledky v neupravené verzi na datasetu Imagenette a Imagewang jsou znázorněny v tabulce 5.2.

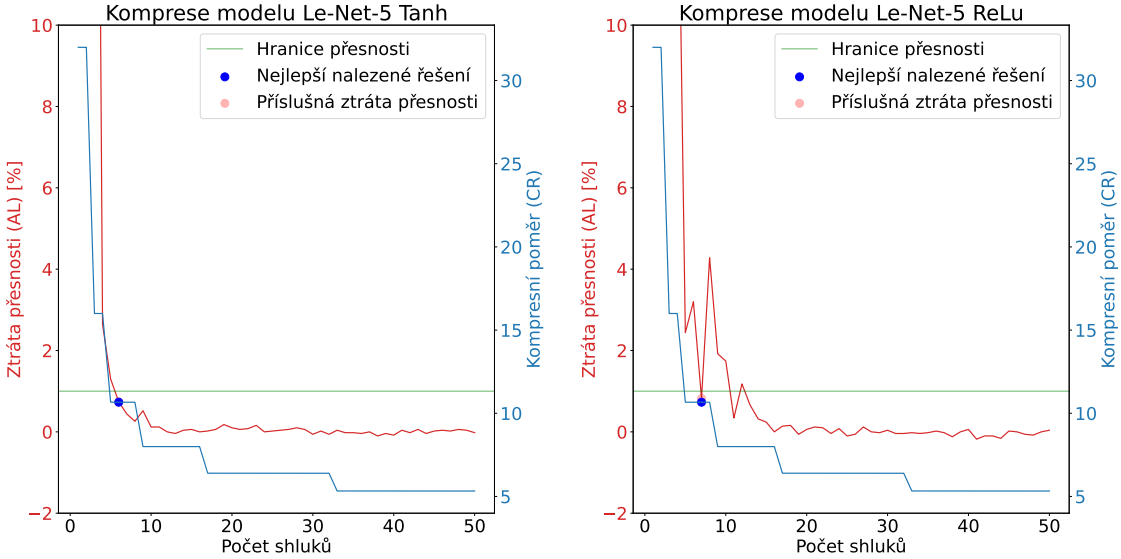
Dataset	Subset	Top-1 ACC [%]	Top-5 ACC [%]
Imagenette	Trénovací	83,09	96,57
	Validační	82,32	96,17
	Testovací	82,89	96,47
Imagewang	Trénovací	82,63	96,40
	Validační	79,54	94,99
	Testovací	80,11	95,74

Tabulka 5.2: Přesnosti sítě MobileNet_v2 na různých subdatasetech ImageNetu dostupných na [14].

[1] Přehled modelů je dostupný na https://pytorch.org/vision/stable/models.html

5.1 Optimalizace technikou Weight-Sharing přes celý model

První experiment prověří účinnost komprese pomocí techniky Weight-Sharing přes celý model pro porovnání účinnosti přístupu po vrstvách se shlukovacím algoritmem K-Means. Jelikož je celá komprese jednoduše popsatelná jedním parametrem – počtem shluků pro celou síť – není nutné zapojovat do výpočtu žádný optimalizační algoritmus. Současně, jelikož v následujících experimentech bude optimalizace prováděna v prostoru, kde na každou vrstvu připadne počet shluků z intervalu $[1, 2, \ldots 50]$, bude i zde pro kompresi přes celý model zvolen tento prostor. Algoritmus pro prohledání tohoto prostoru bude velmi jednoduchý – postupně pro každý prvek z daného intervalu provede kompresi sítě a zapíše výsledné hodnoty. Při tomto experimentu nebude využita metrika fitness, ale pouze její dílčí parametry. Výsledky tohoto algoritmu jsou znázorněny v grafech na obrázku 5.1 a v tabulce 5.3.

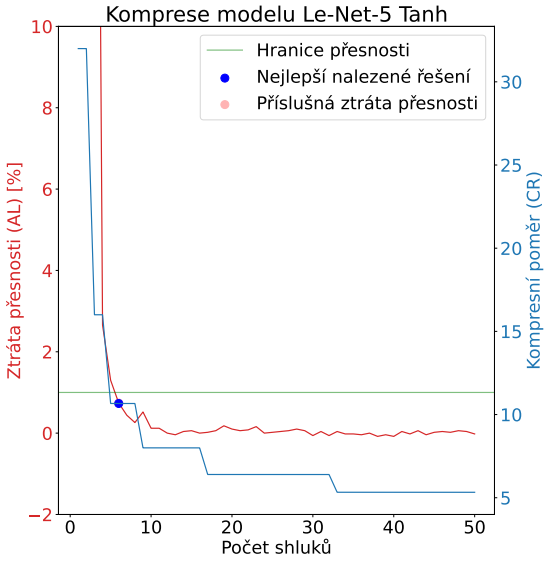


(a) Výsledky se sítí Le-Net-5 Tanh

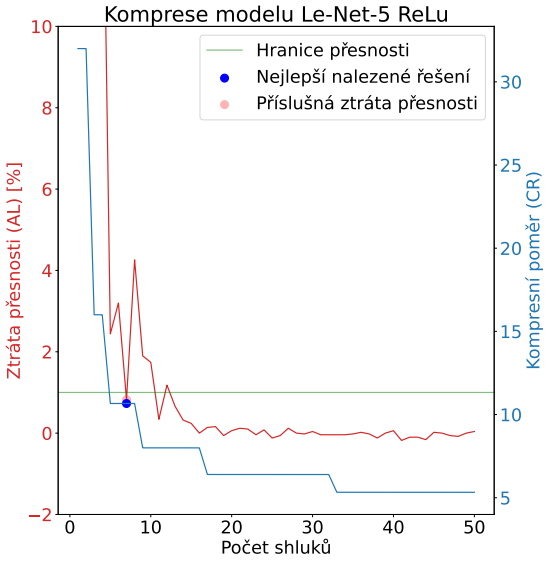
(b) Výsledky se sítí Le-Net-5 ReLu

Obrázek 5.1: Porovnání výsledků kompresí (originální přesnost `float32`)

Z výsledků tedy vyplývá, že tímto způsobem lze dosáhnout nejvýše kompresního poměru 10.66 s přijatelnou ztrátou přesnosti. Možnost hlubší komprese nabízí pouze dodatečná kvantizace na přesnosti `float16` a `float8`. Proto bylo provedeno i měření s tímto nastavením a výsledky těchto pokusů jsou níže na obrázcích 5.2, 5.3 a tabulce 5.3.

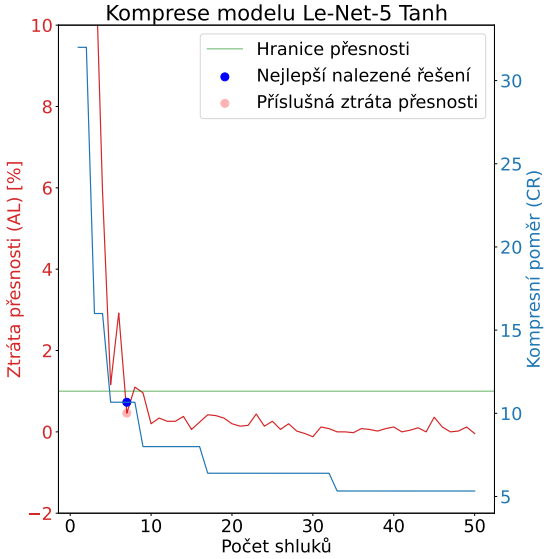


(a) Výsledky se sítí Le-Net-5 Tanh

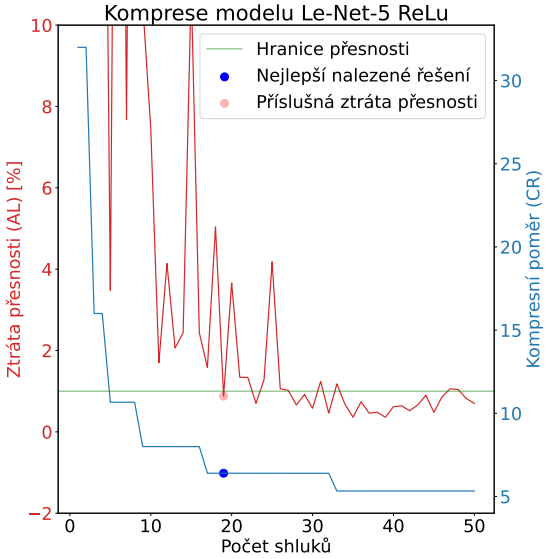


(b) Výsledky se sítí Le-Net-5 ReLu

Obrázek 5.2: Porovnání výsledků kompresí se snížením přesnosti na `float16`



(a) Výsledky se sítí Le-Net-5 Tanh



(b) Výsledky se sítí Le-Net-5 ReLu

Obrázek 5.3: Porovnání výsledků kompresí se snížením přesnosti na `float8`

Druh sítě	Kvantizace	Počet shluků	CR	ACC [%]	AL [%]
Tanh	`float32`	6	10,6642	97,90	0,74
		7	10,6638	98,20	0,44
		8	10,6634	98,38	0,26
		9	7,9977	98,12	0,52
	`float16`	6	10.6649	97,90	0,74
		7	10.6646	98,20	0,44
		8	10.6643	98,38	0,26
		9	7.9982	98,12	0,52
	`float8`	7	10,6650	98,18	0,46
		9	7,9985	97,68	0,96
		10	7,9983	98,44	0,20
		11	7,9982	98,30	0,34
ReLu	`float32`	7	10,6638	97,64	0,82
		11	7,9971	98,12	0,34
		13	7,9966	97,80	0,66
		14	7,9964	98,14	0,32
	`float16`	7	10,6646	97,64	0,82
		11	7,9979	98,12	0,34
		13	7,9975	97,80	0,66
		14	7,9972	98,14	0,32
	`float8`	19	6,3976	97,58	0,88
		23	6,3971	97,76	0,70
		28	6,3965	97,80	0,66
		29	6,3963	97,54	0,92

Tabulka 5.3: Nejlepší výsledky se ztrátou přesnosti pod 1% s různými kvantizacemi

Z výsledků je patrné, že dodatečná kvantizace nepomohla ke znatelně větší kompresi. Globální komprese sice umožňuje uložení pouze jedné tabulky, což je oproti metodě komprese po vrstvách výhodou, ovšem technika Weight-Sharingu více těží z možnosti uložit klíče na co nejmenší počet bitů. Současně je ale při globální kompresi nutné uložit nějaký minimální počet hodnot přes celou síť pro zachování přesnosti. Z tohoto důvodu se zdá komprese přes vrstvy přínosnější, protože umožní uložit v některé z vrstev méně hodnot a snížit tak bitovou zátěž klíčů ve struktuře, což by mělo vést k větší kompresi. Toto tvrzení bude ověřeno v dalším experimentu.

Konečným výsledkem tohoto experimentu tedy je maximální kompresní poměr zhruba 10,66. Zajímavým poznatkem také je, že komprimovat síť Le-Net s aktivačními funkcemi ReLu je obtížnější a celkově je dosaženo nižších kompresních poměrů oproti druhé síti. I z grafů je patrné, že síť je daleko citlivější na zásahy ve vahách a přesnosti jsou daleko rozkmitanější než u druhé sítě.

5.2 Optimalizační algoritmy nad technikou Weight-Sharing

V tomto experimentu budou vyhodnoceny úspěšnosti jednotlivých optimalizačních algoritmů využitých pro optimalizaci techniky Weight-Sharing pracující po vrstvách se shlukovacím algoritmem K-Means. Jednotlivé algoritmy budou porovnány při využití stejného počtu vyhodnocení účelových funkcí s cílem určit jejich úspěšnost a použitelnost při řešení tohoto problému. Testovány budou algoritmy zmíněné v kapitole 3, tedy Random search, Genetický algoritmus, Particle Swarm Optimization a Particle Swarm Optimization obohacený o prvky z Blackhole algoritmu.

Komprese bude prováděna pro všech pět vrstev obsahujících váhy, kde každá vrstva bude shlukována na $[1, 2, \ldots, 50]$ hodnot zvlášť. Tento prostor ovšem bude redukován pomocí zmíněného postupu popsaného v [9], který změří přesnost každého počtu shluků jednotlivě pro všechny vrstvy. Jelikož se v tomto experimentu nebude síť během komprese přetrénovávat, platí, že pokud některá vrstva při kompresi na jistý počet shluků způsobí velký pád přesnosti sítě i bez komprese ostatních vrstev, jedná se o počet shluků, který není třeba dále zkoumat, lze jej rovnou ze zkoumaného prostoru vyřadit. Výsledky jsou znázorněny na grafech na obrázku 5.5 – provedeno rovnou pro více reprezentací, které budou cílem zkoumání v dalších experimentech. Limitem pro připuštění hodnoty do dalšího prohledávání je 97% přesnost sítě po kompresi jedné dané vrstvy. Je nutné podotknout, že tento výpočet je proveden pouze jednou, jeho výsledek uložen a použit pro všechny běhy jednotlivých algoritmů, tedy všechny pracují nad stejným prostorem. Podobně byl zafixován algoritmus K-Means, kterému byl nastaven pevný počáteční stav.

Účelová (fitness) funkce, která je popsána v kapitole 3, bude nastavena na pevný cílový bod popsaný v tabulce 5.4. Pevným bodem je myšleno, že se v průběhu optimalizace bod nepohybuje, jak je popsáno v sekci 3.1.

Druh sítě	Cílová přesnost [%]	Cílová komprese
Le-Net Tanh	99	14
Le-Net ReLu	99	18

Tabulka 5.4: Sítě Le-Net pro pokusy

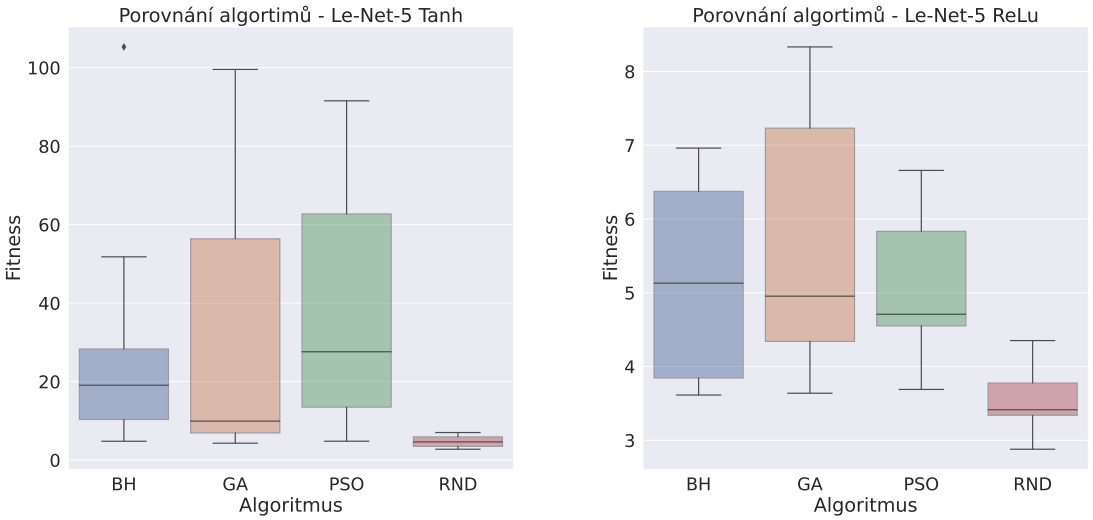
Každý algoritmus je cílen, aby během svého běhu učinil 400 evaluací účelové funkce, přičemž komprese probíhá bez přeučení v pořadí od vstupu k výstupu. Konkrétní nastavení jednotlivých algoritmů jsou následující:

1. Náhodné vyhledávání (dále značeno jako RND) – 400 náhodných vzorků z prostoru.

2. Genetický algoritmus (dále značeno jako GA) – 36 generací s 12 potomky, přičemž se počítá s jedním elitním jedincem (tedy genetický algoritmus využije pouze 397 vyhodnocení účelové funkce). Využit je výběr rodičů pomocí rulety, po kterém následuje jednobodové křížení a mutace s pravděpodobností 20% na každém genu.

3. Particle Swarm Optimization (dále značeno jako PSO) – maximální rychlost v každé dimenzi je 4, hybnost byla zvolena 0.8, využito 20 částic ve 20 časových cyklech.

4. PSO modifikované s Blackhole (dále značeno jako BH) – podobné nastavení jako v případě PSO, částice jsou resetovány, pokud reprezentují stejný bod, jako nejlepší řešení a mají rychlost nižší než 25% maximální rychlosti částice.



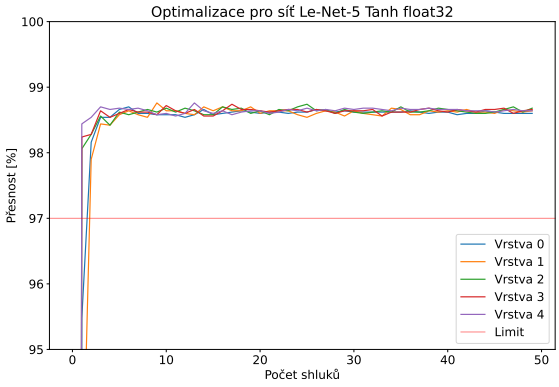
(a) Výsledky se sítí Le-Net-5 Tanh

(b) Výsledky se sítí Le-Net-5 ReLu

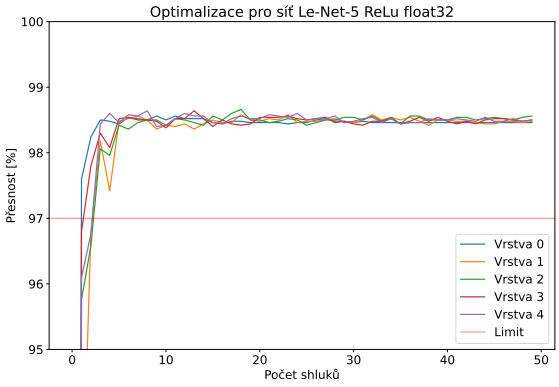
Obrázek 5.4: Porovnání nejlepších výsledků optimalizací z 11 běhů

Pro všechny algoritmy bylo s tímto nastavením provedeno jedenáct běhů.

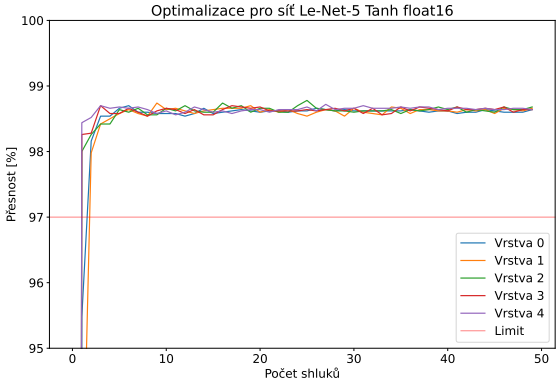
Provedené optimalizace prostoru jsou znázorněny v grafech na obrázku 5.5. I přesto, že podle grafu na první pohled vypadá, že k velké úspoře nedošlo, je třeba si uvědomit, že i odstranění jedné hodnoty počtu shluků pro některou z vrstev vede k poměrně velké redukci prostoru. Kupříkladu v případě optimalizace pro síť Le-Net-5 Tanh s přesností `float32` (z obrázku 5.5a) se prostor z 50^5, což je $3,125 \cdot 10^8$, redukuje na $48 \cdot 48 \cdot 49 \cdot 49 \cdot 49$, což je $2,71063296 \cdot 10^8$, tedy redukce prostoru zhruba o 13,3%.



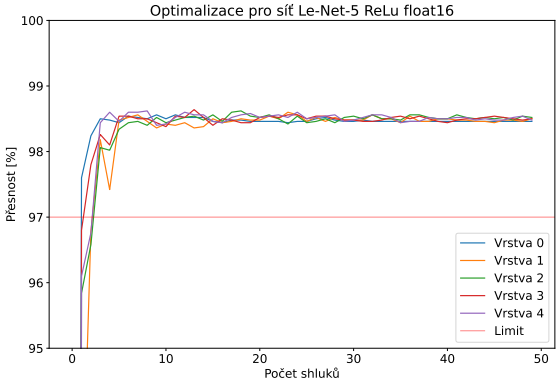
(a) Optimalizace Le-Net-5 Tanh `float32`



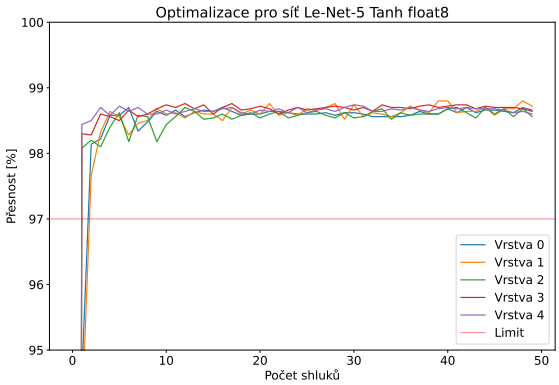
(b) Optimalizace Le-Net-5 ReLu `float32`



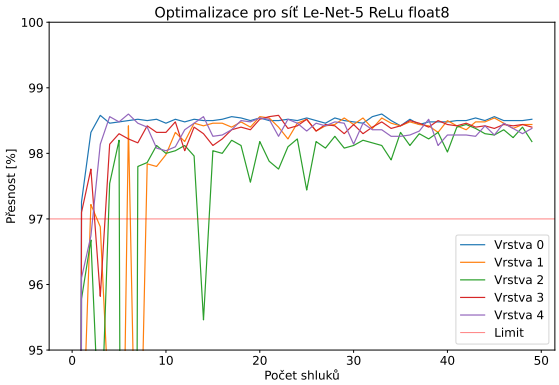
(c) Optimalizace Le-Net-5 Tanh `float16`



(d) Optimalizace Le-Net-5 ReLu `float16`

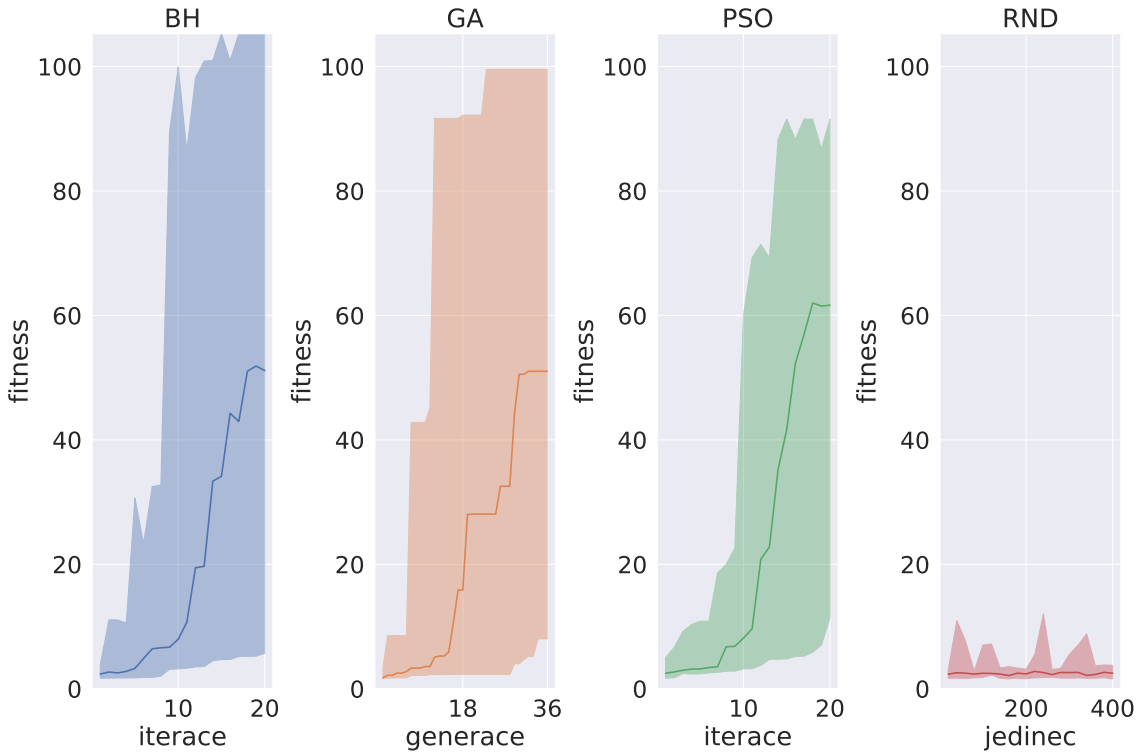


(e) Optimalizace Le-Net-5 Tanh `float8`

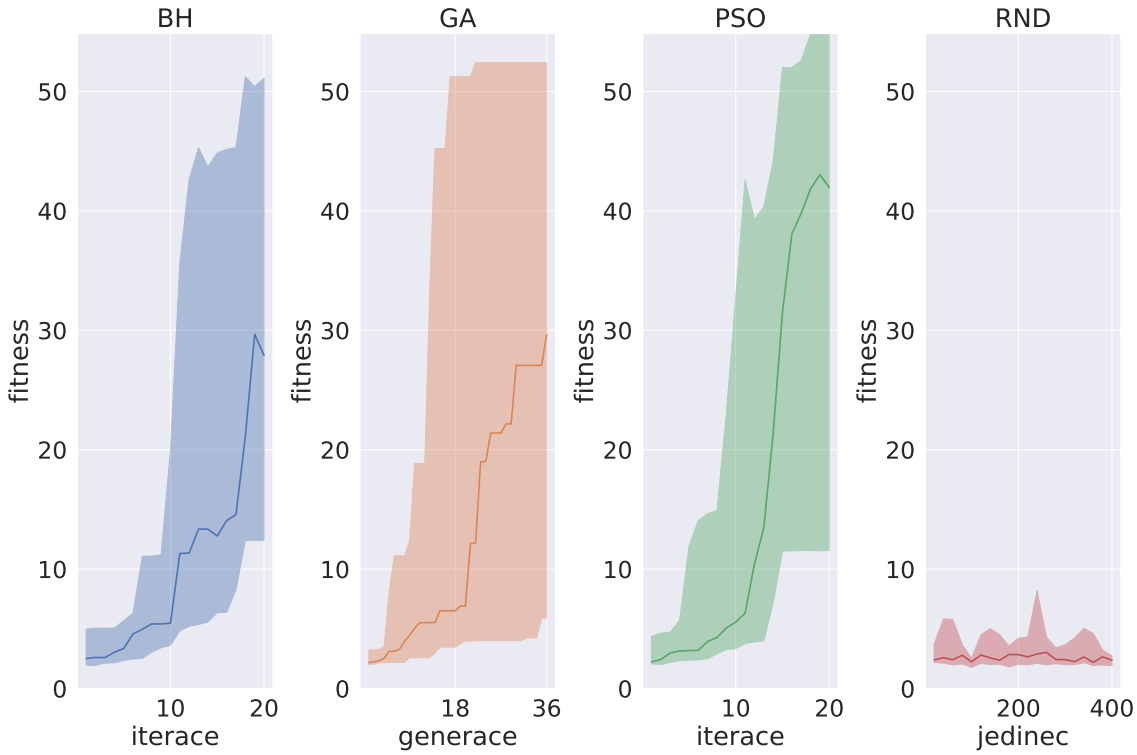


(f) Optimalizace Le-Net-5 ReLu `float8`

Obrázek 5.5: Znázornění provedených optimalizací pro jednotlivé typy sítí a jednotlivé přesnosti.



(a) Výsledky se sítí Le-Net-5 Tanh

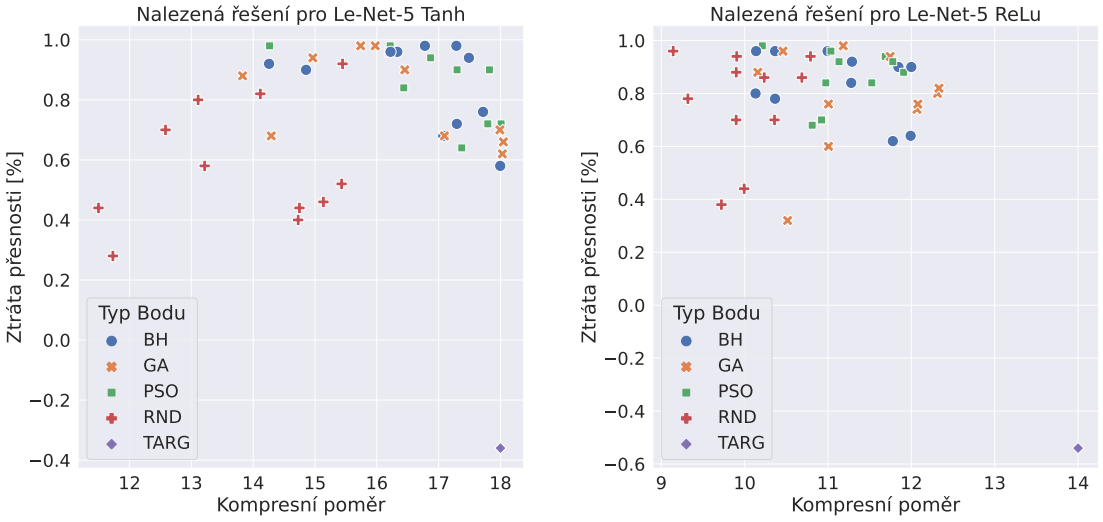


(b) Výsledky se sítí Le-Net-5 ReLu

Obrázek 5.6: Průběh vývoje fitness pro zkoumané optimalizační algoritmy z 11 běhů. Výsledky algoritmu náhodného prohledávání byly pro jednoduší vyobrazení seskupeny po dvaceti tak, aby korespondovaly s ostatními. Medián je znázorněn plnou čárou

Typ sítě Le-Net-5	Tanh			ReLu		
Algoritmus	přesnost [%]	AL%	CR	přesnost [%]	AL%	CR
RND	97,72	0,92	15,4455	97,52	0,94	10,7899
	98,12	0,52	15,4301	97,60	0,64	10,6862
GA	98,02	0,62	18,0307	97,64	0,82	12,3310
	97,98	0,66	18,0472	97,66	0,80	12,3134
PSO	97,92	0,72	18,0101	97,58	0,88	11,9071
	97,74	0,90	17,8216	97,54	0,92	11,7772
BH	98,06	0,58	17,9959	97,56	0,90	11,9992
	97,88	0,76	17,9959	97,82	0,64	11,9913

Tabulka 5.5: Nejlepší dosažené výsledky se sítí Le-Net-5



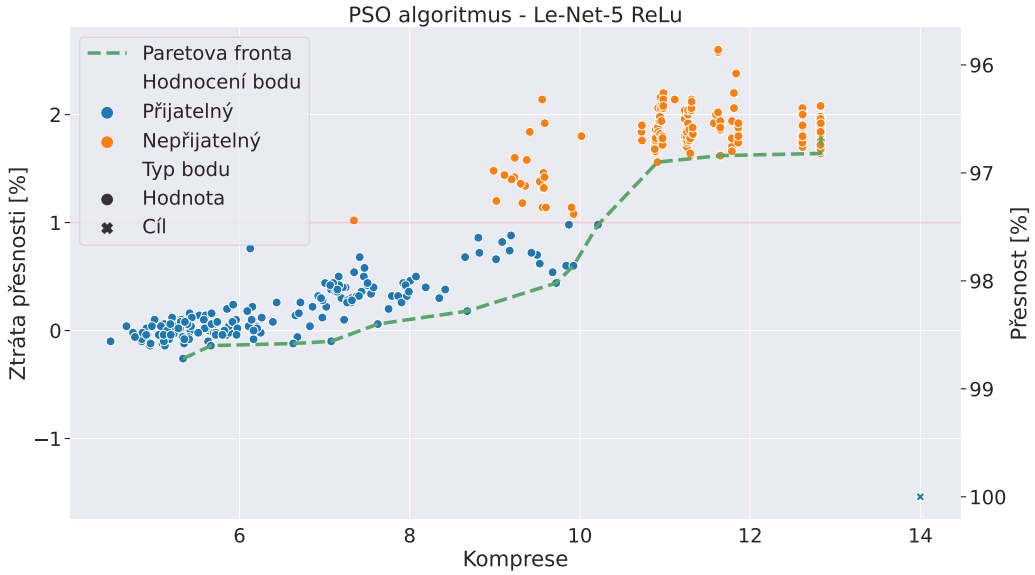
(a) Výsledky se sítí Le-Net-5 Tanh

(b) Výsledky se sítí Le-Net-5 ReLu

Obrázek 5.7: Znázornění nejlepších objevených řešení v každém běhu - TARG označuje cílový bod účelové funkce

Jak je z přiložených grafů 5.4 a 5.6 zřejmé, všechny tři pokročilejší optimalizační algoritmy dosahují výrazně vyšší úspěšnosti než náhodné prohledávání. Dále lze potvrdit poznatek z minulého experimentu, že neuronovou síť s aktivační funkcí hyperbolický tangens lze komprimovat daleko lépe než síť s ReLu. Důležitým poznatkem také ovšem je, že algoritmy PSO a BH bylo daleko náročnější nastavit tak, aby dosáhly uspokojivých výsledků v porovnání s genetickým algoritmem. Každopádně v následujících experimentech již bude algoritmus RND vypuštěn.

Pokud je upuštěno od hodnot účelové funkce a jsou prozkoumána data samotná, je patrné, že pro síť Le-Net Tanh byl cíl nastaven špatně, neboť byla přesažena cílová hodnota (zeleně označené hodnoty v tabulce 5.5). Jelikož účelová funkce hodnotí řešení jako vzdálenost k cílovému bodu, pak řešení lepší než ta nalezená by byla ohodnocena hůře, protože by se nacházela dále od tohoto cílového bodu, což není optimální chování. Lze uvést jednoduchý příklad: Pokud by bylo nalezeno řešení se ztrátou přesnosti 18 a řešení se stejnou



Obrázek 5.8: Příklad prozkoumaných bodů v jednom běhu optimalizace PSO se sítí Le-Net-5 ReLu

přesností, ovšem kompresí 20, účelová funkce by lépe ohodnotila první možnost, protože má nižší vzdálenost k cíli. Řešením by samozřejmě bylo zvolit lépe cílový bod, ovšem to je velmi náročné odhadnout, neboť je k tomu zapotřebí hluboké pochopení a prozkoumání vlastností sítě. Touto problematikou se bude zabývat následující experiment.

Při ohlédnutí zpět k předchozímu experimentu je možné ověřit tvrzení, že optimalizace po vrstvách skutečně vede k většímu kompresnímu poměru a současnému zachování přesnosti, narozdíl od komprese přes celý model. Komprimace po vrstvách prvního řešení algoritmem GA z tabulky 5.5 sítě Le-Net Tanh bez dodatečné kvantizace byla provedena pomocí hodnot $[5, 6, 7, 2, 2]$. Pokud pomineme tabulky s hodnotami, jsou třeba k uložení klíčů v jednotlivých vrstvách počty bitů $[3, 3, 3, 1, 1]$, zatímco nejlepší komprese přes celý model bylo dosaženo počtem shluků 6, což znamená, že všechny klíče potřebují 3 bity na uložení. Jelikož je v modelu počet těchto klíčů k uložení v řádech desetitisíců, je výhodnější uložit více tabulek a získat tak nižší bitovou zátěž klíčů, než se snažit šetřit na tabulce a rozšiřovat klíče. Z tohoto důvodu je technika Weight-Sharing efektivnější po vrstvách.

Zajímavostí je, že kompresí lze dosáhnout i vyšší přesnosti, než kterou měla originální síť. Tento fakt je znázorněný v grafu 5.8. Nastavení komprese s nejvyšší přesností dosahovalo v tomto konkrétním případě přesnosti 98.72% (což znamená zlepšení přesnosti o 0.26%) s kompresním poměrem 5.3366. Teoreticky by se tak dala technika použít i pro lehké doučování sítě, pokud by byla účelová funkce zaměřena pouze na přesnost. Je však možné, že původní síť mohla být lepe natrénována a pak by tato technika v této úloze neobstála.

5.3 Dynamické cíle optimalizace

Jak již bylo naznačeno v minulém experimentu v sekci 5.2, tento experiment se bude zabývat určením správného cíle optimalizace pro účelovou funkci. Jednou z možností je vlastnosti sítě nastudovat a díky tomu určit správný cíl. Tato možnost by ovšem nejspíše vyžadovala několik evaluací a nezanedbatelné množství dalších výpočtů. Z tohoto důvodu byl navržen dynamický cíl účelové funkce, jehož návrh je popsaný v 3.2 a implementace v 4.2. Vyzkoušen bude ve dvou režimech:

1. Přednastavený dynamický – na začátku bude cíl nastavený stejně jako v předchozím experimentu podle tabulky 5.4, pouze nebude statický.

2. Extrémní dynamický – na začátku bude cíl nastavený na hodnoty přesnosti 1% a kompresního poměru 1. Představuje tak extrémní nastavení dynamického cíle, přičemž se zkoumá, zdali se dokáže takto nastavený systém sám pomocí dynamičnosti dostat k podobným či lepším výsledkům než v předchozím případě.

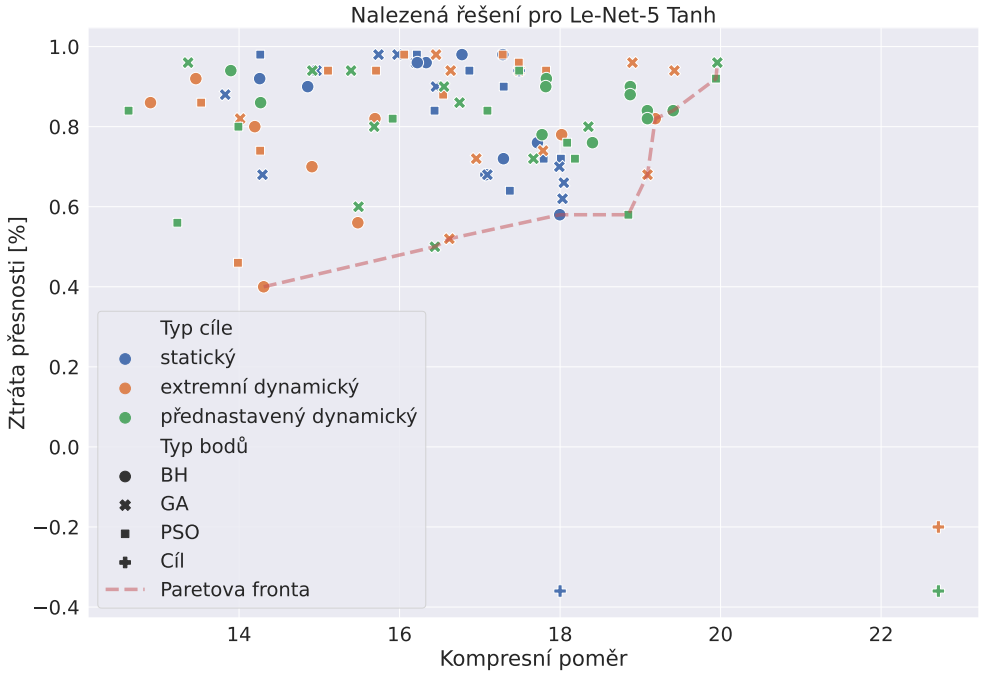
Dále mají tyto dva zkoumané přístupy společné nastavení hraničních hodnot a rozdíly úpravy podle následující tabulky:

Druh parametru	Přesnost [%]	Komprese
Hraniční hodnoty	97	1
Rozdíl úpravy	0,1	0,1

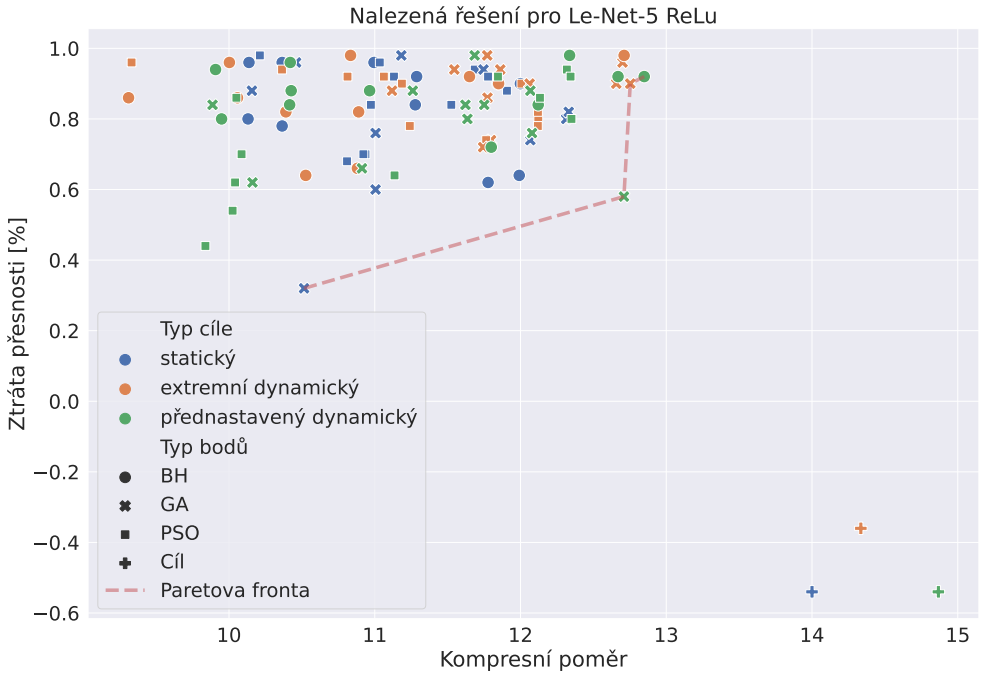
Tabulka 5.6: Společné nastavení parametrů pro dynamický cíl

Tyto hodnoty byly vybrány následujícím způsobem: Hraniční hodnoty jsou zvolené podle základních vlastností neupravené sítě. Kompresní poměr neupravené sítě je 1, přesnost je lehce snížená základní přesnost sítě. Rozdíly úprav jsou v obou metrikách zvoleny tak, aby docházelo k úpravám v rozumných krocích.

Nastavení optimalizačních algoritmů zůstává stejné jako v experimentu 5.2, pouze je vynechaný Random Search optimalizační algoritmus, který podle účelové funkce nepracuje a nevyužívá její hodnoty. V tomto experimentu je opět zvolený shlukovací algoritmus K-Means.



(a) Výsledky se sítí Le-Net-5 Tanh



(b) Výsledky se sítí Le-Net-5 ReLu

Obrázek 5.9: Porovnání průběhů optimalizací s různými nastaveními cílů

Druh sítě	Druh cíle	Opt. alg.	CR	Přesnost [%]	AL [%]
Tanh	statický	GA	18,0471	97,98	0,66
		GA	18,0307	98,02	0,62
		PSO	18,0101	97,92	0,72
		BH	17,9959	98,06	0,58
	před. dyn.	GA	**19,9595**	97,68	0,96
		PSO	19,9428	97,72	0,92
		PSO	19,4094	97,80	0,84
		BH	19,4094	97,80	0,84
	ext. dyn.	PSO	19,9428	97,72	0,92
		GA	19,4245	98,18	0,94
		BH	19,4094	97,68	0,84
		BH	19,1855	98,44	0,82
ReLu	statický	GA	12,3310	97,64	0,82
		GA	12,3134	97,66	0,80
		GA	12,0777	97,70	0,76
		GA	12,0673	97,72	0,74
	před. dyn.	BH	**12,8488**	97,54	0,92
		GA	12,7096	97,88	0,58
		BH	12,6691	97,54	0,92
		PSO	12,3496	97,66	0,80
	ext. dyn.	GA	12,7529	97,58	0,90
		BH	12,7108	97,76	0,98
		GA	12,7001	97,80	0,96
		GA	12,6571	97,54	0,90

Tabulka 5.7: Výsledky optimalizací s různými druhy cílů

Z tabulky lze vyčíst, že díky dynamickému nastavení cíle bylo možné dosáhnout výsledků, které se více blíží stanovené hranici přesnosti, přičemž bylo dosaženo vyšší komprese. Současně lze říct, že pro síť ReLu byl cíl nastavený poměrně dobře, protože se výsledky příliš neliší mezi různými typy cílů. Dále z tabulky vyplývá, že oba přístupy nastavení dynamického cíle vedly k podobným výsledků, i když při extrémním nastavení bylo dosaženo většího rozptylu řešení.

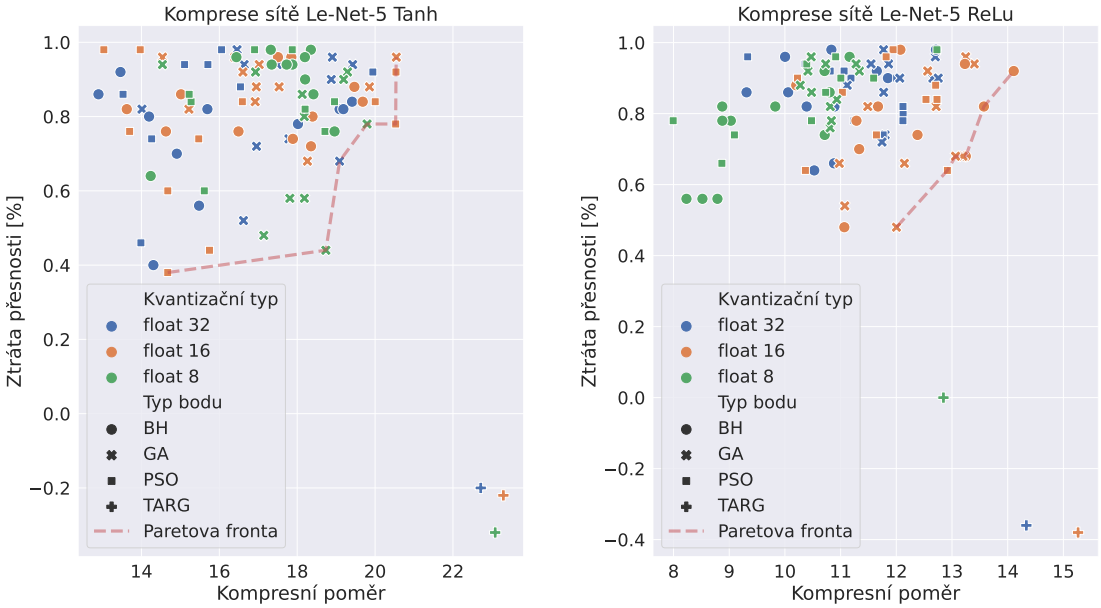
5.4 Weight-Sharing a kvantizace hodnot

V tomto experimentu bude prozkoumán vliv dodatečné kvantizace zástupců jednotlivých shluků s cílem získání vyšší přesnosti. Pro dodatečnou kvantizaci budou zvoleny typy `float16` a `float8`, znázorněné na obrázku 2.10. Pro měření budou využity poznatky z minulých experimentů, hlavně zapojení dynamických cílů. U těch se ukázala jako nejlepší varianta dynamického nastavení, kde se zvolil cíl v rámci hodnot statického cíle a bylo povoleno dynamické chování. Jelikož ale dopředu není známo, jak síť na změnu přesnosti zareaguje, bylo zvoleno extrémní nastavení cíle, které v minulém experimentu dosáhlo také uspokojivých výsledků. Dále je nastavení téměř totožné s předchozím experimentem. Je ale důležité zmínit, že pro každou přesnost je vygenerována vlastní redukce prostoru, jejíž výsledky jsou znázorněny na obrázku 5.5. Výsledky jednotlivých vyhledávání jsou obsaženy v tabulce 5.8 a na obrázku 5.10. Cílem tohoto experimentu je prozkoumání reakce sítě na převody do různých reprezentací čísel v překladoví tabulce techniky Weight-Sharing, proto bude k vizualizaci využita Paretova fronta vytvořená ze všech nalezených řešení. Znázorněna je na obrázku 5.11.

Zde se každá ze sítí projevuje jinak. Klasická implementace Le-Net-5 Tanh vykazuje minimální rozdíly mezi různými druhy kvantizace a dá se tak říci, že ztráta přesnosti vyváží kompresi a není dosaženo žádného znatelného vylepšení. Na druhou stranu Le-Net-5 ReLu využila dodatečnou kompresi na `float16` a podařilo se dosáhnout znatelně lepších výsledků (vyšší kompresní poměr až o 1,4). Kvantizace na `float8` už však byla příliš agresivní a došlo ke značné ztrátě přesnosti. Z výsledků lze tedy vyvodit, že dodatečná komprese může být přínosná, ovšem záleží na dané síti.

Druh sítě	Kvantizace	Opt. alg.	CR	Přesnost [%]	AL [%]
Tanh	`float32`	PSO	19,9428	97,72	0,92
		GA	19,4245	98,18	0,94
		BH	19,4094	97,68	0,84
	`float16`	GA	**20,5451**	97,68	0,96
		PSO	20,5368	97,72	0,92
		PSO	20,5288	97,86	0,78
	`float8`	GA	19,7939	97,86	0,78
		GA	19,2935	97,72	0,92
		GA	19,1924	97,74	0,90
ReLu	`float32`	GA	12,7529	97,58	0,90
		BH	12,7108	97,76	0,98
		GA	12,7001	97,80	0,96
	`float16`	BH	**14,1083**	97,54	0,92
		BH	13,5730	97,64	0,82
		GA	13,3997	97,52	0,94
	`float8`	PSO	12,7291	97,48	0,98
		PSO	11,5933	97,56	0,90
		GA	11,3367	97,54	0,92

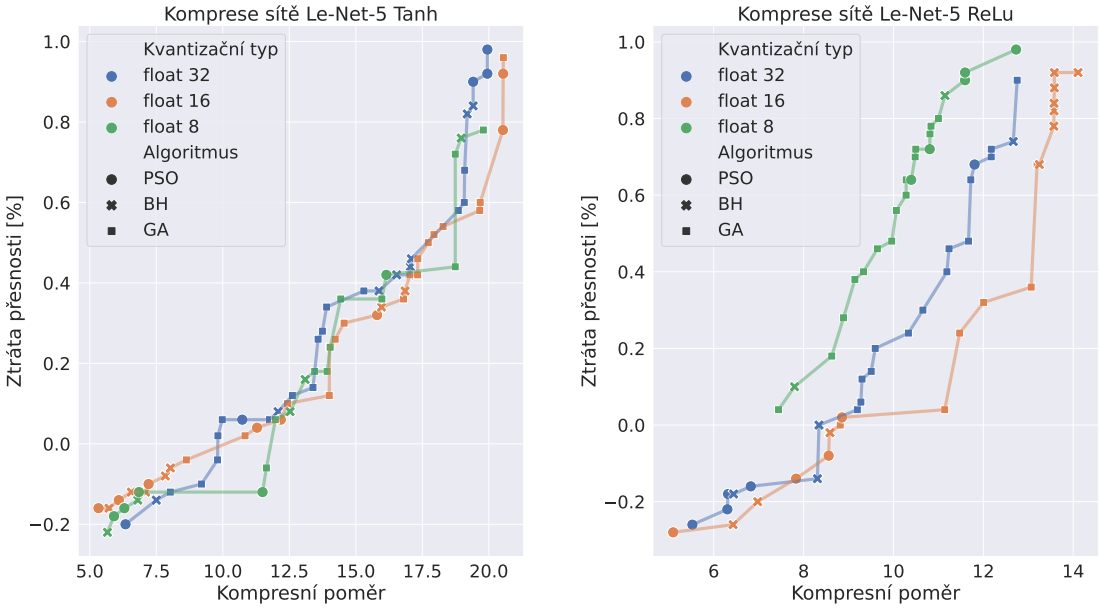
Tabulka 5.8: Výsledky optimalizací s různými dodatečnými kvantizacemi



(a) Výsledky se sítí Le-Net-5 Tanh

(b) Výsledky se sítí Le-Net-5 ReLu

Obrázek 5.10: Znázornění nejlepších objevených řešení v každém běhu pro různé reprezentace čísel a různé optimalizační algoritmy.



(a) Komprimovaná síť Le-Net-5 Tanh

(b) Komprimovaná síť Le-Net-5 ReLu

Obrázek 5.11: Znázornění Paretových front jednotlivých komprimovaných modelů s různými kvantizacemi

Výsledky experimentu je možno vyhodnotit statisticky pomocí vhodně zvoleného lineárního modelu. Model je zvolen tak, že se snaží vysvětlit zjištěnou ztrátu přesnosti jako kvadratickou funkci výsledného kompresního poměru samostatně v každé ze tří skupin vah (`float32`, `float16`, `float8`). Polynom druhého stupně byl zvolen jako vhodný model pro skutečnost, že při určitém nižším poměru komprese dochází ke zlepšení přesnosti sítě, ztráta přesnosti je v určitém úseku záporná a polynom dokáže modelovat změnu trendu. Komplexní lineární model je pak kombinací tří samostatných regresních křivek (pro skupiny float 8, 16, 32), na kterých odhaduje 3×3 regresní koeficienty. V následné analýze rozptylu modelu je možné rozhodnout, zda jsou regresní křivky tří skupin významně odlišné.

Model byl použit pro experimentální data sítě Le-Net-5 Relu (1203 bodů) a sítě Le-Net-5 Tanh (1622 bodů) a redukovaná data z Paretovy fronty pro shodné experimenty těchto sítí (66 záznamů pro síť Le-Net-5 ReLu a 83 záznamů pro Le-Net-5 Tanh).

U sítě Le-Net-5 Relu je patrné z modelu i z jeho grafické prezentace, že váhy `float16` mají nejmenší ztrátu přesnosti při rostoucí kompresi (viz vypočtené parametry v tabulce B.2), lepší než `float32`, zatímco komprese `float8` vykazuje očekávatelně horší výsledky a nedochází u ní ke zlepšení přesnosti ani při nízké kompresi. Model ANOVA dokládá významný rozdíl mezi skupinami tří regresních křivek pro síť ReLu. Následná analýza reziduí neodhalila nadměrnou nehomogenitu reziduí, v datech je částečná systematická heteroskedakicita[2], způsobená tím, že při nižších kompresních poměrech bývají rozdíly u jednotlivých měření za daných podmínek menší než u velkých kompresních poměrů.

U sítě Le-Net-5 Tanh je situace odlišná, kvadratický trend ztráty přesnosti je potvrzen jen pro kompresi `float8`, a ztráty přesnosti u různých skupin vah se chovají velmi podobně. Model ANOVA signalizuje méně významný rozdíl mezi skupinami vah. Vysoké signifikance jsou dány značným rozsahem modelovaného souboru.

Kontrasty mezi sítěmi jsou zvýrazněny, jsou-li ve výše popsaném modelu použita jen data získaná z Paretovy fronty. Statistické modely jsou znázorněny na grafech v obrázcích B.2 a B.3.

[2] Nekonstantnost rozptylu

5.5 Zkouška fine-tuning metody

Tento experiment se bude zabývat vyzkoušením fine-tuning metody popsané v sekci 3.1 definovanou rovnicí 3.1. Zjednodušeně v ní jde o modifikaci prostoru, ve kterém je prováděno shlukování za účelem dosažení vyšší přesnosti při stejné kompresi (počet shluků pro každou vrstvu je pevně daný). V návrhu byly představeny parametry *spread* a $focus$, experimentálně bylo zjištěno, že parametr $focus$ má na změnu přesnosti větší vliv, proto první algoritmus pracuje následujícím způsobem:

1. Pevně se zvolí parametr *spread*, pole možných hodnot pro parametr $focus$, vytvoří se index ukazující na první vrstvu sítě l a založí se prázdné pole pro výsledné hodnoty $results$.
2. Uloží se aktuální (originální) váhy v síti a provede se zadaná komprese.
3. Vytvoří se index h ukazující na první prvek možných hodnot $focus$ a založí se proměnná $best_{acc}$.
4. Vrstva l se obnoví z uložených hodnot a provede se komprese (se stejným algoritmem, s jakým byla řešení vyhledávána – v tomto případě K-Means) s daným parametrem $focus$ daným indexem h a vyzkouší se přesnost.
5. Pokud je proměnná $best_{acc}$ bez hodnoty nebo platí, že aktuální přesnost je vyšší než $best_{acc}$, $results[l]$ se nastaví na hodnotu $focus[h]$ a $best_{acc}$ se přepíše na aktuální přesnost.
6. Inkrementuje se h, pokud je h menší než délka pole hodnot parametru $focus$, opakuje se výpočet od bodu 4.
7. Inkrementuje se l, pokud je l menší, než je počet vrstev v modelu, opakuje se výpočet od bodu 3.

Pro algoritmus jsou zvoleny takové řetězce kompresí, které vycházely jako nejlepší v minulých experimentech, parametr *spread* byl zvolený na pevnou hodnotu 2 a pole hodnot pro parametr $focus$ nastaveno jako $[0, 0.2, 0.4, 0, 6 \ldots 9.8]$. Je třeba podotknout, že při nastavení $focus$ na hodnotu jsou všechny souřadnice y bodů rovné 0, a je tak zastoupená i možnost „bez modifikace“ (úprava prostoru a její vliv na rozdělení shluků je znázorněn na obrázku B.1). Při této implementaci je pro Le-Net-5 nutné vyhodnotit síť 250krát (50 prvků pole hodnot $focus$ × 5 vrstev sítě). Výsledky této metody jsou znázorněny v tabulce 5.9 (rozšířené výsledky jsou v tabulce B.1).

Druh sítě	Kvantizace	CR	ACC [%]	ACC ft [%]	AL [%]	AL ft [%]
Tanh	`float32`	19,9428	97,72	98,10	0,92	0,54
	`float16`	20,5451	97,68	98,02	0,96	0,62
	`float8`	19,7939	97,86	98,04	0,78	0,60
ReLu	`float32`	12,7529	97,58	97,80	0,90	0,66
	`float16`	14,1083	97,54	97,66	0,92	0,80
	`float8`	12,7291	97,48	97,78	0,98	0,68

Tabulka 5.9: Výsledky optimalizací s různými dodatečnými kvantizacemi (ft označuje výsledky s technikou fine-tuning)

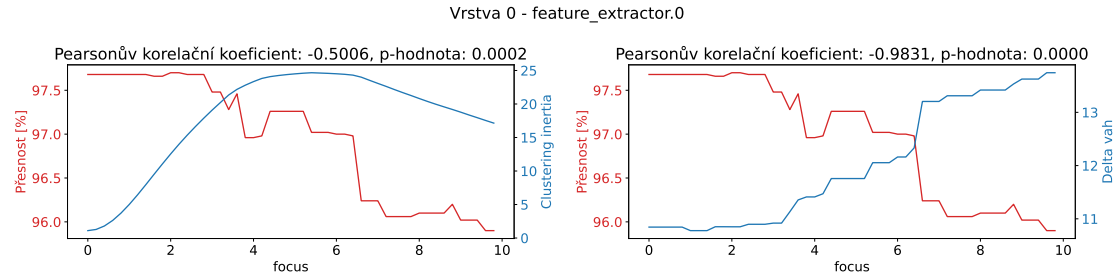
Z výsledků vyplývá, že touto technikou je možné snížit ztrátu přesnosti oproti nekomprimované síti až o třetinu. Jedná se však o relativně výpočetně náročnou operaci (nalezení kompresního řetězce vyžadovalo 400 evaluací, fine-tuning vyžadoval 250 evaluací). Z tohoto důvodu byla snaha o nalezení metriky, která by korelovala s přesností a nebylo by tak nutné síť vyhodnocovat, což by vedlo k úsporám. Byly prozkoumány dvě metriky:

1. Clustering Inertia – čtverec vzdálenosti bodů k centroidu v rámci shluku (v rozšířeném prostoru).

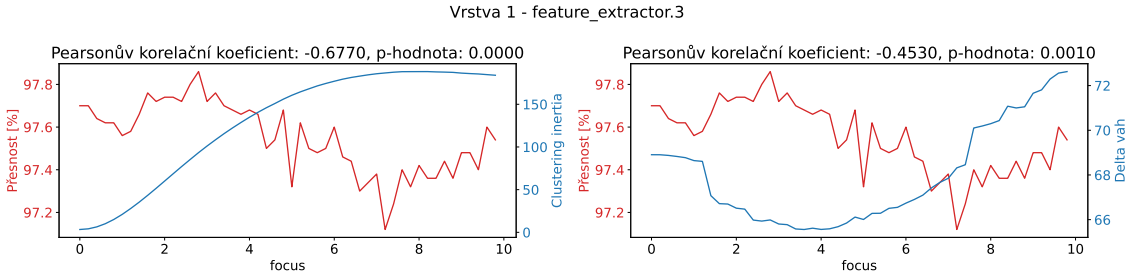
2. Delta vah – vzdálenost originální váhy ke svému zástupci (v nerozšířeném prostoru).

Výkonost těchto metrik je naznačena na obrázku 5.12, kde bude detailněji rozebrána úprava na síti Le-Net-5 Tanh s dodatečnou kvantizací na `float16`. Výsledkem této operace pro pevné hodnoty $spread = 2$ jsou hodnoty $focus = [2.0, 2.8, 4.8, 6.4, 0.0]$. Vylepšení se neprojevilo pouze na testovacím datasetu, ale i na validačním (původní: 98,20% fine-tuned: 98,28%) a trénovacím (původní: 98,37%, fine-tuned: 98,44%). Bohužel se však ani jedna z navržených metrik nejeví jako dobrá náhrada za výpočet přesnosti, neboť korelace nebyly konzistentní v jednotlivých vrstvách (například delta vah ukazuje kladnou korelaci v 5.12c, ovšem zápornou korelaci v 5.12b).

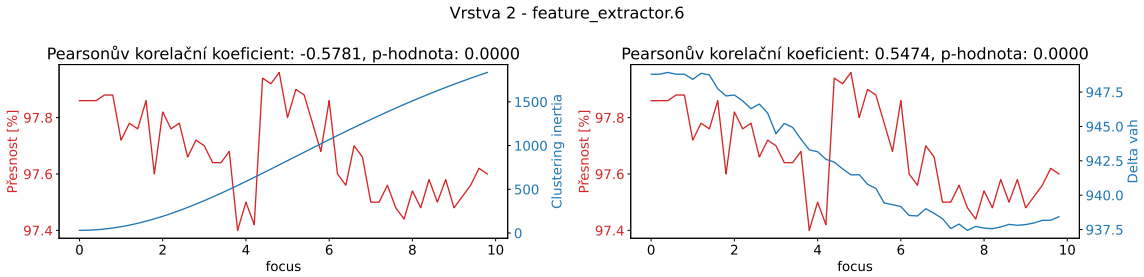
Byly provedeny i pokusy s cílem nahradit tento deterministický postup optimalizačním algoritmem PSO, neboť se jedná o spojitý prostor. PSO ovšem nedokázal se stejným počtem evaluací dosáhnout zdaleka tak dobrého výsledku a v některých případech i zhoršil původní přesnost komprimované sítě.



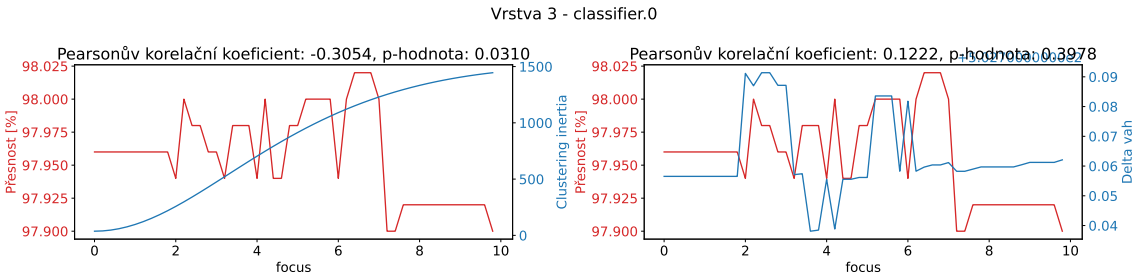
(a) Metoda fine-tuning a metriky na první vrstvě



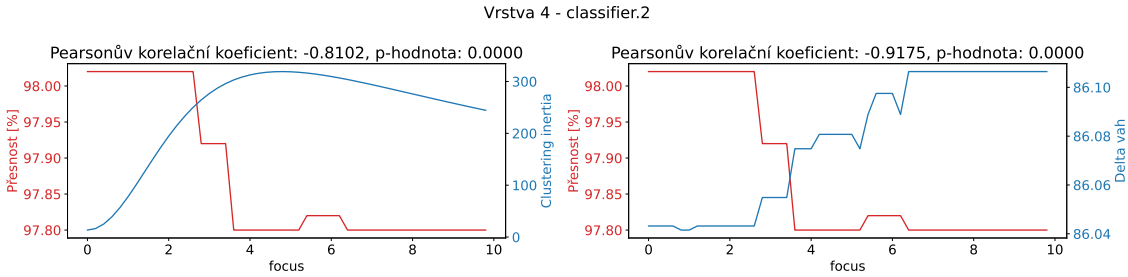
(b) Metoda fine-tuning a metriky na druhé vrstvě



(c) Metoda fine-tuning a metriky na třetí vrstvě



(d) Metoda fine-tuning a metriky na čtvrté vrstvě



(e) Metoda fine-tuning a metriky na páté vrstvě

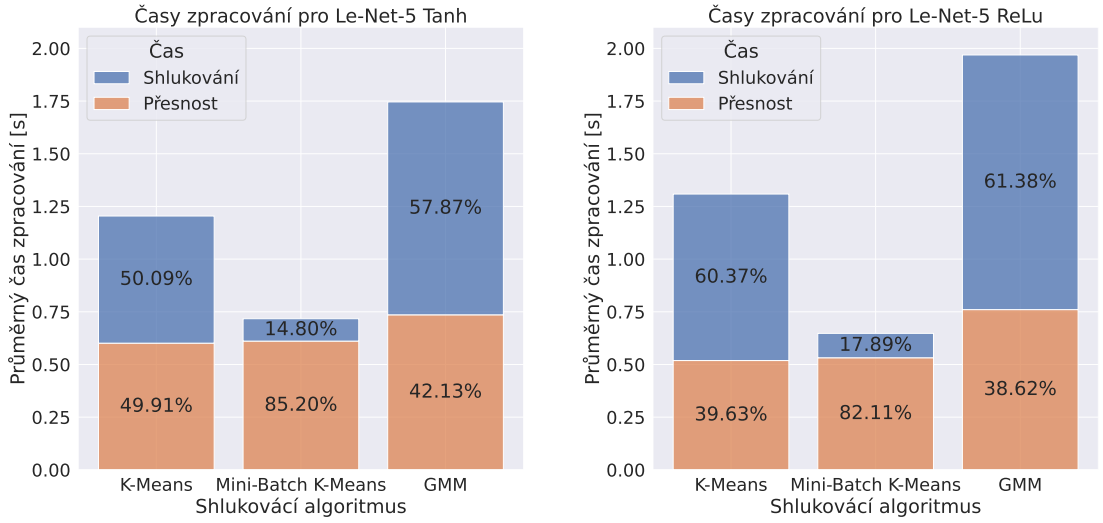
Obrázek 5.12: Průběh metody fine-tuning na síti Le-Net-5 Tanh

5.6 Zkouška shlukovacích algoritmů

Poslední experiment se sítí Le-Net-5 zkouší různé shlukovací algoritmy. Cílem bude zjistit jednak jejich schopnost poskytovat dobrá řešení, ale i časovou náročnost kompletní komprese. Vyzkoušeny budou následující možnosti:

- K-Means: Klasický K-Means algoritmus popsaný v sekci 2.4.
- Mini-Batch K-Means – M-B K-Means: Variace K-Means algoritmu navržena pro redukci časové náročnosti klasické implementace K-Means.
- Gaussian mixture model – GMM: Popsaný v sekci 2.4.

Všechny tyto algoritmy jsou implementovány pomocí knihovny scikit-learn [30]. Ostatní nastavení jsou identická jako u extrémního dynamického cíle z experimentu 5.3. Časy jsou měřeny na procesoru AMD Ryzen 7 4700U při podobných podmínkách. Výsledky jsou znázorněny na grafech v obrázcích 5.13, 5.14 a v tabulce 5.10.



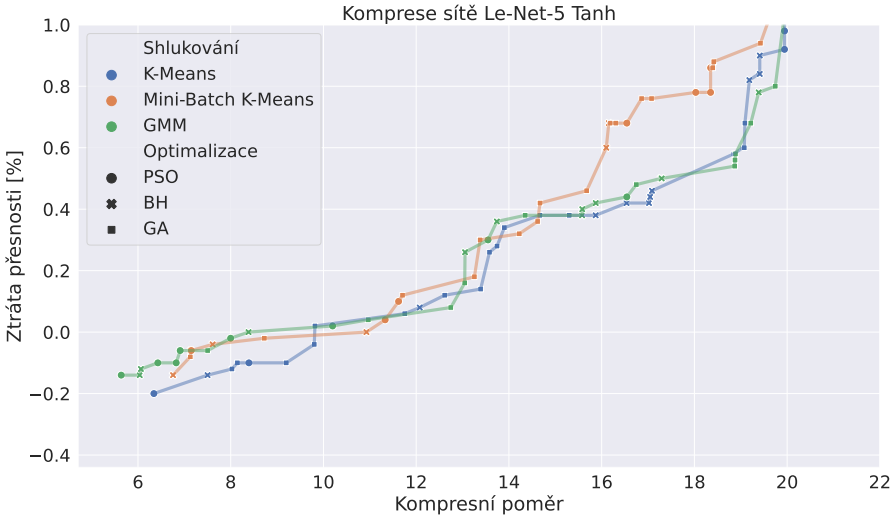
(a) Výsledky se sítí Le-Net-5 Tanh

(b) Výsledky se sítí Le-Net-5 ReLu

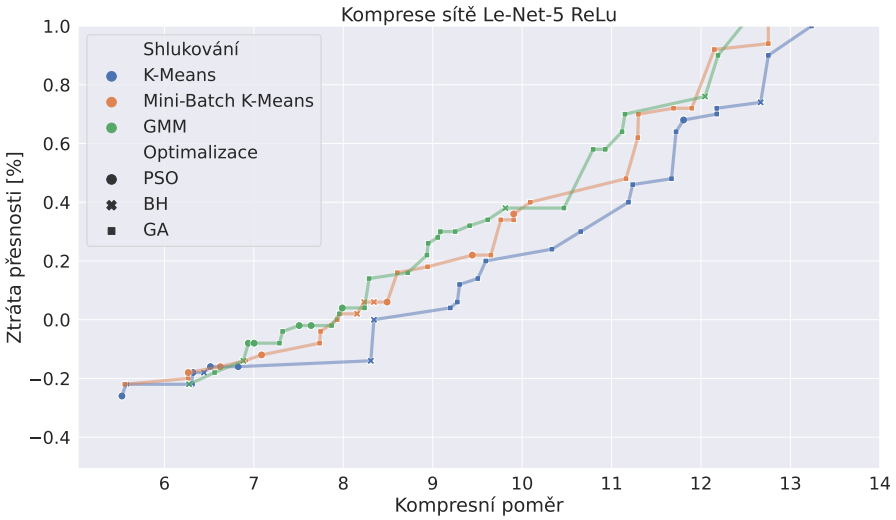
Obrázek 5.13: Průměrné časy zpracování potomka

Druh sítě	Shlukování	ACC čas [ms]	S. čas [ms]	ACC čas [%]	S. čas [%]
Tanh	K-Means	601,1	603,3	49,91	50,09
	M-B K-Means	610,9	106,1	85,20	14,80
	GMM	735,5	1010,2	42,13	57,87
ReLu	K-Means	518,6	790,0	39,63	60,37
	M-B K-Means	531,7	115,8	82,11	17,89
	GMM	760,3	1208,3	38,62	61,38

Tabulka 5.10: Časové rozdíly zpracování potomka jednotlivých typů shlukování (ACC čas – čas vyhodnoceni sítě, S. čas – čas shlukování vah)



(a) Výsledky se sítí Le-Net-5 Tanh



(b) Výsledky se sítí Le-Net-5 ReLu

Obrázek 5.14: Působení různých shlukovacích algoritmů na sítě Le-Net-5 - Paretovy fronty

Nejprve je třeba poukázat na chybu měření, neboť pro shlukování pomocí GMM byl celkově běh pomalejší (vyhodnocení přesnosti sítě trvalo déle než u ostatních, přestože bylo nastaveno totožně). Z tohoto důvodu se nedají poměřovat přímo časy. Místo toho však lze poměřovat poměry časů, které operace zabraly. Z těch lze říci, že nejvíce času trvalo shlukování GMM a o trochu méně času klasické K-Means. Mini-Batch K-Means shlukovalo zdaleka nejrychleji, často skoro dvakrát rychleji než klasické K-Means. Tato rychlost je ovšem vykoupena výsledky, kterých metoda dosáhla, jak lze vidět v grafech 5.14, kde nejlepších výsledků dosahuje algoritmus K-Means. Zde však rozdíly nejsou tak drastické jako v časech.

Pro zhodnocení ztráty přesnosti se vzrůstající kompresí mezi různými algoritmy byl použit shodný statistický model jako v experimentu popsaném v sekci 5.4. Zpracována byla

data z Pareto fronty pro ztrátu přesnosti do 1%. V případě obou sítí se vypočítané parametry (dílčí kvadratické regrese) jednotlivých algoritmů od sebe statisticky odlišují. U sítě Le-Net-5 Relu je nejlepším kandidátem algoritmus K-means, Pro síť Le-Net-5 Tanh má menší ztrátu přesnosti algoritmus K-means u nižších kompresí, zatímco algoritmus GMM dává lepší výsledky v případě vyššího kompresního poměru (nad 16). Výsledky jsou znázorněny v tabulce B.3 a v grafech na obrázku B.4.

5.7 Experimenty na MobileNet_v2

Se znalostmi získanými při předešlých experimentech na síti Le-Net-5 bude proveden pokus o kompresi sítě MobileNet_v2 popsané v sekci 2.2.2. Síť obsahuje 53 vrstev, které je možné optimalizovat technikou Weight-Sharing. Jako shlukovací algoritmus byl pro svoji rychlost vybrán Mini-Batch K-Means, neboť se jedná o velkou síť a je třeba výpočet urychlit. Síť bude při kompresi dodatečně kvantizována na reprezentace typů `float32` (bez dodatečné kvantizace), `float16` a `float8`. Prohledávací prostor bude inspirovaný prací [8] z důvodu možnosti následného porovnání výsledků. V práci [8] bylo dosaženo kompresního poměru 4,85 se ztrátou přesnosti 1,43% v prohledávacím prostoru 2-120 shluků na jednu vrstvu. Přesnost se bude měřit na testovacím subsetu datasetu Imagenette (originální validační subdataset rozdělen v poměru 70% na testovací dataset a 30% na použitý validační), opět s cílem urychlení výpočtu oproti vyhodnocování na ImageNetu. Metrikou přesnosti bude Top-1 ACC. Cílem je dosáhnout podobných, případně lepších výsledků než ve jmenované práci pomocí aplikace dodatečné kvantizace a metodou dodatečného fine-tuningu.

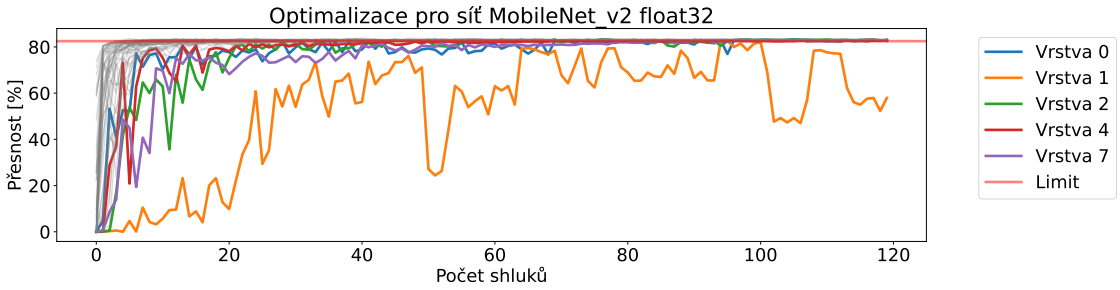
V prvním kroku je provedena optimalizace vyhledávacího prostoru pro všechny kvantizační typy s hranicí top-1 přesnosti 82,5%, jejíž výsledky jsou znázorněny na obrázku 5.15. Již z těchto optimalizací je možné vyčíst, že redukce přesnosti z `float32` na `float16` měla velmi malý dopad na chování sítě. Na druhou stranu nemůže experiment dále pokračovat s kvantizací na `float8`, protože pro některé vrstvy nebyly nalezeny počty shluků, které by splňovaly kritérium. Z grafů lze také vyčíst, že nejcitlivějšími na kompresi jsou vrstvy 1 a 7 (soupis vrstev braných v potaz během komprese je znázorněn v tabulce B.4). Optimalizace prostoru vede k jeho obrovské redukci z cca $1,57 \times 10^{110}$ možných kombinací na $8,10 \times 10^{101}$ v případě `float32` a na $8,58 \times 10^{101}$ v případě `float16`.

Dalším krokem experimentu je tedy komprese samotná. Pro optimalizaci je vybrán Genetický algoritmus, protože jak bylo řečeno v experimentu 5.2, je jednodušší jej nastavit pro dosažení dobrých výsledků. Zvolené nastavení je následující:

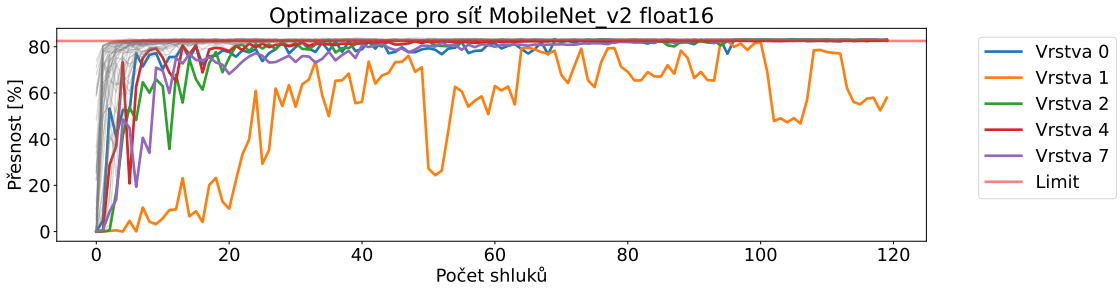
- Výběr rodičů pomocí algoritmu rulety.
- Uniformní křížení rodičů (znázorněno na obrázku 2.9).
- Mutace pro každý gen s pravděpodobností 1%.
- Jeden elitní jedinec v každé generaci.
- 12 jedinců v každé generaci (včetně elitního jednice).

Pro fitness funkci je zaveden dynamický cíl s následujícím nastavením:

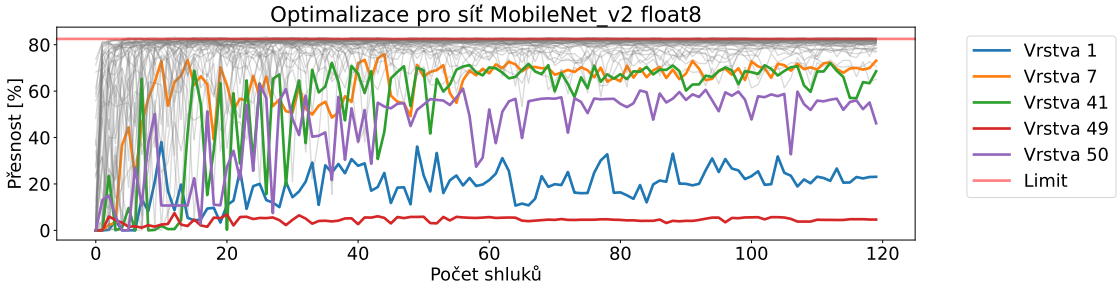
- Počáteční cíl – $CR_{target} = 4,7,\ ACC_{target} = 0,79$
- Limit pro úpravu cíle – $CR_{treshold} = 1,0,\ ACC_{treshold} = 0,79$



(a) Optimalizace pro kvantizaci `float32`



(b) Optimalizace pro kvantizaci `float16`



(c) Optimalizace pro kvantizaci `float8`

Obrázek 5.15: Optimalizace prohledávacího prostoru pro síť MobileNet_v2 s různým nastavením kvantizace

- Odsazení cíle při úpravě – $CR_{offset} = 0,1$, $ACC_{offset} = 0,001$

Cíl je nastaven pomocí znalostí získaných z práce [8], přičemž je umožněn posun k lepším výsledkům. Jak bylo ukázáno v experimentu 5.3, přednastavený cíl lehce vylepší šanci na nalezení lepšího řešení. Prohledávání bylo provedeno pro obě kvantizace po 58 generací. Pro získání hodnot přesnosti je opět využit testovací subset datasetu Imagenette. Výsledky budou poté ověřeny i na trénovacím a validačním subdatasetu Imagenette a na subdatasetu Imagewang (rozdělen na trénovací, validační a testovací stejně jako Imagenette). Prostor nalezených řešení je znázorněn v grafech na obrázku 5.16. V posledním kroku byla na nalezených řešeních provedena technika fine-tuning. Ta byla uplatněna stejně jako v experimentu 5.5 pomocí modulace parametru $focus$ přes hodnoty $[0, 0.2, 0.40.6, \ldots, 9.8]$. Výsledky jsou znázorněny opět v tabulce 5.11.

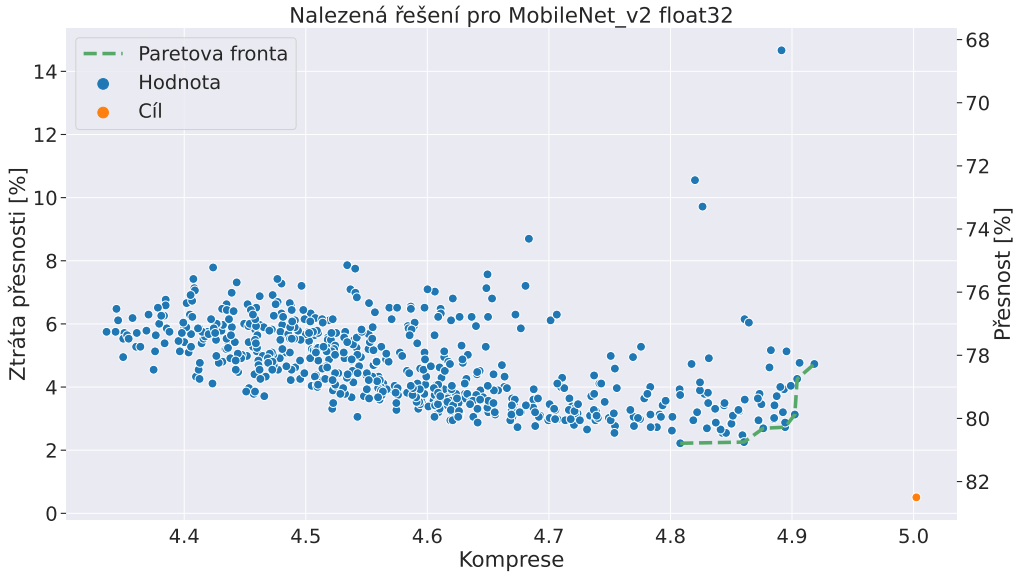
Z výsledků je bohužel patrné, že zvolený subdataset není úplně správný, neboť lze pozorovat přetrénování sítě pouze na data z Imagenette a špatnou generalizaci na další data

Q	CR	Dataset	typ ACC	Subset	ACC [%]	AL [%]	ft ACC [%]	ft AL [%]
float32	4,8943	Imagenette	Top 1	Train	80,27	2,82	82,75	0,34
				Valid	80,96	1,36	83,17	-0,85
				Test	80,67	2,22	83,91	-1,02
			Top 5	Train	95,31	1,26	96,07	0,50
				Valid	95,07	1,10	95,41	0,76
				Test	95,23	1,24	96,06	0,41
		Imagewang	Top 1	Train	79,62	3,01	81,99	0,64
				Valid	70,03	9,51	70,11	9,43
				Test	72,55	7,56	71,86	8,25
			Top 5	Train	95,09	1,31	95,75	0,65
				Valid	92,27	2,72	93,03	1,96
				Test	93,38	2,36	92,29	3,45
float16	4,7617	Imagenette	Top 1	Train	80,47	2,62	82,34	0,75
				Valid	81,73	0,59	84,28	-1,96
				Test	80,13	2,76	82,96	-0,07
			Top 5	Train	95,25	1,32	95,77	0,80
				Valid	95,32	0,85	95,83	0,34
				Test	95,12	1,35	96,06	0,41
		Imagewang	Top 1	Train	79,41	3,22	81,50	1,13
				Valid	65,78	13,76	69,43	10,11
				Test	68,70	11,41	72,48	7,63
			Top 5	Train	94,92	1,48	95,61	0,79
				Valid	91,00	3,99	92,86	2,13
				Test	91,93	3,81	92,87	2,87

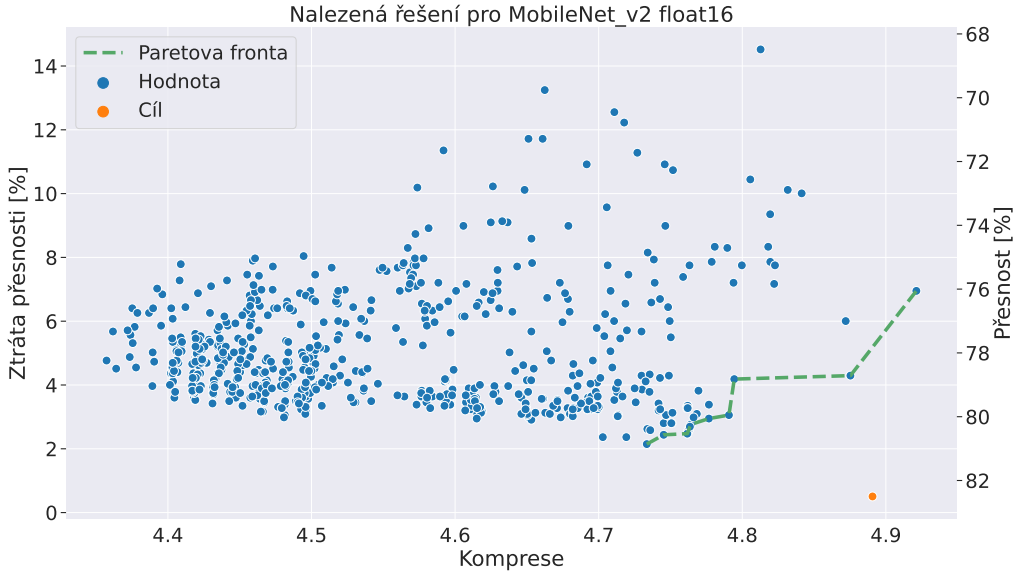
Tabulka 5.11: Přesnosti pro různě komprimované sítě MobileNet_v2. Modré řádky znázorňují datasety použité při kompresi a při fine-tuningu. Červeně znázorněné výsledky ukazují případy, kdy metoda fine-tuning zklamala a zhoršila výslednou přesnost. (vysvětlivky: Q – kvantizace, CR – kompresní poměr, ACC – přesnost, Train – trénovací subdataset, Valid – validační subdataset, Test – testovací subdataset, ft – fine-tuning)

z ImageNetu. Z tohoto důvodu nelze řešení moc dobře porovnat se zmíněnými výsledky v projektu [8][3]. I přesto lze ovšem pozorovat zajímavé vlastnosti jednotlivých přístupů, hlavně výkony mezi různými datasety. Problémy generalizace plynou hlavně ze samotné komprese, neboť již při jejím provedení se objevují největší propady přesnosti u rozšířeného datasetu Imagewang. Dodatečná komprese v tomto případě dle výsledků také spíše uškodila, neboť se nepodařilo dostatečně vyvážit její ztrátu přesnosti kompresí. Technika fine-tuning ovšem ve většině případů výrazně zlepšila výsledek komprese.

[3] v práci [8] je při kompresi 4.85 dosaženo ztráty Top-1 přesnosti klasifikace na ImageNet 1.43%



(a) Prozkoumaná řešení pro `float32`



(b) Prozkoumaná řešení pro `float16`

Obrázek 5.16: Prozkoumaná řešení pro MobileNet_v2 s dodatečnými kvantizacemi

Kapitola 6

Závěr

V rámci diplomové práce se mi podařilo nastudovat problematiku týkající se kompresí neuronových sítí a s ní související problematiku optimalizace parametrů pomocí různých optimalizačních algoritmů, kterou jsem popsal v kapitole 2. Díky tomu byl vytvořen návrh a implementace této techniky v kapitolách 3 a 4. Projekt je zaměřený na prozkoumání techniky Weight-Sharing, její rozšíření o přídavnou kvantizaci za účelem dosažení vyšší komprese a vytvoření metody fine-tuning s cílem zvýšení přesnosti při zachování kompresního poměru.

Následně byla provedena řada experimentů se sítí Le-Net-5 pro získání znalostí ohledně techniky Weight-Sharing. Nejprve experimenty porovnaly schopnosti techniky při přístupu shlukování přes vrstvy a přes celý model, kde byla potvrzena vyšší efektivita přístupu přes vrstvy. Dále byl vyzkoušen způsob optimalizace parametrů pro tuto techniku s dynamickým cílem, který poskytuje větší spolehlivost dosažení hraničních výsledků při optimalizaci. Pro optimalizaci byly využity různé optimalizační a shlukovací algoritmy. Byl zkoumán i přístup dodatečné kvantizace nad technikou Weight-Sharing. Znalosti získané z těchto experimentů byly dále využity při pokusu o kompresi složitější sítě MobileNet_v2.

Výsledky bohužel nebyly porovnatelné s jinými projekty z důvodu nevhodně zvoleného datasetu. Na druhou stranu se podařilo objevit zajímavou metodu k vyladění výsledku, která ve většině případech vedla k vylepšení řešení. Také se podařilo navrhnout a implementovat systém, který automaticky komprimuje konvoluční neuronové sítě, čemuž pomohla implementace účelové funkce na bázi vzdálenosti k ideálnímu bodu a dynamický pohyb tohoto bodu. Dále byla prozkoumána dodatečná kvantizace nad technikou Weight-Sharing.

Další případné pokračování práce by se mohlo zabývat například odhadem přesnosti bez nutnosti inferencí nad příslušným datasetem s vidinou dalšího zrychlení výpočtu, neboť tato část ohodnocení jedince zabírá podstatnou část výpočtu, jak je patrné z experimentu popsaného v sekci 5.6. Dalším směrem je jistě hlubší prozkoumání techniky fine-tuning, kde hlavním vylepšením by byl odhad parametrů bez nutnosti měření přesnosti sítě nad datasetem a sofistikovanější úprava prostoru pomocí funkce, která bude více na míru vahám sítě. To by mohlo vést k zapojení techniky do samotné optimalizace, kde by se operace fine-tuning mohla provádět na všech jedincích, aby byla nalezena ještě lepší řešení. Případně lze tuto techniku vyzkoušet i na jiných sítích s jinými shlukovacími a optimalizačními algoritmy. Možné by bylo také zhodnotit vliv přetrénování sítě během procesu komprese či prozkoumání přístupu, kde bude komprese Weight-Sharing aplikována na síť již během trénování.

Literatura

[1] *IEEE Standard for Floating-Point Arithmetic*. 2019. DOI: 10.1109/IEEESTD.2019.8766229.

[2] Anwar, S., Hwang, K. a Sung, W. *Structured Pruning of Deep Convolutional Neural Networks*. arXiv, 2015. DOI: 10.48550/ARXIV.1512.08571. Dostupné z: https://arxiv.org/abs/1512.08571.

[3] Baskin, C., Liss, N., Schwartz, E., Zheltonozhskii, E., Giryes, R. et al. UNIQ: Uniform Noise Injection for Non-Uniform Quantization of Neural Networks. *ACM Transactions on Computer Systems*. Association for Computing Machinery (ACM). nov 2019, sv. 37, 1-4, s. 1–15. DOI: 10.1145/3444943. Dostupné z: https://doi.org/10.1145%2F3444943.

[4] Blumenstein, M. a Verma, B. A Neural Network for Real-World Postal Address Recognition. In: Chawdhry, P. K., Roy, R. a Pant, R. K., ed. *Soft Computing in Engineering Design and Manufacturing*. London: Springer London, 1998, s. 79–83. ISBN 978-1-4471-0427-8.

[5] Buhmann, M. D. *Radial Basis Functions: Theory and Implementations*. Cambridge University Press, 2003. Cambridge Monographs on Applied and Computational Mathematics. ISBN 0521633389.

[6] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. et al. ImageNet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. June 2009, s. 248–255. DOI: 10.1109/CVPR.2009.5206848. ISSN 1063-6919.

[7] Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*. IEEE. 2012, sv. 29, č. 6, s. 141–142.

[8] Dupuis, E. *Méthode de partage de poids pour compresser des réseaux de neurones profonds sans réentraînement*. 2022. Theses. Université de Lyon. Dostupné z: https://theses.hal.science/tel-03764940.

[9] Dupuis, E., Novo, D., O'Connor, I. a Bosio, A. A Heuristic Exploration of Retraining-free Weight-Sharing for CNN Compression. In: *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2022, s. 134–139. DOI: 10.1109/ASP-DAC52403.2022.9712487.

[10] Elbtity, M. E., Son, H.-W., Lee, D.-Y. a Kim, H. High Speed, Approximate Arithmetic Based Convolutional Neural Network Accelerator. In: *2020 International*

SoC Design Conference (ISOCC). 2020, s. 71–72. DOI: 10.1109/ISOCC50952.2020.9333013.

[11] Han, S., Mao, H. a Dally, W. J. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. arXiv, 2015. DOI: 10.48550/ARXIV.1510.00149. Dostupné z: https://arxiv.org/abs/1510.00149.

[12] He, K., Zhang, X., Ren, S. a Sun, J. *Deep Residual Learning for Image Recognition*. arXiv, 2015. DOI: 10.48550/ARXIV.1512.03385. Dostupné z: https://arxiv.org/abs/1512.03385.

[13] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W. et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017.

[14] Howard, J. *Imagenette*. 2023. Dostupné z: https://github.com/fastai/imagenette/.

[15] Ismail, A. H., Hartono, N., Zeybek, S. a Pham, D. T. Using the Bees Algorithm to solve combinatorial optimisation problems for TSPLIB. *IOP Conference Series: Materials Science and Engineering*. IOP Publishing. apr 2020, sv. 847, č. 1, s. 012027. DOI: 10.1088/1757-899X/847/1/012027. Dostupné z: https://dx.doi.org/10.1088/1757-899X/847/1/012027.

[16] Jin, X. a Han, J. K-Means Clustering. In: Sammut, C. a Webb, G. I., ed. *Encyclopedia of Machine Learning*. Boston, MA: Springer US, 2010, s. 563–564. DOI: 10.1007/978-0-387-30164-8_425. ISBN 978-0-387-30164-8. Dostupné z: https://doi.org/10.1007/978-0-387-30164-8_425.

[17] Jouppi, N. P. a tým, Y. a. In-Datacenter Performance Analysis of a Tensor Processing Unit. *SIGARCH Comput. Archit. News*. New York, NY, USA: Association for Computing Machinery. jun 2017, sv. 45, č. 2, s. 1–12. DOI: 10.1145/3140659.3080246. ISSN 0163-5964. Dostupné z: https://doi.org/10.1145/3140659.3080246.

[18] Krizhevsky, A., Sutskever, I. a Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C., Bottou, L. a Weinberger, K., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, sv. 25. ISBN 9780262528016. Dostupné z: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[19] Kumar, S., Datta, D. a Singh, S. Black Hole Algorithm and Its Applications. *Studies in Computational Intelligence*. Springer. Prosinec 2015, sv. 575, s. 147–170. DOI: 10.1007/978-3-319-11017-2_7.

[20] Liu, Z., Wu, Z. a Tóth, R. *SMOKE: Single-Stage Monocular 3D Object Detection via Keypoint Estimation*. arXiv, 2020. DOI: 10.48550/ARXIV.2002.10111. Dostupné z: https://arxiv.org/abs/2002.10111.

[21] Lorenzo, V. Evolutionary tinkering vs. rational engineering in the times of synthetic biology. *Life Sciences, Society and Policy*. Srpen 2018, sv. 14. DOI: 10.1186/s40504-018-0086-x.

[22] Luu, H. M. a Park, S.-H. *Extending nn-UNet for brain tumor segmentation.* arXiv, 2021. DOI: 10.48550/ARXIV.2112.04653. Dostupné z: https://arxiv.org/abs/2112.04653.

[23] McCulloch, W. S. a Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics.* Springer. 1943, sv. 5, č. 4, s. 115–133.

[24] Miller, J. F. Cartesian Genetic Programming. In: Miller, J. F., ed. *Cartesian Genetic Programming.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 17–34. DOI: 10.1007/978-3-642-17310-3_2. ISBN 978-3-642-17310-3. Dostupné z: https://doi.org/10.1007/978-3-642-17310-3_2.

[25] Mitchell, M. *An Introduction to Genetic Algorithms.* Cambridge, MA, USA: MIT Press, 1998. ISBN 0262631857.

[26] Mocerino, L. a Calimera, A. AxP: A HW-SW Co-Design Pipeline for Energy-Efficient Approximated ConvNets via Associative Matching. *Applied Sciences.* 2021, sv. 11, č. 23. DOI: 10.3390/app112311164. ISSN 2076-3417. Dostupné z: https://www.mdpi.com/2076-3417/11/23/11164.

[27] Park, E., Ahn, J. a Yoo, S. Weighted-Entropy-Based Quantization for Deep Neural Networks. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017, s. 7197–7205. DOI: 10.1109/CVPR.2017.761. ISBN 978-1-5386-0457-1.

[28] Parsopoulos, K. E. a Vrahatis, M. N. Particle Swarm Optimization Method in Multiobjective Problems. In: *Proceedings of the 2002 ACM Symposium on Applied Computing.* New York, NY, USA: Association for Computing Machinery, 2002, s. 603—-607. SAC '02. DOI: 10.1145/508791.508907. ISBN 1581134452. Dostupné z: https://doi.org/10.1145/508791.508907.

[29] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* Curran Associates, Inc., 2019. Dostupné z: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[30] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B. et al. *Scikit-learn: Machine Learning in Python.* 2011.

[31] Rawat, W. a Wang, Z. Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. *Neural Computation.* Červen 2017, sv. 29, s. 1–98. DOI: 10.1162/NECO_a_00990.

[32] Rojas, R. *Neural Networks: A Systematic Introduction.* 1. vyd. Berlin, Heidelberg: Springer-Verlag, 1996. 151–184 s. ISBN 3540605053.

[33] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. a Chen, L.-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2018, s. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[34] Sekanina, L. Neural Architecture Search and Hardware Accelerator Co-Search: A Survey. *IEEE Access*. 2021, sv. 9, s. 151337–151362. DOI: 10.1109/ACCESS.2021.3126685.

[35] Sze, V., Chen, Y.-H., Yang, T.-J. a Emer, J. S. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE*. 2017, sv. 105, č. 12, s. 2295–2329. DOI: 10.1109/JPROC.2017.2761740.

[36] team, T. pandas development. *Pandas-dev/pandas: Pandas*. Zenodo, únor 2020. DOI: 10.5281/zenodo.3509134. Dostupné z: https://doi.org/10.5281/zenodo.3509134.

[37] Wortsman, M. a tým, I. a. *Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time*. arXiv, 2022. DOI: 10.48550/ARXIV.2203.05482. Dostupné z: https://arxiv.org/abs/2203.05482.

[38] Xiao, H., Rasul, K. a Vollgraf, R. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. arXiv, 2017. DOI: 10.48550/ARXIV.1708.07747. Dostupné z: https://arxiv.org/abs/1708.07747.

[39] Yang, T.-J., Chen, Y.-H. a Sze, V. *Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning*. arXiv, 2016. DOI: 10.48550/ARXIV.1611.05128. Dostupné z: https://arxiv.org/abs/1611.05128.

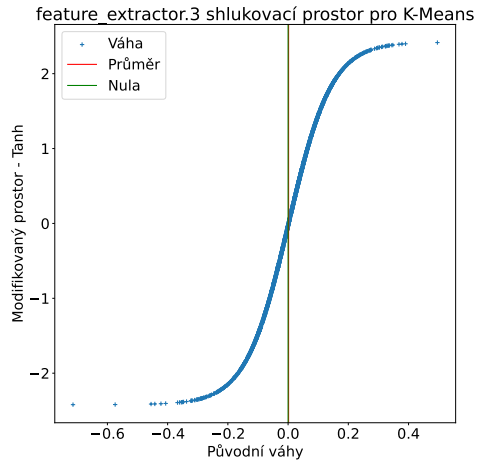
Příloha A

Obsah paměťového média

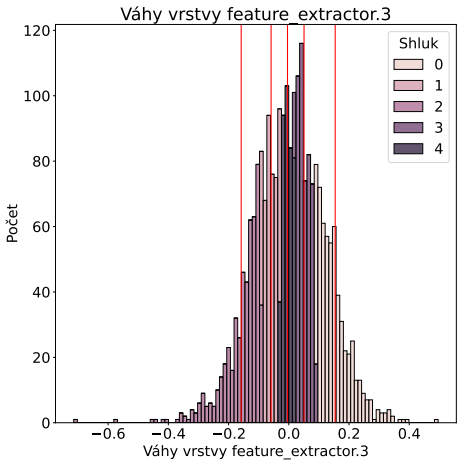
- Paměťové médium
  - `doc` – dokumentační materiály (text práce, prezentace, plakát apod.)
  - `site` – programová dokumentace (přístup přes soubor `index.html`)
  - `src` – zdrojové soubory
    - `compress_optim` – Implementované komprese pro CLI spouštění
      - `compressor_config.py` – Konfigurační soubor komprese
      - `bh_compressor.py` – Komprese s upraveným PSO pomocí Blackhole
      - `genetic_compressor.py` – Komprese s Genetickým algoritmem
      - `pso_compressor.py` – Komprese s PSO
      - `random_compressor.py` – Komprese s náhodným prohledáváním
      - `total_compressor.py` – Komprese přes celý model
    - `data` – Implementace a ukládání datasetů
      - `utils` – Nástroje pro datasety
        - `download.py` – Implementace stahování dat
        - `imagenet_utils.py`– Nástroje pro imagenet datasety
        - `mnist_utils.py` – Nástroje pro MNIST datasety
      - `imagenette.py` – Implementace Imagenette datasetu
      - `mnist.py` – Implementace MNIST datasetu
    - `models` – Data a implementace modelů (převzatý kód označen)
    - `notebooks` – Python notebooky projektu
    - `results` – Hrubé výsledky projektu (dostupné naměřené hodnoty)
    - `utils` – Nástroje pro kompresi
      - `float_prec_reducer` – Implementace dodatečné kvantizace
      - `genetic` – Implementace genetického algoritmu
      - `pso` – Implementace PSO
      - `rnd` – Implementace náhodného prohledávání
      - `fitness_controler.py` – Implementace účelové funkce
      - `plot.py` – Implementace některých grafů
      - `weight_sharing.py` – Implementace komprese technikou Weight-Sharing
  - `lenet_compression.py` – CLI program pro kompresi Le-Net
  - `net_compression.py` – CLI program pro kompresi sítí pracující s Imagenetem
  - `net_finetune.py` – CLI program pro techniku fine-tuning sítí pracující s Imagenetem
  - `net_range_opt.py` – CLI program pro redukci prostoru sítí pracující s Imagenetem
  - `requirements.txt` – Požadované knihovny projektu
  - `run_test.sh` – CLI skript pro běh velkých měření

Příloha B

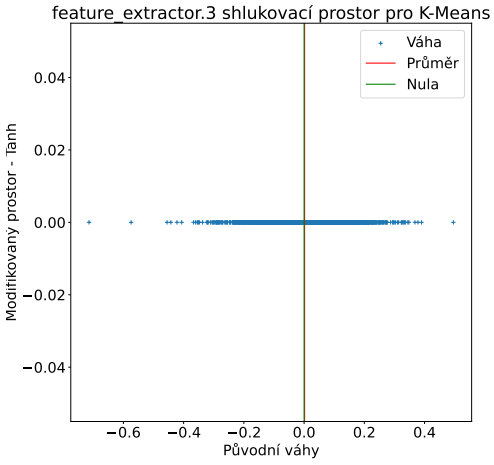
Přílohy



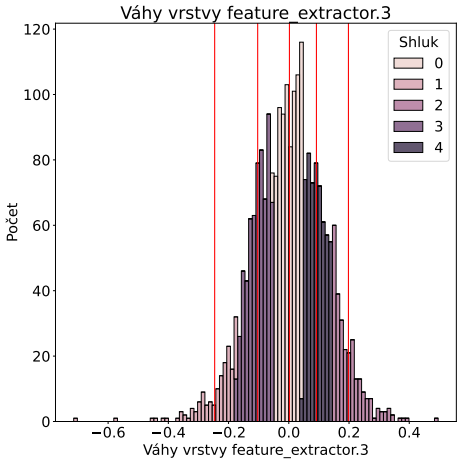
(a) Shlukovací prostor pro $focus = 7$



(b) Rozdělení vah pro $focus = 7$



(c) Shlukovací prostor pro $focus = 0$



(d) Rozdělení vah pro $focus = 0$

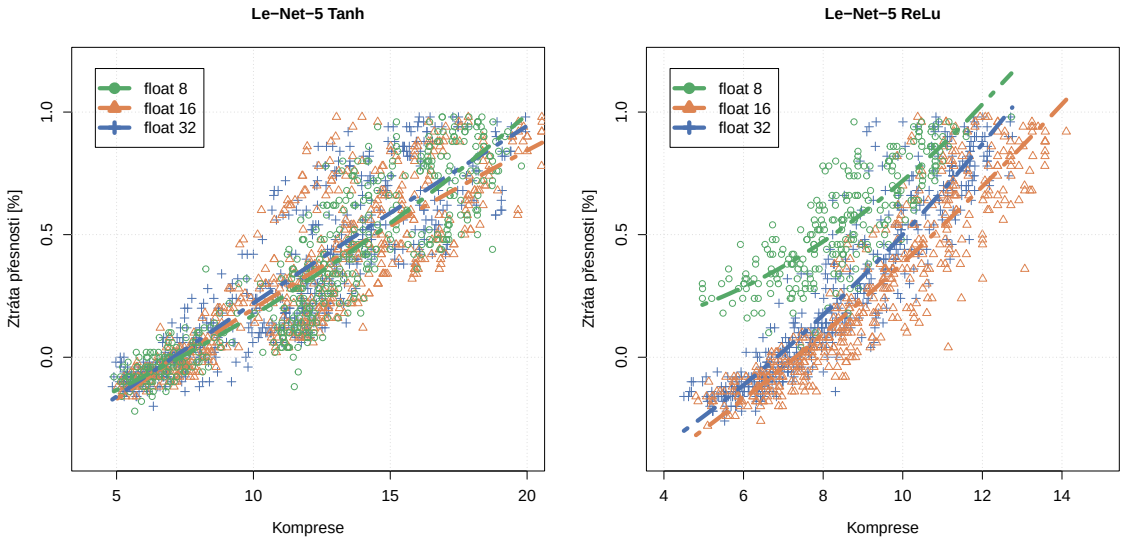
Obrázek B.1: Znázornění rozdílných hodnot $focus$ pro fine-tuning techniku na druhé vrstvě sítě Le-Net-5 Tanh

Druh sítě	Kvantizace	Opt. alg.	Seznam komprese	CR	ACC [%]	AL [%]	ACC ft [%]	AL ft [%]
Tanh	`float32`	PSO	[3, 3, 8, 2, 2]	19,9428	97,72	0,92	98,10	0,54
	`float16`	GA	[3, 5, 3, 2, 2]	20,5451	97,68	0,96	98,02	0,62
	`float8`	GA	[3, 35, 3, 2, 2]	19,7939	97,86	0,78	98,04	0,60
ReLu	`float32`	GA	[3, 4, 6, 4, 7]	12,7529	97,58	0,90	97,80	0,66
	`float16`	BH	[2, 6, 8, 11, 4]	14,1083	97,54	0,92	97,66	0,80
	`float8`	PSO	[2, 50, 10, 21, 4]	12,7291	97,48	0,98	97,78	0,68

Tabulka B.1: Rozšířená tabulka s výsledky metody finetuning

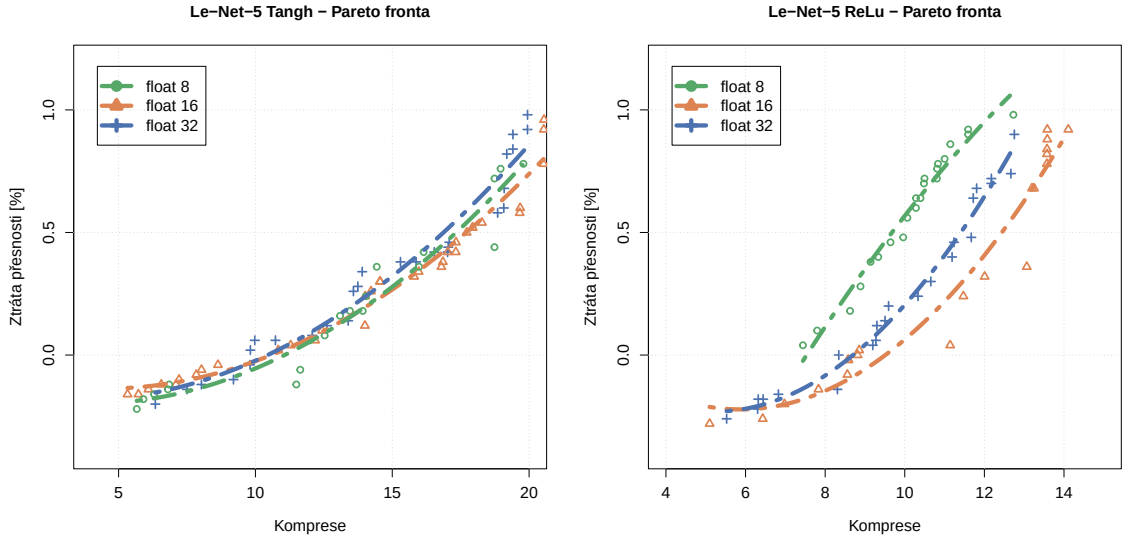
Síť	Data	Parametr	`float32`	`float16`	`float8`	ANOVA p
Tanh	Všechna	1	-0.5538719	-0.5682642	-0.3701156	$< 2 \times 10^{-16}$
		x	0.0800053	0.0825578	0.0403864	$< 2 \times 10^{-16}$
		x^2	-0.0002572	-0.0005991	0.0013816	0.01063
	Pareto	1	-0.1343557	-0.0671488	-0.1396088	$< 2.2 \times 10^{-16}$
		x	-0.0273199	-0.0318044	-0.0302206	$< 2.2 \times 10^{-16}$
		x^2	0.0038474	0.0036057	0.0038717	1.824×10^{-11}
ReLu	Všechna	1	-0.720564	-0.826532	0.098072	$< 2.2 \times 10^{-16}$
		x	0.069947	0.091885	-0.015030	$< 2.2 \times 10^{-16}$
		x^2	0.005212	0.002919	0.007721	7.182×10^{-05}
	Pareto	1	0.292941	0.356248	-2.396726	$< 2.2 \times 10^{-16}$
		x	-0.200111	-0.196759	0.384156	$< 2.2 \times 10^{-16}$
		x^2	0.019131	0.016748	-0.008770	6.111×10^{-10}

Tabulka B.2: Výsledky statistického zkoumání dopadu dodatečné kvantizace. ANOVA p sloupec značí p hodnotu testu ANOVA nad parametrem modelu.



(a) Statistické modely pro síť Le-Net-5 Tanh (b) Statistické modely pro síť Le-Net-5 ReLu

Obrázek B.2: Statistické modely přes všechna data pro Le-Net-5 – zkoumání dodatečné kvantizace

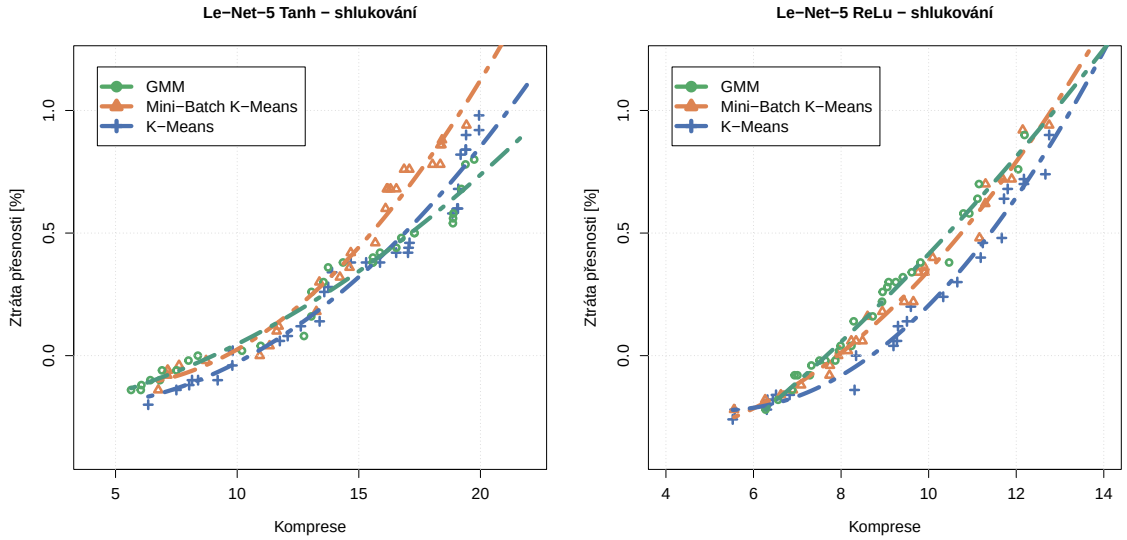


(a) Statistické modely pro síť Le-Net-5 Tanh (b) Statistické modely pro síť Le-Net-5 ReLu

Obrázek B.3: Statistické modely přes data z Paretovy fronty pro Le-Net-5 – zkoumání dodatečné kvantizace

Síť	Parametr	K-Means	Mini-Batch K-Means	GMM	ANOVA p
Tanh	1	0.321165	-0.244520	-0.975998	$< 2.2 \times 10^{-16}$
	x	-0.205035	-0.076583	0.089064	$< 2.2 \times 10^{-16}$
	x^2	0.019343	0.013574	0.004996	$< 2.2 \times 10^{-16}$
ReLu	1	-0.1761495	-0.0118622	-0.2560869	$< 2.2 \times 10^{-16}$
	x	-0.0217882	-0.0493379	0.0109553	$< 2.2 \times 10^{-16}$
	x^2	0.0036616	0.0053062	0.0019365	2.085×10^{-10}

Tabulka B.3: Výsledky statistického zkoumání dopadu různých shlukovacích algoritmů. ANOVA p sloupec značí p hodnotu testu ANOVA nad parametrem modelu.



(a) Statistické modely pro síť Le-Net-5 Tanh (b) Statistické modely pro síť Le-Net-5 ReLu

Obrázek B.4: Statistické modely přes data z Paretovy fronty pro Le-Net-5 – zkoumání různých shlukovacích algoritmů

Index	Jméno vrstvy	Počet vah	Index	Jméno vrstvy	Počet vah
0	features.0.0	864	27	features.10.conv.0.0	24576
1	features.1.conv.0.0	288	28	features.10.conv.1.0	3456
2	features.1.conv.1	512	29	features.10.conv.2	24576
3	features.2.conv.0.0	1536	30	features.11.conv.0.0	24576
4	features.2.conv.1.0	864	31	features.11.conv.1.0	3456
5	features.2.conv.2	2304	32	features.11.conv.2	36864
6	features.3.conv.0.0	3456	33	features.12.conv.0.0	55296
7	features.3.conv.1.0	1296	34	features.12.conv.1.0	5184
8	features.3.conv.2	3456	35	features.12.conv.2	55296
9	features.4.conv.0.0	3456	36	features.13.conv.0.0	55296
10	features.4.conv.1.0	1296	37	features.13.conv.1.0	5184
11	features.4.conv.2	4608	38	features.13.conv.2	55296
12	features.5.conv.0.0	6144	39	features.14.conv.0.0	55296
13	features.5.conv.1.0	1728	40	features.14.conv.1.0	5184
14	features.5.conv.2	6144	41	features.14.conv.2	92160
15	features.6.conv.0.0	6144	42	features.15.conv.0.0	153600
16	features.6.conv.1.0	1728	43	features.15.conv.1.0	8640
17	features.6.conv.2	6144	44	features.15.conv.2	153600
18	features.7.conv.0.0	6144	45	features.16.conv.0.0	153600
19	features.7.conv.1.0	1728	46	features.16.conv.1.0	8640
20	features.7.conv.2	12288	47	features.16.conv.2	153600
21	features.8.conv.0.0	24576	48	features.17.conv.0.0	153600
22	features.8.conv.1.0	3456	49	features.17.conv.1.0	8640
23	features.8.conv.2	24576	50	features.17.conv.2	307200
24	features.9.conv.0.0	24576	51	features.18.0	409600
25	features.9.conv.1.0	3456	52	classifier.1	1280000
26	features.9.conv.2	24576	–	–	–

Tabulka B.4: Vrstvy sítě MobileNet_v2 uvažované při kompresi. Názvy jsou určeny implementací poskytnuté knihovnou Pytorch a je v nich označen také typ vrstvy.