



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

DEPARTMENT OF COMPUTER SYSTEMS

**EVOLUČNÍ NÁVRH KONVOLUČNÍCH NEURONOVÝCH
SÍTÍ S VYUŽITÍM SUPERSÍTĚ**

EVOLUTIONARY DESIGN OF CONVOLUTIONAL NEURAL NETWORKS

UTILIZING A SUPERNET

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. ZBYNĚK LAMAČKA

VEDOUCÍ PRÁCE

SUPERVISOR

prof. Ing. LUKÁŠ SEKANINA, Ph.D.

BRNO 2024

Zadání diplomové práce



154227

Ústav: Ústav počítačových systémů (UPSY)
Student: **Lamačka Zbyněk, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Strojové učení
Název: **Evoluční návrh konvolučních neuronových sítí s využitím supersítě**
Kategorie: Umělá inteligence
Akademický rok: 2023/24

Zadání:

1. Seznamte se s možnostmi využití evolučních algoritmů pro návrh a optimalizaci hlubokých neuronových sítí. Zaměřte se na metody budující supersítě.
2. Seznamte se s knihovnamy pro práci s konvolučními neuronovými sítěmi (CNN) a s dostupnými natrénovanými modely CNN a supersítí.
3. Navrhněte způsob využití vícekritériálního evolučního algoritmu při automatizovaném návrhu konvoluční neuronové sítě s využitím supersítě.
4. Navržený způsob implementujte a experimentálně ověřte jeho přínos na zvolené úloze z oblasti klasifikace obrazu.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

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: 1.11.2023
Termín pro odevzdání: 17.5.2024
Datum schválení: 30.10.2023

Abstrakt

Tato práce se zabývá možnostmi automatizovaného návrhu a optimalizace konvolučních neuronových sítí (CNN) pomocí evolučních algoritmů s využitím konceptu neuroevoluce (NAS – Neural Architecture Search). NAS metody usnadňují práci architektům neuronových sítí a umožňují přístup k neuronovým sítím i lidem, kteří by se za běžných okolností k nim nedostaly. Architektury, které vznikají automatizovanými metodami, jsou schopny překonat architektury, které byly vytvořeny zkušenými architekty. Tyto metody nejsou svázány konvenčními přístupy k návrhu, a proto mohou vznikat inovativní architektury. Cílem této práce je návrh a implementace metody neuroevoluce využívající supersít. Koncept supersítě má za cíl proces automatického návrhu sítě zrychlit a zlevnit. Tato metoda bude vyhodnocena na základě architektur, které vygeneruje. Vyhodnocení architektur je prováděno na základě dvou kritérií – přesnost a složitost sítě. Pro vyhodnocování je použita datová sada ImageNet.

Abstract

This work explores the possibilities of automated design and optimization of convolutional neural networks (CNNs) using evolutionary algorithms with the concept of Neural Architecture Search (NAS). NAS methods facilitate the work of neural network architects and allow access to neural networks by people who would not normally have access to them. Architectures that are created by automated methods are able to outperform architectures that were created by experienced architects. These methods are not bound by conventional design approaches, and therefore innovative architectures can emerge. The goal of this work is to design and implement a neuroevolutionary method using a supernet. The supernet concept aims to make the process of automatic network design faster and cheaper. This method will be evaluated based on the architectures it generates. The evaluation of the architectures considers two criteria – accuracy and complexity of the network. The ImageNet dataset is used for the evaluation.

Klíčová slova

neuronové sítě, konvoluční neuronové sítě, evoluční algoritmy, neuroevoluce, supersít

Keywords

neural networks, convolutional neural networks, evolutionary algorithms, neuroevolution, supernet

Citace

LAMAČKA, Zbyněk. *Evoluční návrh konvolučních neuronových sítí s využitím supersítě*. Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Evoluční návrh konvolučních neuronových sítí s využitím supersítě

Prohlášení

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

.....
Zbyněk Lamačka
16. května 2024

Poděkování

Rád bych poděkoval vedoucímu práce prof. Ing. Lukášovi Sekaninovi, Ph.D. za vedení této práce, za rady a za čas, který mi věnoval. Dále bych chtěl poděkovat doc. Ing. Jiřímu Jarošovi, Ph.D. za poskytnutí přístupu k výpočetním zdrojům.

Obsah

1	Úvod	2
2	Evoluční algoritmy	4
2.1	Inicializace	5
2.2	Selekce	5
2.3	Mutace	6
2.4	Křížení	6
2.5	Ukončovací podmínka	7
2.6	Vícekriteriální evoluční algoritmus	8
3	Neuronové sítě a jejich návrh	9
3.1	Neuronová síť	9
3.2	Konvoluční neuronová síť	15
3.3	Automatizovaný návrh neuronových sítí	18
4	Evoluční návrh CNN pomocí supersítě	22
4.1	Vícekriteriální optimalizace	23
4.2	Reprezentace jedince	24
4.3	Mutace	25
4.4	Křížení	26
4.5	Ohodnocení jedince	26
4.6	Návrh evolučního algoritmu	26
5	Implementace	28
5.1	Struktura projektu	28
5.2	Implementace evolučních algoritmů	29
5.3	Paralelizace	29
5.4	Spouštění projektu	31
6	Vyhodnocení experimentů	34
6.1	Výběr parametrů evolučního algoritmu	34
6.2	Porovnání jednotlivých algoritmů	39
7	Závěr	44
	Literatura	45
A	Obsah příloženého paměťového média	48

Kapitola 1

Úvod

V dnešní době patří pojmy umělá inteligence (artificial intelligence – AI) a strojové učení (Machine learning – ML) mezi jedny z nejvíce skloňovaných pojmů z oboru informačních technologií. O tomto svědčí i fakt, že výrobci hardware začali u svých nových produktů prezentovat jejich AI schopnosti ¹ ².

Mezi jednu z nejvíce používaných technologií umělé inteligence patří neuronové sítě, které mají uplatnění v mnoha odvětvích průmyslu. Mezi nejznámější případy užití neuronových sítí patří rozpoznávání obrazu, strojový překlad a, v dnešní době asi neznámější, velké jazykové modely (large language model – LLM), kde je klasickým případem ChatGPT.

V závislosti na aktivitě, kterou bude neuronová síť vykonávat, je odvozen typ neuronové sítě. Tato práce se bude zabývat konvolučními neuronovými sítěmi (Convolutional neural network – CNN), které se používají pro práci s obrázky (například klasifikace, nebo detekce).

Návrh architektury konvoluční neuronové sítě je složitý proces, který vyžaduje zkušeného návrháře. Odborníci v oblasti IT jsou známí snahou všechno automatizovat. Je vůbec možné zautomatizovat návrh a optimalizaci konvolučních sítí? Odpověď je prostá – ano, je, jak dokládá dnes již obsáhlá literatura [22], [16] a [15].

Cílem této práce bylo nastudovat možnosti evolučního návrhu a optimalizace konvolučních neuronových sítí. Tato práce se zaměřuje na metody používající supersítě. Po nastudování problematiky bylo cílem navrhnout řešení dle nastudovaných přístupů a nakonec toto řešení implementovat.

V kapitole 2 jsou popsány evoluční algoritmy. V sekci 3.1 jsou popsány principy fungování neuronových sítí včetně používaných aktivačních funkcí a v neposlední řadě i princip učení neuronových sítí. Sekce 3.3 se zaměřuje na automatizovaný návrh neuronových sítí. V této sekci je popsána obecná problematika automatizovaného návrhu. Dále je v této sekci popsána technika Neural architecture search (NAS), která se v této oblasti v dnešní době využívá. V závěru této sekce jsou popsány one-shot metody, které pro svoji funkci používají supersítě.

Kapitola 4 se zabývá popisem návrhu implementace řešení. Nejprve je vysvětlen důvod volby použité knihovny a datové sady. Dále je v kapitole popsán návrh vícekritériálního evolučního algoritmu a důvod volby jednotlivých vyhodnocovacích kritérií. Tato kapitola obsahuje i popis křížení a mutace, které jsou v rámci této práce použity. Dále kapitola obsa-

¹<https://www.anandtech.com/show/21185/intel-releases-core-ultra-h-and-u-series-processors-meteor-lake-brings-ai-and-arc-to-ultra-thin-notebooks>

²<https://www.anandtech.com/show/21177/amd-unveils-ryzen-8040-mobile-series-apus-hawk-point-with-zen-4-and-ryzen-ai>

huje popis, jakým způsobem probíhá ohodnocování jedinců. V poslední části této kapitoly je popsán návrh samotného evolučního algoritmu.

Kapitola 5 popisuje implementaci řešení. Dále je v této kapitole popsána struktura vzniklého projektu

V kapitole 6 jsou popsány provedené experimenty. Experimenty jsou v této kapitole vyhodnoceny jak po vizuální stránce, tak po stránce statistické. Tato kapitola nejprve popisuje experimenty s nastavením jednotlivých parametrů a následně popisuje porovnání jednotlivých evolučních algoritmů, které vznikly jak v rámci této práce, tak i mimo ni.

Kapitola 2

Evoluční algoritmy

Evoluční algoritmy (EA) jsou algoritmy, které řeší optimalizační problémy na základě metod inspirovaných principy evoluce a přírodní selekce. Tyto algoritmy mají za cíl nalézt řešení problémů v různých odvětvích, jako je například strojové učení. Optimalizační problém je definován jako nalezení $\alpha^*, x^* \in \Omega$ tak, že $f(x^*)$ je minimální (v případě minimalizační úlohy). f je účelová funkce. Evoluční algoritmy mají tři hlavní charakteristické rysy [25]:

- **Populace:** EA pracují s množinou řešení, populací, za účelem řešení problému paralelně. Populace je základním principem evolučních algoritmů.
- **Fitness:** každé řešení v populaci se nazývá jedinec. Každý jedinec obsahuje informaci o tom, jak je v daném kontextu vhodný. Tato míra vhodnosti se nazývá hodnota fitness funkce. Evoluční algoritmy preferují vhodnější jedince, tedy jedince s lepší hodnotou fitness. Toto je základem optimalizace a konvergence těchto algoritmů.
- **Různorodost:** jedinci prochází operacemi, které mají za cíl zvětšit různorodost populace. Tyto operace (mutace a křížení) mají základ v genetických změnách živých organismů. Různorodost je základem pro prohledávání stavového prostoru.

Základní princip evolučních algoritmů spočívá v modelování evolučního procesu, který je často založen na Darwinově teorii přirozeného výběru. Tento proces zahrnuje opakovanou generaci populací jedinců, kteří reprezentují potenciální řešení daného problému. Jedinci jsou ohodnoceni pomocí fitness funkce (v české literatuře se někdy uvádí pojem účelová funkce), která vyhodnocuje jejich kvalitu. Pomocí selekce se z této populace vyberou jedinci, kteří slouží jako rodiče pro následující generaci. Princip fungování základního evolučního algoritmu je předveden v algoritmu 1.

Algorithm 1: Základní evoluční algoritmus

```
 $P \leftarrow$  Initialize population
while termination condition not met do
  evaluation
   $P_{\text{selected}} \leftarrow$  Select individuals from  $P$  based on fitness
   $P_{\text{offspring}} \leftarrow$  Crossover individuals in  $P_{\text{selected}}$ 
   $P_{\text{mutant}} \leftarrow$  Mutate individuals in  $P_{\text{offspring}}$ 
   $P \leftarrow$  Select individuals from  $P$  and  $P_{\text{mutant}}$  to form the next generation
end
```

Reprezentace problému

Reprezentace problému závisí na konkrétní řešené úloze. Kandidátní řešení může být bitový vektor, vektor reálných čísel, permutace objektů nebo může mít formu stromu.

2.1 Inicializace

Pro inicializaci je nutné znát, jaká je velikost populace. Tato hodnota je zpravidla nastavena jako parametr evolučního algoritmu. Následně je nutné vytvořit požadovaný počet jedinců. Někdy může být k dispozici nějaká znalost, díky které je možné vytvořit vhodnější řešení již ve fázi inicializace. Ve většině případů ale taková znalost k dispozici není, a proto je nutné iniciální jedince vytvořit náhodně. Nejjednodušší přístup je všechny elementy každého jedince vytvořit náhodně na základě rovnoměrného rozdělení.

2.2 Selektce

Selektce se provádí za účelem vybrání rodičů pro následující generaci. Selektce funguje na principu, že lepší jedinci mají vyšší šanci za vybrání. Jedinec může být vybrán vícekrát. Nejčastěji se používají tyto selekční metody:

Ruleta

Pravděpodobnost, že bude jedinec vybrán je proporční k hodnotě jeho fitness funkce. Absolutní hodnoty fitness funkce jedinců se přepočítají na relativní hodnoty. Tyto hodnoty představují pravděpodobnost, s jakou bude jedinec vybrán.

Problémem u této metody je stav, kdy jeden jedinec v populaci má mnohem vyšší hodnotu fitness funkce než ostatní jedinci. Tento jedinec bude s velkou pravděpodobností vybírán a může dojít ke stavu, kdy velká část populace bude tvořena tímto jedincem, což vede na ztrátu diverzity.

Problémem je i stav, kdy všichni jedinci mají velmi podobnou hodnotu fitness funkce. Například fitness funkce má maximální hodnotu 100 a jedinci v populaci mají hodnotu fitness funkce 95, 96 a 97. Každý z těchto jedinců bude mít pravděpodobnost výběru přibližně jednu třetinu, a tedy selektce zdegenerovala na náhodný výběr.

Rank

Jedinci jsou nejprve seřazení podle jejich hodnoty fitness funkce. Selektce se provádí na základě těchto pořadí (rank) jedinců. Díky tomu, že pro selekci není použita přímo hodnota fitness funkce, ale pořadí, je tento způsob selektce odolný vůči oběma problémům rulety.

Turnaj

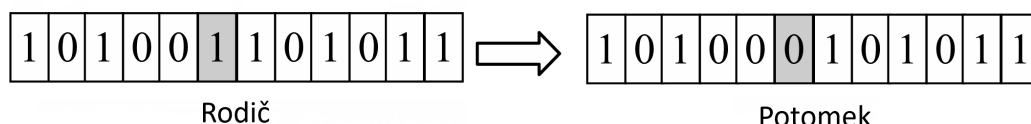
Z celkové populace se náhodně vybere „t“ jedinců (zpravidla 2). Z těchto jedinců je vybrán ten nejlepší. Tento proces se opakuje pro každého hledaného rodiče. Turnaj kombinuje náhodný výběr s výběrem na základě hodnoty fitness funkce. Díky tomu je turnaj odolný vůči oběma problémům rulety. Pomocí volby parametru „t“ je možno upravovat chování selektce. Menší hodnoty parametru „t“ kladou větší důraz na náhodu a vyšší hodnoty kladou důraz na výběr nejlepších jedinců

Deterministický výběr

Z celkové populace se vybere požadovaný počet nejlepších jedinců.

2.3 Mutace

Operátor mutace má za cíl zvětšení diverzity v rámci populace pomocí zavedení náhodné změny. Mutace se provádí na jednom jedinci, kde provede změnu jednoho (nebo více) genu. Pro binárního jedince je mutace znázorněna na obrázku 2.1.



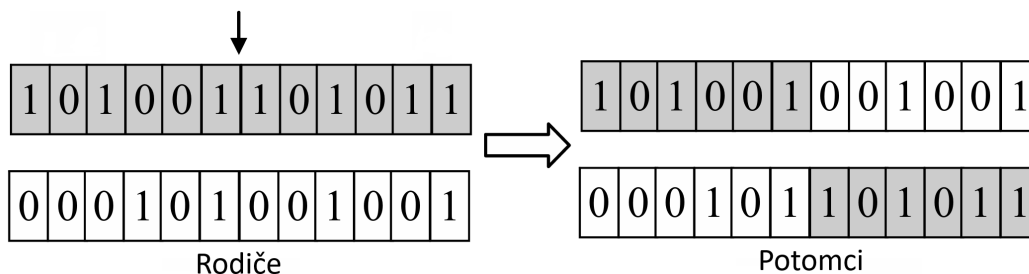
Obrázek 2.1: Příklad mutace. Převzato, upraveno [25].

2.4 Křížení

Operátor křížení má za cíl zvětšení diverzity v rámci populace pomocí kombinace dvou (nebo více) rodičů. Křížení simuluje rozmnožování u živých organismů, kdy potomek zdědí polovinu genů od jednoho rodiče a druhou polovinu genů od druhého rodiče. Na rozdíl od přírodní genetické výměny mezi rodiči je operace křížení v evolučních algoritmech více flexibilní. Nový potomek může zdědit od každého rodiče libovolnou část genetické informace. Nejčastěji se používají tři způsoby křížení.

Jednobodové křížení

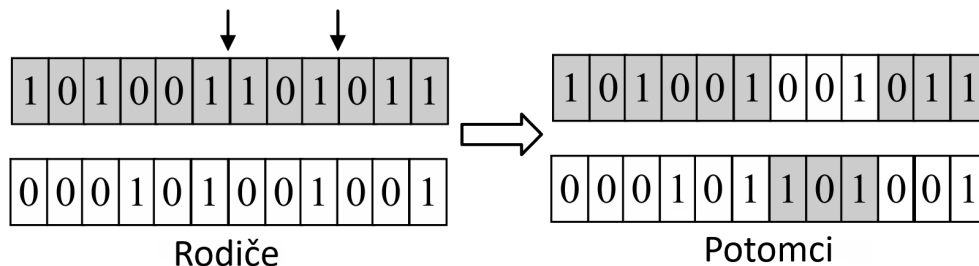
Jak už z názvu vyplývá, jednobodové křížení probíhá v jednom bodě. Náhodně se vygeneruje místo, ve kterém dojde ke křížení. Při uvažování binárních jedinců se vygeneruje jedna celočíselná hodnota, která bude představovat index, ve kterém dojde ke křížení. Potomek vzniká tak, že jeho první část (část po bod křížení) pochází od prvního rodiče a druhá část (část od bodu křížení) pochází od druhého rodiče. Pro názornost je ilustrace na obrázku 2.2



Obrázek 2.2: Příklad jednobodového křížení. Šipka označuje místo křížení. Převzato, upraveno [25].

Vícebodové křížení

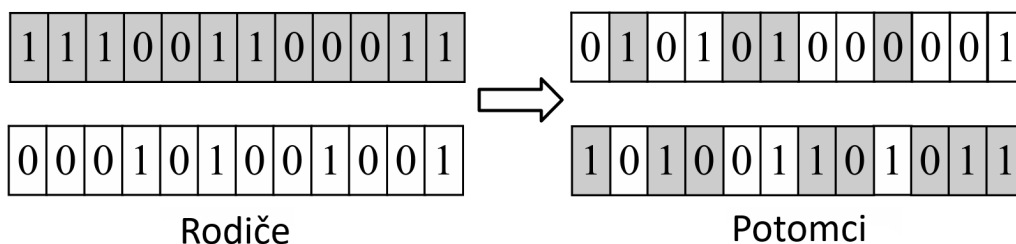
Oproti jednobodovému křížení, křížení vícebodové probíhá mezi více body (většinou dva). Obdobně jako při jednobodovém křížení dojde k náhodnému výběru bodů křížení. V rámci této ilustrace bude uvažováno dvoubodové křížení binárních jedinců. Nejprve se vygenerují dvě celočíselné hodnoty, které opět představují index, kde dojde ke křížení. Potomek vzniká tak, že jeho první část (část po první bod křížení) pochází od prvního rodiče, střední část (část mezi body křížení) pochází od druhého rodiče a poslední část (část od posledního, tedy druhého, bodu křížení) pochází opět od prvního rodiče. Ilustrace je na obrázku 2.3.



Obrázek 2.3: Příklad dvoubodového křížení. Šipky označují místa křížení. Převzato, upraveno [25].

Uniformní křížení

Posledním často používaných způsobem křížení je uniformní křížení. V tomto typu křížení se rozhodování, jestli se hodnota genu použije z prvního, nebo z druhého rodiče, provádí na úrovni jednotlivých genů. Pro každý gen se tedy provádí nezávislý výběr rodiče, jehož hodnota genu se použije. Znázornění uniformního křížení je na obrázku 2.4.



Obrázek 2.4: Příklad uniformního křížení. Převzato, upraveno [25].

2.5 Ukončovací podmínka

Evoluční algoritmus postupně vytváří nové generace jedinců. Algoritmus skončí, jakmile bude splněna ukončovací podmínka. Pro ukončení evolučního algoritmu existují tři základní možnosti. První možností je ukončení algoritmu v případě, že nalezené řešení dosáhlo požadované kvality. Druhou možností je ukončení na základě dosažení maximálního počtu generací. Třetí možnost ukončení nastává v případě, že se kvalita jedinců několik generací nezmění.

2.6 Vícekriteriální evoluční algoritmus

Vícekriteriální evoluční algoritmy řeší optimalizační problémy s více než jedním kritériem. V tomto případě je nutné hledat řešení, která splňují kompromis mezi jednotlivými kritérii. Mezi hlavní přístupy pro vícekriteriální optimalizaci patří:

- **Agregace kritérií:** hlavní myšlenka tohoto přístupu spočívá v kombinaci hodnot všech kritérií do jediné hodnoty, která představuje hodnotu fitness funkce. Tato hodnota se následně používá jako u běžných evolučních algoritmů s jedním kritériem.
- **Omezení:** tento přístup optimalizuje pouze jedno kritérium, ale k „akceptaci“ jedince dojde pouze v případě, že hodnoty ostatních kritérií splňují určité podmínky. Pokud jedinec nesplňuje některé kritérium, je z evoluce diskvalifikován.
- **Relace dominance:** tento přístup hledá Pareto optimální řešení, což znamená, že neexistuje žádné jiné řešení, které by bylo ve všech kritériích alespoň tak dobré a alespoň v jednom kritériu lepší. Pareto optimální řešení jsou mezi sebou navzájem neporovnatelné, protože každé z těchto řešení bude alespoň v jednom kritériu lepší a alespoň v jednom kritériu horší než jiné Pareto optimální řešení. Podrobnější popis Pareto fronty je v sekci 4.1. Pareto optimální řešení se někdy nazývají jako „dominující“ řešení a ostatní řešení jsou nazývána jako „dominovaná“ řešení. NSGA-II je jedním z nejvýznamnějších algoritmů, který používá tento přístup.

Kapitola 3

Neuronové sítě a jejich návrh

3.1 Neuronová síť

Neuronová síť je výpočetní model, který obsahuje řadu programovatelných proměnných (vah), jejichž vhodným nastavením (v procesu učení) je umožněno zajistit požadovaný výpočet. Princip fungování neuronových sítí je odvozen od funkce biologických neuronů v mozku a potažmo fungování celého mozku. Stejně, jak jsou v mozku neurony přiřazeny do určitých skupin, které spolu určitým způsobem interagují, i neurony, které jsou základním kamenem neuronových sítí, jsou seskupeny do vrstev, které jsou jistým způsobem propojeny. Počet neuronů ve vrstvě, počet vrstev a způsob propojení těchto vrstev se nazývá topologie sítě a má zásadní vliv na chování neuronové sítě. Tato kapitola je inspirována knihami [2] a [10] a informace, které jsou v této kapitole obsaženy, jsou z nich převzaty.

Neuron

Neuron je základní stavební jednotka neuronových sítí. Neuron v neuronových sítích je opět inspirován principem funkčnosti biologického neuronu v mozku.

Biologický neuron obsahuje výběžky – dendrity, které slouží jako vstup neuronu. Dále neuron obsahuje výstup, který se nazývá axon. Propojení mezi neurony je tvořeno pomocí synaptických vazeb, kde dochází k propojení axonu jednoho neuronu s dendritem druhého neuronu.

Princip funkčnosti biologického neuronu je modelován následovně – neuron se v závislosti na svých vstupech buď aktivuje, nebo neaktivuje. Biologický neuron je napojen na několik předchozích neuronů, které jsou jeho vstupy, a pokud vstupní stimuly překročí práh excitability, dojde k vytvoření akčního potenciálu (neuron se aktivuje).

Princip funkčnosti umělého neuronu je v principu úplně stejný, pouze došlo k nahrazení biologických procesů za matematický aparát.

Umělý neuron obsahuje n vstupů označených x_1 až x_n . Tento vstup se označuje jako vektor vstupů. Neuron dále obsahuje pro každý vstup jeho váhu označovanou w_1 až w_n . Tyto váhy se nazývají vektor vah. Vektor vstupů a vektor vah musí mít stejnou délku. Neuron navíc obsahuje hodnotu označovanou jako bias b (v české literatuře se může vyskytovat pod názvem předpětí, nebo práh, ale nejčastěji se nepřekládá), která určuje, kde se nachází aktivační hranice. V závislosti na tom, jestli je bias kladná, nebo záporná hodnota, dochází k tomu, že se neuron aktivuje dříve (pro aktivaci stačí menší potenciál), nebo později (pro aktivaci je nutný vyšší potenciál).

Postup výpočtu neuronu má několik kroků. Nejprve dojde k vynásobení každé vstupní hodnoty s k ní příslušnou vahou a následně se tyto dílčí součiny sečtou. Tento výpočet se realizuje jako skalární součin vektoru vstupů s vektorem vah. Následně se k této hodnotě přičte hodnota biasu. Tato hodnota se nazývá vnitřní potenciál neuronu. V dalším kroku se z vnitřního potenciálu vypočítá hodnota aktivační funkce (aktivačním funkcím se věnuje sekce 3.1). Matematický vzorec reprezentující tento výpočet je vypadá takto:

$$a = f \left(\sum_{i=1}^n w_i x_i + b \right) \quad (3.1)$$

kde výstup a představuje hodnotu aktivační funkce. Funkce f představuje nějakou aktivační funkci, vektor w představuje vektor vah neuronu, vektor x představuje vektor vstupů a b představuje bias.

Pro zjednodušení se často bias přidává do vektoru vah na nultou pozici a do vektoru vstupů se na nultou pozici přidá hodnota 1. Díky této úpravě je výpočet zjednodušený – stačí provést skalární součin vektoru vstupů a vektoru vah a následně z této hodnoty spočítat hodnotu aktivační funkce. Tento zjednodušený vzorec vypadá takto:

$$a = f \left(\sum_{i=0}^n w_i x_i \right) \quad (3.2)$$

Aktivační funkce

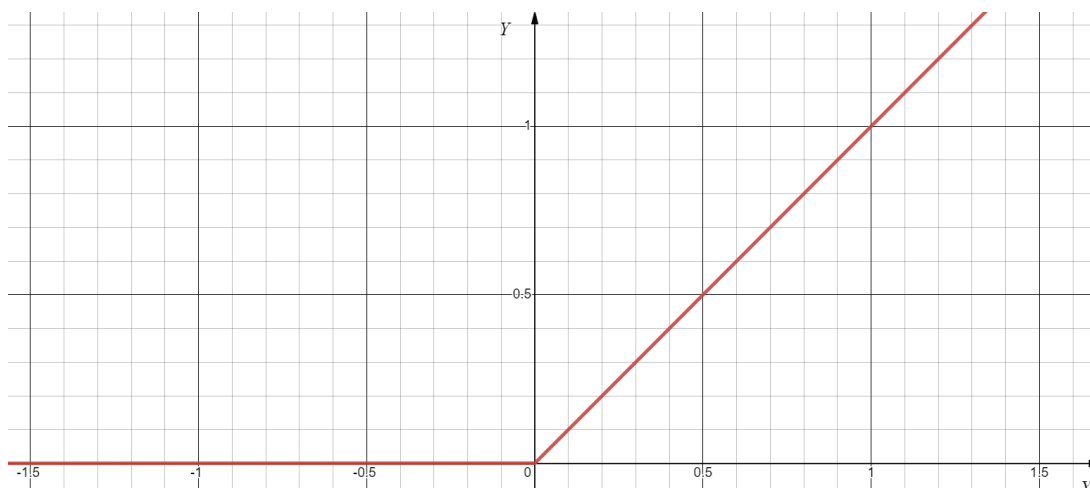
Pro volbu aktivační funkce existuje několik možností. Aktivační funkce musí být nelineární. V případě použití lineární aktivační funkce $f(x) = x$, bude existovat lineární závislost mezi jednotlivými vrstvami sítě. Pokud bude lineární aktivační funkce přítomná v každé vrstvě sítě, bude výstup sítě lineárně závislý na vstupu. V tomto případě dojde k efektivnímu kolapsu celé více vrstvé neuronové do jedné jediné vrstvy. Níže je uvedeno několik aktivačních funkcí, které jsou dnes používány, případně byly používány v minulosti. Informace pro tuto sekci byly převzaty z [1] a [3]. V následujících vzorcích hodnota x označuje vnitřní potenciál neuronu.

Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$

Tato aktivační funkce se používá ve skrytých vrstvách (skrytá vrstva je vrstva, která není vstupní ani výstupní) neuronových sítí. Jeden z hlavních důvodů pro použití této aktivační funkce je její jednoduchost, efektivita výpočtu a rychlost konvergence sítě při učení. Nevýhodou této aktivační funkce je, že může dojít k „úmrtí“ sítě (The Dying ReLU problem) [3].

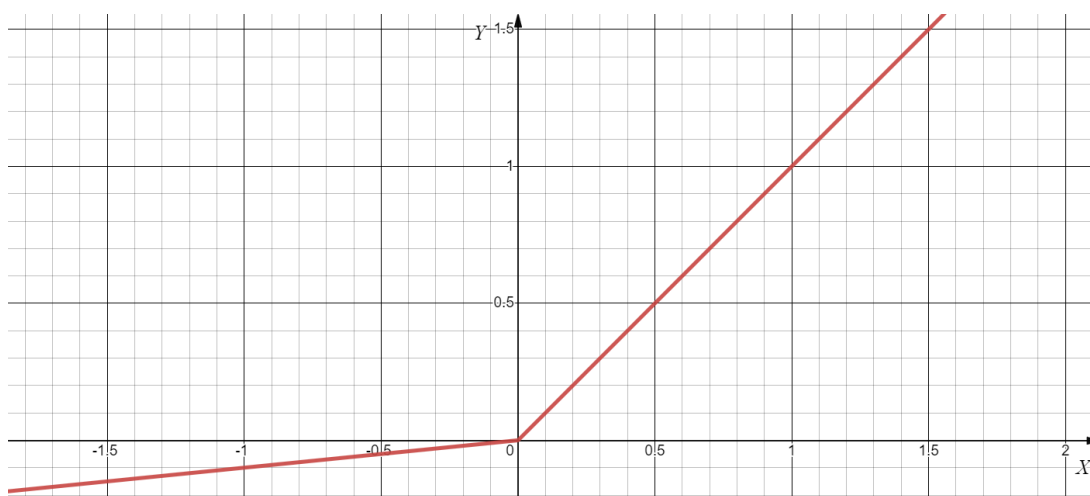
Vzhledem k tomu, že pro záporné hodnoty vnitřního potenciálu neuronu je hodnota aktivační funkce ReLU nulová, bude i gradient nulový. Během úpravy vah pomocí algoritmu zpětného šíření chyby (tento algoritmus je blíže vysvětlen v sekci 3.1) nedojde k úpravě vah neuronů, které nebyly aktivovány. Pokud má neuron nastaveny váhy takovým způsobem, že se neuron nikdy neaktivuje, nedojde nikdy k úpravě jeho vah (tedy nedojde k učení tohoto neuronu) a tento neuron zůstává navždy „mrtvý“. Pokud se v síti nachází velký počet takovýchto neuronů, dochází k „úmrtí“ sítě.



Obrázek 3.1: Průběh funkce ReLU

Tento jev se dá eliminovat pomocí mírné úpravy ReLU funkce. Tato upravená verze se nazývá Leaky ReLU. Principem této úpravy je, že funkce není pro záporné hodnoty nulová, ale záporná.

$$f(x) = \max(\alpha \cdot x, x)$$



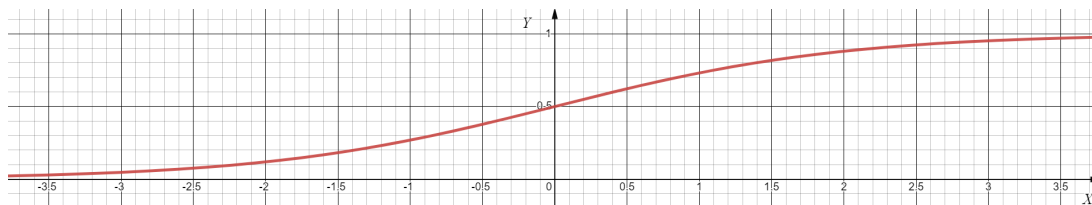
Obrázek 3.2: Průběh funkce Leaky ReLU, parametr $\alpha = 0,1$

α je malé číslo, zpravidla 0,001. Tato aktivací funkce má stejné výhody jako ReLU, ale nezpůsobuje úmrtí sítě.

Sigmoida

$$f(x) = \frac{1}{1 + e^{-x}}$$

Tato aktivací funkce má obor hodnot pouze mezi hodnotami 0 a 1, je tedy vhodná pro reprezentaci pravděpodobnosti. Nevýhodou této funkce je jev označovaný jako miziví gradient [3]. K tomuto jevu dochází v případě, že derivace je velmi blízká nule. Tento jev

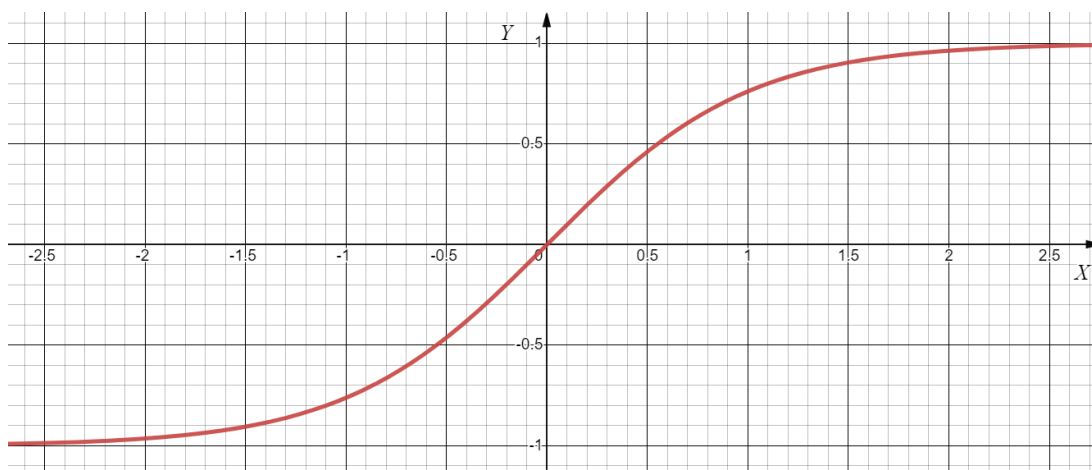


Obrázek 3.3: Průběh funkce Sigmoida

vede na pomalé učení. Pro tuto funkci jsou použitelné hodnoty derivace na definičním oboru od přibližně hodnoty -3 po hodnotu +3.

Hyperbolický tangens

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Obrázek 3.4: Průběh funkce tanh(x)

Tato aktivační funkce je velmi podobná funkci sigmoida. Její obor hodnot je mezi hodnotami -1 a 1 a je vycentrovaný kolem hodnoty 0. Díky tomu je možné výstup chápat jako silně negativní, neutrální a silně pozitivní. Další výhodou je implicitní normalizace hodnot pro následující vrstvy. Stejně jako funkce sigmoida, i hyperbolický tangens trpí problémem mizejícího gradientu.

Softmax

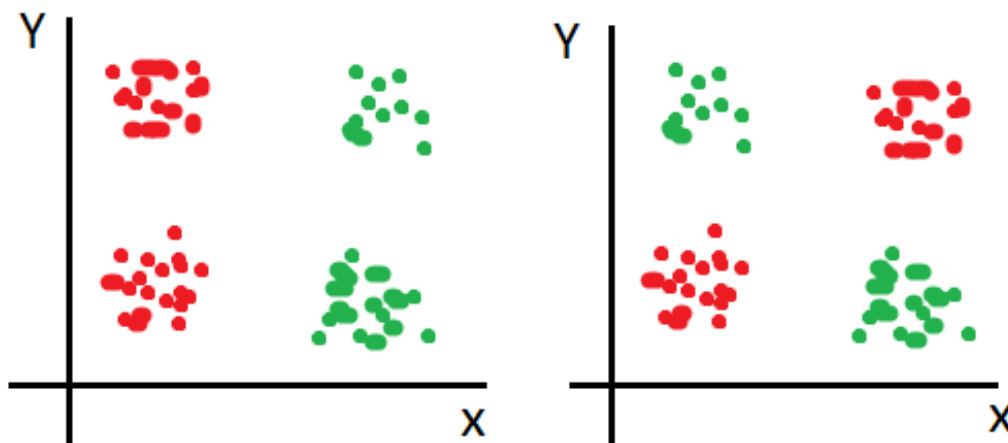
$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Jedná se o zobecněnou funkci sigmoida pro klasifikaci do více tříd. Funkce provádí normalizaci pravděpodobností jednotlivých tříd tak, aby tyto pravděpodobnosti sumovaly do hodnoty 1. Funkce softmax se používá pro klasifikaci do více tříd. Symbol z označuje vektor, který se skládá z n hodnot, které reprezentují n tříd. Tyto třídy prezentují jednotlivé možnosti klasifikace daného vzorku. Hodnota z_i se počítá na základě tohoto vzorce:

$$z_i = \sum_{j=0}^n w_j x_j$$

Lineárně separovatelný problém

Jednotlivý neuron sám o sobě je schopen řešit pouze lineárně separovatelný problém. Následující popis kvůli jednoduchosti a přehlednosti předpokládá 2D prostor a klasifikaci do dvou tříd. Lineárně separovatelný problém je problém, který je možno vyřešit proložením přímkou takovým způsobem, že na jedné straně od přímky jsou pouze hodnoty patřící do třídy A a na druhé straně jsou pouze hodnoty patřící do třídy B. Příklad lineárně separovatelného problému je na obrázku 3.5 vlevo. Příkladem lineárně neseparovatelného problému je XOR, který je zobrazen na obrázku 3.5 vpravo.



Obrázek 3.5: Příklad lineárně separovatelného problému (vlevo) a lineárně neseparovatelného problému (vpravo), pokud je cílem klasifikovat do dvou tříd (zelená, červená).

Učení neuronových sítí

Pro učení klasifikačních neuronových sítí je nutný velký objem dat. Tato data tvoří dvojice vstupní vektor a výsledek. Výsledek v tomto kontextu znamená, do jaké třídy toto konkrétní dato patří. Tyto dvojice tvoří dataset. Pro optimální natrénování sítě je vhodné celkový dataset rozdělit na dvě části – trénovací a testovací. Obvyklé rozdělení je 70 % až 80 % všech dat náleží do trénovacího datasetu, zbylých 20 % až 30 % náleží do testovacího datasetu. Během trénování se používá pouze trénovací dataset a pro vyhodnocení přesnosti sítě se používá testovací dataset.

Učení neuronových sítí probíhá nastavováním vah a biasů neuronů. V současnosti se učení vícevrstevných dopředných neuronových sítí provádí pomocí algoritmu backpropagation (BP). V české literatuře se někdy tento algoritmus nazývá algoritmus zpětného šíření chyby, ale většinou se používá anglický název.

Tento algoritmus je založený na metodě gradientního sestupu (gradient descent), což je optimalizační metoda pro optimalizaci (minimalizaci) chybové funkce s využitím gradientu funkce. S ohledem na využití gradientu je nutné, aby aktivační funkce neuronů v jednotlivých vrstvách byly diferencovatelné. Algoritmus backpropagation distribuuje gradient od výstupní vrstvy ke vstupní. Každý neuron v každé vrstvě si provede úpravu parametrů na základě parciální derivace. Název backpropagation je odvozen od tohoto „zpětného průchodu“ sítí.

Pro použití algoritmu je nutné znát, jaká je chyba sítě. K tomuto účelu se používá chybová funkce (Loss function). Tato funkce je minimalizována pomocí gradientního sestupu. Ten upravuje parametry neuronů na základě záporné hodnoty gradientu. Existuje více variant gradientního sestupu:

Stochastic Gradient Descent (SGD)

Aktualizace vah se provádí pro každý trénovací vzorek zvlášť. Po každém vzorku se aktualizují váhy s ohledem na gradient chybové funkce pouze pro tento jediný vzorek. Mezi výhody patří rychlá aktualizace vah pro každý vzorek a tendence k rychlé konvergenci, zejména v případech s velkým množstvím dat. Jako nevýhoda je nestabilita a výkyvy v průběhu učení.

Batch Gradient Descent (BGD)

Aktualizace vah se provádí až po vyhodnocení celé trénovací sady. Gradient se vypočítá pro celou trénovací sadu naráz. Váhy se aktualizují jednorázově. Mezi výhody patří lepší výpočetní efektivita, zejména pro velká množství dat a stabilnější trénování. Nevýhody jsou velká paměťová náročnost, daná nutností mít v paměti celou trénovací sadu naráz a pomalejší aktualizace vah.

Mini-Batch Gradient Descent (MBDG)

Aktualizace vah se provádí po vyhodnocení malé náhodné podmnožiny (mini-batch) trénovací sady, velikost této podmnožiny se obvykle nastavuje jako hyperparametr. Tento přístup kombinuje SGD a BGD a tedy i kombinuje jejich výhody a nevýhody. Zpravidla se jedná o dobrý kompromis mezi oběma přístupy.

Přetrénování

Problémem při trénování neuronových sítí je tzv. přetrénování (overfitting). Jedná se o jev, kdy se síť místo učení vzorů v datech snaží „zapamatovat“ si jednotlivá data. Důsledkem tohoto jevu je stav, kdy má síť na trénovacích datech velmi vysokou přesnost (může se blížit hranici 100 %), ale na testovacích datech bude přesnost velmi nízká. V tomto případě síť ztratila schopnost generalizace. Pojem generalizace znamená, že síť natrénovaná na určitých datech dokáže správně pracovat i s daty, která nikdy neviděla.

Pro eliminaci přetrénování existuje několik možností. Jednou z možností je už zmíněné rozdělení datasetu na trénovací a testovací dataset. Při trénování se v určitých intervalech provede otestování pomocí testovacích dat. Pokud se chybovost zlepšuje jak na trénovacích, tak na testovacích datech, trénování bude pokračovat. Pokud se ale chybovost na trénovacích datech zlepšuje a chybovost na testovacích datech se zhoršuje, je vhodné trénování ukončit, protože síť začíná být přetrénovaná. Další možností je zavedení dropout vrstev (vícevrstevným neuronovým sítím se věnuje sekce 3.1). Tyto vrstvy během učení náhodně deaktivují určitý počet neuronů (například 10 %). Díky tomu se síť nemůže „spoléhat“ na určité neurony pro detekci určitých vzorů, ale je donucena generalizovat.

Vícevrstvé neuronové sítě

Vícevrstvá síť obsahuje více než jednu výpočetní vrstvu. Tyto vrstvy, které se vyskytují mezi vstupní a výstupní vrstvou, se nazývají skryté vrstvy. Označení skryté vrstvy vychází z toho,

že výpočty, které se v nich provádějí nejsou pro uživatele viditelné. Vícevrstvé neuronové sítě mohou mít různou architekturu. Specifickým typem architektury vícevrstvých sítí je dopředná neuronová síť. Toto označení vychází ze způsobu propojení jednotlivých vrstev, kdy vstupy každé vrstvy jsou připojeny na výstupy předchozí vrstvy.

Základní druhy vrstev

- **Vstupní vrstva:** první vrstva neuronové sítě. Tato vrstva má na starosti přijetí vstupních dat a jejich přenesení do další vrstvy. V této vrstvě se neprovádí žádné výpočty. Počet neuronů v této vrstvě závisí na vstupních datech.
- **Skrytá vrstva:** tyto vrstvy se nachází mezi vstupní a výstupní vrstvou. Zpravidla jich síť obsahuje více. V této vrstvě se provádí samotné výpočty.
- **Výstupní vrstva:** poslední vrstva sítě. Počet neuronů v této vrstvě závisí na řešeném problému (klasifikace, regrese). Aktivační funkce taktéž vyplývá z řešeného problému (sigmoida pro binární klasifikaci, softmax pro klasifikaci do více tříd).

Další druhy vrstev

- **Plně propojená vrstva:** každý neuron v této vrstvě je propojen s každým neuronem v předchozí vrstvě. Tyto vrstvy tvoří základ dopředných neuronových sítí.
- **Konvoluční vrstva:** tato vrstva aplikuje konvoluci na vstupní data, vhodné pro obrázky. Konvoluční neuronové sítě jsou podrobněji popsány v sekci 3.2.
- **Normalizační vrstva:** tato vrstva normalizuje aktivace předchozí vrstvy, což má pozitivní dopad na stabilitu a rychlost učení.
- **Dropout vrstva:** tato vrstva při trénování náhodně deaktivuje některé neurony, což potlačuje přetrénování.

3.2 Konvoluční neuronová síť

Konvoluční neuronové sítě (convolutional neural network – CNN) jsou specifickým typem neuronových sítí pro zpracování dat, které mají strukturu mřížky (grid). Příkladem takových dat mohou být časové řady, které jsou chápány jako 1D mřížka, která obsahuje vzorky v pravidelných intervalech. Pravděpodobně nejčastějším případem jsou ale obrázky, které jsou reprezentovány jako 2D mřížka pixelů. Název těchto sítí vychází z matematické operace konvoluce, která je v těchto sítích použita. Jednoduše řečeno, konvoluční neuronová síť je neuronová síť, která místo klasického maticového násobení vektoru vstupu s vektorem vah (v plně propojených vrstvách) používá konvoluci v alespoň jedné vrstvě [7].

Princip fungování CNN spočívá na základním principu, kdy každá konvoluční vrstva generuje výstup s vyšší úrovní abstrakce než vrstva předchozí [19]. Tyto výstupy se označují jako mapy příznaků (anglicky „feature map“, nebo zkráceně „fmap“) a proces jejich generování se nazývá extrakce příznaků. Mapa příznaků zpravidla obsahuje více kanálů (anglicky „channel“), kde každý kanál odpovídá použití jednoho konkrétního konvolučního filtru, který bývá označován jako jádro (anglicky „kernel“), na vstup.

Každá jedna konvoluce (aplikace konvolučního jádra na vstup) se provádí jako suma přes všechny vstupní kanály a jako výsledek vzniká jediný kanál. Zpravidla se provádí aplikace

několika různých konvolučních filtrů současně v jedné vrstvě. Důvodem pro použití více konvolucí současně je fakt, že jedna konvoluce používající jedno konvoluční jádro je schopna ze vstupu extrahovat pouze jeden druh příznaků, které se ale mohou nacházet na libovolném místě. Zpravidla je žádoucí, aby každá konvoluční vrstva byla schopna extrahovat více různých příznaků naráz.

Pro ilustraci uvažujme stav, kdy máme na vstupu tři kanály (například každý kanál odpovídá jedné barvě RGB) a máme celkem deset konvolučních jader v této vrstvě. Výsledkem bude mapa příznaků, která obsahuje 10 kanálů a každý z těchto výstupních kanálů obsahuje informace ze všech třech vstupních kanálů.

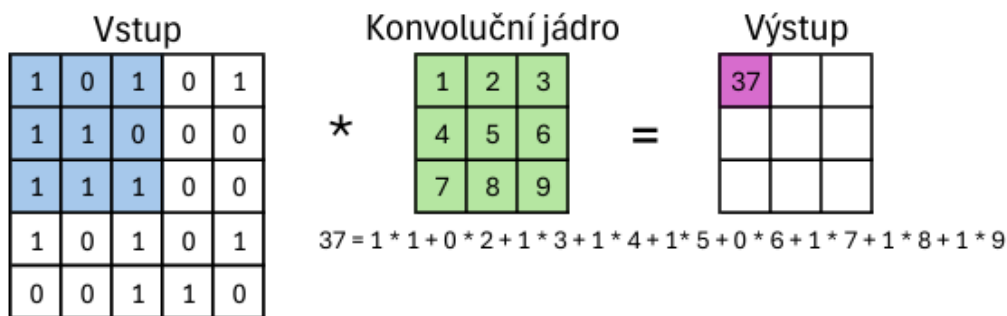
Vzhledem k mírně matoucí terminologii je nutné vysvětlit pojmy konvoluční vrstva (convolution layer) a konvoluční úroveň (convolution stage). Běžná konvoluční vrstva obsahuje tři úrovně [7]. V první úrovni, označované jako konvoluční úroveň, se provádí konvoluce. Výstupem této úrovně je množina lineárních aktivací. V druhé úrovni je každá lineární aktivace použita na vstup nelineární aktivační funkce (nejčastěji ReLU, nebo její derivát). Tato úroveň je někdy označována jako detekční úroveň. Třetí úroveň obsahuje seskupující funkci, která způsobí změnu výstupu. Někdy se může mezi detekční úroveň a seskupující úroveň vložit normalizační úroveň, která má za úkol transformovat distribuce výstupů tak, aby se blížily normálnímu rozdělení.

Konvoluční úroveň

V konvoluční úrovni se provádí operace konvoluce, ta je pro 2D data dána tímto vztahem:

$$I(u, v) = \sum_{i=-a}^b \sum_{j=-c}^d h(i, j)I(u + i, v + j) \quad (3.3)$$

Na obrázku 3.6 je zobrazena konvoluce konvolučního jádra s rozměry 3 x 3 na vstup o velikosti 5 x 5. Krok (anglicky stride) určuje, o kolik se konvoluční filtr posune. Na obrázku je uvažován stride = 1. Kvůli tomu dojde ke zmenšení rozměrů výstupu, na obrázku má výstup rozměry 3 x 3, protože dojde k „ukousnutí“ jednoho pixelu z každé strany, tedy celkově se každá dimenze zmenší o 2. Pokud je toto zmenšení výstupu nežádoucí, je možné vstup rozšířit přidáním nějakých hodnot (zpravidla nula, jedna, nebo duplikace okrajové hodnoty) k okrajům vstupu. Tento proces se označuje jako „Padding“, česky by se dal použít pojem výplň. Padding je zobrazen na obrázku 3.7, kde se jako výplň používá hodnota jedna.



Obrázek 3.6: Příklad jedné operace konvoluce s konvolučním jádrem 3 x 3

1	1	1	1	1	1	1
1	1	0	1	0	1	1
1	1	1	0	0	0	1
1	1	1	1	0	0	1
1	1	0	1	0	1	1
1	0	0	1	1	0	1
1	1	1	1	1	1	1

Obrázek 3.7: Na obrázku jde vidět původní vstup (zobrazen v modré barvě) a padding pomocí hodnoty jedna (zobrazeno oranžově)

Aktivační úroveň

V aktivační úrovni dochází k výpočtu výstupní hodnoty neuronu na základě jeho vnitřního potenciálu. Nejčastěji se jako aktivační funkce používá ReLU, nebo nějaký její derivát. Aktivačním funkcím se věnuje sekce 3.1.

Slučovací úroveň

Ve slučovací úrovni (anglicky „pooling“) dochází k nahrazení výstupu sítě v určité oblasti nějakou statistickou hodnotou z okolí tohoto výstupu. Mezi nejčastější slučovací funkce patří max pooling, který výstup nahradí maximální hodnotou z okolí a average pooling, který výstup nahradí průměrnou hodnotou z okolí. Slučování se zpravidla provádí po nepřekrývajících se blocích [19]. V závislosti na velikosti okolí dochází ke zmenšení výstupu, což vede ke zlepšení výpočetní účinnosti a ke snížení paměťových nároků pro ukládání parametrů. Na obrázku 3.8 je zobrazen příklad fungování max pooling a average pooling na příkladových datech. Každá barva odpovídá jedné slučovací oblasti.

Vstup	Max Pooling	Average Pooling
11	21	12
21	32	13
9	6	3
7	5	3
32		
3		
1		
2		
2		
2		
6		
3		
3		
5		

Obrázek 3.8: Příklad slučování. Každá barva odpovídá jedné slučovací oblasti

3.3 Automatizovaný návrh neuronových sítí

Návrh architektury neuronové sítě značně záleží na znalostech výzkumníka. Toto se týká jak znalostí z oboru neuronových sítí, tak znalostí z cílové domény. Kvůli tomuto je složité pro lidi bez velkých znalostí navrhnout, či modifikovat existující architekturu pro jejich potřeby. Tento problém mají za cíl vyřešit automatizované metody pro návrh neuronových sítí, jako je například Neural architecture search. Zdrojem informací pro tuto kapitolu jsou výzkumy [16] a [22].

Neural architecture search (NAS)

NAS vznikl jako podčást automatizovaného strojového učení (automated machine learning – AutoML). AutoML je proces automatizace všech kroků v procesu strojového učení – od vyčištění dat, přes tvorbu příznaků a jejich výběr až po optimalizaci hyperparametrů a nalezení vhodné architektury. NAS má velký překryv s technikou hyperparameter optimization (HPO), která má za cíl automatizovanou optimalizaci hyperparametrů určitého modelu. NAS je někdy [12] považován za podmnožinu HPO, protože NAS může být chápán jako pouze optimalizace hyperparametrů náležitých určité architektuře. Navzdory tomuto tvrzení jsou techniky používané pro HPO a NAS často velmi odlišné.

Prototypem typického HPO problému je optimalizace různé variace spojitých a kategorie hyperparametrů, jako je například rychlost učení (learning rate), dropout rate, velikost dávky (batch size), aktivační funkce, momentum a normalizační strategie. Oproti tomu se NAS zaměřuje na optimalizaci topologie sítě, což může být komplexnější problém. Topologie je obvykle reprezentována jako orientovaný acyklický graf (directed acyclic graph – DAG), ve kterém jsou uzly nebo hrany chápány operace neuronových sítí.

Přestože je občas možné standardní HPO algoritmy upravit pro NAS, je obvykle rozumnější použít techniky, které jsou navrhovány přímo na použití v NAS. Důvodem je, že většina moderních NAS technik nekončí u klasické black-box optimalizace, ale využívají i prvky specifické pro NAS, jako je například sdílení vah mezi podobnými architekturami pro eliminaci nutnosti trénovat každou architekturu od nuly. NAS je možno realizovat pomocí posilovaného učení, evolučních algoritmů, Bayesovské optimalizace, nebo one-shot metod.

Black-Box optimalizační techniky

Jedna z nejvíce studovaných částí NAS je strategie prohledávání (search strategy). Tato strategie je zodpovědná za nalezení optimální architektury ze stavového prostoru. Strategie prohledávání se dělí na black-box optimalizační techniky a one-shot metody, které jsou popsány v sekci 3.3. Tyto dvě skupiny nejsou striktně disjunktní a existují metody, které mají charakteristiky obou skupin a existují i metody, které nemají charakteristiky ani jedné z těchto skupin.

Black-box optimalizační techniky jsou v dnešní době široce používány a studovány pro jejich dobré výsledky a jednoduchost použití. Obecně tyto techniky mají vyšší výpočetní nároky než one-shot metody, kvůli nutnosti trénovat mnoho architektur od nuly, nezávisle na sobě. Mezi jejich výhody patří robustnost, jednodušší paralelizace, společná optimalizace s jinými hyperparametry a jednodušší adaptace na nové problémy, datasety nebo stavové prostory. Jsou často konceptuálně jednodušší, díky čemuž jsou snazší na implementaci a použití.

Jedna z nejjednodušších možných prohledávacích strategií je prosté náhodné prohledávání (random search), kde jsou architektury vybírány ze stavového prostoru náhodně

a následně jsou plně natrénovány. Na konci běhu algoritmu je jako výsledek vrácena architektura s nejvyšší validační přesností. Navzdory své jednoduchosti a naivnímu přístupu k řešení problému, je tato technika relativně úspěšná [5]. Toto tvrzení obzvláště platí pro velmi dobře navržené stavové prostory, které obsahují velké procento kvalitních architektur. Některé studie ukazují, že náhodné prohledávání není vhodné pro velké a různorodé stavové prostory [4].

Odvozenou strategií je lokální prohledávání (local search). Nejjednodušší formou lokálního prohledávání je iterativní trénování a vyhodnocování všech sousedů dosud nejlepší nalezené architektury. Jako soused se většinou považuje architektura, která se liší právě v jedné operaci nebo hraně. Výzkumy tvrdí, že tato strategie dosahuje dobrých výsledků pro malé [13] i velké [27] stavové prostory.

Další skupina strategií je založena na posilovaném učení (reinforcement learning). Tyto strategie byly nejvíce používány v počátcích moderního NAS. Většina těchto přístupů modeluje architekturu jako sekvenci akcí vygenerovaných kontrolerem. Validací přesnosti této architektury po natrénování je použita jako odměna (reward) pro kontroler pro úpravu parametrů kontroleru. Jako kontroler se zpravidla používá rekurentní neuronová síť (RNN). V dnešní době se strategie založené na posilovaném učení příliš nepoužívají, protože evoluční metody [15] a bayesovská optimalizace [23] dosahují v přímém porovnání lepších výsledků.

V minulosti byly pro automatizovaný návrh a optimalizaci neuronových sítí používány evoluční algoritmy. Tyto algoritmy byly použity pro souběžnou optimalizaci architektury i vah neuronů. V dnešní době se evoluční algoritmy používají pouze pro optimalizaci architektury, pro optimalizaci vah se používá metoda gradientního sestupu spolu s algoritmem backpropagation (popsáno v sekci 3.1).

Evoluční NAS algoritmy iterativně obměňují populaci architektur. V každém kroku je vybrána jedna nebo více architektur jako „rodiče“. Tento výběr je zpravidla založen na validační přesnosti. Z těchto rodičů je pomocí operátorů křížení a mutace vytvořen jeden nebo více potomků. Tito potomci se natrénují a přidávají se do populace, kde nahradí jedince s horšími výsledky.

Jednotlivé evoluční NAS algoritmy se liší způsobem tvorby počáteční populace, způsobem volby rodičů a generováním potomků. Jako počáteční populaci je možno použít nějaké triviální architektury, náhodně vybrat některé architektury ze stavového prostoru, nebo manuálně najít některé architektury, které dosahují dobrých výsledků. Pro výběr rodičů se používají klasické selekce z evolučních algoritmů, jako je turnaj, ruleta nebo prostý výběr nejlepšího jedince. Jedním z úspěšných evolučních algoritmů je regularizovaná evoluce. Ta spočívá v tom, že v každé iteraci je z populace vyřazena architektura, která byla v populaci nejdéle, i přesto, že by se jednalo o architekturu s nejlepšími výsledky. Tato metoda porazila náhodné prohledávání a strategie posilovaného učení a v době svého uvedení dosáhla state-of-the-art výsledků na datasetu ImageNet [15].

One-Shot metody

Problémem raných NAS metod je jejich vysoká výpočetní náročnost, obzvláště pokud nejsou k dispozici žádné techniky pro zrychlení. To je dáno tím, že dochází k iterativnímu výběru architektury ze stavového prostoru a následnému natrénování této sítě. Následně dojde k vyhodnocení kvality sítě a v závislosti na této kvalitě se bude vybírat následující architektura. Vzhledem k tomu, že v průběhu běhu algoritmu může být nutné natrénovat až tisíce jednotlivých sítí od úplného základu, může se výpočetní náročnost pohybovat v řádu tisíců GPU-dnů. Navzdory tomu, že jednotlivé sítě vznikají evolučně, a tedy mezi jednotlivými

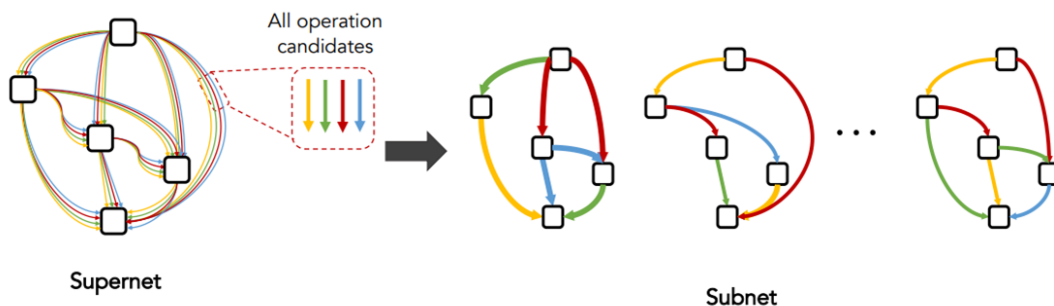
sítěmi jsou určité podobnosti, není možné tento fakt využít během trénování. Každá síť je nezávislá na ostatních, a proto musí být trénována od samotného počátku.

Alternativní technikou jsou one-shot metody, které mají za cíl minimalizovat nutnost trénování každé jedné sítě od nuly. Místo trénování každé architektury zvlášť, tyto metody provedou jediné (odtud název one-shot) natrénování všech architektur ve stavovém prostoru najednou pomocí natrénování nadřazení sítě. Tato nadřazená síť se označuje jako supersít (supernet), nebo hypersít (hypernet).

Hypersít je neuronová síť, která generuje váhy ostatních neuronových sítí. Supersít obsahuje všechny možné architektury ve stavovém prostoru, tyto architektury jsou označovány jako podsítě (subnet). V literatuře se občas pojem supernet používá jako synonymum k „one-shot model“.

Princip supersítě spočívá v tom, že po natrénování supersítě vznikají jednotlivé architektury zděděním vah od příslušných podsítí. Princip tvorby podsítí ze supersítě je zobrazen na obrázku 3.9. Supersítě jsou škálovatelné a výpočetně efektivní díky tomu, že lineární zvýšení počtu kandidátních operací vede na lineární zvýšení výpočetních nákladů během trénování, ale počet podsítí v supersíti se zvýší exponenciálně. Supersít umožňuje natrénovat exponenciální počet architektur za lineární výkonnostní náklady.

Všechny one-shot metody stojí na jednom základním předpokladu – ohodnocení kvality architektury pomocí one-shot přístupu je podobné, jako kdyby byla tato architektura natrénována samostatně, nezávisle na ostatních. To, do jaké míry je tento předpoklad platný, je předmětem mnoha výzkumů. Některé výzkumy předkládají důkazy, že tento předpoklad platí (například [11] a [24]), jiné ukazují, že tento předpoklad neplatí (například [14] a [26]). Z dostupných výzkumů vyplývá závěr, že platnost tohoto předpokladu závisí na návrhu stavového prostoru, technice použité pro trénování one-shot modelu a na samotném použitém datasetu a je složité odhadnout, do jaké míry bude předpoklad platný v konkrétním případě [28].



Obrázek 3.9: Supersít (supernet) se skládá ze všech možných architektur ve stavovém prostoru. Každá architektura je podsít (subnet) v supersíti. [22]

Vyhodnocování výsledků architektur

Porovnávání jednotlivých NAS algoritmů je složitá záležitost. Pro evaluaci existuje mnoho parametrů, kde některé parametry jsou parametry samotného NAS algoritmu a jiné parametry jsou parametry generovaných architektur. Výkonnost NAS algoritmu je odvozena od toho, jak dobré architektury dokáže vygenerovat. Tady vzniká problém, že není jednoduché určit, zda NAS algoritmus A vygeneroval opravdu lepší architekturu než NAS algoritmus B

nebo se jedná pouze o důsledek toho, že horší architektura byla lépe natrénována. Proto je nutné pro správné porovnání NAS algoritmů použít naprosto shodné nastavení trénování.

Velká výhoda NAS algoritmů spočívá v tom, že ve většině případů není nutné znát dopředu dobu, která je k dispozici pro běh algoritmu. Algoritmus může být kdykoliv zastaven a v tomto případě výstup algoritmu bude nejlepší zatím nalezená architektura. S rostoucí délkou běhu algoritmu bude i lepší výsledná síť. Z tohoto důvodu je vhodné pro NAS algoritmus uvádět nejen nejlepší nalezenou architekturu, ale i úspěšnost nalezených architektur během doby běhu algoritmu.

Přestože je NAS převážně doménou klasifikačních problémů (image classification), existují i případy, kdy byl NAS úspěšně použitý pro jiný typ sítě. Příkladem jsou grafové neuronové sítě (GNN) [6], Generative adversarial networks (GANs) [9] a transformery [18].

Kapitola 4

Evoluční návrh CNN pomocí supersítě

Tato kapitola se zabývá popisem návrhu implementace řešení, které bude pomocí evolučních algoritmů schopno generovat konvoluční neuronové sítě ze supersítě. Tato práce je založena na výzkumech [21] a [20]. Autoři zveřejnili zdrojové kódy včetně předtrénovaných modelů a předtrénované supersítě v repozitáři GitHub¹ ². Předtrénované modely vznikly knihovně PyTorch. Z důvodu, že tato práce funkčně navazuje na implementované třídy a metody pro práci se supersítí, musí i tato práce být implementována v knihovně PyTorch.

PyTorch

PyTorch je Python framework pro strojové učení, který je založen na knihovně Torch. Jedná se o open-source software, původně vyvinutý společností Facebook (Meta AI), dnes patřící pod Linux Foundation³.

PyTorch má dva základní případy užití. V první řadě může být využit pro výpočty pomocí tensorů s GPU akcelerací. Druhým případem je práce s hlubokými neuronovými sítěmi (DNN).

Základním prvek je třída Tensor, která uchovává a provádí operace nad homogenními vícerozměrnými poli. Tato třída je podobná třídě ndarray z knihovny NumPy, ale na rozdíl od ní je Tensor možno akcelarovat pomocí platformy CUDA (v případě grafických karet Nvidia) a ROCm (v případě grafických karet AMD).

Dateset

Předtrénované modely byly natrénovány na datasetu ImageNet, který je jedním z nejznámějších a nejpoužívanějších datasetů pro klasifikaci obrázků. ImageNet obsahuje obrázky, které patří do jedné z 1 000 tříd. Je k dispozici 1 281 167 trénovacích obrázků, 50 000 validačních obrázků a 100 000 testovacích obrázků⁴. Vzhledem k velkému počtu obrázků má ImageNet velikost 150 GB, což je pro použití v rámci této práce nepraktické.

¹<https://github.com/facebookresearch/AttentiveNAS>

²<https://github.com/facebookresearch/AlphaNet>

³<https://www.linuxfoundation.org/blog/blog/welcoming-pytorch-to-the-linux-foundation>

⁴<https://www.image-net.org/download.php>

Vzhledem k tomu, že předtrénované modely byly natrénovány na datasetu ImageNet a jsou stavěny na klasifikaci do 1000 tříd, musí i dataset použitý v této práci být založen na ImageNetu a obsahovat data všech 1000 tříd. Z tohoto důvodu není možné použít některý z „klasických“ subsetů ImageNetu, jako je ImageNette, který obsahuje data pouze 10 tříd. Dataset, který splňuje výše uvedené podmínky, je například dataset ImageNet 1000 (mini)⁵. Tento dataset má velikost 8 GB, která je z praktické stránky použitelná v podmínkách této práce. Dataset obsahuje přibližně 39 tisíc obrázků. Z těchto důvodů byl dataset ImageNet 1000 (mini) zvolen jako vhodný dataset pro tuto práci.

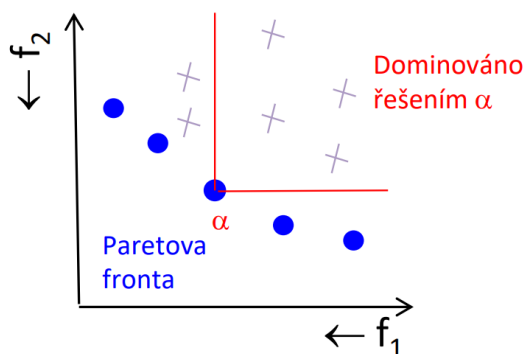
Výpočetní zdroje

Vzhledem k tomu, že modely jsou celkem velké, je nutná akcelerace na grafické kartě (graphics processing unit – GPU). Dalším problémem je velká náročnost na velikost operační paměti grafických karet (Video RAM – VRAM), 8 GB VRAM není dostatečné.

Práce byla prováděna na výpočetním serveru sc-gpu2 (jedná se o GPU server FITu) obsahuje 4 GPU Nvidia RTX A5000, které obsahují 24 GB VRAM. Velikost VRAM je dostatečná a vzhledem k přítomnosti 4 GPU je možné použít paralelizaci pro rychlejší dosažení výsledků.

4.1 Vícekriteriální optimalizace

Při optimalizaci jednoho kritéria (například přesnost sítě) nastávají mezi jednotlivými sítěmi porovnatelné výsledky – nová síť může mít lepší přesnost, stejnou přesnost, nebo horší přesnost. Oproti tomu při optimalizaci více kritérií (například přesnost sítě jako jedno kritérium a počet inferencí, příkon, nebo zpoždění jako druhé kritérium) naráz dochází i ke stavům, které není možno porovnat. K tomuto jevu dojde, pokud nová síť je v jednom kritériu lepší než původní síť, ale v jiném kritériu je horší. Tyto navzájem neporovnatelné řešení tvoří tzv. Pareto frontu. Pareto fronta na obrázku 4.1 kvůli přehlednosti zobrazuje vícekriteriální optimalizaci pro dvě kritéria. Obě kritéria jsou minimalizována. Tato kritéria jsou například chyba a příkon. V tomto případě se dá říct, že čím je řešení více vlevo dole, tím je lepší.



Obrázek 4.1: Příklad Pareto fronty pro dvě kritéria, které se minimalizují, převzato⁶

⁵<https://www.kaggle.com/datasets/iftgotin/imagenetmini-1000>

⁶https://moodle.vut.cz/pluginfile.php/400521/mod_resource/content/1/bin2022_p03.pdf

V rámci této práce bude použita vícekritériální optimalizace se dvěma kritérii. Konkrétně se jedná o přesnost sítě a výpočetní náročnost inference, která je vyjádřena v jednotce FLOPS. Pro výpočetní náročnost inference je v rámci této práce zaveden název „computational cost“, který je označován zkratkou „CC“, jednotka je FLOPS. Důvodem pro volbu těchto dvou kritérií je několik. Prvním z nich je, že přesnost a počet FLOPS (který je proporcionalní k příkonu) jsou dva z nejdůležitějších parametrů neuronových sítí, a proto dává smysl je použít pro vyhodnocení vzniklých řešení. Dalším důvodem je, že tato kritéria jsou použita i v původním výzkumu a je snadné je získat. Přesnost sítě je známá z běhu samotného algoritmu a pro výpočet FLOPS je k dispozici funkce. Tato kritéria nejsou závislá například na konkrétním hardware, na kterém bude daná síť nasazena, na rozdíl od metriky příkon nebo doba inference.

4.2 Reprezentace jedince

Vzhledem k tomu, že tato práce vychází z již existující implementace, je návrh řešení z velké části dán právě touto implementací. Hlavním požadavkem na návrh je zajištění kompatibility evolučního algoritmu, který je předmětem této diplomové práce, s vnitřním fungováním existujícího řešení, které například řeší tvorbu podsítí a jejich vyhodnocení.

Stavový prostor má konečnou velikost a jeho konfigurace je popsána v tabulce 4.1 včetně vysvětlení, co jednotlivé parametry znamenají. Při zvážení všech možností, kterých může jednotlivá podsít nabývat (pronásobení všech možností mezi sebou), se dostaneme k hodnotě přes 927 miliard možností.

Block	Width	Depth	Kernel size	Expansion ratio
Conv	{16, 24}	-	3	-
MBCConv-1	{16, 24}	{1, 2}	{3, 5}	1
MBCConv-2	{24, 32}	{3, 4, 5}	{3, 5}	{4, 5, 6}
MBCConv-3	{32, 40}	{3, 4, 5, 6}	{3, 5}	{4, 5, 6}
MBCConv-4	{64, 72}	{3, 4, 5, 6}	{3, 5}	{4, 5, 6}
MBCConv-5	{112, 120, 128}	{3, 4, 5, 6, 7, 8}	{3, 5}	{4, 5, 6}
MBCConv-6	{192, 200, 208, 216}	{3, 4, 5, 6, 7, 8}	{3, 5}	6
MBCConv-7	{216, 224}	{1, 2}	{3, 5}	6
MBPool	{1792, 1984}	-	1	6
Input resolution:	{192, 224, 256, 288}			

Tabulka 4.1: MBCConv označuje „inverted residual block“ [17]. MBPool označuje „efficient last stage“ [8]. Width určuje všechny možnosti nastavení počtu kanálů, kterých může podsít v této vrstvě nabývat. Depth označuje, kolikrát se tento MBCConv blok bude opakovat. Kernel size určuje, jakou velikost může mít konvoluční jádro. Expansion ratio označuje, kolikrát se zvýší počet výstupních kanálů v rámci inverted residual bloku [17]. Input resolution určuje s jakým rozlišením podsít pracuje. Rozlišení se na rozdíl od ostatních parametrů nenastavuje na každou úroveň, ale na celou síť.

Z tabulky 4.1 je patrné, jaké informace musí jedinec obsahovat, aby bylo možné z něj vytvořit podsít a následně ji evaluovat. Ve vnitřním fungování původního řešení je podsít tvořena z objektů třídy `AttrDict`. Konstruktor této třídy jako parametr bere běžný Python dictionary. Samotná třída `AttrDict` je efektivně ekvivalentní běžnému dictionary, pouze se k jejím atributům přistupuje jiným způsobem. Kvůli tomuto je jedinec reprezentován jako

běžný Python dictionary. Jedinec má fixní velikost, proto všichni jedinci budou mít stejnou strukturu a budou se lišit pouze hodnotami parametrů. Na obrázku 4.2 je zobrazena reprezentace generického jedince pomocí dictionary.

```
# Reprezentace jedince pomocí dictionary
{
    'resolution': resolution,
    'width': width_list,
    'kernel_size': kernel_size_list,
    'expand_ratio': expand_ratio_list,
    'depth': depth_list
}
```

Obrázek 4.2: Na obrázku je znázorněn jedinec jako dictionary

Na obrázku 4.3 je vidět už konkrétní reprezentace jedince. Tento jedinec má rozlišení 224 pixelů, úvodní konvoluční vrstva má 24 kanálů, výstup konvoluce v prvním residual bloku má 16 kanálů. Důležité upozornění je, že `kernel_size`, `expand_ratio` a `depth` mají každý sedm prvků, kde každá hodnota odpovídá jedné residuální vrstvě. Oproti tomu `width` obsahuje hodnot devět, protože kromě sedmi residuálních vrstev obsahuje ještě údaje pro úvodní konvoluci (první hodnota seznamu) a výslednou konvoluci (poslední hodnota seznamu).

```
resolution: 224
width: [24, 16, 32, 32, 72, 112, 192, 216, 1984]
kernel_size: [3, 3, 5, 3, 5, 3, 3]
expand_ratio: [1, 5, 5, 4, 6, 6, 6]
depth: [2, 4, 5, 6, 4, 8, 2]
```

Obrázek 4.3: Na obrázku jde vidět konkrétní jedinec

4.3 Mutace

K mutaci jedince dochází s pravděpodobností p . Pokud k mutaci dojde, provede se m změn. Jakým způsobem probíhal výběr hodnot těchto parametrů je vysvětleno v sekci 6.1. Mutace probíhá jako série několika náhodných pokusů. Nejprve se náhodně vybere jeden jedinec z množiny rodičů. Tento jedinec slouží jako rodič pro tuto mutaci. Po výběru rodiče dojde k „hodu mincí“, jestli k mutaci dojde, nebo ne. Pokud k mutaci nedochází, potomek bude identický jako rodič. Pokud k mutaci dojde, bude se provádět m změn rodiče. První se náhodně vybere atribut, který se bude měnit. Dále se náhodně vybere index seznamu, kde dojde ke změně (s výjimkou atributu `resolution`, který je datového typu integer). Jako poslední dojde ke změně na vybrané pozici za některou náhodně zvolenou povolenou možnost. Povolené možnosti na různých místech jsou popsány v tabulce 4.1. Vzhledem ke způsobu implementace mutace může dojít ke stavu, kdy se sice mutace provede, ale potomek bude identický jako rodič. To se může stát například pokud se za pozici mutace zvolí místo, které obsahuje jedinou možnost, nebo pokud je nově vygenerovaná hodnota stejná, jako původní hodnota rodiče. Dále může dojít k neaktivní mutaci, když se provádí více změn a náhodně budou probíhat dvě (nebo i více) mutací na stejném místě. V tomto případě budou starší hodnoty přepsány novější hodnotou a efektivně jak dojde k „propadnutí“ těchto mutací. V extrémním případě se může stát, že mutace změní hodnotu od rodiče a některá

z následných mutací v témže kole tuto zmutovanou hodnotu vrátí na původní hodnotu od rodiče.

4.4 Křížení

Na rozdíl od mutace ke křížení dochází vždy (pravděpodobnost křížení = 1). Nejprve se z množiny rodičů vyberou dva rodiče, kteří jsou označeni jako matka a otec. Může se stát, že se v obou případech vybere stejný jedinec, potom otec i matka jsou identičtí, ničemu to nevadí. Křížení je implementováno uniformě, pro každý atribut se se náhodně vybírá, jestli se použije hodnota od matky, nebo od otce. V případě, kdy je atribut datového typu seznam, vybírá se celý seznam naráz (všechny hodnoty seznamu jsou buď od matky, nebo od otce). Pokud je otec i matka stejný jedinec, potom i potomek bude identický. Dále se může stát, že všechny atributy potomka se zdědí od jednoho rodiče, potom bude potomek identický s tímto rodičem.

4.5 Ohodnocení jedince

Jedinec je předpis pro vytvoření běžného modelu konvoluční neuronové sítě v PyTorch. Pro ohodnocení jedince je potřeba z něj vytvořit tento model, který se následně vyhodnotí na validačním datasetu. Pro vytvoření modelu, ohodnocení modelu i pro výpočet počtu flops jsou použity funkce z původního řešení.

4.6 Návrh evolučního algoritmu

Původní implementace [21] a [20] používá následující nastavení. Nejprve se pomocí náhodného výběru vybere 512 podsítí (subnet) ze supersítě a provede se výpočet jejich přesnosti na validačním datasetu. Následně se provádí křížení a mutace na 128 nejlepších architekturách. Křížení jedinců probíhá tím způsobem, že nový jedinec zdědí jednotlivé operace od jednoho, nebo druhého rodiče. Pro každou operaci dochází k novému nezávislému výběru. Mutace spočívá v tom, že s určitou pravděpodobností se jedna operace vymění za jinou dostupnou operaci. Z křížení vznikne 128 nových architektur a z mutace rovněž. V jedné iteraci vznikne 256 nových podsítí. Celkem se provádí 20 iterací, což znamená, že na konci běhu evolučního algoritmu vznikne 5 376 sítí. V této práci bude použito jiné nastavení parametrů, než je u původní implementace. Důvodem je vysoká výpočetní a časová náročnost původního nastavení. Pro potřeby této práce se používá nastavení velikosti rodičovské populace na hodnotu 4. Následně se z těchto rodičů náhodným výběrem vybírají dva rodiče pro mutaci, celkem tedy vzniknou dva potomci z mutace. Z populace rodičů se následně vyberou dvě dvojice rodičů (matka a otec) pro křížení, celkem vzniknou dva potomci pomocí křížení. Následně se všichni 4 rodiče a všichni 4 potomci sloučí do nové populace. Každý jedinec z této populace bude ohodnocen. Po ohodnocení se tato populace sloučí s aktuální Pareto frontou (v první iteraci je tato Pareto fronta prázdná) a spočítá se nová Pareto fronta, která se uloží pro použití v další iteraci. Následně se z ohodnocené populace na základě použité fitness funkce deterministicky vyberou 4 nejlepší jedinci jako rodiče pro další generaci. Celkem se provádí 20 iterací. Pro účely statistického vyhodnocení je každý běh spuštěn 20krát.

V této práci jsou použity dva přístupy pro výpočet hodnoty fitness funkce. V prvním přístupu se hodnota fitness funkce počítá z přesnosti sítě a počtu flops sítě. V tomto případě

hodnoty obou kritérií se podílejí na výsledné hodnotě fitness funkce. V druhém případě se používá omezení (constraint). V tomto přístupu se jako hodnota fitness funkce bere přesnost, ale jenom v případě, že hodnota fops je menší než požadovaná hodnota. V opačném případě je jako hodnota fitness funkce zvolena hodnota nula.

Dále pro účely porovnání výsledků je implementováno i náhodné prohledávání (random search). Nastavení parametrů náhodného prohledávání bylo voleno tak, aby pokud možno jeden běh náhodného prohledávání provedl stejný počet ohodnocení jedinců jako evoluční algoritmus.

V poslední řadě byly upraveny i parametry původního algoritmu pro účel porovnání s výsledky dosaženými v této práci. To znamená počet rodičů nastaven na hodnotu 4 a počet potomků vzniklých z mutace a křížení nastaven na hodnotu 2. Vzhledem k odlišnému fungování původního evolučního algoritmu a evolučního algoritmu vytvořeného v rámci této práce, není přímé porovnání mezi těmito algoritmy perfektně férové.

Algoritmus 2 zobrazuje navržený evoluční algoritmus kombinující hodnoty obou kritérií a algoritmus 3 zobrazuje navržený evoluční algoritmus s omezením. Funkce, které jsou zvýrazněny tučně, využívají funkce z původního řešení.

Algorithm 2: Návrh evolučního algoritmu kombinujícího hodnoty obou kritérií

```

P ← generate_parents()
while termination condition not met do
    evaluate_all_parents()
    P_selected ← find_N_genotypes_with_highest_fitness(P)
    P_mutant ← mutate(P_selected)
    P_crossover ← crossover(P_selected)
    P ← P_selected ∪ P_mutant ∪ P_crossover
end

```

Algorithm 3: Návrh evolučního algoritmu s omezením

```

P ← generate_all_parents_with_constraint()
while termination condition not met do
    evaluate_all_parents()
    P_selected ← find_N_genotypes_with_highest_fitness(P)
    P_mutant ← mutate(P_selected)
    P_crossover ← crossover(P_selected)
    P ← P_selected ∪ P_mutant ∪ P_crossover
end

```

Kapitola 5

Implementace

Tato kapitola se zaměřuje na popis implementace navrženého řešení, které je popsáno v kapitole 4. V této kapitole je popsána struktura projektu a způsoby spouštění jednotlivých skriptů.

Projekt je implementován v jazyce Python s využitím knihovny PyTorch. Pro paralelizaci byla použita knihovna `Multiprocessing`, jedná se o systémovou knihovnu, kterou není třeba nijak dodatečně instalovat. Pro izolaci a replikaci běhového prostředí byl použit modul `venv`.

5.1 Struktura projektu

Struktura projektu je z velké části dána původní implementací. Základem toho projektu jsou dva naklonované repozitáře s původní implementací (popsáno v úvodu kapitoly 4). Na tuto implementaci je z pohledu této práce nahlíženo jako na knihovnu, která poskytuje třídy a funkce pro práci s předtrénovanou supersítí (supernet) a pro vyhodnocování jednotlivých podsítí.

Evoluční algoritmy vytvořené v rámci této práce jsou implementované formou Python skriptů. Každý druh evolučního algoritmu je vyčleněn do samostatného souboru. Kromě těchto souborů byl vytvořen i soubor plnicí roli knihovny. Tato knihovna implementuje funkce, které jsou používány v jednotlivých evolučních skriptech.

Dále byly vytvořeny Jupyter notebooky pro vizualizaci a vyhodnocení jednotlivých experimentů, které je popsáno v kapitole 6.

Vytvořené Python skripty

V následujícím seznamu jsou vyjmenované soubory, které byly vytvořeny v rámci této práce. Pro každý zmíněný skript je napsaná i základní úloha daného skriptu.

- `x_evo_base.py`: skript slouží jako knihovna pro další skripty. Implementuje funkce jako je vytvoření rodiče, mutace, křížení a vyhodnocení populace
- `x_evo_constraints.py`: skript implementuje evoluční algoritmus s omezením
- `x_evo_fitness.py`: skript implementuje evoluční algoritmus s kombinací hodnot obou kritérií pro výpočet hodnoty fitness funkce
- `x_random_search.py`: skript implementuje náhodné prohledávání (random search)

5.2 Implementace evolučních algoritmů

Základní princip fungování všech implementovaných evolučních algoritmů je podobný, proto tato sekce popisuje implementaci všech algoritmů naráz. Nejprve dojde k náhodnému vygenerování požadovaného počtu jedinců pomocí funkce `generate_parent`. U algoritmu s omezením musí vygenerovaní rodiče splňovat podmínku na velikost sítě, tato generace je blíže popsána v sekci 5.3.

Následně se provede ohodnocení populace pomocí funkce `evaluate_all_parents`. Tato funkce je podrobněji popsána v sekci 5.3. Toto ohodnocení se používá ze dvou důvodů. Prvním důvodem je použití těchto hodnot pro výběr nejlepších jedinců pro následující iteraci. Druhým důvodem je tvorba Pareto fronty ze vznikajících řešení. Po každé iteraci se provádí tvorba nové Pareto fronty na základě Pareto fronty z předchozí iterace a hodnocení aktuálně vzniklých sítí. Tato Pareto fronta má formát seznamu seznamů, kde každá položka odpovídá Pareto frontě jedné iterace. Tato Pareto fronta jedné iterace je seznam hodnot sítí, které byly v dané iteraci Pareto-optimální. Celkový výstup algoritmu je Pareto fronta poslední iterace.

V dalším kroku se pomocí funkce `find_N_genotypes_with_highest_fitness` vybere N nejlepších sítí, které slouží jako rodiče pro další generaci (s výjimkou náhodného prohledávání, kde se nová generace generuje náhodně). V obou evolučních algoritmech se výběr provádí podle hodnoty fitness funkce, kde vyšší hodnota znamená lepší řešení. Rozdíl mezi oběma přístupy je ve způsobu, jakým se hodnota fitness funkce počítá, a tedy i to, co reprezentuje. Evoluční algoritmus s omezením používá jako hodnotu fitness funkce přesnost sítě. Pokud je velikost sítě menší než omezení, je hodnota fitness funkce nastavena na přesnost, pokud je velikost sítě větší než omezení, pak je hodnota fitness funkce nastavena na hodnotu nula. Tímto dojde k „diskvalifikaci“ sítě z evolučního procesu. Druhý algoritmus využívá pro výpočet hodnoty fitness funkce hodnoty obou kritérií a vhodným způsobem je kombinuje. V rámci této práce byly použity dva způsoby výpočtu hodnoty fitness funkce. První verze pro výpočet hodnoty fitness funkce používá vzorec:

$$\text{fitness} = \text{acc} \times 2 - \frac{\text{CC}}{10} \quad (5.1)$$

Druhý způsob počítá hodnotu fitness podle vzorce:

$$\text{fitness} = (0.5 \times (1 - \text{normalized_CC})) + (0.5 \times \text{normalized_acc}) \quad (5.2)$$

V obou rovnicích `acc` představuje přesnost sítě v procentech na validačních datech, `CC` reprezentuje výpočetní náročnost jedné inference, jednotkou je MFLOPS. Hodnoty `normalized_acc` a `normalized_CC` představují normalizované hodnoty `acc` a `CC`.

Z těchto rodičů vzniknou další jedinci pomocí funkcí `mutate` a `crossover`. Nová populace vznikne sjednocením potomků vzniklých z mutace a křížení s množinou rodičů. Pro náhodné prohledávání se mutace ani křížení neprovádí. Pokud nebyl dosažen zadaný počet iterací, použije se tato nová populace v další iteraci, kde se tato populace použije jako parametr pro funkci `evaluate_all_parents`.

5.3 Paralelizace

Díky tomu, že výpočetní server `sc-gpu2` obsahuje 4 GPU, je možné použít paralelizaci. Prakticky jediná operace, která trvá dlouho, je vyhodnocení jednotlivých podsítí. Všechny ostatní operace oproti tomu trvají zanedbatelně krátkou dobu. Vyhodnocení se provádí na

GPU. Je tedy možné provádět až 4 ohodnocení naráz, protože není možné na jedné GPU provádět víc než jednu evaluaci současně. Protože velikost populace, která se musí ohodnotit obsahuje 8 jedinců, každé GPU provede sekvenční vyhodnocení dvou sítí.

Tato paralelizace je řešena ve funkci `evaluate_all_parents`, která má na starost rozdělení populace mezi jednotlivé procesy a jejich následné spuštění. Funkce nejprve zjistí počet dostupných grafických karet (na serveru `sc-gpu2` je to 4). Tento počet odpovídá počtu vytvořených procesů, protože nemá smysl vytvářet proces, který nebude mít přístup ke „své“ grafické kartě. Následně dojde k rozdělení populace mezi tyto procesy. Populace je reprezentována jako seznam jedinců. Populace je rozdělena mezi jednotlivé procesy sekvenčním způsobem, kdy první dva jedinci (jejich indexy jsou 0 a 1) budou zpracováni prvním procesem. Procesy jsou číslovány podle identifikace GPU, která k procesu patří, pro 4 GPU jsou to hodnoty 0, 1, 2 a 3.

Po vytvoření všech evaluačních procesů, včetně jim příslušných parametrů, dojde ke spuštění těchto procesů. Každý z těchto procesů vykonává funkci `evaluate_parent`. Tato funkce provádí samotnou evaluaci jedinců. Každý proces má k dispozici svoji kopii dat. Procesy mezi sebou nijak nekomunikují a jsou navzájem izolované, což je jak z pohledu náročnosti implementace, tak z hlediska výkonu ideální stav, protože se nemusí řešit případné konflikty při přístupu ke sdíleným proměnným. Každý proces provádí evaluaci mu příslušné podmnožiny populace. Tato podmnožina obsahuje dva jedince. Nejprve dojde na základě jedince k vytvoření modelu (k tomuto je využita metoda `create_model` z původní implementace). Následně se tento model přesune na GPU, která přísluší tomuto procesu. Na GPU se nejprve provede kalibrace modelu a následně se provede samotné vyhodnocení modelu. Pro vyhodnocení je použita funkce `validate_one_subnet` z původní implementace. Pro potřeby implementovaných evolučních algoritmů jsou podstatné pouze hodnoty `acc1` a `flops`. Výsledkem je trojice (tuple) (jedinec, `acc1`, `flops`). Tato trojice se uloží do seznamu výstupů. Protože každý proces vyhodnocuje dva jedince, seznam výstupů bude na konci obsahovat dvě trojice. Jako poslední krok se tento seznam uloží do fronty (queue). Tato fronta je objekt třídy z knihovny `multiprocessing` a slouží pro vrácení výsledků z vytvořených výpočetních procesů.

Po skončení všech evaluačních procesů, funkce `evaluate_all_parents` provede „posbírání“ dílčích výsledků. Tyto výsledky jsou předávány přes už zmíněnou frontu. Každý z dílčích výsledků obsahuje kompletní informace ohledně jedince a jeho ohodnocení, je tedy jedno, jakým způsobem se tyto dílčí výsledky agregují. Jediná důležitá věc je, aby došlo k agregaci všech výsledků, ale na jejich pořadí už nezáleží. Agregované výsledky se skládají do výsledného seznamu. Jakmile jsou data od všech procesů „posbírána“, funkce vrátí tento výsledný seznam jako celkový výsledek evaluace celé populace.

Paralelizace je dále použita v evolučním algoritmu s omezením. V tomto případě je paralelně prováděna generace iniciální populace. Protože generace rodičů probíhá náhodně, není možné zajistit, aby vygenerovaní rodiče splňovali omezení (omezení je například $CC \leq 500$ MFLOPS). Generování rodičů probíhá tím způsobem, že se náhodně vygeneruje jedinec a následně se pomocí knihovní funkce `count_net_flops_and_params` provede výpočet velikosti sítě. Tato funkce se opět provádí na GPU. Pokud tato síť splňuje podmínku, je tento jedinec zvolen jako jeden z iniciálních rodičů, pokud tato síť podmínku nespĺňuje, je vygenerován nový jedinec a cyklus se opakuje.

Vzhledem k tomu, že velikost rodičovské populace je rovna hodně 4, je možné každého rodiče generovat na jednom GPU. Tato paralelizace funguje na stejném principu jako paralelizace při ohodnocování jedinců. Nejprve se zavolá funkce

`generate_all_parents_with_constraint` s parametry kolik rodičů má být vygenerováno (vždy je rovno 4) a jaká je maximální velikost sítě. Následně se vytvoří 4 procesy, kde každý proces vykonává funkci `generate_parent_with_constraint`. Tato funkce v nekonečné smyčce generuje jedince a počítá jejich velikost. Pokud jedinec splňuje podmínku na velikost, tento jedinec se uloží do fronty a proces se ukončí. Až všechny generující procesy skončí, provede se ve funkci `generate_all_parents_with_constraint` vytvoření celkové iniciální populace z dílčích výsledků. Výsledná populace je seznam jedinců, kde každý jedinec je jeden z dílčích výsledků.

5.4 Spouštění projektu

Jak už bylo zmíněno, tato práce používá virtuální prostředí (`venv`). Pro vytvoření toho prostředí je připraven soubor `requirements.txt`, který obsahuje informace o tom, jaké knihovny jsou potřeba a v jaké verzi. Základním požadavkem pro spuštění této práce je mít nainstalovaný Python a nástroj `pip`.

V první řadě je nutné vytvořit virtuální prostředí. K tomu slouží příkaz

```
python -m venv <venvname>
```

Po vytvoření virtuálního prostředí je nutné jej aktivovat. To se dělá tímto příkazem

```
source <venvname>/bin/activate
```

Po aktivaci virtuálního prostředí je nutné nainstalovat všechny požadované knihovny, které jsou specifikované v souboru `requirements.txt`. To se provede následovně

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

Po nainstalování všech knihoven je možné spouštět jednotlivé skripty. Skript při spuštění musí mít zadaný konfigurační soubor jako parametr. Zadáváno přepínačem `--config-file`. V tomto konfiguračním souboru je specifikována například cesta k datasetu, předtrénované síti nebo parametry evolučního algoritmu. Konfigurační soubor pro spouštění evolučních algoritmů je `./configs/x_evo_cfg.yml`. Volitelným přepínačem je `--output-file`. Tento přepínač specifikuje, do jakého souboru se uloží výsledek běhu skriptu. Pokud tento parametr není zadán, výsledek se do žádného souboru nebude ukládat. Ať je výstupní soubor zadán, nebo ne, vždy dochází k výpisu výsledku na standardní výstup. Na standardní výstup se během chodu vypisují informace o průběhu evoluce (například kolikátá iterace běží, jak vypadají jedinci a co zrovna skript provádí).

Připravené shell skripty

Pro spuštění každého Python skriptu byl vytvořen shell skript, který cílový Python skript spustí zadaným počtem běhů a výsledky uloží do zadané cesty.

- **`run_script_constraints.sh`**: spustí algoritmus s omezením s výchozím nastavením. To znamená, že hodnota omezení se bude postupně zvětšovat a bude nabývat každé z těchto hodnot: 500, 700, 900, 1100 a 1300 MFLOPS. Pro každou hodnotu omezení budou provedeny 4 běhy. Při tomto nastavení se provede stejný počet evaluací jako u skriptu `run_script_fitness.sh`, a porovnání mezi nimi bude férové.
- **`run_script_constraints_boxplot.sh`**: spustí algoritmus s omezením s vysokým počtem běhů (30) pro každou hodnotu omezení.

- **run_script_constraints_unlimited.sh**: spustí algoritmus s omezením s „unlimited“ nastavením, které má nastavenou maximální velikost sítě na 1150 MFLOPS a počet běhů na 20. V tomto případě se očekává, že v konfiguračním souboru bude nastaven počet iterací na vyšší hodnotu (například 50 iterací).
- **run_script_fb.sh**: spustí evoluční algoritmus z původní implementace, konfigurace se nastavuje v souboru `./configs/parallel_supernet_evo_search.yml`.
- **run_script_fitness.sh**: spustí algoritmus využívající kombinaci hodnot pro výpočet hodnoty fitness funkce. Výchozí nastavení je 20 běhů.
- **run_script_random_search.sh**: spustí náhodné prohledávání. Výchozí nastavení je 20 běhů.

Vytvořené Jupyter notebooky

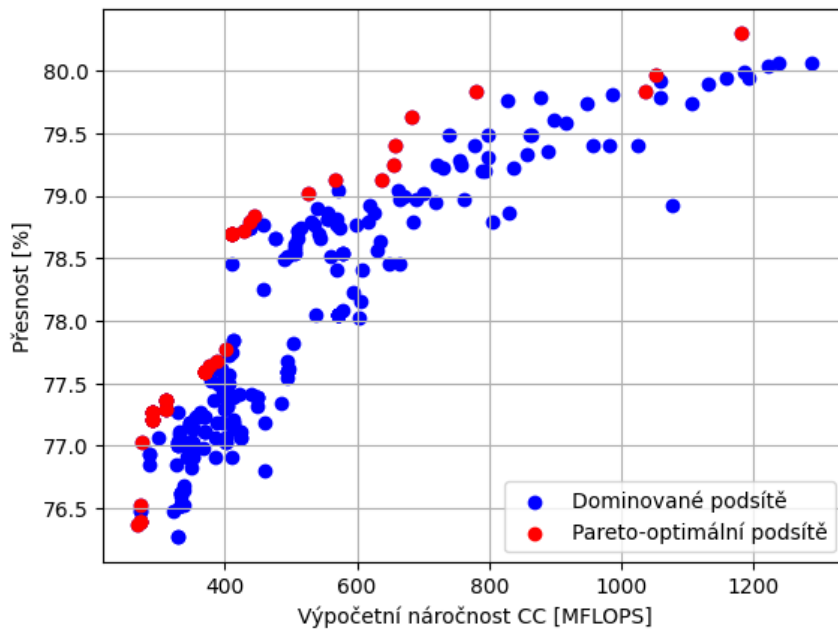
Vytvořené Jupyter notebooky jsou ve složce `results/`. Tato složka obsahuje i výstupy experimentů. Složka je rozdělena do podsložek dle experimentu, který obsahuje. Tyto experimenty jsou blíže popsány v kapitole 6.

- **boxplot**: vliv hodnoty omezení na přesnost generovaných řešení
- **mutation_prob**: pravděpodobnost mutace pomocí algoritmu kombinujícího hodnoty obou kritérií pro výpočet hodnoty fitness funkce
- **mutation_prob_constraint**: pravděpodobnost mutace pomocí algoritmu s omezením
- **mutation_number**: počet mutací jedince
- **fitness**: porovnání dvou použitých fitness funkcí
- **epoch_number**: vliv počtu generací na výsledné řešení
- **different_methods**: porovnání různých algoritmů
- **unlimited_constraint**: porovnání algoritmu s omezením s velkým počtem generací a původního řešení

Jupyter notebooky používají knihovnu `matplotlib` pro vizualizaci dat a knihovnu `scipy` pro statistické vyhodnocení dat. Notebooky pracují s pevně definovanou strukturou dat. Tato struktura dat odpovídá struktuře, kterou generují Python skripty pomocí přepínače `output` a je navíc plně kompatibilní s výstupem spouštěcích shell skriptů.

Každá datová složka obsahuje dvojici souborů pro každé spuštění – soubor `.out`, který obsahuje výstup běhu evolučního algoritmu a soubor `.log`, který obsahuje standardní výstup evolučního algoritmu. Logovací soubor je ignorován a data jsou načítána ze souboru `.out`. Data jsou ve formátu seznam seznamů. Kde vnější seznam obsahuje seznam pro každou iteraci evolučního algoritmu. Tento seznam obsahuje výsledek aktuálně provedené evaluace spojený s globální Pareto frontou. Seznam na posledním indexu tedy obsahuje výstup po poslední iteraci, a tedy výsledek celého běhu. Soubory jsou pojmenovány dle pořadového čísla běhu, tedy například dvojice `0.out` a `0.log` a `1.out` a `1.log`.

Notebooky obsahují funkci pro načítání výsledné (poslední) generace ze všech běhů v dané složce. Dále je připravená funkce pro transformaci těchto dat do dvou seznamů – jeden pro přesnost sítě a druhý pro flops sítě. Další funkce na základě těchto seznamů vytvoří bodový graf (dominované sítě jsou zobrazeny modrou barvou) a vyznačí výsledné Pareto-optimální řešení červeně. Tento graf je zobrazen na obrázku 5.1. Data pro tento graf jsou agregována ze všech běhů algoritmu s daným nastavením. Z každého běhu se použijí hodnoty z poslední iterace, které reprezentují Pareto-optimální řešení. Tato data jsou následně vykreslena do grafu. Graf tedy zobrazuje všechna Pareto-optimální řešení, která vznikla v rámci všech uvažovaných běhů. Dále jsou tyto seznamy použité pro statistické vyhodnocení experimentů. Podrobnější vysvětlení ohledně vyhodnocení experimentů je popsáno v následující kapitole 6.



Obrázek 5.1: Kompromisní řešení mezi přesností klasifikace a FLOPS získaná pomocí algoritmu využívajícího kombinaci hodnot obou kritérií pro výpočet hodnoty fitness funkce. Data byla získána z 20 běhů algoritmu na validační datové sadě. Algoritmus provedl 10 generací v každém běhu, pravděpodobnost mutace je 0.5, počet změn mutace je 3.

Kapitola 6

Vyhodnocení experimentů

Tato kapitola se zabývá popisem provedených experimentů a jejich vyhodnocením. Vyhodnocování experimentů probíhalo na datasetu ImageNet 1000 (mini) (blíže popsáno v sekci 4). Nastavení parametrů evoluce probíhalo experimentálně. Všechny parametry, až na jeden, byly zafixovány a byl spuštěn evoluční algoritmus. S každým nastavením parametru bylo spuštěno 20 běhů algoritmu, aby bylo možné vyvodit statisticky významný závěr. Při nízkém počtu spuštění by rozdílné výsledky mohly vzniknout kvůli náhodné povaze evolučních algoritmů než kvůli rozdílnému nastavení. Po provedení všech běhů s každým nastavením daného parametru bylo vyhodnoceno, které nastavení daného parametru vedlo k nejlepším výsledkům. Po provedení všech experimentů s daným parametrem bylo analogicky prováděno experimentování s dalším parametrem. Po vybrání nejlepších parametrů bylo provedeno porovnání jednotlivých evolučních algoritmů.

Vyhodnocení se provádí pomocí dvou přístupů. Vizuelní vyhodnocení se provádí pomocí grafů, kde jsou do jednoho grafu zaneseny výsledky z jednoho nastavení i výsledky z druhého (případně dalších) nastavení.

Druhým přístupem k vyhodnocování experimentů je statistická metoda. Tady je použita analýza rozptylu (ANOVA). Vzhledem k tomu, že se tato práce zabývá vícekritériální optimalizací (konkrétně dvě kritéria), je porovnáváno každé kritérium zvlášť. Nejprve se provede statistické vyhodnocení přesnosti a následně statické vyhodnocení velikosti sítě (na pořadí nezáleží). Tyto testy říkají, jestli je mezi jednotlivými nastaveními statisticky významný rozdíl, nebo není.

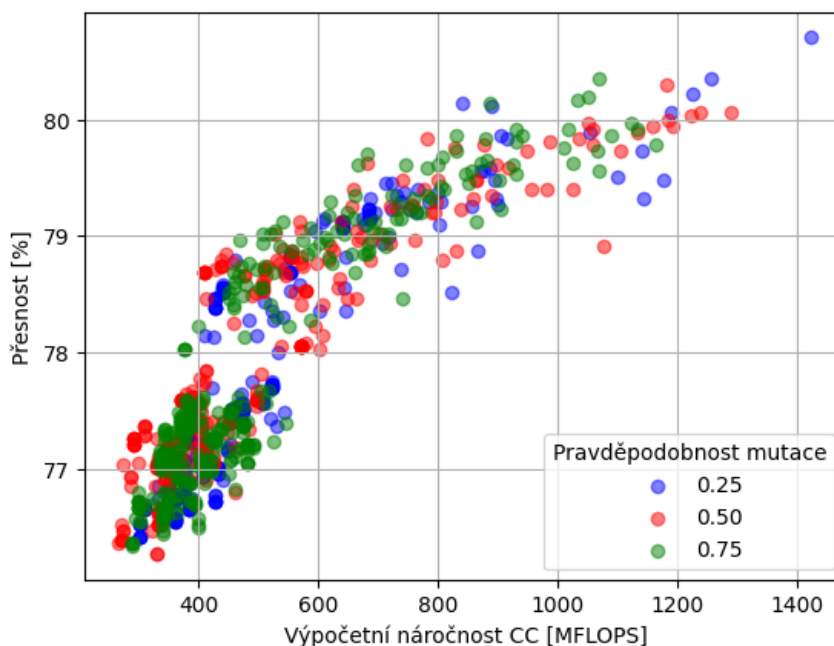
Při statistickém vyhodnocení přesnosti je výsledek testu udáván jako p-hodnota. Pokud je p-hodnota menší než hladina významnosti α (v rámci všech experimentů je hladina významnosti zvolena na hodnotu 0.05), nulovou hypotézu zamítáme v prospěch alternativní hypotézy. Pokud je p-hodnota větší než hladina významnosti α , nulovou hypotézu nezamítáme.

6.1 Výběr parametrů evolučního algoritmu

Nejprve byly provedeny experimenty ke zjištění optimálního nastavení parametru pravděpodobnosti mutace. Na obrázku 6.1 jde vidět vizualizace pomocí grafu. Různé barvy označují různé pravděpodobnosti mutace. Legenda popisuje, která barva odpovídá jaké pravděpodobnosti. Každý graf je zkonstruován s daty ze všech běhů algoritmu s daným nastavením. Z každého běhu se použijí hodnoty z poslední iterace, které reprezentují Pareto-optimální řešení. Každý bod v grafu reprezentuje hodnoty acc a CC pro síť v poslední generaci v rámci

některého běhu algoritmu s daným nastavením. Všechna nastavení parametrů a bližší informace o experimentu jsou vysvětleny v popisu jednotlivých obrázků. Z grafu jde vidět, že mezi jednotlivými nastaveními není patrný velký rozdíl. Největší shluk je kolem bodu 400 MFLOPS a 77 % přesnost. Zbytek Pareto fronty taktéž vypadá velmi podobně. Modrá podsít poblíž hranice 1400 MFLOPS je považována za odlehlou hodnotu.

Pro účely statistického vyhodnocení je nutné stanovit nulovou a alternativní hypotézu. Jako nulová hypotéza je tvrzení, že přesnosti (velikosti) různého nastavení mají stejnou střední hodnotu. Jako alternativní hypotéza je tvrzení, že střední hodnota přesnosti a velikosti dosahuje pro hodnotu pravděpodobnosti mutace = 0.75 lepších výsledků. Tato alternativní hypotéza byla zvolena na základě grafu 6.1.

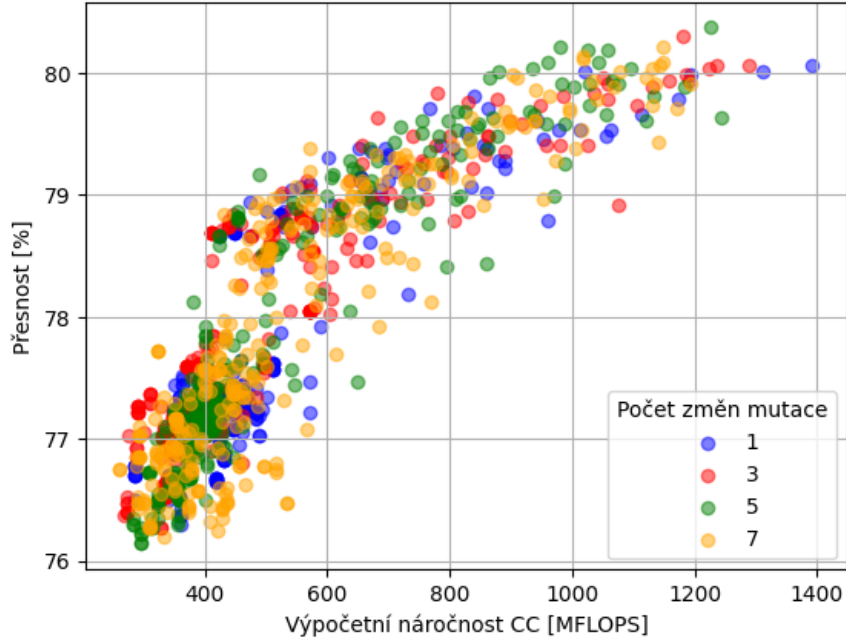


Obrázek 6.1: Kompromisní řešení mezi přesností klasifikace a FLOPS získaná pomocí algoritmu využívajícího kombinaci hodnot obou kritérií pro výpočet hodnoty fitness funkce. Pro výpočet byla použita „Klasická“ varianta fitness funkce. Data byla získána z 20 běhů algoritmu na validační datové sadě. Algoritmus provedl 10 generací v každém běhu, počet změn mutace je 3, pravděpodobnost mutace je zkoumaným parametrem.

Při provedení statistického testu na rovnost středních hodnot přesnosti je p-hodnota rovna 0.225, tedy nulovou hypotézu nezamítáme. Při testu velikosti sítě je p-hodnota rovna 0.295, nulovou hypotézu tedy nezamítáme. Na základě provedených statistických testů není možné říct, že by některé nastavení pravděpodobnosti mutace vedlo ke statisticky lepším výsledkům. Více informací a vizualizací ohledně tohoto experimentu obsahuje soubor `mutation_prob_evaluation.ipynb`.

Dále bylo experimentálně zjištěno nastavení počtu změn při mutaci. Na obrázku 6.2 je vidět graf, který vizualizuje jednotlivé nastavení parametru. Legenda vysvětluje, která barva odpovídá jaké hodnotě parametru. Po vizuální stránce všechny nastavení vypadají velmi podobně. Opět pro všechny nastavení vznikl shluk kolem bodu 400 MFLOPS a 77 % přesnosti a zbylá část Pareto fronty taky vypadá podobně.

Jako nulová hypotéza je stanoveno tvrzení, že střední hodnoty přesnosti a velikosti sítě všech čtyř nastavení parametru jsou stejné. Alternativní hypotéza je tvrzení, že pro hodnotu počet změn = 5 dosahují přesnost, respektive velikost sítě lepších výsledků. Alternativní hypotéza je stanovena na základě grafu 6.2.



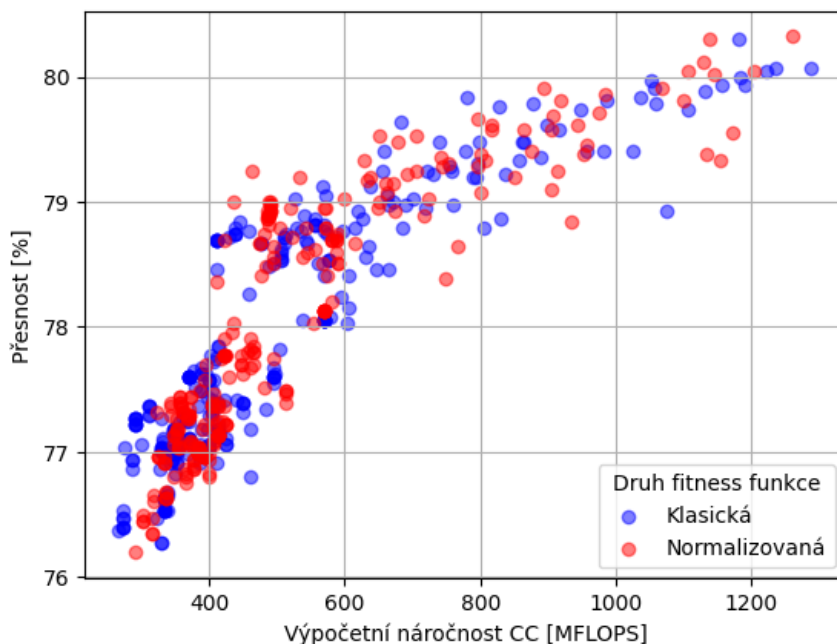
Obrázek 6.2: Kompromisní řešení mezi přesností klasifikace a FLOPS získaná pomocí algoritmu využívajícího kombinaci hodnot obou kritérií pro výpočet hodnoty fitness funkce. Data byla získána z 20 běhů algoritmu na validační datové sadě. Algoritmus provedl 10 generací v každém běhu, pravděpodobnost mutace je 0.5, počet změn mutace je zkoumaným parametrem.

Statistický test na rovnost středních hodnot přesností má p-hodnotu rovnou hodnotě 0.134, nulovou hypotézu nezamítáme. Pro kritérium velikosti sítě dosahuje statistický test hodnoty 0.606, nulovou hypotézu nezamítáme. Obdobně jako u pravděpodobnosti mutace, ani u počtu změn mutace není možné říct, že by některé nastavení vedlo na statisticky lepší výsledky. Více informací a vizualizací ohledně tohoto experimentu obsahuje soubor `mutation_changes_evaluation.ipynb`.

Na základě dosavadních pozorování panovalo podezření, že „problémem“ je způsob, jakým se počítá hodnota fitness funkce. Pro ověření, nebo vyvrácení tohoto podezření byl implementován nový způsob výpočtu hodnoty fitness funkce. Původní verze, v grafu označena jako „Klasická“, počítá hodnotu fitness funkce dle vzorce 5.1. Nově implementovaný výpočet fitness funkce, v grafu označený jako „Normalizovaná“, používá normalizaci hodnot obou kritérií. Celý výpočet je dán vztahem 5.2.

Z grafu 6.3 je vidět velmi podobný stav, jako v předchozích experimentech. Obě fitness funkce tvoří shluk kolem bodu 400 MFLPOS a 77 % přesnost. Zbývá část Pareto fronty obsahuje sítě vzniklé pomocí obou fitness funkcí rovnoměrně rozdělené. Na rozdíl od předchozích experimentů není ani možné na základě grafu stanovit, které nastavení generuje lepší výsledky pro účely stanovení alternativní hypotézy. Z tohoto důvodu je jako alterna-

tivní hypotéza bráno tvrzení, že střední hodnoty si nejsou rovny. Nulová hypotéza je opět tvrzení, že střední hodnoty jednotlivých nastavení si jsou rovny.

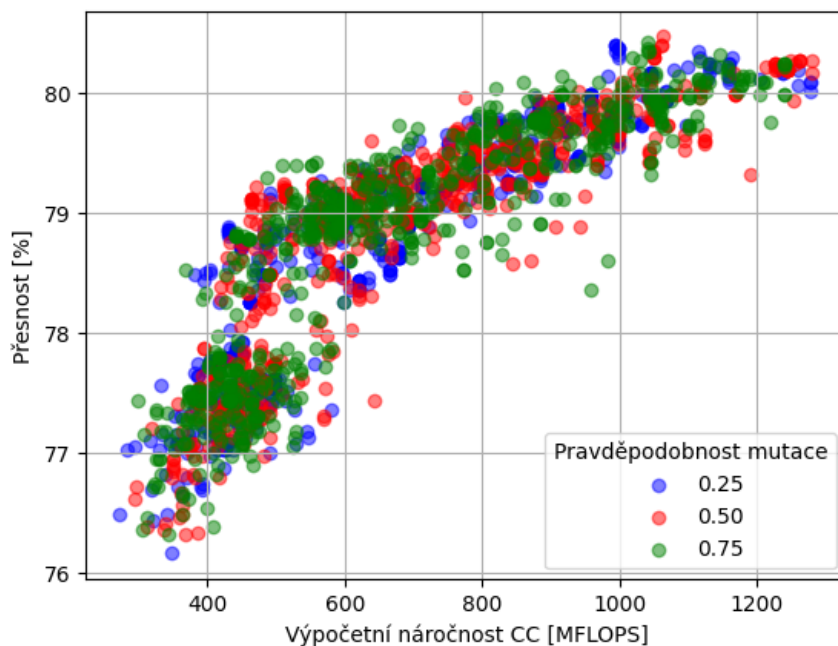


Obrázek 6.3: Kompromisní řešení mezi přesností klasifikace a FLOPS získaná pomocí algoritmu využívajícího kombinaci hodnot obou kritérií pro výpočet hodnoty fitness funkce. Pro výpočet byla použita „Klasická“ varianta fitness funkce. Data byla získána z 20 běhů algoritmu na validační datové sadě. Algoritmus provedl 10 generací v každém běhu, pravděpodobnost mutace je 0.5, počet změn mutace je 3, druh použité fitness funkce je zkoumaným parametrem.

Při provedení statistického testu na rovnost středních hodnot přesností je výsledná p-hodnota rovna 0.536 a pro rovnost středních hodnot rozptylů je p-hodnota rovna 0.591. Dle očekávání ani jednu z nulových hypotéz nezamítáme. Vzorec, podle kterého se provádí výpočet hodnoty fitness funkce nemá statisticky významný vliv na chování evolučního algoritmu. Více informací a vizualizací ohledně tohoto experimentu obsahuje soubor `fitness_evaluation.ipynb`.

S ohledem na předchozí výsledky byl ještě zopakován experiment na pravděpodobnost mutace. V tomto případě ale nebyl použit evoluční algoritmus využívající kombinaci hodnot obou kritérií pro výpočet hodnoty fitness funkce, ale byl použit evoluční algoritmus s omezením. Vzhledem ke způsobu, jakým tento algoritmus funguje, byl zvětšen počet běhů pro každé nastavení parametru na 50 běhů (10 běhů pro každou hodnotu omezení). Tento experiment je zobrazen na obrázku 6.4. Po vizuální stránce je vidět, že graf vypadá velmi podobně, jako v předchozích případech. To znamená, že kolem hodnoty 400 MFLOPS a 77 % přesnosti se opět vyskytuje shluk sítí a zbylá část Pareto fronty taktéž vypadá velmi podobně.

Pro statistické ověření byl opět použit statistický test. Nulová hypotéza je rovnost středních hodnot, alternativní hypotéza je tvrzení, že střední hodnoty si nejsou rovny. Pro kritérium přesnosti vychází p-hodnota na hodnotu 0.216, tedy nulovou hypotézu nezamítáme. Pro kritérium velikost sítě je p-hodnota rovna hodnotě 0.156, nulovou hypotézu taky ne-



Obrázek 6.4: Kompromisní řešení mezi přesností klasifikace a FLOPS získaná pomocí algoritmu s omezením. Data byla získána z 50 běhů (10 pro každou hodnotu omezení) algoritmu na validační datové sadě. Algoritmus provedl 10 generací v každém běhu, počet změn mutace je 5, pravděpodobnost mutace je zkoumaným parametrem.

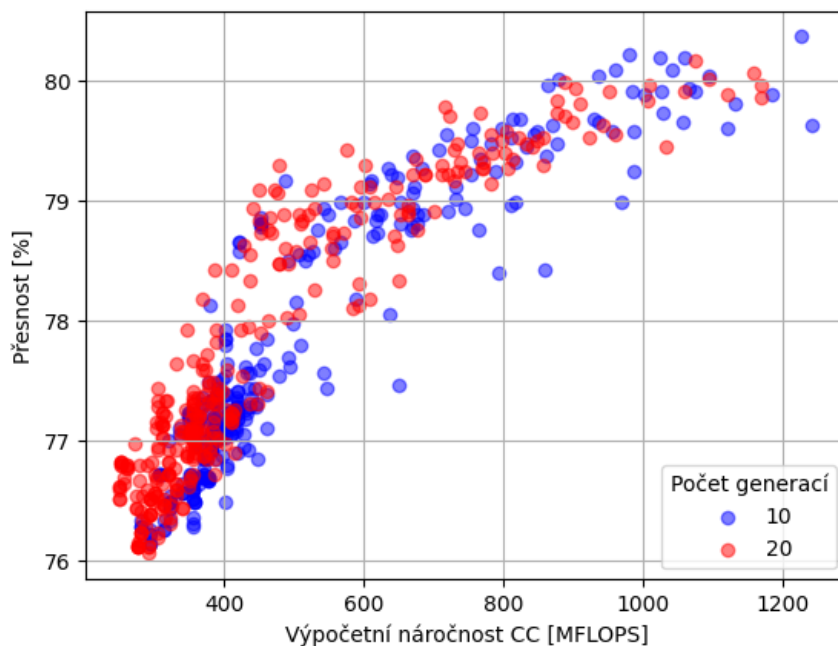
zamítáme. K těmto nižším p-hodnotám dochází pravděpodobně díky většímu počtu běhů, a tedy i samotnému počtu dat. Díky tomu jsou statistické testy přesnější. S ohledem na to, že pro ani jedno kritérium nulovou hypotézu nezamítáme, není možné ze statistického hlediska říct, že by některé nastavení vedlo k lepším výsledkům. Více informací a vizualizací ohledně tohoto experimentu obsahuje soubor `mutation_prob_evaluation_constraints.ipynb`.

Posledním parametrem, který je třeba vyhodnotit je počet generací. Základní předpoklad je, že vyšší počet generací povede na lepší řešení. Pro účely tohoto experimentu bylo zvoleno nastavení 10 generací a 20 generací, vizualizace je na grafu 6.5. Z grafu je patrné, že při nastavení vyššího počtu generací vznikají lepší sítě. Pro potvrzení, nebo vyvrácení tohoto pozorování je opět použit statistický test. Nulová hypotéza je rovnost středních hodnot, alternativní hypotéza je, že vyšší počet generací dosahuje lepších výsledků.

Při statistickém testu kritéria přesnosti sítě je p-hodnota rovna 0.682, nulovou hypotézu tedy nezamítáme. Při vyhodnocení kritéria velikosti sítě je p-hodnota rovna 0.005 a nulovou hypotézu tedy zamítáme v prospěch alternativní hypotézy. Počet generací tedy nemá statisticky významný vliv na přesnost generovaných řešení, ale má statisticky významný vliv na velikost těchto řešení, kde vyšší počet generací vede k menším sítím. Více informací a vizualizací ohledně tohoto experimentu obsahuje soubor `epoch_evaluation.ipynb`.

S ohledem na výše popsané experimenty je složité vybrat optimální nastavení jednotlivých parametrů, protože ze statistického hlediska žádné optimální nastavení neexistuje. Na druhou stranu ani neexistuje žádné nevhodné nastavení, protože každé nastavení parametrů je ze statistického hlediska stejné, co se týká kvality generovaných řešení.

Jediný parametr, který prokazatelně ovlivňuje kvalitu výsledných řešení, je počet generací v rámci jednotlivých běhů. Jestli je výhodné prodloužit dobu běhu algoritmu (zdvojná-



Obrázek 6.5: Kompromisní řešení mezi přesností klasifikace a FLOPS získaná pomocí algoritmu využívajícího kombinaci hodnot obou kritérií pro výpočet hodnoty fitness funkce. Pro výpočet byla použita „Klasická“ varianta fitness funkce. Data byla získána z 20 běhů algoritmu na validační datové sadě. Pravděpodobnost mutace je 0.5, počet změn mutace je 5, počet generací algoritmu v každém běhu je zkoumaným parametrem.

sobení počtu generací povede zhruba na zdvojnásobení doby běhu algoritmu) k tomu, aby vznikaly lepší řešení, není možné v rámci této práce vyhodnotit. Pro vyhodnocení by bylo nutné znát, jaký je k dispozici výpočetní výkon, kolik je k dispozici času a neposlední řadě i to, jak kvalitní sítě jsou potřeba. Pokud například v zadaných podmínkách není nutné mít nejlepší možné řešení, ale stačí jedno z nejlepších řešení, pak pravděpodobně nebude nutné používat vysoký počet generací.

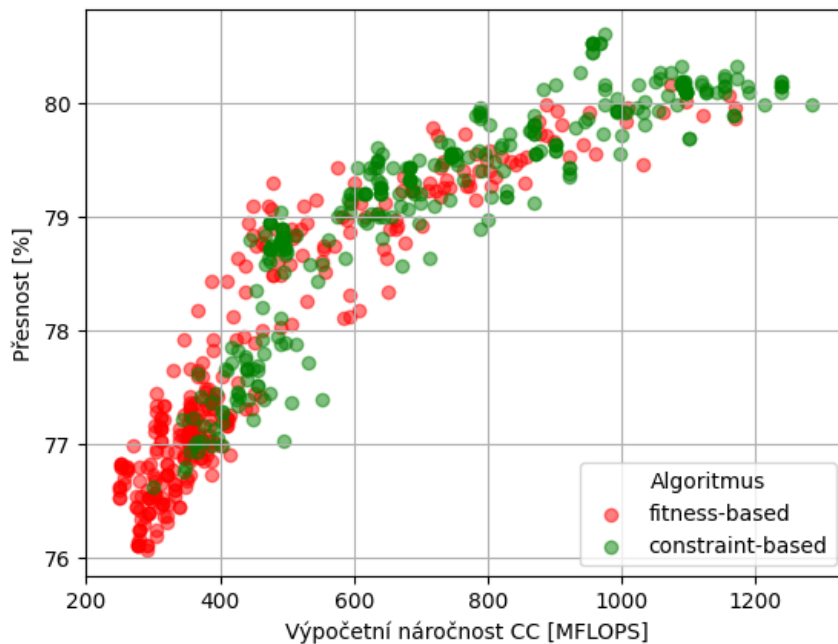
6.2 Porovnání jednotlivých algoritmů

Tato sekce se zaměřuje na popis a vyhodnocení experimentů, které porovnávaly jednotlivé evoluční algoritmy. V rámci této práce byly implementovány dva evoluční algoritmy a náhodné prohledávání (random search). Dále byla použita původní implementace. Byly však u ní upraveny parametry, aby bylo porovnání férovější. Tato evoluční metoda funguje na jiném principu než evoluční metody, které vznikly v rámci této práce, a tedy porovnání není úplně férové.

Pro přehlednost je vyhodnocení provedeno ve dvou skupinách. Nejprve jsou porovnány dva evoluční algoritmy, které vznikly v této práci a následně je lepší z těchto algoritmů porovnán s ostatními.

Na grafu 6.6 je zobrazeno porovnání evolučního algoritmu používající kombinaci hodnot obou kritérií pro výpočet hodnoty fitness funkce (na grafu označenou červeně). Zelenou barvou je zobrazen algoritmus, který pracuje s omezením. Z grafu je patrné, že každý z těchto algoritmů generuje řešení na jiné části Pareto fronty. Algoritmus používající kombinaci hod-

not pro výpočet fitness funkce generuje menší a méně přesné sítě. Oproti tomu algoritmus pracující s omezením generuje větší, ale přesnější sítě.



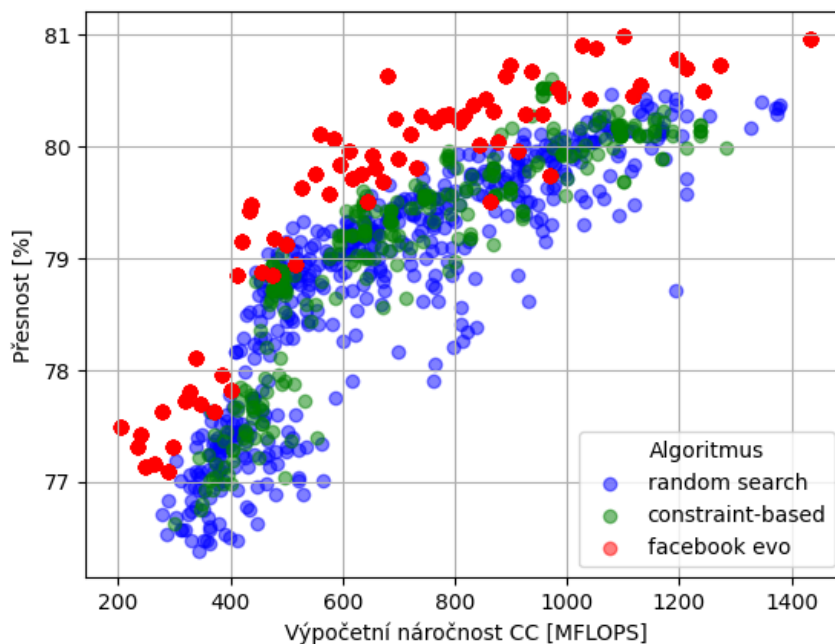
Obrázek 6.6: Kompromisní řešení mezi přesností klasifikace a FLOPS získaná pomocí algoritmu s omezením (constraint-based) a algoritmu s kombinací hodnot pro výpočet hodnoty fitness funkce (fitness-based). Pro výpočet byla použita „Klasická“ varianta fitness funkce. Data byla získána z 20 běhů na validační datové sadě. Oba algoritmy provedly 20 generací v každém běhu, a tedy oba algoritmy měly k dispozici stejný počet evaluací, v každém běhu algoritmu bylo provedeno 164 evaluací. Pravděpodobnost mutace byla 0.5, počet změn mutace byl 5.

Statistický test prokáže, že toto pozorování je statisticky významné. Nulová hypotéza je opět rovnost středních hodnot přesnosti (respektive velikosti) sítě. Alternativní hypotéza je tvrzení, že střední hodnoty si nejsou rovny. Statistický test pro kritérium přesnost dosahuje p-hodnotu v řádu 10^{-53} , kritérium velikost sítě dosahuje p-hodnotu v řádu 10^{-44} . Obě nulové hypotézy rozhodně zamítáme. Statistický test potvrdil, že každý z algoritmů generuje jiná řešení.

Následuje porovnání evolučního algoritmu, který vznikl v rámci této práce, s náhodným prohledáváním a s evolučním algoritmem z původní implementace (facebook evo). Na grafu 6.7 je toto porovnání zobrazeno. Z grafu je patrné, že původní evoluční algoritmus (označen červeně) dosahuje nejlepších výsledků. Evoluční algoritmus s omezením (zobrazen zeleně) generuje řešení na úzké parato frontě. Oproti tomu náhodné prohledávání (modrá) generuje řešení na široké Pareto frontě.

Pojmy úzká a široká Pareto fronta je myšleno, jak blízko (nebo daleko) jsou generovaná řešení od hypotetické Pareto fronty tvořené těmito daty. Je důležité si uvědomit, že každý z těchto bodů byl v poslední iteraci některého běhu Pareto optimální.

Při statistickém testu kritéria přesnosti sítě vychází p-hodnota v řádu 10^{-42} , nulovou hypotézu o rovnosti středních hodnot zamítáme. Pro zjištění, která dvojice hodnot se liší, je použita post-hoc analýza. Tato analýza používá Studentův t-test. Pro kritérium přesnosti



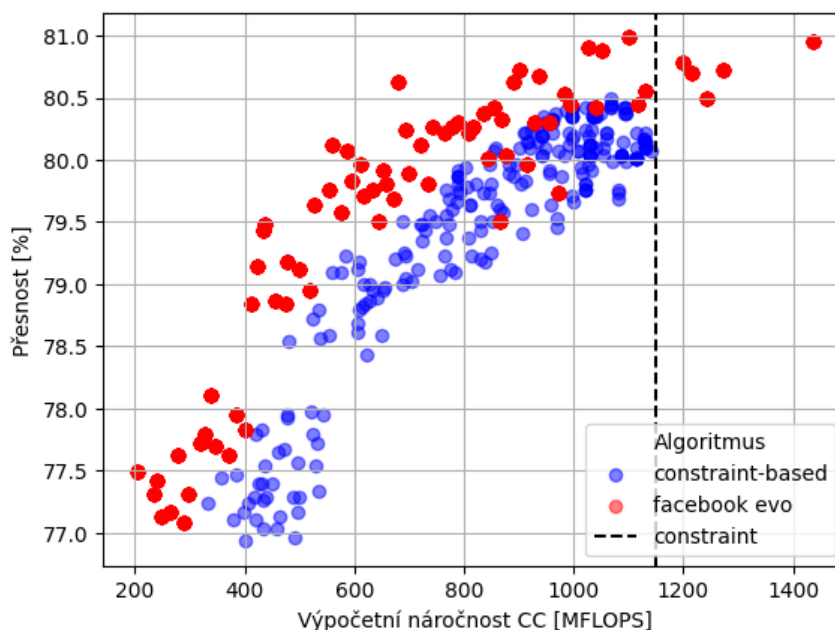
Obrázek 6.7: Kompromisní řešení mezi přesností klasifikace a FLOPS získaná pomocí náhodného prohledávání (random search) a algoritmu s omezením (constraint-based) a evolučního algoritmu z původní implementace (facebook evo). Data byla získána z 20 běhů na validační datové sadě. Všechny algoritmy provedly 20 generací v každém běhu. Pravděpodobnost mutace byla 0.5, počet změn mutace byl 5. Pro algoritmus od facebooku bylo použito výchozí nastavení parametrů, tedy pravděpodobnost mutace 0.2 a jedna změna na mutaci.

se liší všechny tři dvojice. Všechny algoritmy dle přesnosti je možné seřadit v tomto pořadí (od nejpřesnější po nejméně přesný): původní implementace od facebooku, evoluční algoritmus s omezením, který byl vytvořen v rámci této práce a na posledním místě je náhodné prohledávání.

Při statistickém vyhodnocení velikosti sítě má p-hodnota hodnotu 0.028 a nulovou hypotézu o rovnosti středních hodnot zamítáme. Z post-hoc analýzy vychází, že největší sítě jsou generovány evolučním algoritmem s omezením. Pro velikost sítí generovaných náhodným prohledáváním a původním evolučním algoritmem nulovou hypotézu nezamítáme. Tedy největší sítě generuje evoluční algoritmus s omezením, ale další pořadí není možné ze statistického hlediska určit. Více informací a vizualizací týkajících se této sekce obsahuje soubor `method_evaluation.ipynb`.

Další experiment se zaměřuje na to, zda je možné pomocí evolučních algoritmů vytvořených v této práci dosáhnout stejné přesnosti, jako u evolučního algoritmu z původního řešení. Pro účely tohoto experimentu byl použit evoluční algoritmus s omezením, protože generuje větší a přesnější sítě. Hladina omezení byla nastavena na hodnotu 1150 MFLOPS. Počet iterací byl nastaven na hodnotu 50. Celkem bylo provedeno 12 běhů. Vizuelní porovnání tohoto evolučního algoritmu s původním řešením je zobrazeno na grafu 6.8. Z grafu je patrné, že algoritmus s omezením (na grafu modrá barva) dosahuje po stránce přesnosti podobných výsledků, jako původní řešení (na grafu červená barva). Po stránce velikosti sítě jde vidět, že algoritmus s omezením generuje mnohem větší sítě než původní implementace.

Vzhledem k nastavenému omezení na 1150 MFLOPS není možné vygenerovat větší sítě, než je tato hodnota (na grafu znázorněno černou přerušovanou čarou).



Obrázek 6.8: Kompromisní řešení mezi přesností klasifikace a FLOPS získaná pomocí algoritmu s omezením (constraint-based) a evolučního algoritmu z původní implementace (facebook evo). Data byla získána z 12 běhů na validační datové sadě. Algoritmus s omezením provedl 50 generací v každém běhu a tedy provedl více evaluací. Pravděpodobnost mutace byla 0.5, počet změn mutace byl 3. Pro algoritmus od facebooku bylo použito výchozí nastavení parametrů, tedy pravděpodobnost mutace 0.2 a jedna změna na mutaci. Svislá černá přerušovaná čára (constraint) představuje hodnotu omezení.

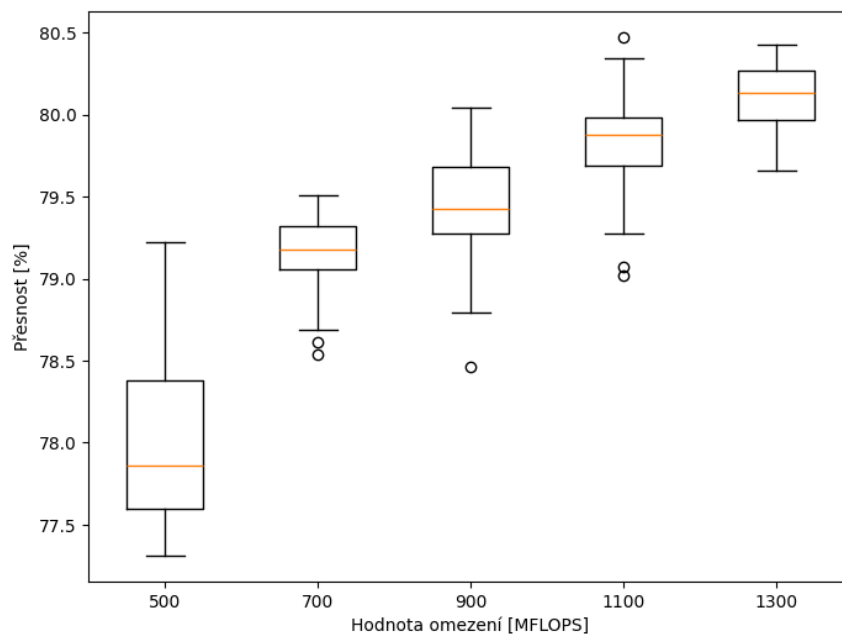
Pro statistické vyhodnocení byla opět použita ANOVA. Jako nulová hypotéza bylo stanoveno tvrzení, že střední hodnoty přesnosti (velikosti sítě) se rovnají. Jako alternativní hypotézy byla stanovena tvrzení, že střední hodnota přesnosti původního řešení je vyšší, respektive, že střední hodnota velikostí sítí generovaných původním řešením je menší.

Pro kritérium přesnosti je p-hodnota rovna hodnotě 0.393, a proto nulovou hypotézu nezamítáme. Pro kritérium velikosti sítě je p-hodnota v řádu 10^{-15} , tedy nulovou hypotézu zamítáme v prospěch alternativní hypotézy.

Ze statistických testů plyne, že po stránce přesnosti mezi oběma metodami není statisticky významný rozdíl. Po stránce velikosti sítě je vidět, že sítě generované původním řešením jsou statisticky významně menší. Tento výsledek odpovídá očekávání, protože evoluční algoritmus s omezením nijak nepenalizuje velikost sítě (za předpokladu, že je menší než omezení). Více informací a vizualizací týkajících se této sekce obsahuje soubor `unlimited_vs_fb.ipynb`.

Poslední experiment se zaměřuje na vyhodnocení přesnosti vzniklých sítí pomocí krabicového grafu (boxplot). Pro účely toho experimentu byl použit algoritmus s omezením s parametry pravděpodobnost mutace 0.5, počet změn mutace 5. Pro každé nastavení omezení bylo provedeno 30 běhů, každý běh měl 10 generací. Z poslední generace každého běhu byl vybrán jedinec s nejvyšší přesností (je uvažováno pouze toto jedno kritérium), který

reprezentuje nejlepšího jedince daného běhu. Tito jedinci jsou agregováni podle hodnoty omezení, během kterého byli vytvořeni. Tento graf je zobrazen na obrázku 6.9. Z grafu je patrné, že algoritmus s omezením generuje přesnější sítě při použití vyšší hodnoty omezení. Tento experiment je zpracován pomocí souboru `boxplot.ipynb`.



Obrázek 6.9: Porovnání nejlepších nalezených jedinců algoritmem s omezením na základě hodnoty omezení. Pro každou hodnotu omezení bylo provedeno 30 běhů s 10 generacemi. Pravděpodobnost mutace je 0.5 a počet mutací je 5.

Kapitola 7

Závěr

Cílem této práce bylo nastudovat možnosti evolučního návrhu a optimalizace konvolučních neuronových sítí se zaměřením na metody používající supersít. Dalším cílem této práce bylo navržení způsobu implementace vícekriteriálního evolučního algoritmu pro automatizovaný návrh konvolučních neuronových sítí s využitím supersítě.

V první kapitole byly popsány evoluční algoritmy včetně popisu stěžejních přístupů, které se v dané problematice používají. Druhá kapitola je zaměřena na popis a návrh neuronových sítí. Nejprve byl vysvětlen princip fungování neuronu, který tvoří základní stavební kámen neuronových sítí. Dále byly popsány aktivační funkce, které se v současnosti v neuronových sítích používají. Dále byl popsán algoritmus zpětného šíření, který se používá pro učení vícevrstvých neuronových sítí. Následně byly popsány konvoluční neuronové sítě. V poslední části této kapitoly byl popsán automatizovaný návrh neuronových sítí se zaměřením na neural architecture search (NAS). V třetí kapitole byl popsán návrh řešení. Nejprve byla vysvětlena volba knihovny a datové sady, následně byl popsán návrh vícekriteriální optimalizace a v poslední části kapitoly byl popsán návrh samotného evolučního algoritmu. Ve čtvrté kapitole byla popsána implementace navrženého řešení. V poslední kapitole byly popsány a vyhodnoceny jednotlivé experimenty.

Na základě provedených experimentů je vidět, že implementované algoritmy jsou schopny nacházet kvalitní řešení. V porovnání s původní implementací od facebooku je vidět, že je zde stále prostor pro zlepšení.

Pro zlepšení by bylo vhodné implementovat a vyhodnotit další metody selekce jedinců, jiné křížení a mutaci. S ohledem na výsledky ze sekce 6.1 by bylo vhodné provést podrobnější experimenty s nastavením parametrů a s vyšším počtem běhů pro zlepšení kvality statistických testů. Problémem toho přístupu by byla velmi vysoká náročnost na výpočetní výkon, a tím i na samotný čas.

Možným důvodem proč většina nastavení generuje podobné výsledky může být samotný tvar stavového prostoru. Pokud například velká část podsítí se nachází poblíž Pareto fronty, pak většina algoritmů bude generovat tato řešení, bez ohledu na nastavení parametrů.

Literatura

- [1] *Activation Functions - ML Glossary documentation* [online]. Sep 2021 [cit. 2024-01-23]. Dostupné z: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html.
- [2] AGGARWAL, C. C. *Neural networks and deep learning: A textbook*. Springer, 2018. ISBN 978-3-319-94463-0.
- [3] BAHETI, P. *Activation functions in neural networks [12 types & use cases]* [online]. May 2021 [cit. 2024-01-23]. Dostupné z: <https://www.v7labs.com/blog/neural-networks-activation-functions>.
- [4] BENDER, G., LIU, H., CHEN, B., CHU, G., CHENG, S. et al. *Can weight sharing outperform random architecture search? An investigation with TuNAS*. 2020. Dostupné z: <https://arxiv.org/pdf/2008.06120.pdf>.
- [5] CHEN, L.-C., COLLINS, M. D., ZHU, Y., PAPANDREOU, G., ZOPH, B. et al. *Searching for Efficient Multi-Scale Architectures for Dense Image Prediction*. 2018. Dostupné z: <https://arxiv.org/pdf/1809.04184.pdf>.
- [6] GAO, Y., YANG, H., ZHANG, P., ZHOU, C. a HU, Y. Graph Neural Architecture Search. In: BESSIERE, C., ed. *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. International Joint Conferences on Artificial Intelligence Organization, červenec 2020, s. 1403–1409. DOI: 10.24963/ijcai.2020/195. Main track. Dostupné z: <https://doi.org/10.24963/ijcai.2020/195>.
- [7] GOODFELLOW, I., BENGIO, Y. a COURVILLE, A. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [8] HOWARD, A., SANDLER, M., CHU, G., CHEN, L.-C., CHEN, B. et al. *Searching for MobileNet V3*. 2019. Dostupné z: <https://arxiv.org/pdf/1905.02244>.
- [9] KOBAYASHI, M. a NAGAO, T. A Multi-objective architecture search for generative adversarial networks. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: Association for Computing Machinery, 2020, s. 133–134. GECCO '20. DOI: 10.1145/3377929.3390004. ISBN 9781450371278. Dostupné z: <https://doi.org/10.1145/3377929.3390004>.
- [10] KRIESEL, D. *A Brief Introduction to Neural Networks*. 2007. Dostupné z: [availableathttp://www.dkriesel.com](http://www.dkriesel.com).

- [11] LI, L., KHODAK, M., BALCAN, M.-F. a TALWALKAR, A. *Geometry-Aware Gradient Algorithms for Neural Architecture Search*. 2021. Dostupné z: <https://arxiv.org/pdf/2004.07802.pdf>.
- [12] LI, L. a TALWALKAR, A. *Random Search and Reproducibility for Neural Architecture Search*. 2019. Dostupné z: <https://arxiv.org/pdf/1902.07638.pdf>.
- [13] OTTELANDER, T. D., DUSHATSKIY, A., VIRGOLIN, M. a BOSMAN, P. A. N. *Local Search is a Remarkably Strong Baseline for Neural Architecture Search*. 2020. Dostupné z: <https://arxiv.org/pdf/2004.08996.pdf>.
- [14] POURCHOT, A., DUCAROUGE, A. a SIGAUD, O. *To Share or Not To Share: A Comprehensive Appraisal of Weight-Sharing*. 2020. Dostupné z: <https://arxiv.org/pdf/2002.04289.pdf>.
- [15] REAL, E., AGGARWAL, A., HUANG, Y. a LE, Q. V. *Regularized Evolution for Image Classifier Architecture Search*. 2019. Dostupné z: <https://arxiv.org/pdf/1802.01548.pdf>.
- [16] REN, P., XIAO, Y., CHANG, X., HUANG, P.-y., LI, Z. et al. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *ACM Comput. Surv.* New York, NY, USA: Association for Computing Machinery. May 2021, sv. 54, č. 4. DOI: 10.1145/3447582. ISSN 0360-0300. Dostupné z: <https://doi.org/10.1145/3447582>.
- [17] 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.
- [18] SU, X., YOU, S., XIE, J., ZHENG, M., WANG, F. et al. *ViTAS: Vision Transformer Architecture Search*. 2021. Dostupné z: <https://arxiv.org/pdf/2106.13700>.
- [19] SZE, V., CHEN, Y.-H., YANG, T.-J. a EMER, J. *Efficient Processing of Deep Neural Networks: A Tutorial and Survey*. 2017. Dostupné z: <https://arxiv.org/abs/1703.09039>.
- [20] WANG, D., GONG, C., LI, M., LIU, Q. a CHANDRA, V. AlphaNet: Improved Training of Supernet with Alpha-Divergence. *ArXiv preprint arXiv:2102.07954*. 2021. Dostupné z: <https://arxiv.org/pdf/2102.07954.pdf>.
- [21] WANG, D., LI, M., GONG, C. a CHANDRA, V. AttentiveNAS: Improving Neural Architecture Search via Attentive Sampling. *ArXiv preprint arXiv:2011.09011*. 2020. Dostupné z: <https://arxiv.org/pdf/2011.09011.pdf>.
- [22] WHITE, C., SAFARI, M., SUKTHANKER, R., RU, B., ELSKEN, T. et al. *Neural Architecture Search: Insights from 1000 Papers*. 2023. Dostupné z: <https://arxiv.org/pdf/2301.08727.pdf>.
- [23] YING, C., KLEIN, A., REAL, E., CHRISTIANSEN, E., MURPHY, K. et al. *NAS-Bench-101: Towards Reproducible Neural Architecture Search*. 2019. Dostupné z: <https://arxiv.org/pdf/1902.09635.pdf>.

- [24] YU, K., RANFTL, R. a SALZMANN, M. *How to Train Your Super-Net: An Analysis of Training Heuristics in Weight-Sharing NAS*. 2020. Dostupné z: <https://arxiv.org/pdf/2003.04276.pdf>.
- [25] YU, X. a GEN, M. *Introduction to Evolutionary Algorithms*. Springer, leden 2010. ISBN 978-1-84996-128-8.
- [26] ZELA, A., SIEMS, J. a HUTTER, F. *NAS-Bench-1Shot1: Benchmarking and Dissecting One-shot Neural Architecture Search*. 2020. Dostupné z: <https://arxiv.org/pdf/2001.10422.pdf>.
- [27] ZELA, A., SIEMS, J., ZIMMER, L., LUKASIK, J., KEUPER, M. et al. *Surrogate NAS Benchmarks: Going Beyond the Limited Search Spaces of Tabular NAS Benchmarks*. 2022. Dostupné z: <https://arxiv.org/pdf/2008.09777.pdf>.
- [28] ZHANG, Y., LIN, Z., JIANG, J., ZHANG, Q., WANG, Y. et al. *Deeper Insights into Weight Sharing in Neural Architecture Search*. 2020. Dostupné z: <https://arxiv.org/pdf/2001.01431.pdf>.

Příloha A

Obsah přiloženého paměťového média

Na přiloženém paměťovém médiu se nacházejí všechny zdrojové soubory použité v rámci této práce. Na médiu jsou přiloženy i všechny výsledky, které byly dosaženy. K těmto výsledkům jsou i přiloženy skripty pro jejich vyhodnocení. Složka `src/` obsahuje zdrojové soubory, které se v rámci práce používaly. Složka `results/` obsahuje výstupní data experimentů včetně skriptů pro jejich vyhodnocení. Soubor `doc.pdf` obsahuje text této práce. Soubor `README` obsahuje návod pro práci s implementovaným projektem.

Adresářová struktura paměťového média:

```
root
|- src/
|- results/
|- doc.pdf
|- README
```