



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 NEURONOVÝCH SÍTÍ

EVOLUTIONARY DESIGN OF NEURAL NETWORKS.

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

JAN KASTNER

VEDOUCÍ PRÁCE

SUPERVISOR

prof. Ing. LUKÁŠ SEKANINA, Ph.D.

BRNO 2024

Zadání bakalářské práce



154226

Ústav: Ústav počítačových systémů (UPSY)
Student: **Kastner Jan**
Program: Informační technologie
Název: **Evoluční návrh neuronových sítí**
Kategorie: Umělá inteligence
Akademický rok: 2023/24

Zadání:

1. Seznamte se s problematikou neuronových sítí (NN), evolučních algoritmů (EA) a využitím EA pro návrh a optimalizaci NN.
2. S využitím EA navrhnete metodu automatizovaného návrhu architektury NN, která bude řešit zvolený problém z oblasti klasifikace. Volbu problému konzultujte s vedoucím práce.
3. Navržené řešení implementujte ve zvoleném prostředí. Můžete využít existující nástroje pro učení NN.
4. Proved'te sadu experimentů pro různá nastavení navržené implementace.
5. Experimentálně vyhodnot'te účinnost EA a kvalitu navržených NN.
6. Zhodnot'te dosažené výsledky a diskutujte možná pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 a 2 zadání.

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: 9.5.2024
Datum schválení: 30.10.2023

Abstrakt

Tato práce je věnována implementaci metody pro řešení problémů v oblasti automatizovaného návrhu architektury konvolučních neuronových sítí (CNN). Optimalizace dvou základních a často protichůdných charakteristik, počtu parametrů a kvality klasifikace CNN, je prováděna pomocí vícekritériálního optimalizačního genetického algoritmu (NSGA-II). Pro zakódování tohoto problému je využita technika kartézského genetického programování (CGP), která umožňuje reprezentaci široké škály architektur CNN a současně lze parametrizací vhodně omezit prohledávaný prostor. Experimenty byly prováděny na datasetu MNIST za účelem pochopení vlivu velikosti populace na kvalitu výsledného řešení. Z výsledků experimentů je také patrné, že kvalita nalezených architektur dokáže konkurovat již etablovaným modelům. Jedná se tedy o alternativní přístup, který v porovnání s manuálním návrhem nevyžaduje lidskou intervenci.

Abstract

The thesis deals with the implementation of a problem-solving method for the automated design of convolutional neural networks (CNN) architectures. The optimization of two fundamental and often conflicting characteristics, the number of parameters and the quality of CNN classification, is performed using a multi-criteria optimization genetic algorithm (NSGA-II). To encode this problem, the Cartesian genetic programming (CGP) technique is used, which enables the wide range of CNN architectures to be represented, and at the same time, the searched area can be appropriately limited by parameterization. Experiments were performed on the MNIST dataset to understand the effect of population size on the quality of the resulting solution. It is also evident from the results of the experiments that the quality of the architectures found can compete with already established models. This is therefore an alternative approach that does not require human intervention compared to manual design.

Klíčová slova

neuroevoluce, evoluční algoritmy, kartézské genetické programování, optimalizace, hluboké neuronové sítě, konvoluční neuronové sítě, strojové učení

Keywords

neuroevolution, evolutionary algorithms, cartesian genetic programming, optimization, deep neural networks, convolutional neural networks, machine learning

Citace

KASTNER, Jan. *Evoluční návrh neuronových sítí*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Evoluční návrh neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou 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.

.....

Jan Kastner
8. května 2024

Poděkování

Rád bych poděkoval panu prof. Ing. Lukáši Sekaninovi, Ph.D. za odborné vedení mé bakalářské práce. Jsem vděčný za jeho trpělivost, vstřícnost a cenné rady, které mi poskytl. Vážím si veškerého času, který ke zhotovení této práce věnoval.

Obsah

1	Úvod	3
2	Evoluční algoritmy	5
2.1	Obecný evoluční algoritmus	5
2.2	Genetické programování	7
2.3	Kartézské genetické programování	11
2.4	Vícekritériální optimalizační genetický algoritmus	13
3	Neuronové sítě	17
3.1	Perceptron	17
3.2	Dopředná neuronová síť	19
3.3	Chybová funkce	20
3.4	Výstupní vrstva	20
3.5	Skryté vrstvy	22
3.6	Algoritmus zpětného šíření chyby	23
3.7	Konvoluční neuronová síť	24
4	Evoluce umělých neuronových sítí	31
4.1	Evoluce vah	31
4.2	Automatický návrh architektury	31
4.3	Evoluce algoritmů učení	36
5	Návrh řešení	37
5.1	Zakódování jedince	37
5.2	Sdílení parametrů	40
5.3	Evoluční algoritmus	42
6	Implementace	44
6.1	Pytorch	44
6.2	Struktura implementovaného řešení	44
6.3	Problém mizejícího gradientu	48
7	Experimenty	50
7.1	Datová sada	50
7.2	Nastavení parametru modelu	51
7.3	Výsledky experimentů	51
7.4	Analýza vlivu velikosti populace na kvalitu výsledného řešení	52
7.5	Zhodnocení výsledků	60

7.6 Pokračování práce	60
8 Závěr	63
Literatura	64

Kapitola 1

Úvod

Algoritmy inspirované přírodou se staly preferovaným přístupem pro řešení mnoha složitých výpočetních problémů. Tyto algoritmy se nezaměřují pouze na nalezení globálně optimálního řešení, ale spíše na nalezení řešení, které je dostatečně dobré pro daný účel. Mezi ně patří neuronové sítě, které čerpají inspiraci z fungování síťové struktury neuronů v mozku. Historicky prošly neuronové sítě obdobím pozitivních očekávání, ale i obdobím skepse, spojeným s poklesem zájmu o tuto oblast výzkumu. Před přibližně deseti lety však opět získaly na popularitě, především díky hernímu průmyslu, který vyžadoval rostoucí výpočetní kapacitu, což vedlo k rozvoji grafických procesorových jednotek (graphics processing unit, GPU) umožňujících zpracování velkého množství paralelních výpočtů. Brzy poté začal být tento typ hardware využíván pro trénování a evaluaci neuronových sítí, což vedlo k rozvoji hlubokého učení, které je dnes dominantním modelem strojového učení. Největší úspěchy byly dosaženy v oblasti klasifikace obrazových dat pomocí konvolučních neuronových sítí (convolutional neural network, CNN). Úspěšnost CNN závisí na jejich architektuře a dobrém natrénování, které vyžadují expertní znalosti v dané oblasti [12], [11]. Dalším typem algoritmů jsou evoluční algoritmy, které čerpají z Darwinovy teorie a poznatků moderní genetiky. Tyto algoritmy umožňují efektivně řešit některé netriviální optimalizační problémy [22]. Kombinace těchto typů algoritmů vedla k vzniku neuroevoluce, která může být využita na třech úrovních: evoluci vah, evoluci architektury a evoluci učících se algoritmů [31]. Zvláštní pozornost je věnována výzkumu v oblasti automatického návrhu struktur neuronových sítí, který vedl k vytvoření mnoha nových řešení, jež dosáhly srovnatelných nebo dokonce lepších výsledků než ručně navržené varianty [31], [27].

Cílem této práce je navrhnout a implementovat metodu pro automatizovaný návrh architektury neuronových sítí a na základě experimentů, prováděných za různých konfigurací této implementace, vyhodnotit kvalitu nalezených neuronových sítí a účinnost aplikovaných evolučních algoritmů.

Kapitola 2 je zaměřena na problematiku evolučních algoritmů s důrazem na kartézské genetické programování (Cartesian Genetic Programming, CGP) a nedominantní řadící genetický algoritmus (Non-dominated Sorting Genetic Algorithm, NSGA-II). Kapitola 3 podrobněji analyzuje koncepty neuronových sítí s důrazem na CNN. Kapitola 4 zkoumá teoretický rámec neuroevoluce a její aplikace na třech úrovních, zahrnující evoluci vah, architektury a algoritmů učení. Kapitola 5 se zabývá návrhem metody pro automatický návrh architektury CNN pomocí evolučního algoritmu NSGA-II, který je uplatněný pro optimalizaci sítí z hlediska počtu parametrů a kvality klasifikace. Pro reprezentaci jedinců v tomto procesu jsou aplikovány techniky, které nacházejí uplatnění v CGP. Kapitola 6 se věnuje implementaci navrženého řešení v programovacím jazyce Python s využitím knihovny

PyTorch, která představuje vhodný nástroj pro vývoj a trénování CNN. Závěrečná část práce je zaměřena na experimenty, v rámci kterých je provedena analýza vlivu velikosti populace na kvalitu výsledného řešení. Dále dochází k porovnání získaných architektur s již etablovanými modely, což přináší pohled na relevanci nalezených řešení.

Kapitola 2

Evoluční algoritmy

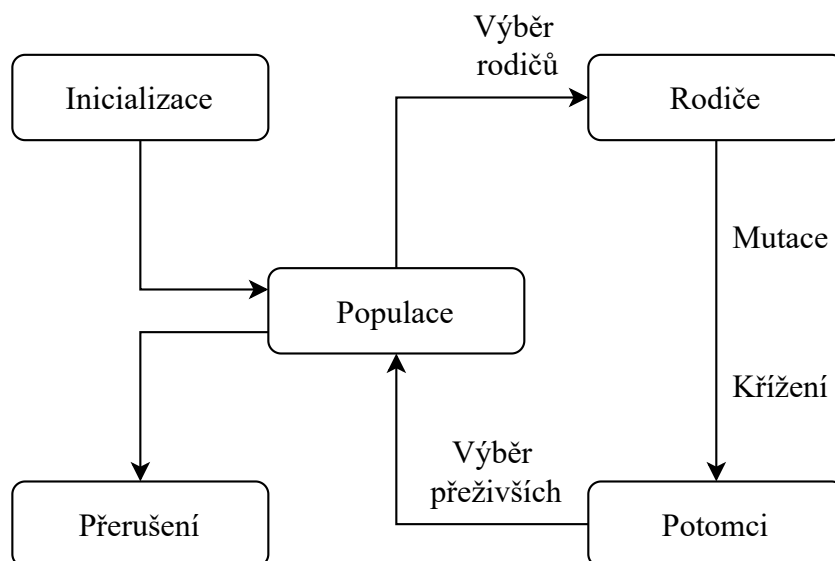
Příroda se stala zdrojem inspirace pro mnoho výzkumů, jejichž cílem je nalézt řešení pro netriviální inženýrské problémy. V tomto kontextu je netriviální problém definován jako takový, pro který v současnosti neexistuje algoritmus schopný nalézt řešení v polynomiálním čase. V takových situacích je nezbytné přistoupit k alternativnímu přístupu a hledat řešení, které je dostatečně přesné.

Rozvojem vědy a vlivem moderní genetiky bylo zjištěno, že každá forma života vyskytující se na Zemi, má zakódovanou genetickou informaci pomocí deoxyribonukleové kyseliny (deoxyribonucleic acid, DNA). Chromozom je větší struktura skládající se z DNA, histonových bílkovin a dalších molekul [30]. Gen je konkrétní úsek DNA, který může existovat v různých variantách nazývaných alely a který představuje základní funkční jednotku dědičnosti. Geny kódují fenotyp, což jsou pozorovatelné vlastnosti organismu. Pohlavní bunky (gamety) obsahují pouze jednu sadu chromozomů typickou pro dané pohlaví. Po spojení pohlavních buněk vznikne zygota, která se pomocí procesu ontogeneze vyvine do nového organismu, aniž by došlo k další změně genetického materiálu. Podle Darwinovy teorie se jedinci vyskytují v prostředí s omezenými zdroji. Jedinci mezi sebou soutěží o zdroje a šance na jejich získání je podmíněna jejich fenotypem. Schopnost získat zdroje je úzce provázána s šancí jedince přežít. Schopnější jedinci mají více potomků. Jejich geny se dostávají do dalších generací a multiplikují se, což je základem přirozeného výběru. Variabilita fenotypu je zajištěna reprodukcí jedinců a malou šancí náhodné změny zvané mutace, která při reprodukci může nastat [10]. Darwinova teorie a poznatky moderní genetiky poskytly základ pro vývoj algoritmů inspirovaných přírodou, známých jako evoluční algoritmy [22]. Tato kapitola byla sepsána zejména s využitím zdroje [7].

2.1 Obecný evoluční algoritmus

V současné době existuje mnoho kategorií evolučních algoritmů, které se liší ve variantách použitých komponent, ale principální myšlenka je zachována. Do prostředí, které nabízí jen omezené množství zdrojů, je vložena počáteční populace. Schopnost jedince získat zdroje je měřena fitness funkcí a na jejích hodnotách je závislá selekce jedinců sloužících jako základ nové populace. Pro získání potomků je nutné aplikovat operátory křížení a mutace. Pro křížení jsou typicky vybráni dva jedinci a kombinací jejich genetického materiálu vznikne potomek. Operátor mutace je aplikován na jednotlivce a způsobuje náhodné změny jeho genetického materiálu. Následuje proces nahrazení, při kterém spolu soutěží jedinci z nové a

původní populace. Komponenty evolučních algoritmů mají velmi často stochastický základ. Schéma průběhu evolučního algoritmu znázorňuje obrázek 2.1.



Obrázek 2.1: Schéma evolučního algoritmu. Převzato z [7].

Z historického hlediska existuje několik směrů evolučních algoritmů, které vznikaly nezávisle na sobě a které jsou charakterizovány zejména způsobem reprezentace jedince a implementací genetických operátorů. Evoluční programování používá přímé kodování problému ve formě automatu. Genetické algoritmy, které byly původně zaměřeny na reprezentaci jedince pomocí binárního řetězce, jsou snadno rozšiřitelné na řetězce reálných hodnot. Evoluční strategie využívají vektory reálných hodnot, zatímco genetické programování pracuje se stromy [1].

Reprezentace problému

Jedná se o způsob, jakým jsou reálná řešení zakódována pro potřeby počítače. Při tomto procesu velmi často dochází ke zjednodušení reálného problému a výběr správné reprezentace je obvykle ponecháván do rukou expertů. Třída objektů představující možná řešení v rámci původního problému je nazývána fenotyp, zatímco třída těchto objektů převedených do reprezentace odpovídajícího evolučního algoritmu je nazývána genotyp. Proces, kde se mapuje fenotyp na genotyp, se nazývá kódování, zatímco proces, kde se mapuje genotyp na fenotyp, se nazývá dekódování.

Fitness funkce

V rámci řešeného problému fitness funkce udává, jak dobrý je jedinec. Pomocí fitness funkce jsme schopni identifikovat správnější řešení a umožnit tak jedincům, kteří ho kódují, větší šanci na úspěch. Tato funkce se tak stává základní složkou selekce. Její sestavení je klíčové pro správný běh evolučního algoritmu a měla by co nejlépe vystihovat vlastnosti požadovaného řešení. Fitness funkce umožňuje kombinovat více zkoumaných vlastností, což vede k vícekritériální optimalizaci [20].

Populace

Populace uchovává reprezentace řešení v rámci daného kontextu. Metrika, která udává počet různých řešení vyskytujících se v dané populaci, se nazývá diverzita. Pro měření diverzity se používá počet různých hodnot fitness funkce, počet různých fenotypů nebo genotypů. Kvalita populace se v čase mění a na rozdíl od jedinců se jedná o dynamickou strukturu. V rámci některých evolučních algoritmů byly zavedeny přístupy, které zohledňují prostorovou strukturu, a tím lépe simulují reálnou evoluci. Velikost populace je velmi často pevně daná a vytváří se tak prostředí, ve kterém jsou jedinci nuceni mezi sebou soupeřit.

Výběr rodičů

Výběr rodičů a mechanismus nahrazení stojí za zlepšováním populace. Jedinci z dané populace jsou vybíráni s cílem mezi nimi provést křížení a vytvořit tak nové potomstvo. Velmi často se jedná o stochastickou metodu, kde i ti nejhůře hodnocení jedinci mají šanci být vybráni, aby se předešlo uváznutí algoritmu v lokálních minimech.

Mutace

Mutace je unární operátor, který slouží k provádění drobných změn v genomu jedince. Tento operátor je aplikován s určitou pravděpodobností a jedná se tedy o stochastickou metodu. Je tedy klíčovým prvkem pro udržení diverzity populace. V různých typech evolučních algoritmů hraje mutace odlišnou roli, například v evolučním programování funguje jako jediný variační operátor.

Křížení

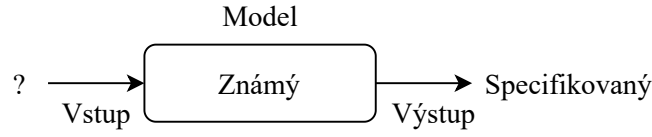
Křížení je primárně binární operátor, avšak existují přístupy pracující s vyšší aritou [8]. Při křížení jsou části genetického materiálu rodičů využity k vytvoření genetického materiálu potomka. Podobně jako u mutace je křížení opět stochastickou metodou.

Nahrazení

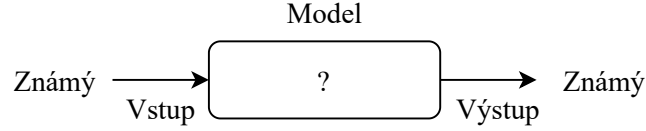
Stejně jako u selekce se jedná o algoritmus, který stojí za zlepšováním kvality populace. Na rozdíl od selekce a mutace se obvykle nejedná o stochastickou metodu. Nahrazení se provádí po vytvoření potomků. Velikost populace je téměř vždy pevně stanovená a jedinci soutěží o místo v další populaci. Často je také využíván koncept věku.

2.2 Genetické programování

Z historického hlediska se genetické programování zaměřovalo na evoluci počítačových programů, ale dnes má široké uplatnění v mnoha oblastech, kterými jsou například evoluce matematických výrazů nebo strojové učení. Na rozdíl od klasických evolučních algoritmů, které jsou zejména nástrojem pro optimalizaci, se genetické programování snaží o nalezení výpočetního modelu, to znamená o automatizovaný návrh. Rozdíl mezi těmito úlohami je znázorněn na obrázcích 2.2 a 2.3.



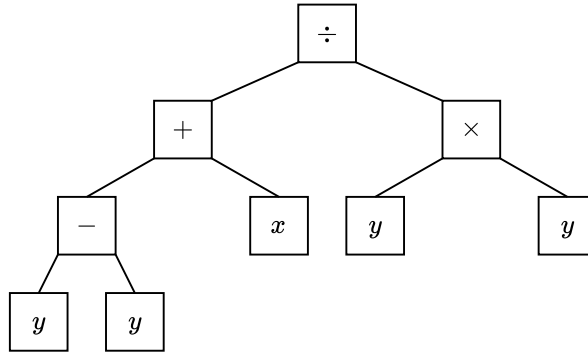
Obrázek 2.2: Klasická úloha evolučního algoritmu. Převzato z [7].



Obrázek 2.3: Klasická úloha genetického programování. Převzato z [7].

Reprezentace

Na rozdíl od genetických algoritmů, které používají zejména binární řetězce k reprezentaci jedinců, genetické programování využívá reprezentaci prostřednictvím stromů. Tvoří hierarchickou strukturu, která se skládá z funkcí a terminálů, jež jsou vybírány z definované množiny. Vnitřní uzly jsou vybírány z množiny funkcí, zatímco listové uzly jsou vybírány z množiny terminálů [25]. Příklad stromové struktury jedince je znázorněn na obrázku 2.4



Obrázek 2.4: Stromová struktura jedince odpovídající výrazu $\frac{(y-y)+x}{y^2}$. Převzato z [25].

Při inicializaci počáteční populace je obvykle definována maximální hloubka stromů a jedinci jsou vytvářeni z množiny funkcí F a množiny terminálů T .

Výběr rodičů

V genetickém programování je výběr rodičů obvykle prováděn pomocí metody proporční selekce, kde se pravděpodobnost výběru jedince odvíjí od jeho fitness funkce ve srovnání s celkovou hodnotou fitness v populaci. Konkrétně se pravděpodobnost výběru jedince i vypočítá podle vztahu $P(i) = f_i / \sum_{j=1}^{\mu} f_j$, kde f_i představuje fitness jedince i a μ označuje velikost populace.

Během několika let výzkumu byly identifikovány určité nedostatky spojené s touto selekční metodou. Mezi ně patří jev nazývaný předčasná konvergence, kdy se v populaci objevují jedinci s výrazně vyšší fitness, kteří rychle vytlačují méně zdatné jedince, což může vést k omezení diverzity a uvíznutí v lokálním maximu.

Jiným problémem je situace, kdy jsou fitness hodnoty jedinců velmi podobné. V takovém případě nastává malý selekční tlak, a tedy zlepšení fitness nepředstavuje pro jedince výraznou výhodu. Později v průběhu evoluce, když jsou již v populaci eliminováni nejméně zdatní jedinci, dochází pouze k pomalému zlepšování průměrné populační zdatnosti.

Dalším nedostatkem je změna chování selekčního mechanismu při transpozici fitness funkce. Tento jev je znázorněn v tabulce 2.1, kde dochází ke změně pravděpodobnosti výběru, ale extrémní funkce zůstávají zachovány.

Tabulka 2.1: Transpozice fitness funkce. Převzato z [7].

Jedinec	Fitness pro f	Pravd. výb. pro f	Fitness pro $f + 10$	Pravd. výb. pro $f + 10$	Fitness pro $f + 100$	Pravd. výb. pro $f + 100$
A	1	0.1	11	0.275	101	0.326
B	4	0.4	14	0.35	104	0.335
C	5	0.5	15	0.375	105	0.339
součet	10	1.0	40	1.0	310	1.0

Pro eliminaci problémů nedostatečného selekčního tlaku a změny chování selekčního mechanismu při transpozici fitness funkce lze využít metodu nazývanou windowing [7]. Tato procedura pracuje na základě odečítání hodnoty β^t od fitness hodnoty jedince. Hodnota β^t je závislá na čase a jedna její možná definice je, že představuje fitness hodnotu nejhůře hodnoceného jedince populace P^t . Alternativním přístupem může být průměrování hodnoty β^t přes jednotlivé generace. Tento přístup nachází využití zejména v případě výrazných oscilací hodnoty β^t mezi jednotlivými generacemi.

Mutace

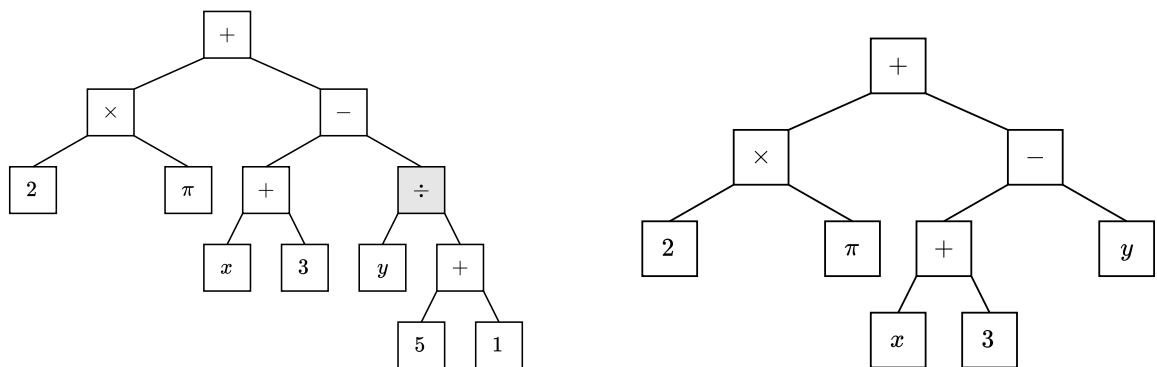
V tradičním přístupu k mutaci v genetickém programování se často pracuje s náhodným výběrem uzlu stromu, který je následně nahrazen novým podstromem. Tento nový podstrom je obvykle vytvořen podobným způsobem jako při inicializaci, přičemž jsou znovu uplatněna omezení na jeho hloubku. Takto vzniklý strom může mít větší hloubku než strom původní a vzniká tak negativní efekt zvaný bloat, při kterém stromová struktura expanduje bez výrazného zlepšení fitness hodnoty. Často obsahuje redundantní uzly. Pro předejití bloatu byly zavedeny metody, které zabrání provádění variačních operací, pokud by to mělo vést k překročení maximálně předem definované hloubky stromu. Existují také sofistikovanější metody, které zavádějí penalizaci fitness hodnot pro hluboké stromy. Ukázka mutace stromu pro výraz $2 \cdot \pi + ((x + 3) - y)$ je zobrazena na obrázku 2.5.

Křížení

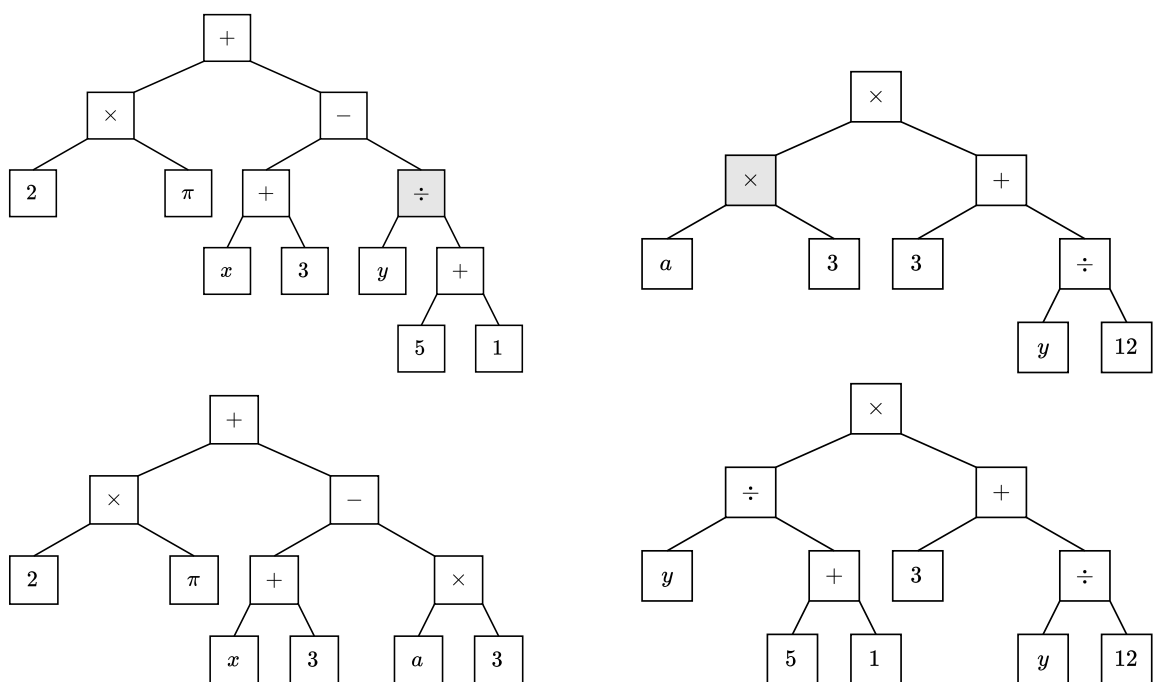
Nejčastěji se jedná o binární operátor. Při křížení dochází k náhodnému výběru uzlu ve stromové struktuře každého z rodičů. Potomci vznikají výměnou podstromů rodičů. Proces křížení je znázorněn na obrázku 2.6.

Nahrazení

V genetickém programování je tradičním přístupem metoda zvaná generační nahrazení, která vychází z obecného přístupu nazývaného nahrazení založené na věku. Tato metoda



Obrázek 2.5: Ilustrace mutace pro stromovou strukturu. Uzel vyznačený šedou barvou je vybrán pro mutaci. Nový, náhodně vygenerovaný strom, je list y . Převzato z [7].



Obrázek 2.6: Ilustrace křížení pro stromovou strukturu. Horní dva stromy reprezentují rodiče, zatímco spodní dva potomky. Uzly vyznačené šedou barvou jsou vybrány pro křížení. Prohozením podstromů rodičů vznikají potomci. Převzato z [7].

nebere v potaz fitness kondici jedince. Každý jedinec existuje v evolučním algoritmu určitý počet generací. Udržování nejlepších jedinců je možné pouze tehdy, když při křížení a mutaci nedochází k jejich modifikaci. Pro efektivní fungování této metody je klíčové v evolučním algoritmu vytvářet dostatečně výrazný selekční tlak pro výběr rodičů a používat variační operátory, které nezpůsobují příliš velké změny v genomu jedince. Pro genetické programování je typické, že počet potomků je roven velikosti populace, což zajišťuje, že každý jedinec v evolučním algoritmu absolvuje právě jeden cyklus.

2.3 Kartézské genetické programování

CGP je forma genetického programování vyvinutá mezi lety 1999-2000 Julianem Millerem. Jedná se o techniku vycházející z reprezentace, která byla použita pro evoluci elektronických obvodů [19]. Jedinci představují funkční programy, kde jsou jednotlivé funkce spojeny hranami a genotyp je vyjádřen jako seznam celých čísel.

Reprezentace

Fenotyp jedince je v CGP reprezentován pomocí orientovaného acyklického grafu (directed acyclic graph, DAG) a genotyp je posloupnost celých čísel. Tato reprezentace poskytuje metodě možnost zakódovat široké spektrum problémů, ať už se jedná o neuronové sítě, elektronické obvody nebo například strategie pro hraní počítačových her [19], [27], [29]. Fenotyp jedince tedy obsahuje neaktivní uzly, jejichž alternací nedojde ke změně funkce. Tato reprezentace kóduje graf, kde ne všechny uzly musí být nutně propojeny. Je však nutné podotknout, že obsahuje alespoň jeden propojený podgraf. DAG je vsazen do dvourozměrné mřížky specifikované počtem řádků r a počtem sloupců c . Uzly v i -tém sloupci mohou být propojeny pouze s uzly ve sloupcích $i - 1$ až $i - LevelBack$, kde parametr *LevelBack* je uživatelem definované celé číslo v rozsahu od 1 do počtu sloupců, které mřížka obsahuje. Obecné schéma reprezentace jedince v CGP je zobrazeno na obrázku 2.7.

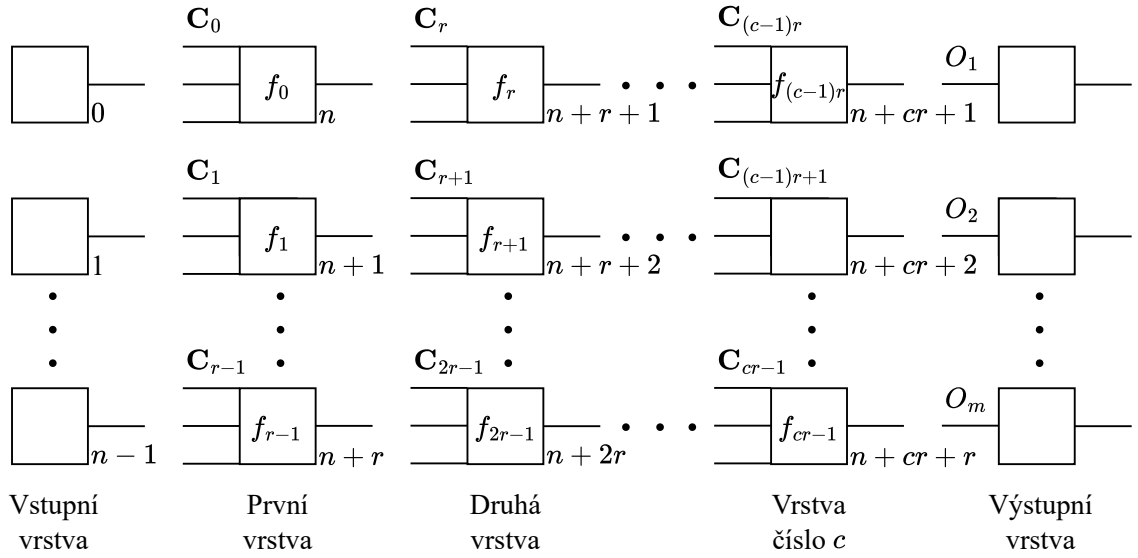
DAG je kódován ve formě řetězce celých čísel

$$C_0, f_0; C_1, f_1; \dots; C_{cx-1}, f_{cx-1}; O_1, O_2, \dots, O_m;$$

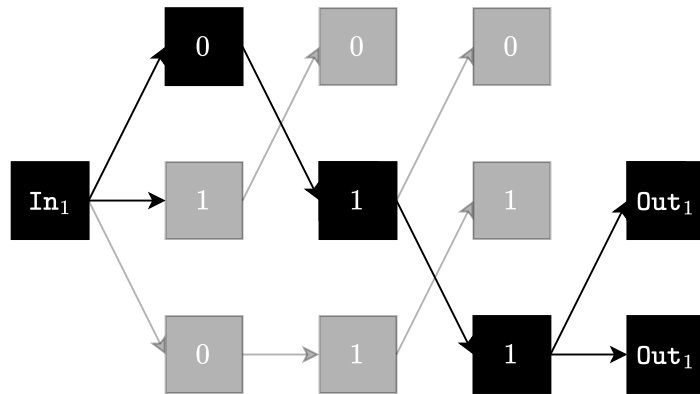
kde C_i je množina obsahující celočíselné identifikátory výstupů, ke kterým je uzel připojen. Číslování výstupů uzlů začíná v levém horním rohu a počáteční uzel má číslo výstupu rovné počtu programových vstupů. Číslování pokračuje postupně shora dolů, zleva doprava a každým dalším uzlem je číslo o jedna vyšší. Uzel grafu také obsahuje informace o funkci, kterou reprezentuje. Identifikátor f_i je číslo odkazující do lookup tabulky obsahující množinu dostupných funkcí. Pokud se arita jednotlivých funkcí liší, pak je počet vstupů do uzlů roven nejvyšší aritě, která se v těchto funkcích vyskytuje. Ostatní vstupy zůstávají nevyužity [18]. Způsob dekódování genotypu je na obrázku 2.8.

Variační operátory

CGP na rozdíl od genetického programování používá výhradně pouze mutaci. Pro operátor křížení bylo zavedeno několik přístupů, ale nebyla prokázána jejich užitečnost. Zatím nebyl nalezen způsob křížení, který by zachovával charakteristiky obou rodičů. Pro CGP je zavedeno několik mutačních mechanismů, mezi které patří pravděpodobnostní a bodová mutace a mutace právě jednoho aktivního genu. Bodová mutace je určena parametrem u_r , který určuje kolik genů se podrobí mutaci. Parametr pravděpodobnostní mutace pak stanovuje



Obrázek 2.7: Obecné schéma grafu kartézského genetického programování znázorněné pro n vstupů, m výstupů a mřížce velikosti r řádků a c sloupců. Každý uzel je definován množinou vstupů C a funkci f z množiny dostupných funkcí. Vlevo se nachází programové vstupy zatímco vpravo programové výstupy. Převzato z [18].

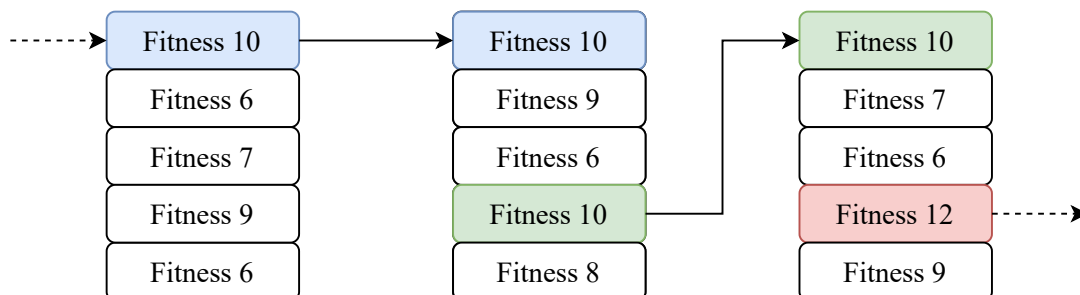


Obrázek 2.8: Kandidátní řešení v CGP s parametry $n = 1$, $m = 2$, $a = 1$, $c = 3$, $r = 3$ a genotypem 00, 01, 00, 20, 11, 31, 50, 61, 51, 99.

pravděpodobnost, že se gen podrobí mutaci. V CGP je využíváno hlavně bodové mutace, která je efektivnější vzhledem k výpočtu. Je třeba poznamenat, že u pravděpodobnostní a bodové mutace může dojít ke změně genu, který nemusí být aktivní. Tato skutečnost vedla k motivaci zavést efektivní mutaci, která by prováděla změnu právě jednoho aktivního genu. V této metodě není použita metrika pro určení míry mutace, ta je závislá na délce genomu jedince. Neaktivní uzly se mohou stát znovu aktivními a v průběhu evoluce mohly prodělat výrazné změny. Důsledkem toho se v CGP objevuje efekt zvaný neutrální drift, který se ukázal být velice prospěšným [17][18].

Prohledávací algoritmus

U CGP je obvykle process evoluce řízen algoritmem $(\mu + \lambda)$, nejčastěji $\mu = 1$ a $\lambda = 4$. Jedná se o jednoduchý algoritmus selekce, kde μ udává počet rodičů a λ počet potomků. Konkrétně v CGP se na jedince z množiny rodičů aplikuje operátor mutace, dokud nevznikne množina potomků o kradinalitě λ . Nový jedinci se podrobí evaluaci a následně jsou sloučeni s původní generací rodičů. Z takto vytvořené množiny se vybere μ nejlépe hodnocených jedinců. Vždy je vybrán potomek, který má stejnou nebo lepší fitness než rodič. Průběh algoritmu je znázorněn na obrázku 2.9



Obrázek 2.9: Průběh selekčního algoritmu $(\mu + \lambda)$ s hodnotami parametrů $\mu = 1$ a $\lambda = 4$. Rodič se nachází vždy nahoře. Cílem je maximalizovat fitness. Převzato z [17]

2.4 Vícekriteriální optimalizační genetický algoritmus

Využití vícekriteriální optimalizace často vede k vytvoření množiny řešení, což je odlišné od případu jednokriteriálních problémů, které mají pouze jedno výsledné řešení. Tato řešení se nachází na Pareto frontě, což jsou taková řešení, u nichž nelze zlepšit jedno kritérium bez zhoršení alespoň jednoho jiného kritéria. Z této množiny není možné vybrat nejlepší řešení bez dodání dodatečné informace o preferenci jedinců. Cílem metody je nalézt co nejvíce řešení. Často jsou vícekriteriální problémy převedeny na jednokriteriální, ale v takovém případě dochází k nalezení pouze jedno Pareto optimální řešení, a proto je nutné algoritmus spustit vícekrát. Tato skutečnost vedla k vytvoření vícekriteriálních evolučních algoritmů, které jsou schopny nalézt Pareto frontu. NSGA-II je metoda, která se snaží efektivně nalézt množinu Pareto optimálních řešení a zajišťuje diverzitu populace. Tato sekce byla převzata z [5].

Algoritmus pro řazení jedinců podle dominance

Algoritmus umožňuje roztrídění populace do tříd podle dominance jedinců nad jinými jedinci. Označme $i \prec j$. Potom platí, že prvek i dominuje prvek j , a žádná hodnota kritéria jedince i není horší než odpovídající hodnota kritéria jedince j , a alespoň jedno kritérium jedince i je lepší než odpovídající kritérium jedince j .

Málo efektivním přístupem k určení dominance by mohlo být porovnání každého jedince s každým jiným v populaci. Při počtu kritérií M a velikosti populace N by složitost algoritmu pro nalezení jedinců v první nedominující třídě byla $O(MN^2)$. Tento proces by se opakoval pro každou třídu, a pokud by v každé třídě byl pouze jeden jedinec, celková složitost by dosáhla $O(MN^3)$.

Pro rychlejší zařazení jedinců do tříd podle dominance lze použít algoritmus s vyšší paměťovou náročností, avšak s nižší časovou složitostí $O(MN^2)$. Tento algoritmus přiřazuje každému jedinci p počet n_p jedinců, kteří ho dominují, a množinu S_p jedinců, kteří jsou dominováni jedincem p . Jedinci, kteří nejsou dominováni žádným jiným jedincem, jsou zařazeni do první fronty \mathcal{F}_1 . Pro každého jedince p z této fronty je proveden výběr každého prvku q z množiny S_p a hodnota n_q je dekrementována o jedničku. Jedinci q , u kterých hodnota n_q klesla na nulu, jsou zařazeni do druhé fronty \mathcal{F}_2 a proces se opakuje, nyní pro vyšší třídy dominance. Každému prvku q je přiřazena hodnota q_{rank} reflektující v jaké třídě se prvek nachází. Tento proces je popsán algoritmem 1.

Udržení diverzity populace

Kromě dosažení konvergence je rovněž důležitá diverzita jedinců v nalezené sadě řešení. Pro udržení této rozmanitosti využívá algoritmus metriku shlukovací vzdálenosti $i_{distance}$, která je založena na odhadu hustoty řešení obklopující konkrétní řešení v populaci. Pro získání tohoto odhadu hustoty v okolí konkrétního řešení je nejprve nutné vypočítat vzdálenost dvou bodů nacházejících se po levé a pravé straně tohoto řešení vzhledem ke každému kritériu. Poté je pro výpočet shlukovací vzdálenosti potřeba seřadit prvky vzhledem ke každému kritériu ve sestupném pořadí. Pro hraniční prvky je nastavena shlukovací vzdálenost na nekonečně vysokou hodnotu a pro vnitřní prvky je rovna součtu normalizovaných vzdáleností podél každého kritéria. Algoritmus 2 popisuje postup výpočtu shlukovací vzdálenosti.

Zde $\mathcal{I}[i].m$ označuje hodnotu m -tého kritéria i -tého prvku v množině řešení \mathcal{I} . Hodnoty f_m^{max} a f_m^{min} představují nejvyšší a nejnižší hodnoty m -tého kritéria.

Po přiřazení shlukovací vzdálenosti všem prvkům je možné je porovnávat pomocí operátoru \prec_n . Jestliže prvky i a j pocházejí z různých dominovaných tříd, platí, že $i \prec_n j$, pokud i_{rank} je menší než j_{rank} . V případě, že prvky pocházejí ze stejné dominované třídy, platí, že $i \prec_n j$, jestliže shlukovací vzdálenost $i_{distance}$ je větší než shlukovací vzdálenost $j_{distance}$. Tím je upřednostněn výběr jedinců, v jejich okolí se nenachází jiná řešení blízko u sebe, což přispívá k větší diverzitě populace.

Hlavní smyčka algoritmu NSGA-II

Algoritmus začíná náhodným vytvořením počáteční populace P_0 . Každý prvek q této populace je ohodnocen podél všech kritérií a je mu přiřazena hodnota q_{rank} a $q_{distance}$. Pro vytvoření množiny potomků Q_0 o velikosti N prvků je použit turnajový výběr rodičů s využitím operátoru \prec_n , po němž následuje křížení a mutace. Hodnoty q_{rank} a $q_{distance}$ jsou poté přiřazeny i potomkům z množiny Q_0 na základě jejich fitness. Následuje vytvoření populace $R_0 = P_0 \cup Q_0$ obsahující $2N$ prvků.

Algorithm 1 Algoritmus pro řazení jedinců podle dominance. Převzato z [5].

Require: P , množina řešení.

```
1: for each  $p \in P$  do
2:    $S_p = \emptyset$ 
3:    $n_p = 0$ 
4:   for each  $p \in P$  do
5:     if  $p \prec q$  then
6:        $S_p = S_p \cup \{q\}$ 
7:     end if
8:     if  $q \prec p$  then
9:        $n_p = n_p + 1$ 
10:    end if
11:  end for
12:  if  $n_p = 0$  then
13:     $p_{rank} = 1$ 
14:     $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$ 
15:  end if
16: end for
17:  $i = 1$ 
18: while  $\mathcal{F}_i \neq \emptyset$  do
19:    $Q = \emptyset$ 
20:   for each  $p \in \mathcal{F}_i$  do
21:     for each  $q \in S_p$  do
22:        $n_q = n_q - 1$ 
23:       if  $n_q = 0$  then
24:          $q_{rank} = i + 1$ 
25:          $Q = Q \cup \{q\}$ 
26:       end if
27:     end for
28:    $i = i + 1$ 
29:    $\mathcal{F}_i = Q$ 
30: end for
31: end while
32: return  $\mathcal{F}$ 
```

Algorithm 2 Výpočet shlukovací vzdálenosti. Převzato z [5].

Require: \mathcal{I} , množina řešení.

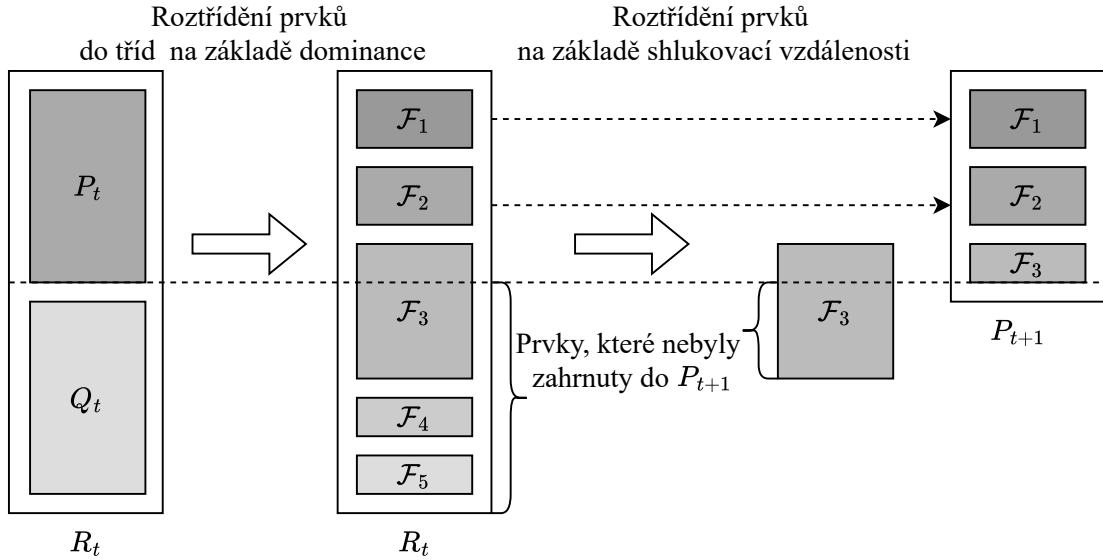
```

1:  $l = |\mathcal{I}|$  ▷ počet řešení v  $\mathcal{I}$ 
2: while  $i = 1$  to  $l$  do ▷ Inicializace vzdálenosti
3:    $\mathcal{I}[i]_{\text{distance}} = 0$ 
4:    $i = i + 1$ 
5: end while
6: for each objective  $m$  do
7:    $\mathcal{I} = \text{sort}(\mathcal{I}, m)$  ▷ Vzestupné uspořádání podle kritéria  $m$ 
8:    $\mathcal{I}[1]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} = \infty$ 
9:   while  $i = 2$  to  $l - 1$  do ▷ Výpočet vzdálenosti pro vnitřní body
10:     $\mathcal{I}[i]_{\text{distance}} = \mathcal{I}[l]_{\text{distance}} + (\mathcal{I}[i + 1].m - \mathcal{I}[i - 1].m) / (f_m^{\max} - f_m^{\min})$ 
11:   end while
12: end for
13: return  $\mathcal{I}$ 

```

Algoritmus zaručuje elitismus, protože populace R_0 obsahuje rodiče i potomky. Do nové populace P_1 jsou zahrnuty všechny prvky z první třídy. Zbývající prvky jsou vybrány z dalších tříd, aby populace P_1 obsahovala alespoň N jedinců. Pokud se do této populace nevejdou všechny prvky dané třídy, jsou prvky této třídy seřazeny sestupně podle operátoru \prec_n , a do populace P_1 je zařazeno pouze prvních n prvků této třídy, aby populace P_1 obsahovala N jedinců.

Noví potomci jsou následně vytvořeni z populace P_1 pomocí mutace a křížení, a tento proces se opakuje. Průběh hlavní smyčky pro populaci P_t je ilustrován na obrázku 2.10.



Obrázek 2.10: Ilustrace hlavní smyčky vícekritériálního optimalizačního algoritmu NSGA-II pro populaci P_t . Převzato z [5]

Kapitola 3

Neuronové sítě

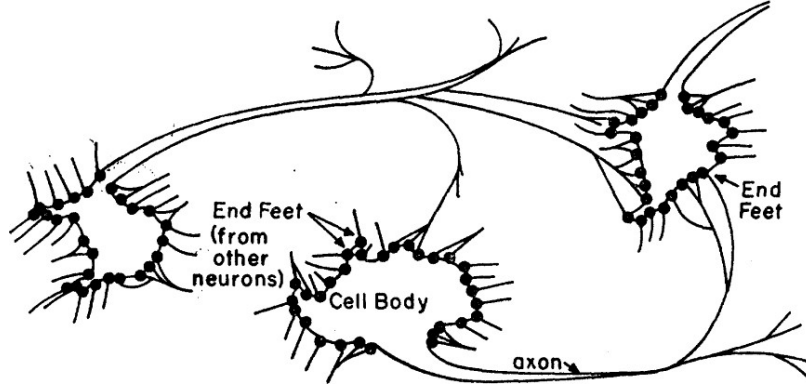
Umělé neuronové sítě (artificial neural network, ANN) se v posledních letech staly dominantním výpočetním modelem strojového učení. ANN našly široké uplatnění v oblastech počítačového vidění, zpracování přirozeného jazyka a dalších, kde dosahují významných výsledků. Jedná se o výpočetní model inspirovaný strukturou a funkcionalitou biologického mozku. Základní výpočetní jednotkou je umělý neuron, který je analogií k biologickému neuronu [23]. ANN zahrnují širokou škálu architektur, z nichž každá má své specifické využití. V této kapitole je popsána architektura známá jako vícevrstvý perceptron (Multilayer Perceptron, MLP), která je základní architekturou, a architektura CNN, která se využívá především v oblastech počítačového vidění a zpracování obrazu [32].

První umělé neurony byly představeny v roce 1944 dvěma výzkumníky, Warrenem McCullochem a Walterem Pittem. Signál se mezi neurony přenáší přes spoje mezi axonem jednoho neuronu a dendrity druhého. Pokud do neuronu vstoupí určité množství signálů z okolních neuronů, překročí se práh a dochází k vybuzení impulsu. Umělé neurony pracovaly na stejném principu a výstup těchto neuronů byl 1 nebo 0 v závislosti na tom, zda byl práh překročen [16]. Pro tyto neurony nebyl zaveden žádný učící se mechanismus a neurony nebyly uspořádány do vrstev. Hlavní myšlenkou a přínosem tedy bylo, že o mozku lze uvažovat jako o výpočetním zařízení. Dalším důležitým milníkem bylo představení perceptronu Frankem Rosenblattem roku 1957. Perceptron se stal prvním trénovatelným modelem podobný moderním neuronovým sítím [23]. Výzkum v této oblasti probíhal až do roku 1959, kdy Minsky a Papert dokázali, že provádění běžných výpočtů pomocí perceptronu je neefektivní. Do roku 1980 však byly představeny učící algoritmy dostatečně efektivní pro vícevrstvé neuronové sítě, pomocí kterých bylo možné provádět komplexnější výpočty [12]. Tato kapitola byla převážně převzata z [11].

3.1 Perceptron

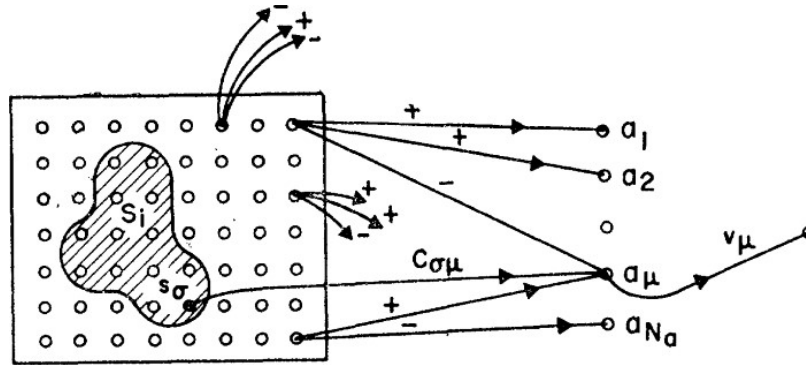
Mozek může svou strukturou připomínat elektronické počítače. Jeho komplexnost není založená na rozmanitosti základních komponent, ale na obrovském množství spojů mezi neurony. Neuron je tedy základní komponentou mozku, která umožňuje předávání elektrochemického signálu přes synapse. Impulzy putují z těl neuronů do výběžků zvaných axony. Synaptické zakončení se nachází na konci axonu a vytváří spojení s dendrity neuronu sloužících k přijímání signálů. Tento popis platí jen pro vnitřní neurony. Senzorické neurony jsou aktivovány vnějšími podněty jako je světlo, teplo a další stimuly. Motorické neurony mají zakončení ve svalovém vláknu a jejich úkolem je prostřednictvím signálu způsobit jeho kon-

trakci. V celém mozku se nachází odhadem 10^{10} neuronů vytvářejících spojení s jednotkami až tisíci jiných neuronů. V digitálních počítačích musí být všechna spojení přesná, jinak může být jeho chování zcela nesprávné. Tato skutečnost však neplatí pro spojení mezi neurony v mozku. Tento popis funkcionality mozku není sice přesný, ale sloužil jako inspirace pro vznik neuronových sítí. Spojení mezi neurony je znázorněno na obrázku 3.1.



Obrázek 3.1: Spojení mezi neurony. Převzato z [2].

Rosenblatt poskytl model, který byl dostatečně přesný pro testování, dostatečně složitý pro poskytování zajímavého chování, jednoduchý pro analýzu a v souladu s biologickými fakty. Perceptron vychází z modelu mozku a je znázorněn na obrázku 3.2.



Obrázek 3.2: Jednoduchý perceptron. Převzato z [2].

Model se skládá z N_s senzorických jednotek, N_a asociačních jednotek a n podnětů. Sensorická jednotka je značena s_σ , asociační jednotka a_μ a podnět jako S_i . Spojení mezi sensorickou jednotkou s_σ a asociační jednotkou a_μ je značeno $C_{\sigma\mu}$ a nabývá hodnot $+1$, -1 nebo 0 . Pokud dojde k stimulaci sensorické jednotky podnětem, tak jednotka odešle signál. Pokud algebraický součet těchto signálů znázorněný rovnicí 3.1 překročí práh θ , dojde k vybuzení a přenosu signálu v_μ na jednotku způsobující odezvu.

$$\alpha_{\mu^i} = \sum_{s_\sigma \in S_i} C_{\sigma\mu} \quad (3.1)$$

Celkový signál u vstupující do jednotky způsobující odezvu je dán algebraickým součtem všech signálů asociačních jednotek, které do ni vstupují. Když je $u > \Theta$, kde Θ představuje vhodně zvolené kladné číslo, je odpověď jednotky $+1$. Pro případ $u < \Theta$ je odpověď jednotky

-1 , a pro situaci, kdy $|u| \leq \Theta$, odpověď je 0 . Pokud $A(S_i)$ je množina senzorů aktivovaných podnětem S_i a pokud vztah 3.2:

$$e_{\mu i} = \begin{cases} 1, & \text{pro } a_\mu \in A(S_i) \\ 0, & \text{pro } a_\mu \notin A(S_i), \end{cases} \quad (3.2)$$

pak se vstupní signál přicházející do jednotky způsobující odezvu při prezenatci podnětu S_i vyjádří rovnicí 3.3:

$$u_i = \sum_{\mu} v_{\mu} e_{\mu i}. \quad (3.3)$$

Perceptron je binárním klasifikátorem. Pokud budeme pro podněty uvažovat dvě třídy klasifikace (-1 a 1) a podnětu S_i je přiřazena třída ρ_i , pak je správná klasifikace vyjádřena rovnicí 3.4:

$$\rho_i u_i = \sum_{\mu} v_{\mu} e_{\mu i} \rho_i > \Theta \quad (3.4)$$

Aby perceptron správně klasifikoval, je nutné zavést algoritmus učení. Tento algoritmus spočívá v úpravě signálů vysílaných asociačními jednotkami. Počáteční hodnoty vektoru signálů $(v_1^0, v_1^0, \dots, v_{\mu}^0, \dots, v_{N_a}^0)$ jsou náhodná reálná čísla. Pokud je podnět správně klasifikován, nedochází k žádné úpravě těchto signálů. V případě nesprávné klasifikace jsou však signály v_{μ} aktivních asociačních jednotek a_{μ} zvýšeny o $\eta \rho_i$, kde η představuje koeficient rychlosti učení [2].

Perceptron dokáže řešit úlohy, kde jsou jednotky lineárně separovatelné. V opačném případě lze problém vyřešit sekvenčním zapojením více vrstev perceptronů. Jelikož nebyl k dispozici dostatečně efektivní algoritmus učení, došlo k pozastavení výzkumu v této oblasti. Až v průběhu 80. let se objevily dostatečně efektivní algoritmy učení a tato oblast se stala opět aktivní [12].

3.2 Dopředná neuronová síť

Dopředné neuronové sítě jsou základním modelem strojového učení. Tyto sítě jsou typicky složeny z většího počtu funkcí tvořících DAG. Když jsou data vložena do modelu, procházejí sekvenčně sérií výpočtů, a odtud pochází název dopředná neuronová síť. Z toho modelu vycházejí další modely jako jsou například konvoluční neuronové sítě, které se používají především pro rozpoznávání objektů na obrázcích, a sítě se zpětnou vazbou, nazývané rekurentní neuronové sítě používané pro zpracování přirozeného jazyka. Cílem dopředných neuronových sítí je aproximovat funkci $f^*(\mathbf{x})$. Model poskytuje funkci $f(\mathbf{x}, \theta)$ a algoritmus učení upravuje váhy θ tak, aby se tato funkce co nejvíce podobala funkci $f^*(\mathbf{x})$. Počet operací spojených v řetězci za sebou udává hloubku neuronové sítě. Poslední vrstva tohoto modelu se nazývá výstupní. Při učení neuronové sítě je chování poslední vrstvy přesně specifikováno, ale chování ostatních vrstev není pevně určeno. Tyto vrstvy se nazývají skryté. Každá skrytá vrstva je typicky funkce, jejímž vstupem i výstupem je vektor. Tyto vrstvy se skládají z více jednotek zvaných perceptrony. Vrstvy neuronových sítí většinou provádějí nelineární transformaci dat, jak je popsáno v rovnici 3.5:

$$\mathbf{h} = g(\mathbf{W}^T \mathbf{x} + \mathbf{c}), \quad (3.5)$$

kde \mathbf{W} představují váhy lineární transformace, \mathbf{c} je posunutí a g je aktivační funkce. Nejpoužívanější aktivační funkcí v současné době je rektifikační lineární jednotka (Rectified Linear Unit, ReLU).

3.3 Chybová funkce

Aby bylo možné model trénovat, je nutné definovat ztrátovou funkci. Díky nelinearitě používané v neuronových vrstvách se jejich ztrátové funkce často stávají nekonvexními. Pro trénování neuronových sítí se využívají algoritmy založené na gradientu. Metoda optimalizace je aplikována na model opakovaně v cyklech, kdy je hodnota ztrátové funkce postupně snižována až na velmi malou hodnotu. Nicméně metoda gradientního sestupu není schopna zaručit dosažení globálního minima. Konvergence algoritmu učení závisí na počáteční inicializaci parametrů.

Ve většině případů je parametrický model poskytovaný neuronovou sítí definován rozdělením $p(\mathbf{y}|\mathbf{x}; \theta)$ a pro odhad parametrů je tedy možné použít metodu maximální věrohodnosti (maximum likelihood), která se snaží maximalizovat pravděpodobnost pozorovaných dat v závislosti na parametrech modelu. V rovnicích 3.6, 3.7, 3.8 je uvedena maximalizace funkce věrohodnosti [11]

$$\hat{\theta}_{\text{ML}} = \arg \max_{\theta} p_{\text{model}}(\mathbb{X}; \theta) \quad (3.6)$$

$$= \arg \max_{\theta} \left(\prod_{i=1}^n p_{\text{model}}(x_i; \theta) \right) \quad (3.7)$$

$$= \arg \max_{\theta} \sum_{i=1}^n \log p_{\text{model}}(x_i; \theta), \quad (3.8)$$

kde θ jsou parametry modelu a $\mathbb{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)}\}$ je množina příkladů nezávisle získaných z pravého ale neznámého pravděpodobnostního rozdělení $p_{\text{data}}(\mathbf{x})$ generujícího data. Cost funkce je vyjádřena rovnicí 3.9

$$J(\theta) = -\mathbb{E}_{\mathbb{X}, y \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(y|\mathbf{x}), \quad (3.9)$$

kde \hat{p}_{data} je empirické pravděpodobnostní rozdělení definované trénovacími daty \mathbb{X} , \mathbf{y} jsou příznaky korespondující k trénovacím datům a \mathbb{E} je střední hodnota. Díky metodě maximální věrohodnosti jsme schopni odvodit ztrátovou funkci a není potřeba ji definovat pro různé modely. Gradient této funkce musí být dostatečně velký, aby mohl algoritmus učení efektivně fungovat. Může se stát, že tato podmínka není splněna vlivem saturace u jednotek obsahujících v aktivační funkci exponenciálu. Takovou aktivační funkcí je například logistická funkce. Metoda maximální věrohodnosti tento problém řeší, neboť logaritmus funguje jako inverzní operace k exponenciální funkci.

3.4 Výstupní vrstva

Aktivační funkce výstupní vrstvy ovlivňuje tvar funkce s maximální věrohodností. Poslední vrstva upravuje výstup sítě tak, aby odpovídal požadovanému formátu.

Lineární jednotka

Jedná se o jednotku provádějící lineární funkci s afinní transformací. Transformace vstupu \mathbf{h} na výstup $\hat{\mathbf{y}}$ pomocí vah \mathbf{W} a posunutí \mathbf{b} je dán rovnicí 3.10:

$$\hat{\mathbf{y}} = \mathbf{W}^{\top} \mathbf{h} + \mathbf{b} \quad (3.10)$$

Lineární vrstvy se jako výstupní vrstvy často používají pro odhad střední hodnoty a rozptylu normálního rozdělení. V lineárních funkcích nenastává saturace, což může představovat výzvu při tréninku pomocí gradientních metod.

Logistická jednotka

Používá se pro řešení problému klasifikace do dvou tříd. Metoda maximální věrohodnosti se zaměřuje na definici Bernoulliho rozdělení pro y , podmíněného vstupem \mathbf{x} . Úkolem neuronové sítě je predikovat hodnotu $P(y = 1|\mathbf{x})$, která je platná v intervalu $[0, 1]$.

Splnění této podmínky by bylo možné například pomocí lineární jednotky a oříznutím hodnot tak, aby byly v rámci intervalu $[0, 1]$, jak je znázorněno rovnicí 3.11.

$$P(y = 1|\mathbf{x}) = \max\{0, \min\{1, \mathbf{w}^\top \mathbf{x} + b\}\} \quad (3.11)$$

Přestože se jedná o validní rozložení, není vhodné ho používat, protože pokud hodnota $\mathbf{w}^\top \mathbf{x} + b$ vychází mimo interval, má výstup jednotky gradient rovný nule. V tomto případě je učení náročné pro optimalizační metody založené na gradientu.

Řešení tohoto problému je možné pomocí logistické funkce, která má v každém bodě nenulový gradient. Tato funkce spolu s metodou maximální věrohodnosti tvoří přístup poskytující silný gradient pro optimalizační metody založené na gradientu. Logistická jednotka je popsána rovnicí 3.12, kde nejdříve dochází k výpočtu lineární vrstvy $z = \mathbf{w}^\top \mathbf{x} + b$, na jejíž výstup je aplikována aktivační funkce σ , kterou je logistická funkce.

$$\hat{y} = \sigma(\mathbf{w}^\top \mathbf{x} + b) \quad (3.12)$$

Při použití metody maximální věrohodnosti, která má tvar $-\log P(y|\mathbf{x})$, dojde k vyrušení exponenciálního prvku v logistické funkci logaritmem, což vede k dostatečnému gradientu, i když dochází k saturaci logistické funkce. Rovnice 3.13, 3.14 a 3.15 převádějí ztrátovou funkci, která je vyjádřena pomocí logistické funkce, na variantu vyjádřenou softplus funkcí ζ .

$$J(\theta) = -\log P(y|\mathbf{x}) \quad (3.13)$$

$$= -\log \sigma((2y - 1)z) \quad (3.14)$$

$$= \zeta((1 - 2y)z) \quad (3.15)$$

Pouze při extrémně negativním $(1 - 2y)z$ dochází k saturaci softplus funkce ζ . Z tohoto charakteru vyplývá, že saturace nastává pouze v případě, když model již generuje správnou odpověď. Takže pokud model poskytuje nesprávnou odpověď, gradient chybové funkce je silný, což vede k rychlé korekci chyby.

Softmax jednotka

Softmax je běžně používaná aktivační funkce ve výstupní vrstvě neuronových sítí určených pro klasifikaci. Jedná se o zobecnění logistické funkce, která umožňuje reprezentovat pravděpodobnostní rozložení pro náhodnou proměnnou s n hodnotami, na rozdíl od logistické funkce, která je určena pro náhodnou proměnnou s dvěma hodnotami.

Úkolem neuronové sítě je vyprodukovat vektor hodnot $\hat{\mathbf{y}}$, kde $\hat{y}_i = P(y = i|\mathbf{x})$. Stejně jako u logistické funkce musí hodnoty ležet v intervalu $[0, 1]$ a navíc součet celého vektoru je roven 1.

Linární vrstva neuronové sítě poskytuje nenormalizované logaritmické pravděpodobnosti. Výstup této vrstvy \mathbf{z} je dán rovnicí 3.16:

$$\mathbf{z} = \mathbf{W}^\top \mathbf{h} + \mathbf{b}, \quad (3.16)$$

kde $z_i = \log(\hat{P}(y = i|\mathbf{x}))$. Softmax funkce je pak definována rovnicí 3.17:

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)}, \quad (3.17)$$

kde $\mathbf{z} = (z_1, z_2, \dots, z_n)$ je vektor vstupních hodnot. Stejně jako u logistické funkce funguje softmax dobře v kombinaci se ztrátovou funkcí logaritmu maximální věrohodnosti. Logaritmus v této funkci vyruší exponenciálu vyskytující se ve funkci softmax a poskytuje tak dostatečně silný gradient pro učení. Kombinace softmax funkce s metodou maximální věrohodnosti je dána rovnicí 3.18:

$$\log \text{softmax}(\mathbf{z})_i = z_i - \log \sum_{j=1}^n \exp(z_j). \quad (3.18)$$

Výstupní vektor softmax funkce reaguje na rozdíly mezi hodnotami ve vstupním vektoru. Pokud je prvek z_i vektoru \mathbf{z} mnohem větší než ostatní prvky, pak odpovídající hodnota $\text{softmax}(\mathbf{z})_i$ saturuje k 1. Naopak, pokud je jiný prvek vektoru mnohem větší než ostatní, pak funkce saturuje k 0. Tato vlastnost může způsobit problém nízkého gradientu u ztrátových funkcí, které tuto situaci nekompenzují.

3.5 Skryté vrstvy

Existuje mnoho aktivačních funkcí používaných ve skrytých vrstvách neuronové sítě, ale ne všechny jsou diferencovatelné v každém bodě. V praxi je velmi nepravděpodobné, že by došlo k vyhodnocení funkce přesně v určitém bodě, spíše kvůli zaokrouhlení dojde k posunu o malou hodnotu ϵ . Softwarové implementace často řeší tento problém použitím náhradní hodnoty, aby se vyhnuly problému nedefinovaných derivací. Skryté vrstvy neuronové sítě jsou definovány výpočtem afinní transformace $\mathbf{z} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$, na jejíž jednotlivé prvky je aplikována aktivační funkce.

ReLU jednotka

ReLU je jednotka velmi podobná lineární funkci. Může být buď aktivní nebo neaktivní, což je dáno nulovým gradientem na polovině jejího definičního oboru. Na druhé polovině je gradient roven 1. Druhá derivace je však vždy nulová. Výhodou ReLU je, že má silný a stabilní gradient. ReLU je popsána rovnicí 3.19:

$$g(z) = \max\{0, z\} \quad (3.19)$$

V určitých situacích může být nevýhodou této jednotky, že má nulový gradient na polovině svého definičního oboru. Existuje přístup řešící tento problém. Pokud je vstupní hodnota $z_i < 0$, použije se princip popsáný rovnicí 3.20:

$$h_i = g(z_i, \alpha_i)_i = \max(0, z_i) + \alpha_i \min(0, z_i), \quad (3.20)$$

kde α_i je neulový argument. Pokud je hodnota parametru α_i nastavena na -1 , pak se jedná o absolutní hodnotu rectifikace. Pokud je α_i nastavena na malé číslo kolem 0.01, pak se jedná o leaky ReLU. A pokud je α_i parametrem schopným učení, pak se jedná o PReLU (parametrickou ReLU).

Maxout

Jedná se o jednotku, která je zobecněním ReLU. Vstupní vektor \mathbf{z} je rozdělen do tříd po k prvcích. Maxout je znázorněn rovnicí 3.21:

$$g(\mathbf{z})_i = \max_{j \in \mathbb{G}^{(i)}} z_j, \quad (3.21)$$

kde $\mathbb{G}^{(i)}$ jsou indexi pro vstup i , $\{(i-1)k+1, \dots, ik\}$. Nejedná se o jednotku s ustálenou aktivační funkcí, jde o jednotku, která se snaží naučit samotnou aktivační funkci. To je možné díky schopnosti jednotky aproximovat libovolnou konvexní funkci pro dostatečně velká k .

3.6 Algoritmus zpětného šíření chyby

Tento algoritmus je základem pro výpočet gradientu ztrátové funkce vzhledem k parametřům modelu, označený jako $\nabla_{\mathbf{x}} J(\theta)$. Tvoří základ pro trénování neuronové sítě pomocí metod založených na gradientním sestupu. Ačkoliv by bylo možné provést analytické vyhodnocení gradientu, numericky by bylo příliš náročné na výpočet. Algoritmus zpětného šíření poskytuje jednoduché a efektivní řešení pro výpočet gradientu.

Pravidlo řetězového derivování

Pravidlo řetězového derivování je používáno pro derivování zřetěžených funkcí. Tato metoda je založena na předpokladu, že derivace jednotlivých funkcí jsou známy a derivace složené funkce je rovna součinu derivace funkce vnější a vnitřní. Tuto metodu vyjadřuje rovnice 3.22,

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}, \quad (3.22)$$

kde se provádí derivace funkce $z = f(g(x))$, přičemž $y = g(x)$. V tomto případě funkce f a g mapují reálné číslo na reálné číslo, ale metoda je rozšiřitelná pro funkce jejichž vstupem a výstupem je vektor. Zápis 3.23 je zobecněním této metody,

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_i}, \quad (3.23)$$

kde $g : \mathbb{R}^m \rightarrow \mathbb{R}^n$ a $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Ekvivaletním zápisem ve vektorové notaci je rovnice 3.24

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^\top \nabla_{\mathbf{y}} z, \quad (3.24)$$

kde $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ je Jakobiho matice funkce g . Metoda lze dále zobecnit pro tenzory zápisem 3.25,

$$\nabla_{\mathbf{x}} z = \sum_j (\nabla_{\mathbf{x}} \mathbf{Y}_j) \frac{\partial z}{\partial \mathbf{Y}_j} \quad (3.25)$$

kde je j uspořádaný seznam indexů.

Obecný algoritmus zpětného šíření chyby

Výpočet gradientu z s ohledem na proměnnou \mathbf{x} začíná stanovením $\frac{\partial z}{\partial z} = 1$. Gradient z vzhledem k jakémukoliv jeho předchůdci ve výpočetním grafu je vypočten jako součin aktuálního gradientu a Jakobiho maticí operace, která vyprodukovala proměnnou z . Tento postup se opakuje, dokud není dosaženo proměnné \mathbf{x} . V případě, že existuje několik cest zpětné propagace z proměnné z do některého uzlu grafu, je jejich gradient spočten jako součet gradientů vstupujících do tohoto uzlu. Tento postup spolu s vytvořením výpočetního grafu pro derivaci je popsán algoritmy 3 a 4.

Algoritmy uvažují graf \mathcal{G} , kde je každý uzel grafu tenzorem \mathbf{V} . Proměnná \mathbf{V} je asociována s následujícími operacemi:

- **get_operation(\mathbf{V})**: Vrací operaci, která provádí výpočet tenzoru \mathbf{V} . Tato operace je spojena s funkcí \mathbf{f} , jež tuto operaci implementuje. Dále je propojena s funkcí **bprop**, která umí vypočítat Jakobiho matici korespondující k dané operaci.
- **get_consumers(\mathbf{V}, \mathcal{G})**: Vrací seznam proměnných, které jsou potomky proměnné \mathbf{V} ve výpočetním grafu \mathcal{G} .
- **get_inputs(\mathbf{V}, \mathcal{G})**: Vrací seznam proměnných, které jsou rodiči proměnné \mathbf{V} ve výpočetním grafu \mathcal{G} .

Funkce **bprop(inputs, \mathbf{X}, \mathbf{G})** je formálně popsána rovnicí 3.26

$$\sum_i (\nabla_{\mathbf{X}} \text{op}.\mathbf{f}(\text{inputs})_i) \mathbf{G}_i, \quad (3.26)$$

kde **inputs** jsou vstupy funkce \mathbf{f} , \mathbf{X} je proměnná, jejíž gradient se počítá, a \mathbf{G} je gradient na výstupu operace.

Algorithm 3 Vnější kostra algoritmu zpětné propagace. Převzato z [11].

Require: \mathbb{T} , množina proměnných, vzhledem k nimž se má vypočítat gradient.

Require: \mathcal{G} , výpočetní graf.

Require: z , proměnná, která má být diferencována.

- 1: Nechť \mathcal{G}' je \mathcal{G} upravený tak, aby obsahoval pouze uzly, které jsou předky z a potomky uzlů \mathbb{T} .
 - 2: Inicializace **grad_table**, datové struktury, která přiřazuje uzlům jejich gradienty.
 - 3: **grad_table**[z] $\leftarrow 1$
 - 4: **for** \mathbf{V} in \mathbb{T} **do**
 - 5: **build_grad**($\mathbf{V}, \mathcal{G}, \mathcal{G}', \text{grad_table}$)
 - 6: **end for**
 - 7: **return** **grad_table** podle proměnných \mathbb{T} .
-

3.7 Konvoluční neuronová síť

CNN jsou využívány k analýze dat s mřížkovou strukturou, jako jsou časové řady ve formě 1D mřížky nebo obrázky, jejichž pixely jsou uspořádané v 2D mřížce. CNN provádějí lineární operaci zvanou konvoluce. Kvalita neuronové sítě je dána do velké míry její architekturou. Výzkum v oblasti architektur neuronových sítí je velmi intenzivní a nové architektury jsou prezentovány poměrně často. Motivací pro použití konvolučních sítí je využití jejich vlastností sdílení parametrů, řídké interakce a ekvivariance, které přispívají k jejich užitečnosti.

Algorithm 4 Vnitřní smyčka podrutiny `build_grad(V, G, G', grad_table)`, volané algoritmem zpětné propagace definovaným v algoritmu 3. Převzato z [11].

Require: V , proměnná, jejíž gradient má být přidán do grafu G a struktury `grad_table`

Require: G , graf, který má být pozměněn

Require: G' , omezení G na uzly, které se podílejí na gradientu

Require: `grad_table`, datová struktura přiřazující uzlům jejich gradienty

```

1: if  $V$  is in grad_table then
2:   return grad_table[V]
3: end if
4:  $i \leftarrow 1$ 
5: for  $C$  in get_consumers(V, G') do
6:    $op \leftarrow \text{get\_operation}(C)$ 
7:    $D \leftarrow \text{build\_grad}(C, G, G', \text{grad\_table})$ 
8:    $G^{(i)} \leftarrow op.bprop(\text{get\_inputs}(C, G'), V, D)$ 
9:    $i \leftarrow i + 1$ 
10: end for
11:  $G \leftarrow \sum_i G^{(i)}$ 
12: grad_table[V] \leftarrow G
13: Vlož  $G$  a operace vytvářející tento gradient do  $G$ 
14: return  $G$ 

```

Operace konvoluce

Konvoluce je operace, která pracuje s dvěma funkcemi, jež mají reálné číslo jako argument. Základní značení konvoluce je definováno rovnicí 3.27:

$$s(t) = (x * w)(t), \quad (3.27)$$

kde funkce x představuje vstup a funkce w je nazývána jádrem konvoluce. Pro výpočty na počítačích je obvykle tato funkce diskretizována, a pokud uvažujeme, že argument t je celé číslo, pak lze tuto operaci zapsat rovnicí 3.28

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \quad (3.28)$$

V oblasti strojového učení je vstup konvoluce obvykle reprezentován tenzorem dat a jádro je tenzor vah, které se upravují během procesu učení. Rovnice 3.28 představuje konvoluci pomocí nekonečného součtu, avšak funkce vstupující do konvoluce mají nenulové hodnoty pouze na omezeném intervalu. Proto je možné konvoluci provést pomocí konečného součtu přes všechny nenulové hodnoty těchto funkcí.

V neuronových sítích se běžně používá operace křížové korelace, která se od konvoluce liší tím, že nemá převrácené jádro. Na rozdíl od konvoluce není křížová korelace komutativní. Avšak pro účely neuronových sítí se tato vlastnost obvykle považuje za zanedbatelnou, protože operace konvoluce není aplikována samostatně, ale často se vyskytuje v kombinaci s další funkcí, která stejně naruší komutativitu u převráceného jádra. V některých knihovnách je operace křížové korelace nazývána konvolucí, a v této práci je pro obě tyto operace bude používán termín konvoluce. Rovnice 3.29 popisuje operaci konvoluce s nepřevráceným jádrem.

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i+m, j+n)K(m, n). \quad (3.29)$$

Konvoluce v neuronových sítích

Konvoluce v neuronových sítích je chápána jako operace, která zahrnuje paralelní aplikaci několika konvolucí současně. Vstup do CNN má obvykle další dimenzi, která představuje kanály. Pokud se jedná o obrázky, jednotlivé kanály reprezentují intenzity červené, zelené a modré barvy. Tímto způsobem je vrstva neuronové sítě schopna extrahovat více rysů z jednoho místa. Vrstvy vícevrstevných konvolučních neuronových sítí typicky generují jako výstup tenzor, který obsahuje hodnoty získané aplikací několika konvolučních jader na každé pozici. V případě zpracování obrázků takto vzniká 3-D tenzor, kde jeden index označuje různé kanály a zbývající dva určují souřadnice v rámci každého kanálu. Rovnice 3.30 popisuje proces konvoluce aplikovaný na data s více kanály.

$$\mathbf{Z}_{i,j,k} = \sum_{l,m,n} \mathbf{V}_{l,j+m,k+n} \mathbf{K}_{i,l,m,n}, \quad (3.30)$$

kde \mathbf{K} je 4-D tenzor, kde element $\mathbf{K}_{i,j,k,l}$ poskytuje váhu mezi jednotkou v kanálu vstupu i s posunem o k řádků a l sloupců a jednotku výstupu v kanálu j s posunem o k řádků a l sloupců. Tenzor \mathbf{V} je vstupní hodnota, kde element $\mathbf{V}_{i,j,k}$ udává hodnotu v rámci kanálů i na řádku j a řádku k . Tenzor \mathbf{Z} má stejný formát jako vstup. Součet je proveden přes platné indexy do tenzorů.

Pro snížení výpočetní náročnosti lze zavést krok s , kterým bude konvoluce prováděna. Avšak tato metoda může negativně ovlivnit extrakci vlastností ze vstupu. Konvoluce s krokem s je popsána rovnicí 3.31.

$$\mathbf{Z}_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} [\mathbf{V}_{l,j \times s + m, k \times s + n} \mathbf{K}_{i,l,m,n}]. \quad (3.31)$$

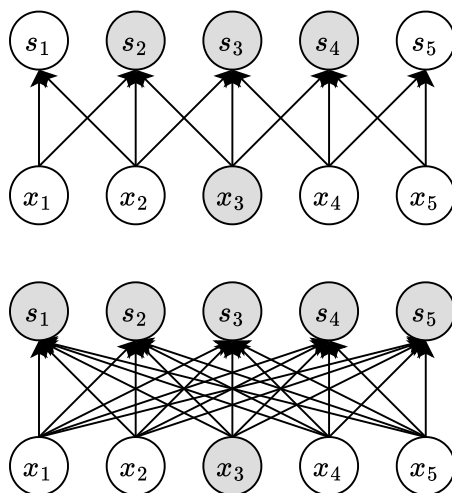
V rámci CNN je možné tenzor \mathbf{V} vyplnit nulami, čímž lze ovlivnit velikost výstupního tenzoru při aplikaci konvoluce. Existují tři způsoby, jak nastavit nulovou výplň. První způsob nezahrnuje žádnou nulovou výplň, což vede ke zmenšení tenzoru a omezení počtu konvolučních vrstev v neuronové síti. Nicméně každý výstup je vypočten ze stejného počtu vstupních jednotek. Druhý způsob se nazývá zero-padding a udržuje velikost tenzoru pomocí přidání nulové výplně. Nevýhodou je, že jednotky na okrajích mají menší vliv na výstup. Poslední způsob používá dostatečně velkou výplň, aby na každou jednotku bylo aplikováno stejné množství konvolucí. Tento přístup však může mít problém se zajištěním, aby konvoluční jádra fungovala správně na každé pozici.

Řidké interakce

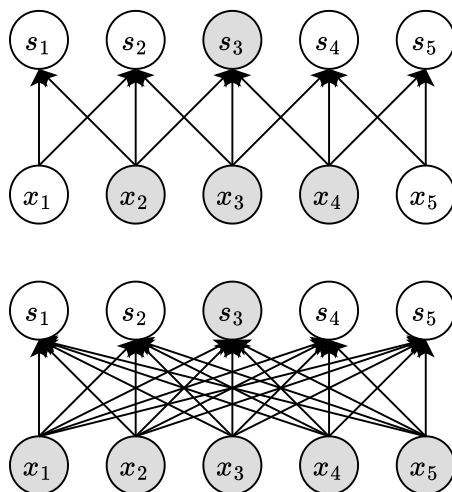
V tradičních neuronových sítích je každá vstupní jednotka propojena s každou výstupní jednotkou. V CNN je díky menší velikosti jádra oproti vstupu dosaženo úspory paměti a vyšší efektivity výpočtů. Obrázky 3.3 a 3.4 zobrazují propojení jednotek v běžných neuronových sítích ve srovnání s CNN. Pokud je v CNN zapojeno více vrstev za sebou, dochází k složitějším interakcím mezi jednotkami a jednotky, které přímo neinteragují, mohou navzájem interagovat nepřímo. Tento koncept je znázorněn na obrázku 3.5.

Sdílení parametrů

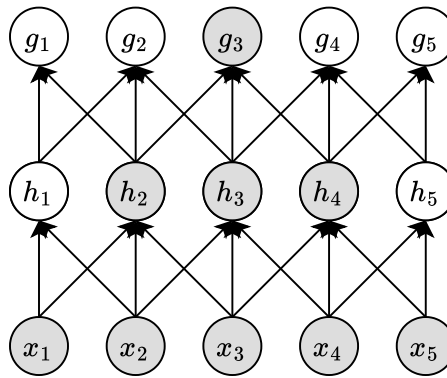
CNN se od tradičních liší v tom, že parametry nejsou využity pouze jednou při jejich aplikaci na výstup jedné jednotky, který vstupuje do druhé jednotky, ale každý parametr je



Obrázek 3.3: Obrázek ilustruje řídkce propojené vrstvy v horní části a plně propojené vrstvy ve spodní části. Prvky následující vrstvy, které jsou ovlivněny prvky předchozí vrstvy, jsou zvýrazněny šedou barvou. Převzato z [11].

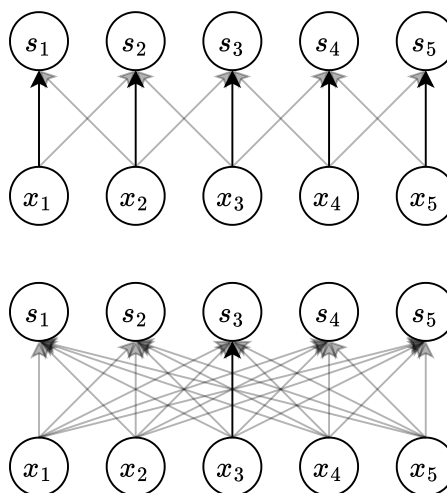


Obrázek 3.4: Obrázek ilustruje řídkce propojené vrstvy v horní části a plně propojené vrstvy ve spodní části. Prvky předchozí vrstvy, které ovlivňují prvek následující vrstvy, jsou zvýrazněny šedou barvou. Převzato z [11].



Obrázek 3.5: Obrázek ilustruje řídké, sekvenční propojení třech vrstev neuronové sítě. Prvky, které spolu navzájem interagují, jsou vyznačeny šedou barvou. Převzato z [11].

využít vícekrát. Tím, že se učíme pouze jednu sadu parametrů místo více sad pro každou oblast modelu, dochází k radikální úspoře paměti. Obrázek 3.6 znázorňuje koncept sdílení parametrů.



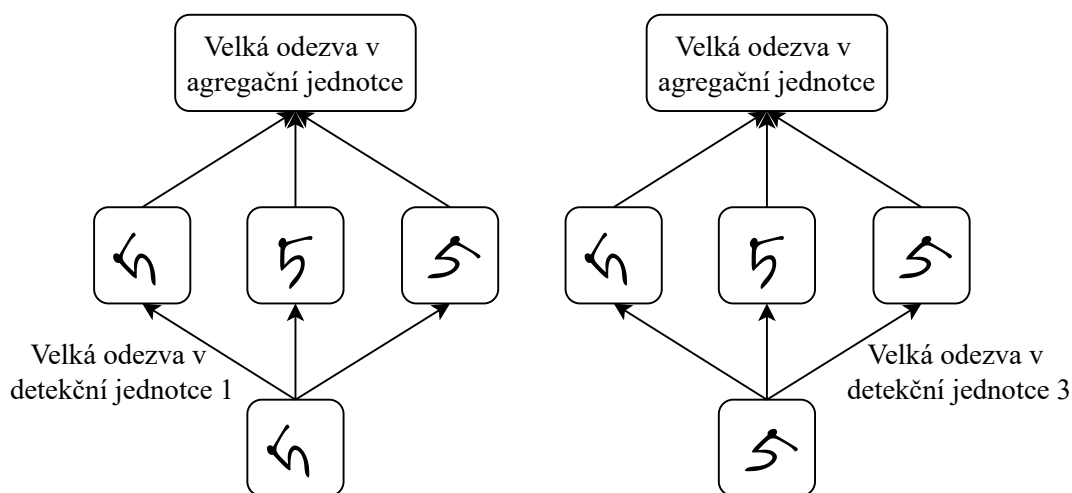
Obrázek 3.6: Obrázek demonstruje vícenásobné využití jednoho parametru v konvolučních vrstvách v horní části obrázku a jeho použití pouze na jednom místě v plně propojených vrstvách v dolní části obrázku. Černou barvou je zvýrazněn tentýž parametr. Převzato z [11].

Ekvivariance

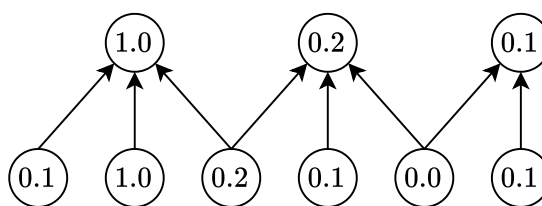
Díky sdílení parametrů v konvolučních neuronových sítích vzniká vlastnost ekvivariance vůči posunutí. To znamená, že pokud se na vstupní data aplikuje posunutí, pak bude i výstup této neuronové sítě posunutý odpovídajícím způsobem. Jinými slovy, výstup bude stejný bez ohledu na to, zda se nejprve aplikuje posunutí a pak výpočty neuronové sítě, nebo naopak. Například při zpracování obrázků dochází v první vrstvě k detekci hran a tyto vlastnosti jsou následně sdíleny v pozdějších vrstvách, což umožňuje detekci objektu bez ohledu na jeho posunutí.

Agregace

Agregace (pooling) je v konvolučních neuronových sítích operace, která se obvykle vykonává v rámci vrstvy dané sítě. Po provedení konvoluce a aplikaci nelineární aktivační funkce na každý prvek výstupu je následně provedena agregace. AgregáčnÍ funkce slouží k nahrazení výstupů určité oblasti neuronové sítě souhrnnou statistikou. Existuje mnoho druhů těchto funkcí. Například max pooling vrací největší číslo z určité obdelníkové oblasti, zatímco average pooling vrací průměr hodnot z této oblasti. Díky agregáčnÍm jednotkám se model stává invariantním vůči malému posunutí. Pokud se rysy ve vstupních datech opakují s malým posunutím, může mít tato vlastnost pozitivní vliv na efektivitu sítě. Pokud je agregace použita na výstupy separovaně parametrizovaných konvolucí, pak se model může naučit být invariantní vůči určité transformaci. Tento princip je ilustrován na obrázku 3.7. Díky vlastnosti agregace, která umožňuje získávat souhrnné statistiky z určitého okolí, není nutné získávat tyto statistiky z okolí od sebe vzdálených jednu jednotku, ale je možné je získávat z okolí vzdálených k jednotek. Tento koncept je ilustrován na obrázku 3.8. Tato skutečnost vede k tomu, že další vrstva sítě má přibližně k krát méně vstupů ke zpracování, což vede k výraznému zlepšení výpočetní efektivy a snížení paměťových nároků modelu. Proměnný posun mezi agregovanými oblastmi umožňuje modelu adaptovat se na vstupy různých velikostí.



Obrázek 3.7: Příklad separovaně parametrizovaných konvolucí naučených pro detekci ručně psaného čísla pět. Každý filtr je konstruován tak, aby zachytil mírně odlišnou orientaci tohoto čísla. Díky použití jednotky max pooling, která generuje vysokou odezvu bez ohledu na to, který z filtrů byl aktivován, je síť invariantní vůči rotaci.



Obrázek 3.8: Ilustrace agregace vrstvy konvoluční neuronové sítě založené na výběru největší hodnoty z regionu 3×1 . Převzato z [11].

Kapitola 4

Evoluce umělých neuronových sítí

Evoluční algoritmy lze použít při návrhu neuronových sítí na třech úrovních. Na první úrovni se nachází evoluce vah, která je zejména užitečná v systémech, kde je učení pomocí gradientních algoritmů obtížné. Mezi takové systémy patří například rekurentní neuronová síť (Recurrent Neural Network, RNN). Na druhé úrovni se nachází evoluce architektur, což umožňuje síti adaptovat se na různé problémy a eliminuje potřebu lidské intervence. Na poslední úrovni je evoluce učících algoritmů. Tato podkapitola byla převzata z [31] [15].

4.1 Evoluce vah

V mnoha ANN se pro učení využívá zpětná propagace založená na gradientním sestupu, při které dochází k iterativní úpravě vah s cílem minimalizovat chybovou funkci. Ačkoli je tento přístup v současné době nejčastěji používaný, nese si určité nedostatky. Algoritmy založené na gradientním sestupu mají tendenci uvíznout v lokálním minimu. Alternativním přístupem k učení je využití evolučních algoritmů. Tyto algoritmy jsou schopny prozkoumat prostory, které jsou obrovské. U evolučních algoritmů nehrozí uvíznutí v lokálním minimu. Nepotřebují informace o gradientu, který může být obtížné určit nebo vypočítat. Evoluční algoritmy jsou nezávislé na konkrétním problému a při učení tak odpadá lidská intervence. Do chybové funkce lze snadno zahrnout regularizaci. I když mohou evoluční algoritmy hledat řešení pomaleji než algoritmy založené na gradientu, na rozdíl od nich nejsou ovlivněny počáteční inicializací vah, kde u algoritmů založených na gradientním sestupu může dojít k uvíznutí v lokálním optimu v blízkosti počátečního nastavení. Tato skutečnost vedla k myšlence kombinace EA a metod založených na gradientním sestupu. V takovém případě jsou evoluční algoritmy využity pro globální prohledávání, a když je nalezen vhodný region, jsou následně použity algoritmy založené na gradientním sestupu k dosažení téměř optimálního řešení.

4.2 Automatický návrh architektury

Architektura neuronové sítě má významný dopad na její výkon a efektivitu. Návrh této architektury je komplexní úloha, která v současnosti nemá analytické řešení a vyžaduje znalost v daném oboru. Pro dosažení optimální architektury se využívá metoda NAS, což je alternativní přístup k ručnímu návrhu, který nevyžaduje lidskou intervenci. NAS lze

chápat jako optimalizační problém, jak je vyjádřeno rovnicí 4.1

$$\begin{cases} \arg \max_A = \mathcal{L}(A, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}) \\ \text{kde } A \in \mathcal{A}, \end{cases} \quad (4.1)$$

kde \mathcal{A} reprezentuje množinu možných architektur neuronových sítí, A je jedna konkrétní architektura a $\mathcal{L}(\cdot)$ vyjadřuje kvalitu sítě na testovacích datech $\mathcal{D}_{\text{test}}$ po natrénování na trénovací sadě $\mathcal{D}_{\text{train}}$.

NAS představuje výpočetně náročný proces, který je velmi často soustředěn na optimalizaci více vzájemně konfliktních kritérií. Pro řešení tohoto problému byly navrženy tři přístupy: algoritmy založené na posilovaném učení, algoritmy založené na gradientu a algoritmy založené na evolučních algoritmech.

Metoda NAS založená na evolučních algoritmech ENAS (Evolutionary Neural Architecture Search) se liší od ostatních dvou přístupů tím, že nevyžaduje odbornou znalost v dané oblasti ani informace o gradientu a má potenciál nalézt globálně optimální řešení. V minulosti, kdy modely neuronových sítí obvykle obsahovaly malé množství skrytých vrstev, se evoluční algoritmy používaly pro evoluci vah i pro evoluci architektur těchto sítí. V současné době jsou evoluční algoritmy využívány zejména pro evoluci architektury, zatímco hodnoty vah jsou získávány pomocí metod založených na gradientu.

Prohledávaný prostor

Často enormě velký prohledávaný prostor, někdy i neomezený, zahrnuje všechny možné architektury, které jsou předmětem zkoumání. Neuronové sítě jsou obvykle složeny ze základních jednotek a propojení mezi nimi. Konfigurací těchto jednotek, uspořádáním spojení mezi nimi a určením dalších parametrů lze specifikovat strukturu prohledávaného prostoru. Na prohledávaný prostor je vhodné aplikovat omezení, která mohou zlepšit efektivitu algoritmu. Omezení se nejčastěji týkají pevně definované hloubky sítí, částečně pevně definované architektury a sestavení počáteční populace pomocí známých architektur. Prohledávaný prostor lze klasifikovat podle použitých základních jednotek. Mezi nejběžnější přístupy patří prostory založené na vrstvách, blocích nebo buňkách.

Prostor založený na vrstvách bere v úvahu klasické vrstvy neuronových sítí, jako jsou konvoluční, agregační a plně propojené vrstvy, jako základní jednotky. V rámci tohoto přístupu jsou běžně aplikována omezení. Například agregační vrstvy mohou následovat po konvolučních vrstvách a plně propojené vrstvy mohou následovat po agregačních vrstvách. Tento přístup může být neefektivní pro velké neuronové sítě obsahující stovky vrstev, neboť velikost prohledávaného prostoru roste exponenciálně s počtem vrstev. Dalším problémem je, že v závislosti na zvoleném způsobu kódování nemusí být možné modelovat skip connections, které se ukázaly jako jedna z klíčových komponent ručně navržených sítí.

Prostor založený na blocích předpokládá, že základními jednotkami jsou bloky složené z tradičních vrstev neuronové sítě s topologickými vztahy navrženými odborníky tak, aby dosahovaly dobrého výkonu. Modely postavené na tomto přístupu jsou obecně výkonnější. Efektivita evolučních algoritmů je též výrazně vyšší zejména u hlubokých modelů, neboť jednotkami v tomto přístupu jsou komplexní bloky složené z více základních vrstev, což snižuje komplexitu prohledávaného prostoru ve srovnání s přístupem založeným na vrstvách.

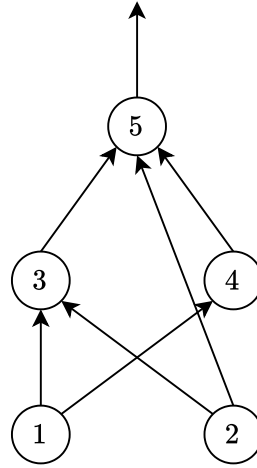
Prostor založený na buňkách lze rozdělit na mikro a makro část. Mikro část se zaměřuje na topologické vztahy v rámci základní jednotky nazývané buňky, zatímco makro část se soustředí na topologické vztahy mezi těmito buňkami. V přístupu založeném na blocích jsou bloky předem definované a prohledávání probíhá pouze v makro části. Obecně se

přístup založený na buňkách zaměřuje zejména na evoluci mikro části, což představuje opak přístupu založeného na blocích.

Způsob reprezentace jedince

Určuje, jakým způsobem je reálný problém zakódován do genotypu jedince. Existují dva základní přístupy kódování jedince: přímé kódování, které uchovává veškeré informace a nepřímé kódování, které se zaměřuje na omezené informace. Dalším kritériem pro rozlišování způsobů kódování je pevná a proměnlivá délka reprezentace jedince.

Přímé kódování umožňuje každé spojení uzlu zakódovat do genotypu jedince. Architekturu neuronové sítě obsahující N uzlů lze například reprezentovat pomocí matice $C = (c_{i,j}) \in \mathbb{R}^{N \times N}$, kde prvek $c_{i,j} \in \{0, 1\}$ označuje existenci, respektive neexistenci, spojení mezi uzlem i a j . Existuje přístup, kde jsou prvky $c_{i,j} \in \mathbb{R}$ pro evoluci architektury a vah současně. Tento přístup umožňuje mapování C jedna ku jedné na korespondující architekturu neuronové sítě. Konkatenací řádků matice vznikne binární řetězec, který představuje genotyp jedince. Na obrázku 4.1 je možné vidět zobrazení architektury dopředné neuronové sítě. K této architektuře odpovídá matice C , která je vyjádřena vztahem 4.2.



Obrázek 4.1: Ilustrace architektury neuronové sítě reprezentované genotypem matice popsané vztahem 4.2. Převzato z [31].

$$C = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (4.2)$$

U dopředných neuronových sítí je možné zapsat genotyp jedince pomocí zkráceného zápisu, protože tyto sítě využívají pouze horní trojúhelníkovou matici. Zkrácený zápis genotypu je dán vztahem 4.3.

$$0110 \ 101 \ 01 \ 1 \quad (4.3)$$

Pomocí tohoto kódování je možné zakódovat i RNN. V takovém případě však není možné použít zkrácený zápis a genotyp jedince je reprezentován prostou konkatenací řádků matice C . Toto kódování je jednoduché na implementaci a umožňuje nalezení přesného řešení,

protože je možné modifikovat spojení mezi jednotlivými uzly. Tento přístup také poskytuje velkou flexibilitu v definici fitness funkce, která může zahrnovat kromě chyby na datech také čas potřebný k natrénování sítě nebo počet uzlů a spojení mezi nimi. Problémem této reprezentace je, že pro velké neuronové sítě je zapotřebí vytvořit velmi rozsáhlé matice, které je reprezentují.

Nepřímé kódování eliminuje délku genotypu. V takovémto případě jsou do genotypu zahrnuty jen omezené charakteristiky. Detaily jsou buď pevně stanoveny pomocí znalostí v dané oblasti, nebo jsou zakódovány pomocí pravidel, které se v průběhu evoluce vyvíjí. Nepřímé kódování zahrnuje několik typů reprezentace. Jedním z přístupů je parametrická reprezentace, kde architektura je specifikována sadou parametrů, jako je například počet skrytých vrstev, počet uzlů v každé vrstvě nebo například počet spojení mezi vrstvami. Tato reprezentace umožňuje definovat vzor architektury, přičemž detailní spojení mezi uzly je specifikováno implicitními pravidly. Použití tohoto přístupu umožňuje prozkoumat pouze omezený prostor architektur a je vhodné ho použít v případě, kdy je známý typ architektury pro řešení daného problému. Reprezentace vývojových pravidel je další variantou nepřímého kódování. Tento přístup umožňuje zakódovat všechna pravidla do jednoho jedince, nebo každé pravidlo do jednoho jedince. Matice udávající konektivitu mezi uzly je postupně generována pomocí pravidel na zprvu jednoprvkovou matici, dokud matice neobsahuje pouze terminální symboly. Příkladem pravidel pro konstrukci genotypu je předpisem 4.4.

$$\begin{aligned}
S &\rightarrow \begin{bmatrix} A & B \\ C & D \end{bmatrix} \\
A &\rightarrow \begin{bmatrix} a & a \\ a & a \end{bmatrix} & B &\rightarrow \begin{bmatrix} i & i \\ i & a \end{bmatrix} & C &\rightarrow \begin{bmatrix} i & a \\ a & c \end{bmatrix} & D &\rightarrow \begin{bmatrix} a & e \\ a & e \end{bmatrix} \\
a &\rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & c &\rightarrow \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} & e &\rightarrow \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} & i &\rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}
\end{aligned} \tag{4.4}$$

$$S \rightarrow \begin{bmatrix} A & B \\ C & D \end{bmatrix} \rightarrow \left[\begin{bmatrix} a & a \\ a & a \\ i & a \\ a & c \end{bmatrix} \quad \begin{bmatrix} i & i \\ i & a \\ a & e \\ a & e \end{bmatrix} \right] \rightarrow \left[\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \right] \tag{4.5}$$

Reprezentace s pevnou délkou je způsobem kódování, při kterém mají všichni jedinci během evolučního procesu stejnou délku zakódování. Tento přístup umožňuje využití standardních operátorů mutace a křížení během evoluce. Nevýhodou je předem určená hloubka sítě, jejíž stanovení vyžaduje rozsáhlé znalosti v dané oblasti. Tato omezení vedou k tomu, že ENAS není zcela automatizovaný.

Reprezentace s proměnlivou délkou umožňuje zakódovat jedince různých délek. Na rozdíl od reprezentace s pevnou délkou, toto kódování umožňuje adaptaci hloubky neuronové sítě a nalezení řešení blízkých optimálním bez nutnosti hlubších znalostí pro určení pevné délky genotypu. Kvůli proměnlivé délce však nelze použít standardní metody křížení a může dojít k významnému snížení efektivity ENAS kvůli vzniku nadměrně velkých jedinců, jejichž vyhodnocení je náročné.

Počáteční populace

Existují tři způsoby inicializace architektur v počáteční populaci. Prvním přístupem je inicializace pomocí triviálních počátečních podmínek, kdy jsou vytvořeni jedinci obsahující pouze základní vrstvy. Tento přístup, díky zahrnutí co nejmenšího množství znalostí, umožňuje objevení zcela nových architektur, které se odlišují od ručně navržených. Druhým způsobem je inicializace pomocí známých architektur, což umožňuje získání výkonných architektur na začátku evoluce, avšak vzácně vede k objevu nových architektur stejné kvality. Posledním způsobem je inicializace pomocí náhodně vybraných architektur z prohledávaného prostoru, což minimalizuje potřebu lidské intervence.

Prohledávací strategie

Definuje, jakým způsobem bude prozkoumáván prohledávaný prostor. Výběr strategie obvykle zahrnuje způsob selekce jedinců a evoluční operátory, jež jsou odpovědné za aktualizaci populace. V kontextu ENAS jsou nejčastěji využívány genetické algoritmy, genetické programování a evoluční strategie.

Vyhodnocení výkonnosti jedinců

Hodnocení výkonnosti architektury je obvykle prováděno vyhodnocením přesnosti na testovací sadě za předpokladu, že model byl natrénován na trénovací sadě. Nalezení optimálních parametrů sítě lze chápat jako optimalizační problém, který lze vyjádřit rovnicí 4.6

$$\arg \max_{\theta_A} E_{\text{train}}(A, \theta_A, D_{\text{train}}) \quad (4.6)$$

kde θ_A představuje parametry architektury A a E_{train} vyjadřuje výkon sítě na trénovací sadě D_{train} .

Poté lze vyhodnocení výkonnosti architektury na testovací sadě D_{test} zapsat rovnicí 4.7

$$E_{\text{fitness}}(A, \arg \max_{\theta_A} E_{\text{train}}(A, \theta_A, D_{\text{train}}), D_{\text{test}}). \quad (4.7)$$

Metody hodnocení výkonnosti modelů obvykle spadají do čtyř kategorií: trénování od nuly, trénování založené na hrubé aproximaci, ukládání hodnot fitness funkce a dědičnost vah.

Trénování od nuly představuje nejpřesnější metodu, kde je jedinec trénován na trénovací sadě a následně se evaluován na testovací sadě. Tato metoda je sice velmi přesná, avšak v praxi se vzhledem k vysokým výpočetním nárokům využívá velmi málo.

Metody hrubé aproximace při trénování zahrnují přístupy, které předpokládají, že výkon architektury na zjednodušené náhradní úloze je srovnatelný s výkonem na původním problému. Tato metodika je často využívána pro efektivní vyhodnocení výkonnosti modelů. Jednou z těchto metod je brzké zastavení trénování neuronové sítě, která předpokládá, že pořadí modelů natrénovaných pouze po omezený počet epoch koreluje s pořadím výkonnosti těchto modelů, které jsou kompletně natrénované. Dalším přístupem je využití podmožiny nebo alternativní datové sady pro trénink modelu.

Metoda ukládání hodnot fitness funkce je často využívána v evolučních algoritmech s elitismem, kde jsou z populace uchovávány nejlepší jedinci a jejich výkon není potřeba opakovaně vyhodnocovat, protože je dostupný z uložených hodnot.

Metoda dědičnosti vah využívá informace o váhách již natrénovaných modelů pro inicializaci parametrů sítě s podobnou architekturou.

Vícekriteriální optimalizace

Pokud jsou metody ENAS využity k optimalizaci více kritérií, jako jsou například paměťové nároky nebo latence modelu, mohou nalézt architektury, které se přibližují kvalitě nejlepších dostupných technologií. Neuronová síť typicky obsahuje miliony až miliardy parametrů, z nichž je každý obvykle minimálně jednou využit při zpracování vstupních dat. Snížení paměťových nároků a latence neuronové sítě je zásadní zejména pro mobilní telefony a IoT (Internet of Things) zařízení.

Pro různá zařízení jsou optimální různé architektury. Původní přístupy byly zaměřeny na optimalizaci klasifikace neuronové sítě spolu s počtem parametrů, počtem operací s plovoucí řádovou čárkou za sekundu FLOPS (Floating-Point Operations per Second, FLOPS) nebo počtem operací MAC (Multiply-and-Accumulate). Tyto parametry jsou jednoduché na vyhodnocení u každé neuronové sítě. Pokročilým přístupem je optimalizace latence pro konkrétní architekturu. Existuje několik přístupů, z nichž jeden využívá metodu založenou na LUT (Lookup Table), kde jsou uloženy latence a další parametry pro každou operaci, získané buď měřením nebo z technické dokumentace. Na základě těchto údajů je pak odhadnuta celková latence modelu pro konkrétní architekturu.[24].

4.3 Evoluce algoritmů učení

Efektivita algoritmů použitých pro učení neuronových sítí často závisí na architektuře těchto sítí. Správný výběr algoritmu je závislý na znalostech o této architektuře. V případě nedostatku znalostí o dané architektuře lze využít evolučních algoritmů k nalezení vhodného algoritmu pro danou architekturu. Tento přístup umožňuje adaptaci učícího algoritmu na konkrétní architekturu a řešený problém. Výzkum v této oblasti je důležitý z hlediska možnosti nalezení nových algoritmů, které mohou být efektivní v komplexních a dynamických prostředích. Pokud je architektura, sítí na kterých je prováděno testování účinnosti daného algoritmu fixní, pak je algoritmus učení optimalizován právě pro tuto architekturu. Při hodnocení účinnosti na různých typech sítí je důležité provést průměr výsledků, aby nedošlo k přílišnému zaměření algoritmu na určitou architekturu.

Kapitola 5

Návrh řešení

Cílem této kapitoly je návrh klasifikátoru obrazu pomocí CNN, která bude automatizovaně získaná pomocí EA. V oblasti hlubokého učení došlo v posledních letech k významnému pokroku v řešení problémů strojového učení, především v oblasti rozpoznávání obrazu s využitím CNN. Úroveň přesnosti, s jakou CNN řeší daný problém, závisí na konkrétní architektuře sítě. CNN se typicky skládají z konvolučních, agregujících a plně propojených vrstev. Architektura je popsána sadou parametrů, jako je hloubka sítě, propojení vrstev a parametry jednotlivých vrstev. Identifikace vhodné architektury je náročný proces, který vyžaduje odborné znalosti v dané oblasti, nebo uchýlení se k přístupu pokus-omyl. Evoluční algoritmy jsou široce využívány v oblasti ANN, zejména pro návrh jejich architektur. Díky návrhu architektur pomocí EA odpadá nutnost lidské intervence. Existují dva přístupy k reprezentaci sítě v evolučních algoritmech – přímé kódování popisující uzly a jejich spojení a nepřímé kódování, které uchovává informace o pravidlech pro generování architektury sítě. V této práci je použito nepřímého kódování. Jedinec je reprezentován grafem CGP, což umožňuje variabilní délku a větvení neuronových sítí. Cílem je architekturu sítě optimalizovat z pohledu počtu parametrů a přesnosti klasifikace na testovací sadě. Pro nalezení řešení je využit NSGA-II [31].

5.1 Zakódování jedince

Architektura CNN obvykle zahrnuje konvoluční, agregační a plně propojené vrstvy. Parametry architektury jako je hloubka neuronové sítě, typy a parametry jednotlivých vrstev a spojení mezi nimi, činí vývoj takové architektury složitým procesem, který vyžaduje buď odborné znalosti, nebo systematické zkoušení výkonu různých architektur. V případě nedostatku odborných znalostí je často využíváno metod založených na evolučních algoritmech pro automatický návrh neuronových sítí. Jako prostředek pro nepřímé zakódování architektury neuronové sítě byla v této práci použita metoda kódování jedince pomocí DAG, která vychází z techniky používané CGP. Vzhledem ke složitosti této architektury nejsou jednotlivé prvky sítě a jejich spojení zakódovány přímo do genotypu, ale uzly grafu tvoří moduly, jako jsou konvoluční, konkatenční a agregační vrstvy. Tato metoda poskytuje vysokou flexibilitu při návrhu architektury, umožňuje snadno upravit hloubku neuronové sítě, modelovat její větvení a vynechat určité vrstvy. Zatímco počet vrstev v neuronové síti může být proměnlivý, délka genotypu zůstává konstantní. Kromě konvolučních a agregačních vrstev jsou do návrhu architektury zahrnuty také vrstvy pro konkatenaci a sčítání. Tato podkapitola byla převzata z [28].

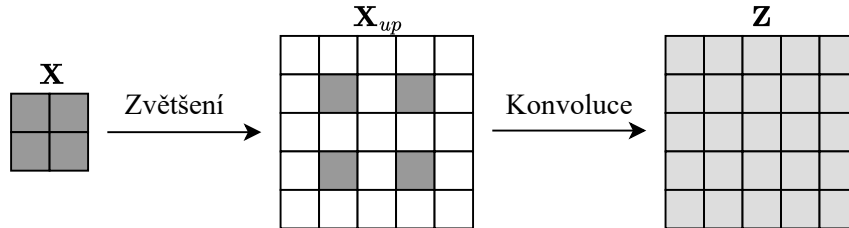
Transponovaná konvoluční vrstva

V CNN se kromě výše zmíněných tradičních vrstev může nacházet vrstva transponované konvoluce, která operuje s definovaným krokem. Tato vrstva představuje další základní komponentu CNN. Na rozdíl od standardní konvoluce, která redukuje prostorový rozměr kanálů vstupního tenzoru v závislosti na zvoleném kroku, transponovaná konvoluce zvětšuje prostorový rozměr kanálů vstupního tenzoru. Původní přístupy využívaly interpolaci, která však, na rozdíl od transponované konvoluce, neobsahuje váhy, jež by mohly být modifikovány algoritmem učení. Metody nezahrnující učení však zaostávají za metodami, které jsou schopné učení.

Transponovaná konvoluce je varianta klasické konvoluce. Parametry transponované konvoluce zahrnují velikost konvolučního jádra \mathbf{K} , krok \mathbf{S} , počet vstupních kanálů C_{in} , počet výstupních kanálů C_{out} , dilataci \mathbf{D} a výplň \mathbf{P} . Nejprve je na vstup $\mathbf{X} \in \mathbb{R}^{C_{in} \times H \times W}$, kde C_{in} představuje počet vstupních kanálů, H výšku kanálu a W šířku kanálu, aplikována operace, která zvětší vstup vyplněním nulami, takže rozměr výstupu je $\mathbf{X}_{up} = \mathbb{R}^{C_{in} \times H_{up} \times W_{up}}$, kde $H_{up} = (H - 1) \cdot \mathbf{S}_0 - 2 \cdot \mathbf{P}_0$ a $W_{up} = (W - 1) \cdot \mathbf{S}_1 - 2 \cdot \mathbf{P}_1$. Pokud platí $\mathbf{P} = [0 \ 0]$, pak jsou prvky tenzoru \mathbf{X}_{up} dány rovnicí (5.1).

$$X_{up}[c, i, j] = \begin{cases} X[c, \lfloor i/\mathbf{S}_0 \rfloor, \lfloor j/\mathbf{S}_1 \rfloor], & \text{pro } (i \bmod \mathbf{S}_0 = 0) \text{ a } (j \bmod \mathbf{S}_1 = 0) \\ 0, & \text{jinak.} \end{cases} \quad (5.1)$$

Na tenzor \mathbf{X}_{up} je následně aplikována klasická konvoluce se symetrickým krokem $\mathbf{S} = [1 \ 1]$ a výstupem je tenzor $\mathbf{Z} = \mathbb{R}^{C_{out} \times H_{out} \times W_{out}}$, kde $H_{out} = (H - 1) \cdot \mathbf{S}_0 - 2 \cdot \mathbf{P}_0 + \mathbf{D}_0 \cdot (\mathbf{K}_0 - 1) + 1$ a $W_{out} = (W - 1) \cdot \mathbf{S}_1 - 2 \cdot \mathbf{P}_1 + \mathbf{D}_1 \cdot (\mathbf{K}_1 - 1) + 1$ [14]. Operace transponované konvoluce je znázorněna na obrázku 5.1.

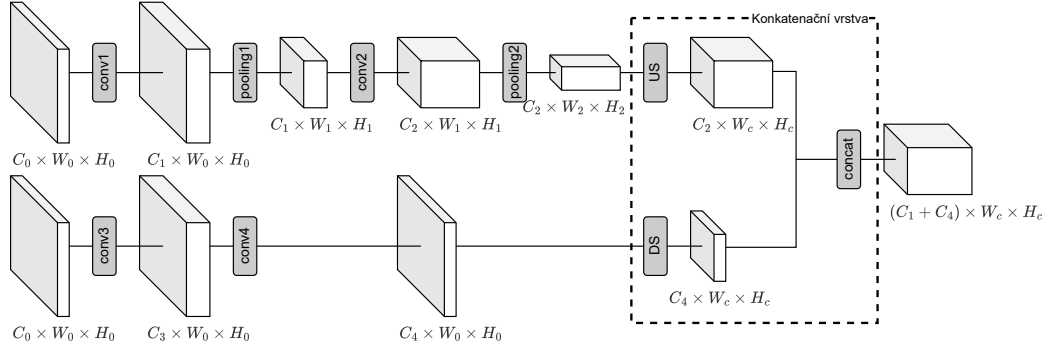


Obrázek 5.1: Ilustrace aplikace operace transponované konvoluce na matici \mathbf{X} . Matice \mathbf{X}_{up} zobrazuje výsledek zvýšení rozlišení s parametry $\mathbf{S} = [2 \ 2]$ a $\mathbf{P} = [1 \ 1]$. Matice \mathbf{Z} ukazuje výsledek operace konvoluce aplikované na matici \mathbf{X}_{up} .

Konkatenáční vrstva

Konkatenace je binární operace, která umožňuje kanálově sloučit výstupy větví neuronové sítě. Vstupy do této operace jsou tenzory $\mathbf{X} \in \mathbb{R}^{C_0 \times H_0 \times W_0}$ a $\mathbf{Y} \in \mathbb{R}^{C_1 \times H_1 \times W_1}$. Pro provedení konkatenace je nezbytné, aby výška a šířka kanálů tenzorů vstupujících do operace byla stejná. Dochází k určitému zjednodušení tím, že vstupní tenzor neuronové sítě má symetrickou výšku a šířku kanálů a během dopředného průchodu sítě jsou použity operace se symetrickým krokem a velikostí jádra. Výstupní tenzor také sdílí tuto vlastnost. Pokud mají vstupní tenzory různou velikost kanálů, použije se operace zmenšení rozlišení kanálu pomocí agregační vrstvy na tenzor s vyšší šířkou a výškou kanálu, a operace zvýšení rozlišení pomocí transponované konvoluce na tenzor s nižší šířkou a výškou kanálu. Parametry

těchto operací jsou nastaveny tak, aby výstupní tenzory měly rozměry $\mathbf{X}_{out} \in \mathbb{R}^{C_0 \times H_c \times W_c}$ a $\mathbf{Y}_{out} \in \mathbb{R}^{C_1 \times H_c \times W_c}$. Na rozdíl od vyžadované rovnosti ve velikosti kanálů mezi vstupními tenzory není tato podmínka nutná pro počet kanálů. Po provedení konkatenace na tenzory \mathbf{X}_{out} a \mathbf{Y}_{out} je výstupem tenzor $\mathbf{Z} \in \mathbb{R}^{(C_0+C_1) \times H_c \times W_c}$. Ilustrace konkatenace tenzorů dvou větví konvoluční neuronové sítě je zobrazena na obrázku 5.2.



Obrázek 5.2: Konkatenace dvou větví neuronové sítě. Prvky **conv1** až **conv4** ilustrují konvoluční vrstvy. Prvky **pooling1** a **pooling2** znázorňují agregační vrstvy. Prvek **US** představuje zvětšení rozlišení kanálů (upsampling, US), zatímco prvek **DS** zmenšuje rozlišení kanálů (downsampling, DS). Prvek **concat** provádí kanálovou konkatenaci tenzorů. Konkatenací vrstva je ohraničena čerchovanou oblastí.

Sčítací vrstva

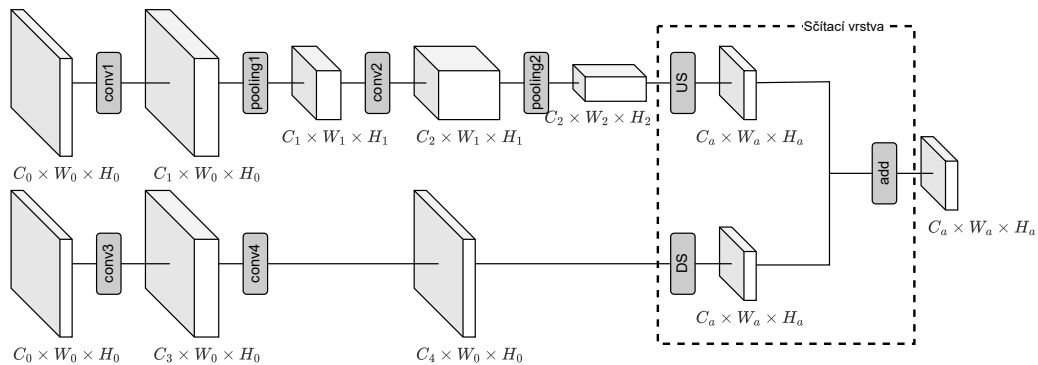
Stejně jako konkatenace i sčítací vrstva je binární operací, která slouží k sečtení prvků výstupních tenzorů dvou větví neuronové sítě. Na rozdíl od konkatenace však sčítací vrstva vyžaduje, aby vstupní tenzory měly nejen stejnou velikost kanálů, ale také stejný počet kanálů. Pro úpravu velikosti tenzorů je stejně jako u konkatenace využívána agregační vrstva pro zmenšení rozlišení a transponovaná konvoluce pro zvětšení rozlišení, kde počet výstupních kanálů odpovídá počtu kanálů v druhém tenzoru. Na takto upravené tenzory $\mathbf{X}_{out} \in \mathbb{R}^{C_a \times H_a \times W_a}$ a $\mathbf{Y}_{out} \in \mathbb{R}^{C_a \times H_a \times W_a}$ je aplikována operace sčítání, která je popsána rovnicí 5.2. Sčítání tenzorů dvou větví CNN je ilustrováno obrázkem 5.3.

$$Z[c, i, j] = X_{out}[c, i, j] + Y_{out}[c, i, j] \quad \text{pro } 0 \leq c < C_a, 0 \leq i < H_a, 0 \leq j < W_a. \quad (5.2)$$

Omezení prohledávaného prostoru

Mnoho architektur neuronových sítí se inspiroje obecnými principy. V klasických architekturách jsou často používány konvoluční vrstvy, které jsou následovány agregačními vrstvami v opakujícím se vzoru. Poslední část sítě pak tvoří několik plně propojených vrstev, jež se snaží klasifikovat obraz do n tříd. Běžnou praxí je také umístit za agregační vrstvu dvojici konvolučních vrstev, což umožňuje extrakci složitějších vzorů. Při použití konvoluce je užitečné doplňovat okraje kanálů nulami, aby nedocházelo k redukci rozměrů kanálů během konvolučního procesu [21]. Poslední plně propojená vrstva neuronové sítě není nutně zahrnována v genotypu, protože je pevně stanovená a není součástí prohledávání.

Aby se síť zakódovaná pomocí DAG využívaného CGP inspirovala obecnými vzory pro architekturu CNN, jsou na uzly grafu s propojením mezi nimi kladena následující omezení:



Obrázek 5.3: Sčítání dvou větví neuronové sítě. Prvky **conv1** až **conv4** ilustrují konvoluční vrstvy. Prvky **pooling1** a **pooling2** znázorňují agregační vrstvy. Prvek **US** představuje zvětšení rozlišení kanálů (upsampling, US), zatímco prvek **DS** zmenšuje rozlišení kanálů (downsampling, DS). Prvek **add** provádí sčítání tenzorů po prvcích. Sčítací vrstva je ohrazena čerchovanou oblastí.

- V každém sloupci se nachází uzly reprezentující vrstvy právě z jedné z těchto tříd: konvolučních vrstev, agregačních vrstev, konkatenčních a sčítacích vrstev nebo plně propojených vrstev.
- Nelze přímo propojit uzly reprezentující agregační vrstvy.
- Po prvním sloupci obsahujícím uzly reprezentující plně propojené vrstvy mohou následovat sloupce reprezentující pouze plně propojené vrstvy.
- V prvním sloupci mohou být pouze uzly reprezentující prvky z třídy konvolučních vrstev.

Omezení kladená na graf kartézského genetického programování jsou ilustrována na obrázku 5.4.

5.2 Sdílení parametrů

Hledání optimální architektury neuronových sítí je často náročným úkolem, který vyžaduje efektivní výpočetní prostředky. Během procesu evolučního algoritmu jsou potomci vytvářeni pomocí replikace a mutace rodičů. V některých situacích mohou být určité části fenotypu sdíleny mezi rodiči a jejich potomky. V těchto případech je výhodné využít již naučené hodnoty parametrů sítě rodičů jako výchozí hodnoty parametrů pro síť potomků. Síť, pro jejíž inicializaci parametrů byly využity hodnoty parametrů naučených na stejné datové sadě, mají silný předpoklad na rychlejší konvergenci.

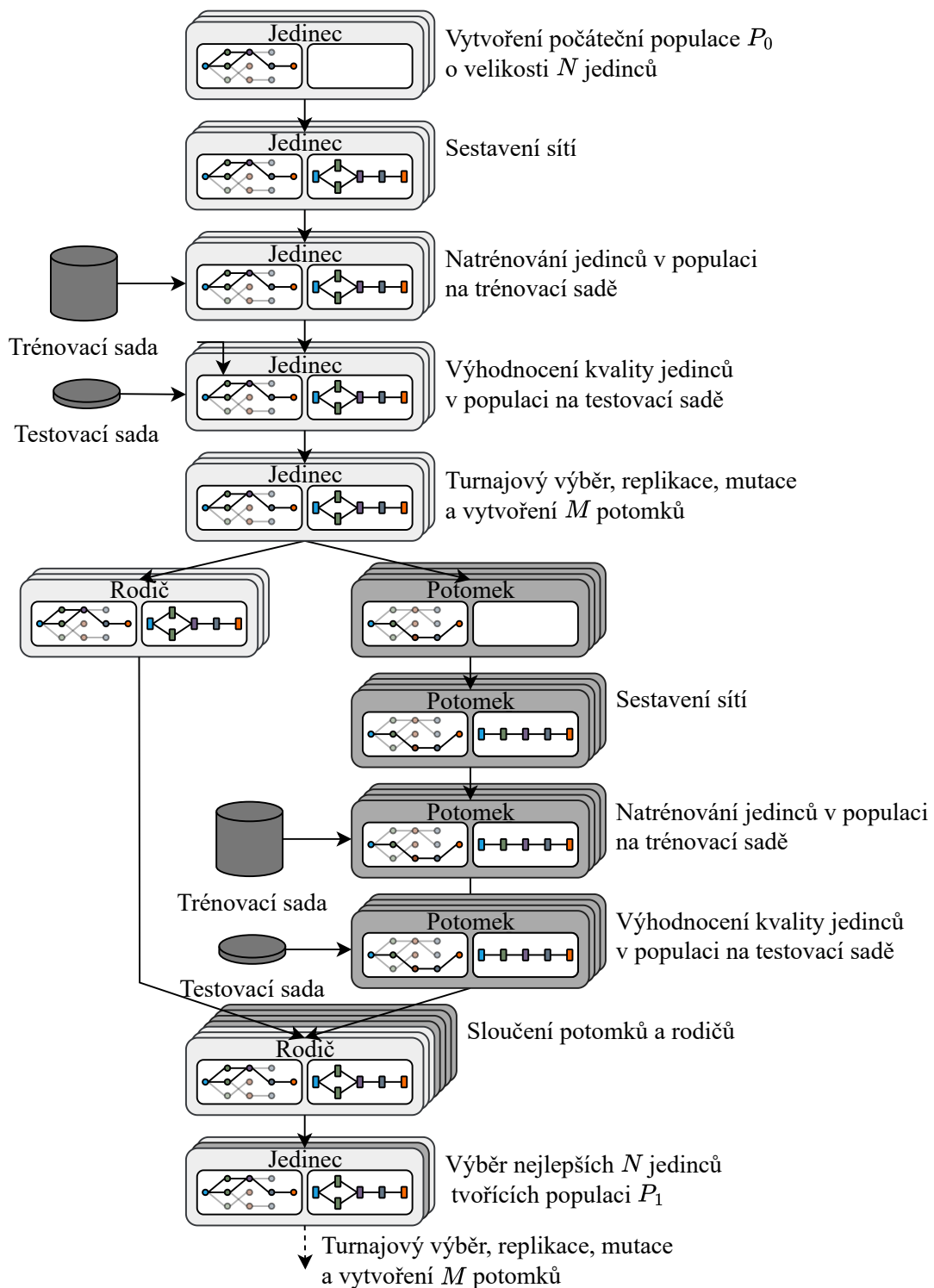
Přestože je stejný typ vrstvy využíván jak rodičem, tak potomkem, může se lišit počet parametrů, které každá vrstva využívá. V takových případech dochází k úpravě rozměrů tenzorů sdílených hodnot parametrů rodiče, buď jejich ořezáním nebo naopak doplněním náhodnými hodnotami. Tím se zajistí, že sdílené hodnoty odpovídají počtu parametrů využívaných vrstvou potomka [11].

5.3 Evoluční algoritmus

Optimalizace architektury CNN s ohledem na počet parametrů a přesnosti představuje multikriteriální optimalizační úlohu. Jedná se o dvě vzájemně protichůdné charakteristiky, a ideální architektura by neměla žádné parametry a dosahovala by 100% přesnosti na testovacích datech. Tyto charakteristiky lze snadno překonvertovat, aby se jednalo o úlohu s cílem minimalizace cílové funkce. Fitness funkce je složena z počtu parametrů CNN a chyby, kterou síť vykazuje na testovací sadě. Vzhledem k jejich konfliktní povaze je však výsledkem soubor řešení, u nichž nelze jednoznačně určit, které je lepší, bez dalších informací.

Pro nalezení těchto řešení je využito modifikovaného NSGA-II. Každý jedinec je reprezentován acyklickým orientovaným grafem, což vychází z metody kódování jedinců v CGP. V rámci CGP dosud neexistuje ustálený způsob křížení, který by byl prokázán jako užitečný. Modifikovaný algoritmus NSGA-II využívá pouze mutaci pro vytvoření nové populace potomků. Během mutace může dojít k modifikaci pouze neaktivních uzlů, což vede k jevu nazývanému genetický drift. Jedinci, u nichž došlo k mutaci pouze neaktivních uzlů, jsou nerozlišitelní na základě hodnoty fitness funkce. S ohledem na tuto skutečnost je do nové populace P_{t+1} zahrnuto vždy pouze N jedinců a nedochází k zahrnutí celé první třídy nedominovaných řešení.

Průběh evolučního algoritmu začíná vytvořením N potomků, kterými se formuje počáteční populace P_0 . Na základě genotypů těchto jedinců jsou konstruovány CNN, jejichž parametry jsou inicializovány náhodnými čísly. Tyto sítě jsou poté trénovány na trénovacích datech a vyhodnoceny na testovacích datech. Následuje turnajový výběr jedinců pro vytvoření M potomků, kteří tvoří populaci Q_0 , a to pomocí mutace. Turnajový výběr využívá operátor \prec_n pro porovnávání jedinců, jehož definice je popsána v kapitole 2.4. U potomků jsou poté zkonstruována CNN, a pokud je to možné, jsou jejich parametry převzaty od rodičů. Následuje trénování CNN potomků na trénovací sadě a vyhodnocení kvality na testovací sadě. Populace potomků a rodičů je sloučena a seřazena pomocí operátoru \prec_n sestupně. Do nové populace P_1 je vybráno prvních N jedinců. Následně probíhá turnajový výběr a vytvoření nové populace potomků Q_1 . Tento proces se opakuje pro další generace, dokud není splněna ukončující podmínka. Průběh evolučního algoritmu je ilustrován obrázkem 5.5.



Obrázek 5.5: Průběh evolučního algoritmu. V jedinci je zobrazen acyklický orientovaný graf po levé straně a odpovídající síť je ilustrována po pravé straně.

Kapitola 6

Implementace

Program byl realizován v jazyce Python, který se stal velice populárním a je hojně využíván pro implementaci aplikací v oblasti strojového učení. Python nabízí širokou škálu knihoven, z nichž byla pro projekt klíčová knihovna Pytorch, která představuje populární nástroj umožňující flexibilní konstrukci modelů a efektivní trénink neuronových sítí na GPU.

6.1 Pytorch

Hluboké učení vyžaduje provádění výpočetně náročných operací, protože neuronové sítě obsahují často milióny až miliardy parametrů. Současné CPU provádějí tyto operace neefektivně a vyžadují značné množství procesorového času. GPU, původně navržené pro renderování grafiky v počítačových hrách, se v oblasti hlubokého učení ukázaly jako velice prospěšné. Díky paralelním výpočtům je provádění složitých operací, například násobení matic, mnohonásobně rychlejší než na CPU.

Pro úspěch modelů hlubokého učení je klíčová kvalitní datová sada. Velké společnosti jako Facebook nebo Google v průběhu let nashromáždily rozsáhlé množství dat různých formátů. Vznikly tak standardizované sady, které se využívají v soutěžích pro porovnávání výkonu modelů hlubokého učení. Mezi tyto sady patří například MNIST, CIFAR, ImageNet nebo FashionMNIST.

Dříve byla pro sestavení modelů hlubokého učení nezbytná znalost C++ a CUDA. Frameworky jako PyTorch nebo TensorFlow odstraňují mnoho komplikací, které sestavování modelů hlubokého učení přináší, a usnadňují práci vývojářům.

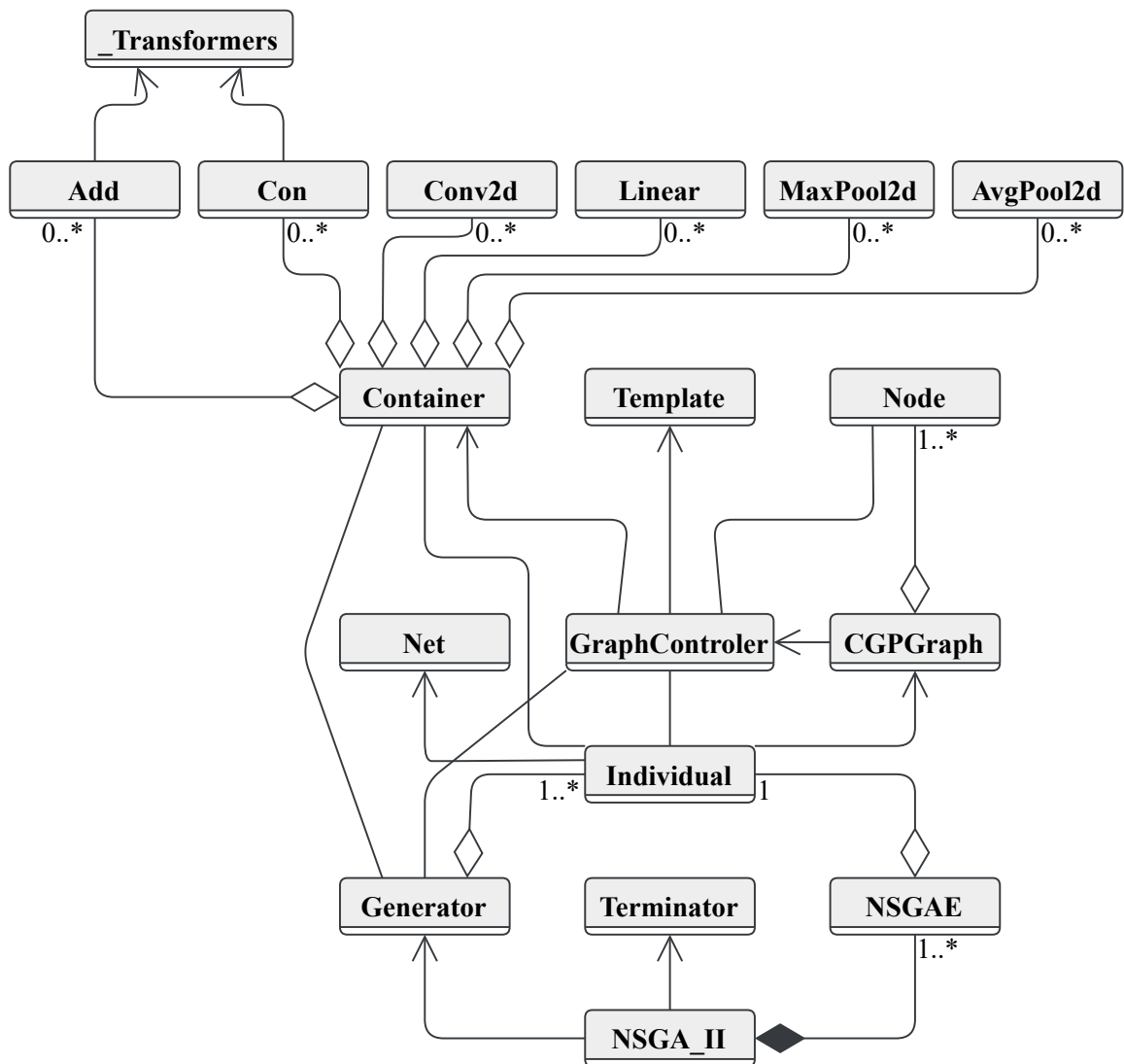
Díky jednoduchosti použití, možnosti provádět výpočty na GPU, dostupnosti standardizovaných datových sad a velké flexibilitě při sestavování architektur neuronových sítí se knihovna PyTorch stala velmi populární [26].

6.2 Struktura implementovaného řešení

Obrázek 6.1 zobrazuje strukturu implementace programu včetně vztahů mezi jednotlivými třídami.

Wrappery

Během konstrukce CNN z DAG, který reprezentuje fenotyp jedince, není možné předem určit, jaké vrstvy budou mezi sebou propojeny. V důsledku toho nelze přesně stanovit ani



Obrázek 6.1: Ilustrace struktury programu a vztahů mezi jednotlivými třídami.

velikosti tenzorů, které budou předávány jako vstup do těchto vrstev. Tento nedostatek je řešen implementací tříd `Conv2d`, `Linear`, `MaxPool2d` a `AvgPool2d`, které umožňují předvypočítat velikost výstupního tenzoru na základě specifikace vstupního tenzoru. Tyto hodnoty jsou pak využity během konstrukce neuronové sítě k dynamické specifikaci konkrétních tříd reprezentujících vrstvy neuronové sítě.

_Transformers

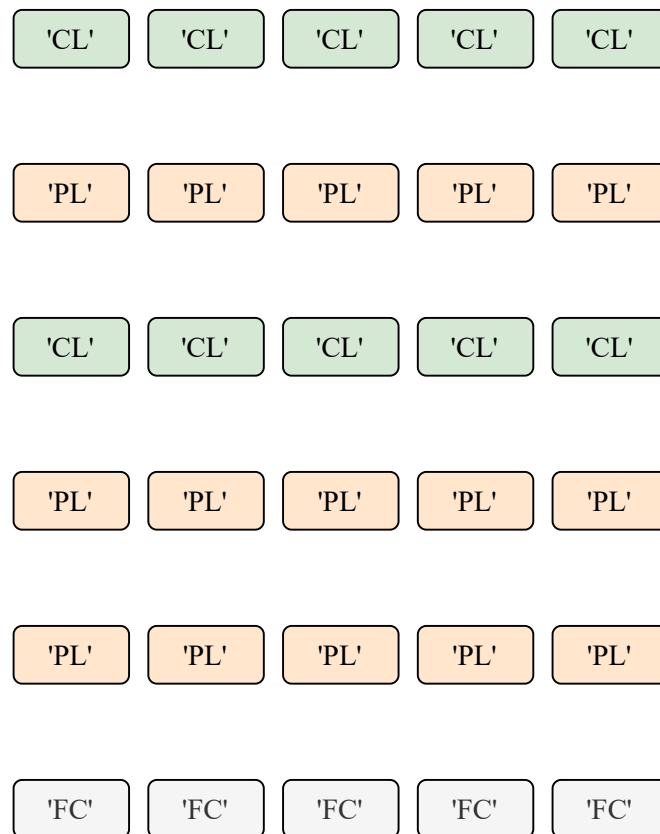
Úkolem této třídy je upravit velikosti tenzorů dvou větví neuronové sítě pro potřeby operace sčítání nebo konkatenace. Na základě specifikace vstupních tenzorů vytváří třídy reprezentující agregační vrstvu a vrstvu transponované konvoluce s vhodně nastavenými parametry tak, aby aplikace těchto vrstev na vstupní tenzory upravila jejich velikost pro potřeby operací provádějících sloučení větví.

Container

Tato třída umožňuje definovat soubor dostupných modulů, které reprezentují jednotlivé vrstvy neuronové sítě, jež se mohou potenciálně vyskytnout v architektuře navržené pomocí implementované metody.

Template

Umožňuje definovat šablonu pro graf CGP, čímž umožňuje uplatnit určitá omezení na architekturu. Správné nastavení může přispět k rychlejší konvergenci metody a k nalezení optimálních řešení. Ilustrace této šablony je uvedena na obrázku 6.2.



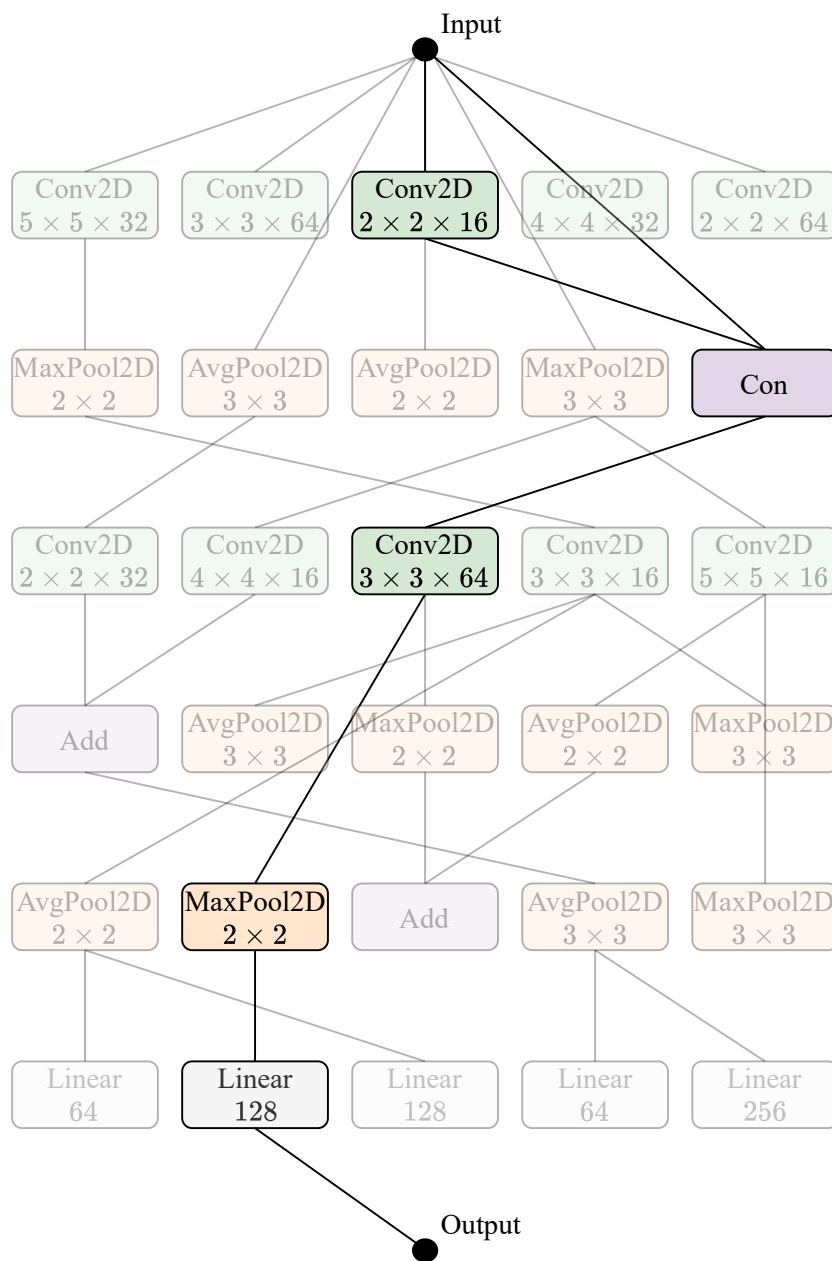
Obrázek 6.2: Ilustrace šablony pro graf artézského genetického programování zobrazuje různé třídy vrstev. Třída konvolučních vrstev je znázorněna zeleným obdélníkem, třída slučovacích vrstev oranžovým a třída plně propojených vrstev šedým.

GraphControler

Na základě tříd `Container` a `Template` definuje doménu hodnot pro jednotlivé geny genomu jedince. Důležitým parametrem `GraphControler` je parametr `ML_probability`, který určuje pravděpodobnost, s jakou může být agregační vrstva nahrazena slučovací vrstvou.

CGPGraph

Třída reprezentující DAG jež je využíván metodou CGP pro zakódování řešení. V rámci programu slouží jako prostředek pro nepřímé zakódování architektury CNN. Obrázek 6.3 ilustruje graf CGP v souladu s nově definovanými restrikcemi.



Obrázek 6.3: Graf CGP.

Net

Tato třída je zodpovědná za dynamické sestavení CNN během evaluace jedince na základě acyklického orientovaného grafu, který tvoří jeho genotyp.

Individual

Tato třída představuje jedince v rámci evolučního algoritmu a implementuje operace bodové mutace a evaluace. Evaluace zahrnuje konstrukci sítě, kterou jedinec reprezentuje, její trénink na trénovacích datech a následné vyhodnocení na testovacích datech. Třída uchovává informace o počtu parametrů a kvalitě klasifikace sítě na validačních datech, které jsou poskytovány evolučnímu algoritmu formou účelové funkce. Uvnitř třídy je implementována optimalizační metoda pro sdílení učících se parametrů mezi sítěmi rodiče a potomka.

Generator

Jedná se o třídu zodpovědnou za vygenerování počáteční populace P_0 .

Terminator

Určuje ukončovací podmínky pro evoluční algoritmus. Tato třída umožňuje zastavení algoritmu po dosažení určitého počtu generací, určitého času stráveného nad výpočty, nebo kombinaci obou těchto kritérií.

NSGAE

Tato třída představuje jedince q modifikovaného evolučního algoritmu **NSGA_II**. Zahrnuje třídu **Individual** a uchovává atributy **dominated**, který určuje, kolik jedinců řešení p dominuje, a **dominates**, pole obsahující jedince, které jsou dominovány řešením p . Dále obsahuje atribut **rank**, označující třídu, do které jedinec patří, a atribut **crowding distance** určující míru hustoty jedinců v okolí řešení q . Kromě toho třída **NSGAE** implementuje operátor \prec_n .

NSGA_II

Tato třída implementuje modifikovaný evoluční algoritmus **NSGA_II**, jak je popsáno v kapitole Návrh v sekci Evoluční algoritmus 5.3.

6.3 Problém mizejícího gradientu

Při trénování neuronových sítí docházelo k mizení gradientu. V hlubokých modelech, složených z několika vrstev, směr gradientu určuje, jak mají být upraveny parametry, za předpokladu, že parametry nižších vrstev zůstanou nezměněny. Avšak v praxi se všechny parametry často mění najednou, což může vést k nečekaným výsledkům. Pro stabilizaci a urychlení procesu učení se používá metoda Batch Normalization, která může provádět normalizaci dat každé vstupní nebo skryté vrstvy neuronové sítě. Normalizace matice \mathbf{H} , představující výstup vrstvy pro soubor vzorků vstupních dat, kde každý řádek obsahuje výsledky aktivačních funkcí pro jeden vzorek, lze popsat rovnicí 6.1

$$\mathbf{H}' = \frac{\mathbf{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}, \quad (6.1)$$

kde vektor $\boldsymbol{\mu}$ reprezentuje průměry a vektor $\boldsymbol{\sigma}$ obsahuje směrodatné odchylky každé jednotky. Tyto vektory se aplikují na každý řádek matice \mathbf{H} , kde normalizace probíhá po prvcích v rámci řádků. Konkrétně je hodnota $H_{i,j}$ normalizována pomocí odpovídajících

průměru μ_j a směrodatné odchyly σ_j . Výpočet průměru a směrodatné odchyly je definován rovnicemi 6.2 a 6.3:

$$\boldsymbol{\mu} = \frac{1}{m} \sum_i \mathbf{H}_{i,:} \quad (6.2)$$

$$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m} \sum_i (\mathbf{H} - \boldsymbol{\mu})_i^2}, \quad (6.3)$$

kde δ je malá kladná hodnota, aby se zabránilo výpočtu nedefinovaného gradientu pro $\sqrt{z} = 0$. Tímto jsou aktivační hodnoty každé funkce normalizovány tak, aby jejich průměr byl blízký 0 a směrodatná odchyly blízká 1. Gradient je zpětně propagován přes operace výpočtu směrodatných odchylek, průměrů a samotné operace normalizace [11].

Kapitola 7

Experimenty

V experimentech byla využita datová sada MNIST, která je běžně používána pro porovnání výsledků modelů strojového učení a rozpoznávání obrazu. V rámci těchto experimentů byly provedeny běhy s různými velikostmi populace a počtem generací s cílem zkoumat vliv těchto faktorů na kvalitu výsledného řešení. V závěru kapitoly bylo provedeno srovnání kvality architektur neuronových sítí, které vzešly z experimentů, se známými architekturami neuronových sítí.

7.1 Datová sada

MNIST je standardizovaný dataset obsahující obrázky ručně psaných číslic. Tato datová sada vychází z původní černobílé datové sady NIST. Čísla z této sady byla vycentrována a jejich velikost byla normalizována tak, aby se vešla do čtverce o rozměrech 20×20 pixelů. Tyto čtverce byly následně vloženy do obrázků o rozměrech 28×28 pixelů. Obrázky mohou obsahovat odstíny šedi z důvodu použití techniky antialiasingu. Dataset obsahuje 60,000 trénovacích a 10,000 testovacích obrázků. Jedná se o jednoduchou datovou sadu, která vyžaduje minimální úsilí v předzpracování a formátování. Kromě toho je k dispozici rozsáhlá literatura, která poskytuje informace pro srovnání výkonnosti algoritmů [6].



Obrázek 7.1: Ukázka datové sady MNIST.

7.2 Nastavení parametru modelu

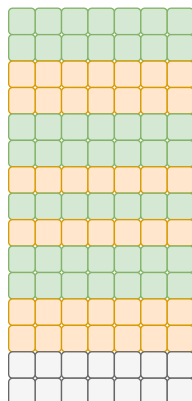
Experimenty byly provedeny na dvou různých počítačových zařízeních: jeden s operačním systémem Windows a grafickou kartou NVIDIA RTX 2080 (Turing) s 8GB RAM, druhý s operačním systémem Linux a grafickou kartou NVIDIA RTX 2060 (Turing) s 8GB RAM. Experimenty byly provedeny pro čtyři různá nastavení parametrů, přičemž struktura populace byla inspirována evoluční strategií $(\mu + \lambda)$, která je standardní metodou v CGP. V každém evolučním běhu bylo trénováno a vyhodnoceno právě 300 jedinců, avšak s variací v počtu rodičů μ , vygenerovaných potomků λ a počtu generací. Nastavení parametrů je uvedeno v tabulce 7.1. Šablona pro experimenty pro modifikovaný graf CGP je ilustrována obrázkem 7.2.

Tabulka 7.1: Nastavení parametrů modelu pro experimenty

Nastavení parametrů	
Hodnoty parametrů společné pro všechny experimenty	
Počet sloupců grafu	7
Počet řádků grafu	15
L-back parametr	2
Batch size	4
Velikost trénovací sady	20 000
Velikost testovací sady	10 000
Míra mutace	0.1
Počet epoch	4
Hodnoty parametrů pro experimentální sadu I	
μ	3
λ	12
Počet generací	20
Hodnoty parametrů pro experimentální sadu II	
μ	4
λ	16
Počet generací	15
Hodnoty parametrů pro experimentální sadu III	
μ	5
λ	20
Počet generací	12
Hodnoty parametrů pro experimentální sadu IV	
μ	6
λ	24
Počet generací	10

7.3 Výsledky experimentů

Byly provedeny celkem čtyři sady experimentů, přičemž každá probíhala po 14 evolučních bžích. Hlavním účelem těchto experimentů bylo zkoumat vliv velikosti populace na kvalitu finálního řešení. Během každého evolučního běhu bylo trénováno 300 jedinců. Průměrná doba trvání evoluce na počítačovém zařízení s operačním systémem Linux a grafickou kartou



Obrázek 7.2: Šablona pro experimenty. Zelená políčka značí třídu konvolučních, žlutá agregačních a šedá plně propojených vrstev.

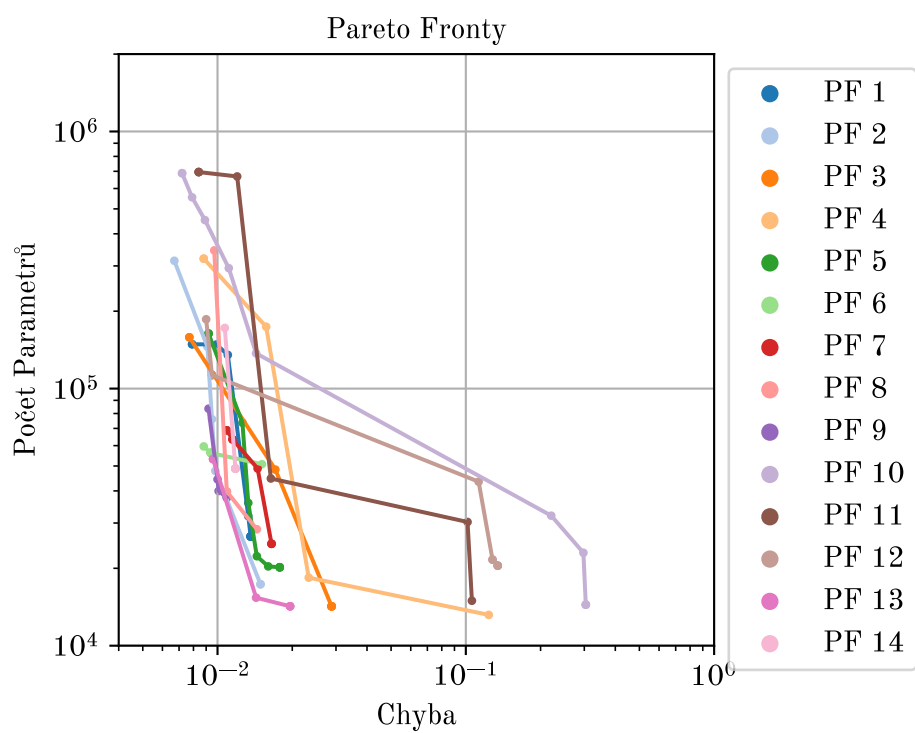
NVIDIA RTX 2060 (Turing) s 8 GB paměti RAM činila 2 hodiny, 42 minut a 42 sekund. Pro druhé počítačové zařízení s operačním systémem Windows a grafickou kartou NVIDIA RTX 2080 (Turing) s 8 GB paměti RAM byla průměrná doba evoluce 5 hodin, 19 minut a 16 sekund. Grafy 7.3 až 7.6 vizualizují Pareto fronty evolučních běhů, přičemž každý graf zobrazuje Pareto fronty jednotlivých sad experimentů.

7.4 Analýza vlivu velikosti populace na kvalitu výsledného řešení

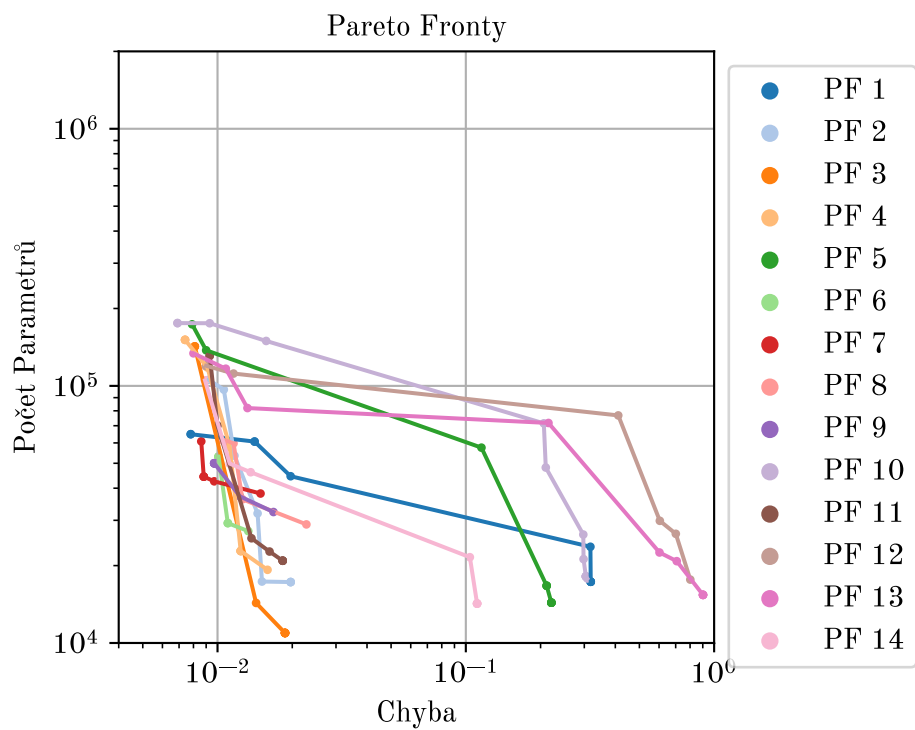
Běžně není skutečná Paretova fronta známa a výsledkem algoritmů pro vícekritériální optimalizaci je pouze aproximace této skutečné fronty. Kvalita této aproximace závisí na dvou hlavních faktorech: přesnosti aproximace bodů na skutečné Pareto frontě v porovnání s body na aproximované frontě a rozmanitosti bodů na aproximované frontě. Metrika hypervolume se stala standardem pro hodnocení kvality aproximace Pareto fronty, protože bere v úvahu oba výše zmíněné aspekty [3]. Tato metrika je koncipována tak, že pokud jedna aproximace Pareto fronty dominuje nad druhou, pak hypervolume té první je větší než hypervolume druhé. Klíčové pro správné použití této metriky je vhodná volba referenčního bodu, ke kterému se hypervolume počítá, neboť tato volba může zásadně ovlivnit výsledky analýzy.

Normalizace

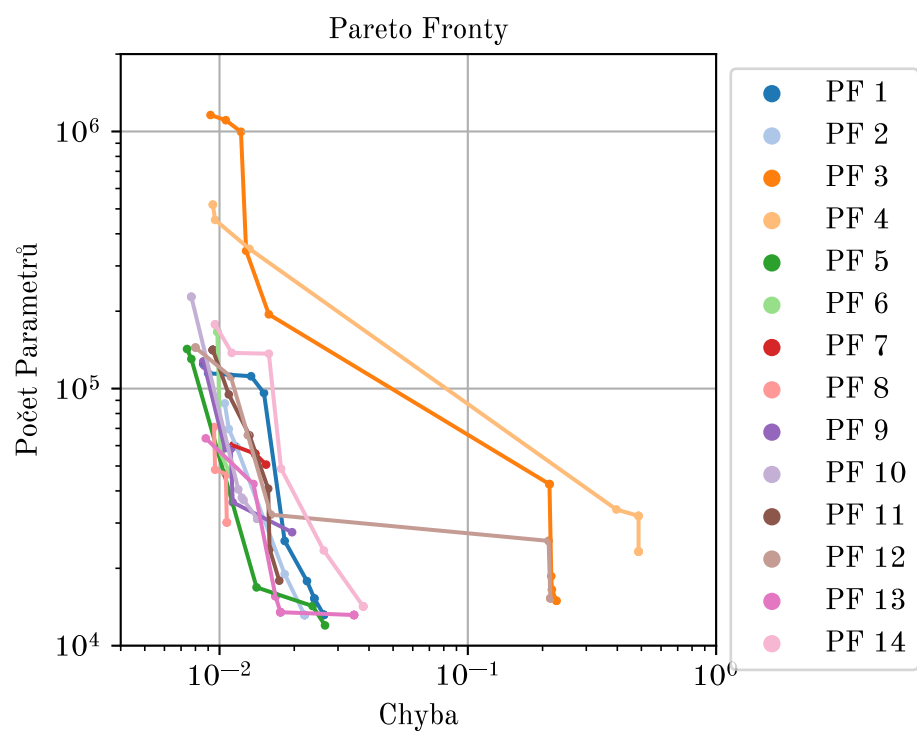
V dvoudimenzionálním prostoru lze bod na Pareto frontě zapsat jako dvojici $(f_1(\xi), f_2(\xi))$, kde $f_1(\xi)$ a $f_2(\xi)$ představují hodnoty účelové funkce podle kritéria 1 a 2. Pro usnadnění analýzy byl vektor hodnot kritérií normalizován do intervalu hodnot $[0, 1]^C$, kde C je počet



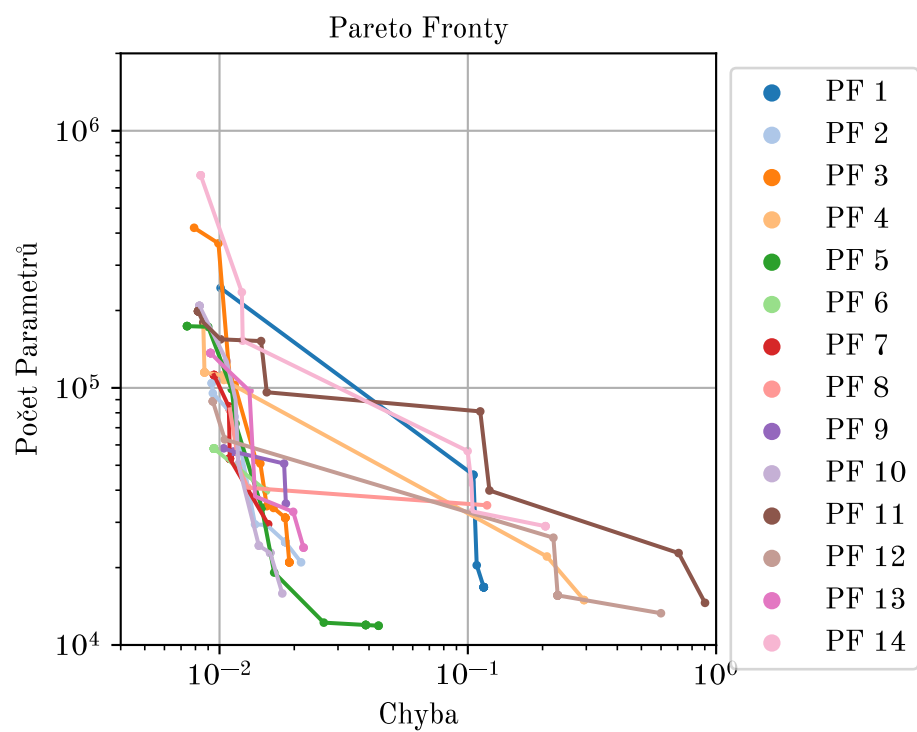
Obrázek 7.3: Graf vizualizující Pareto fronty sady experimentů I.



Obrázek 7.4: Graf vizualizující Pareto fronty sady experimentů II.



Obrázek 7.5: Graf vizualizující Pareto fronty sady experimentů III.



Obrázek 7.6: Graf vizualizující Pareto fronty sady experimentů IV.

zkoumaných kritérií. Normalizace je znázorněna rovnicí 7.1

$$f_j(\xi)^* = \left| \frac{f_j(\xi) - f_j(\xi)_{\text{worst}}}{f_j(\xi)_{\text{best}} - f_j(\xi)_{\text{worst}}} \right|, \quad j = 1, 2, \dots, C \quad (7.1)$$

kde $f_j(\xi)_{\text{worst}}$ odpovídá nejvyšší, respektive nejnižší hodnotě (v případě maximalizace účelové funkce) kritéria j , v rámci PF , a $f_j(\xi)_{\text{best}}$ koresponduje s nejnižší, respektive nejvyšší hodnotou (v případě maximalizace účelové funkce) kritéria j , též množiny.

Hypervolume

Metrika hypervolume měří objem prostoru kritérií, pro jehož prvky platí že jsou slabě dominovány body na Pareto frontě PF . Hodnota tohoto ukazatele je závislá na referenčním bodě $\mathbf{r} = (r_1, r_2, \dots, r_c) \in \mathbb{R}^C$, vůči kterému je vyhodnocována. Jeho formální definice je uvedena v rovnici 7.2

$$I_H(PF, \mathbf{r}) = \lambda \left(\bigcup_{\mathbf{s} \in PF} \text{space}(\mathbf{s}, \mathbf{r}) \right), \quad (7.2)$$

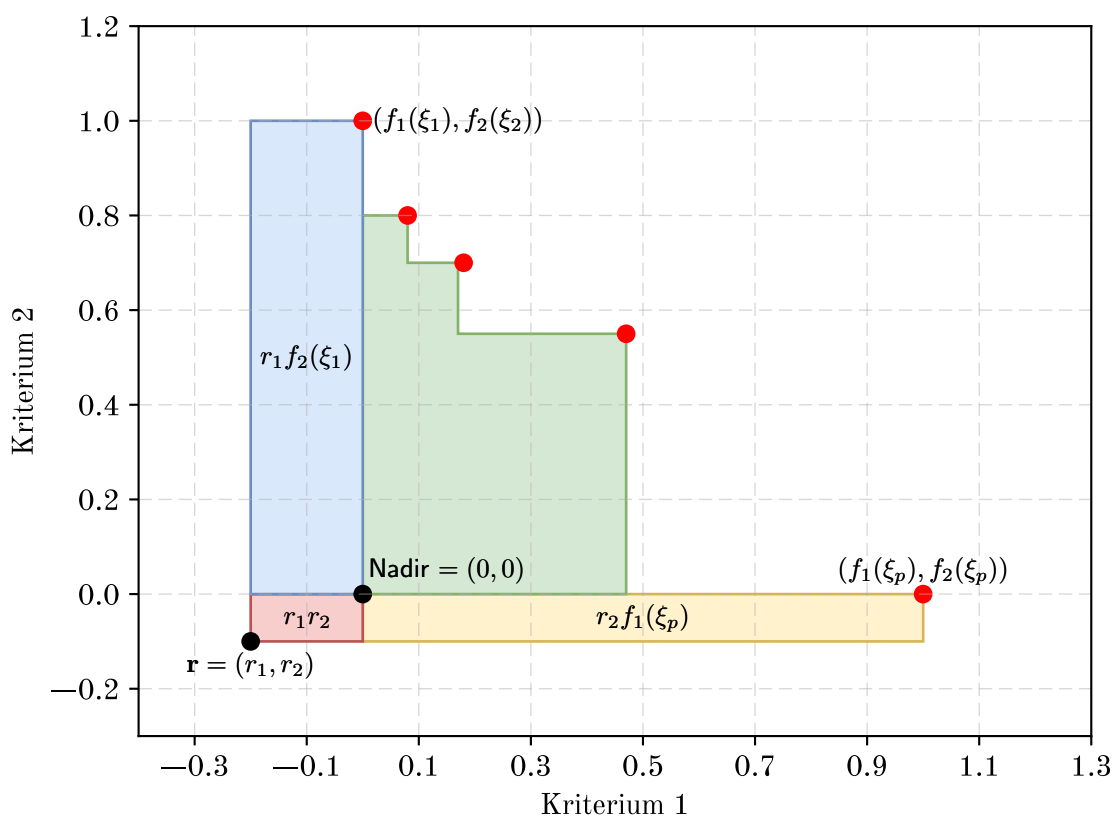
kde $\text{space}(\mathbf{s}, \mathbf{r}) = \{\mathbf{v} \in \mathbb{R}^C | \mathbf{r} \prec \mathbf{v} \preceq \mathbf{s}\}$ je podmnožina prostoru, která obsahuje všechny vektory hodnot kritérií $\mathbf{v} \in \mathbb{R}^C$, které jsou slabě dominovány prvky $\mathbf{s} \in PF$ a zároveň dominují referenčnímu bodu \mathbf{r} . Symbol λ zastupuje Lebesgueovu míru. Referenční bod je často zvolen jako vektor prvků, které odpovídají nejhorším hodnotám podle každého kritéria, nebo jako bod s hodnotami mírně horšími než tento bod. Obrázek 7.7 ilustruje metriku hypervolume.

Porovnání Pareto front pomocí ukazatele míry příspěvku

Jak uvádí článek [3], pro posouzení kvality aproximace Pareto front PF_1, PF_2, \dots, PF_n , získaných z n měření, se předpokládá existence skutečné Pareto fronty, která však není známa. Pro nejlepší odhad této fronty jsou všechny aproximace Pareto front PF_1, PF_2, \dots, PF_n sjednoceny do jedné množiny a na základě dominance prvků je sestavena Pareto fronta PF_S , která představuje nejlepší dostupnou aproximaci skutečné Pareto fronty. Následně jsou konstruovány fronty $PF'_1, PF'_2, \dots, PF'_n$ tak, že $PF'_i = PF_i \cap PF_S$. Jinými slovy, fronty $PF'_1, PF'_2, \dots, PF'_n$ obsahují pouze ty vektory kritérií, které nejsou kolektivně dominovány frontou PF_S . Dále je provedena normalizace fronty PF_S podle rovnice 7.1. Jelikož fronty $PF'_1, PF'_2, \dots, PF'_n$ obsahují pouze prvky, které nejsou kolektivně dominovány frontou PF_S , je zaručeno, že i všechny vektory v těchto frontách budou normalizované. Poté je vypočítán hypervolume pro každou standardizovanou aproximaci Pareto fronty PF'_i jako $I_H(PF'_i, \mathbf{r})$ a pro frontu nejlepší aproximace PF_S jako $I_H(PF_S, \mathbf{r})$. Výpočet ukazatele míry příspěvku pro prvek PF_i je demonstrován rovnicí 7.3 [3].

$$CR(PF_i, PF_S) = \frac{I_H(PF'_i, \mathbf{r})}{I_H(PF_S, \mathbf{r})} \quad (7.3)$$

V průběhu vyhodnocování experimentů se ukázalo, že vlivem velkého počtu porovnávaných front často kolektivně dominovala zkonstruovaná fronta PF_S všem prvkům fronty PF_i , tudíž poskytované výsledky nebyly validní a krok sestavování front $PF'_1, PF'_2, \dots, PF'_n$ byl vynechán. Kolektivně dominované prvky nebyly z aproximačních front odstraněny a z tohoto důvodu bylo nutné provést normalizaci popsanou rovnicí 7.1 tak, aby $f(\xi)_{\text{worst}}$



Obrázek 7.7: Hypervolume pro hypotetickou dvourozměrnou Pareto frontu složenou z pěti bodů, která je výsledkem minimalizace účelové funkce. Každý bod na Pareto frontě je označen červenou barvou. Referenční bod je zvolen tak, aby byl mírně horší než bod Nadir, což je bod reprezentující nejhorší hodnoty podle každého kritéria. Hypervolume je součtem objemů všech barevných ploch. Převzato z [3].

odpovídala nejvyšší, respektive nejnižší hodnotě (v případě maximalizace účelové funkce) kritéria j získané ze sjednocení aproximací Pareto front PF_1, PF_2, \dots, PF_n , a $f(\xi)_{\text{best}}$ korespondovala s nejnižší, respektive nejvyšší hodnotou (v případě maximalizace účelové funkce) kritéria j téže množiny.

Normalita dat

Mnoho statistických testů požaduje, aby zkoumaná data vykazovala normální rozdělení. Pokud však data nejsou normálně distribuována, testy, které předpokládají normální rozdělení, mohou poskytnout nepřesné výsledky. Proto je klíčové provést test normality před použitím statistických testů.

Jedním z účinných způsobů, jak posoudit, zda data pocházejí z normálního rozdělení, jsou metody založené na grafech. Tyto grafické metody jsou užitečným nástrojem pro ověření předpokladů o distribuci dat. V případě jejich nesplnění mohou sloužit k identifikaci potřebných korekčních opatření. Existují také testovací metody, které poskytují určitou míru jistoty, zda data vykazují normální rozdělení, avšak ne vždy poskytují detailní vysvětlení případného odchýlení od normality.

Jednoduchým a běžně využívaným nástrojem pro posouzení normality je sestavení histogramu. Histogram je grafické znázornění, ve kterém jsou na vodorovnou osu umístěny hodnoty zkoumané proměnné a na svislou osu je zaznamenána jejich frekvence. V případě normálního rozdělení by histogram měl přibližně odpovídat tvaru Gaussovy křivky. Prostřednictvím histogramu je možné identifikovat odlehle body, posunutí dat k jedné straně nebo zda jsou hodnoty symetricky rozloženy.

Dalším způsobem je sestavení kvantilového grafu, kde na vodorovnou osu jsou umístěny kvantily hypotetického normálního rozdělení a na svislou osu kvantily zkoumaných dat. Když se kvantily obou distribucí shodují, bod představující kvantil pozorovaných dat leží na přímce, která reprezentuje kvantily hypotetického rozdělení.

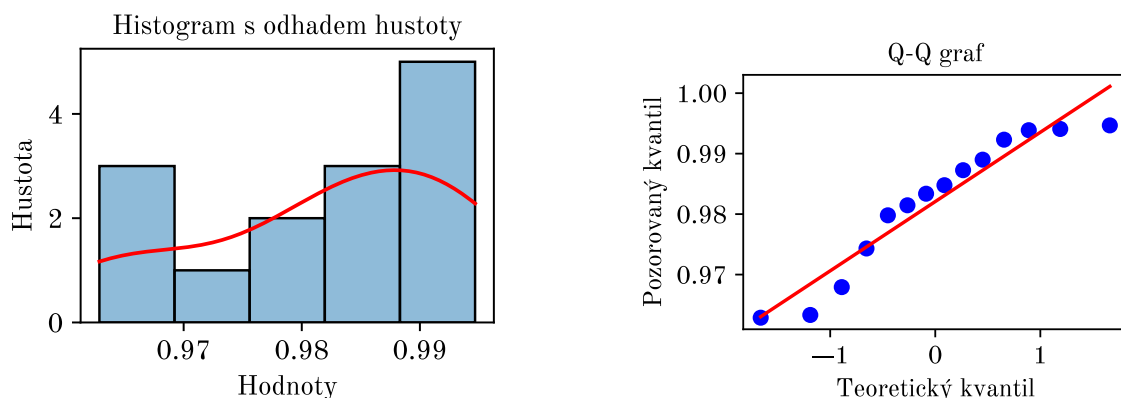
Jedním z nejpobulárnějších statistických testů pro normalitu dat je Shapiro-Wilkův test. Tento test porovnává pozorovaná data s prvky normálního rozdělení, které mají stejný průměr a směrodatnou odchylku jako zkoumaná data. Statistický test je definován rovnicí 7.4.

$$W = \frac{(\sum a_i y_i)^2}{\sum (y - \bar{y})^2}, \quad (7.4)$$

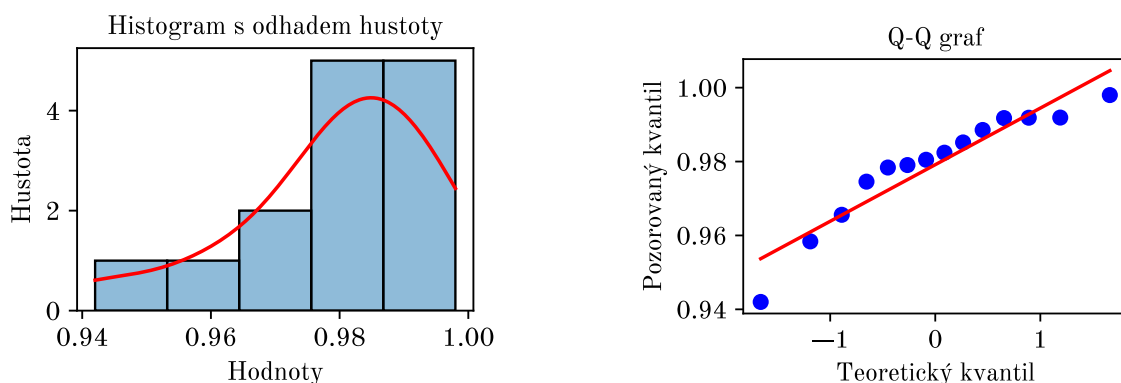
kde y_i je hodnota i -tého pozorovaného prvku a a_i je i -tá očekávaná hodnota pocházející z referenčního normálního rozdělení. Test Shapiro-Wilk má dobré vlastnosti z hlediska síly. Pokud je hodnota W velmi malá, nulová hypotéza, která předpokládá, že data pocházejí z normálního rozdělení, je zamítnuta [4].

Analýza statisticky významného rozdílu distribucí pozorovaných skupin

Během analýzy vlivu velikosti populace na kvalitu výsledného řešení byla použita míra příspěvku aproximativních Pareto front. Pro posouzení významných rozdílů mezi distribucemi skupin I až IV byl zvolen Kruskal-Wallisův test. Při analýze grafů 7.8 až 7.11 nebylo zjištěno, že by histogramy ani Q-Q grafy naznačovaly, že by data pocházela z normálního rozdělení. Normalitu nepodpořil ani Shapiro-Wilkův test, jehož výsledky jsou zachyceny v tabulce 7.2, kde u skupiny III byla zjištěna p -hodnota nižší než 0.05. Z tohoto důvodu byl zvolen Kruskal-Wallisův test, který umožňuje porovnávat více skupin a je robustní vůči



Obrázek 7.8: Na pravé straně se nachází histogram míry příspěvků aproximativních Pareto front skupiny I, zatímco na levé straně jsou zachyceny kvantily této míry nanesené na svislou osu a kvantily hypotetického normálního rozdělení nanesené na vodorovnou osu



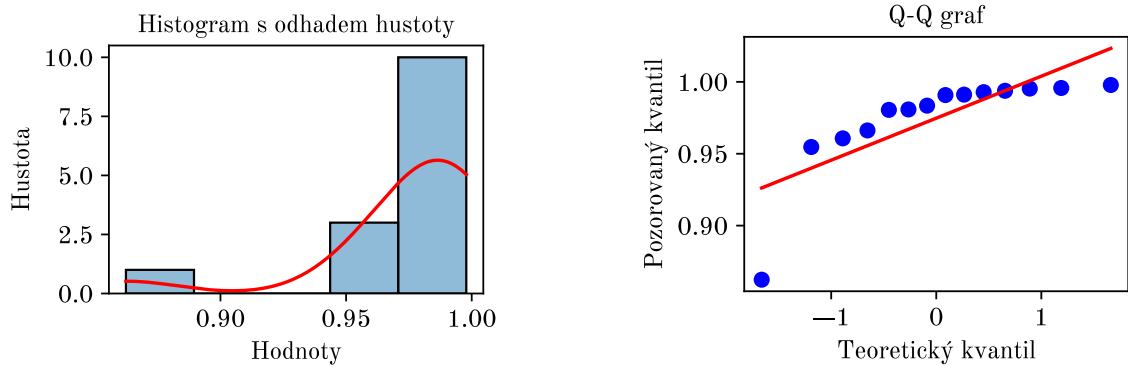
Obrázek 7.9: Na pravé straně se nachází histogram míry příspěvků aproximativních Pareto front skupiny II, zatímco na levé straně jsou zachyceny kvantily této míry nanesené na svislou osu a kvantily hypotetického normálního rozdělení nanesené na vodorovnou osu.

odchylkám pozorovaných dat od normálního rozdělení. Pro statistické výpočty byla využita Python knihovna `scipy`, zatímco pro vizualizaci dat byla použita knihovna `matplotlib`.

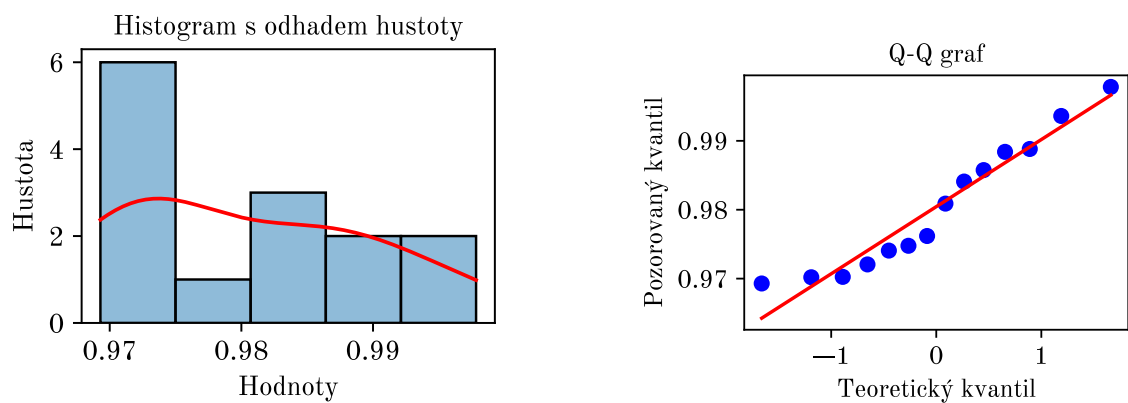
Kruskal-Wallisův test představuje neparametrickou alternativu k parametrické metodě analýzy rozptylu známé jako ANOVA. Jeho účelem je zjistit, zda alespoň jedna skupina ve srovnání s ostatními projevuje statisticky významné rozdíly. Test Kruskal-Wallis neidentifikuje konkrétní skupiny, které se odlišují, ani počet těchto skupin. Nulovou a alternativní hypotézu lze tedy definovat rovnicí 7.5

$$\begin{aligned} H_0 : F_1 = \dots = F_k = F^0 \text{ versus} \\ H_a : \text{alespoň jedna } F_i \text{ není rovna } F^0, \end{aligned} \quad (7.5)$$

kde F_1, F_2, \dots, F_k představují kumulativní distribuční funkce pro $K \geq 2$ zkoumaných skupin, z nichž bylo vybráno N nezávislých vzorků. Pokud uvažujeme náhodné nezávislé prvky X_{ij} , kde $i = 1, 2, \dots, K$, $j = 1, 2, \dots, N_i$, R_{ij} je pořadí i -tého prvku j -té skupiny



Obrázek 7.10: Na pravé straně se nachází histogram míry příspěvků aproximativních Pareto front skupiny III, zatímco na levé straně jsou zachyceny kvantily téže míry nanesené na svislou osu a kvantily hypotetického normálního rozdělení nanesené na vodorovnou osu.



Obrázek 7.11: Na pravé straně se nachází histogram míry příspěvků aproximativních Pareto front skupiny IV, zatímco na levé straně jsou zachyceny kvantily téže míry nanesené na svislou osu a kvantily hypotetického normálního rozdělení nanesené na vodorovnou osu.

Tabulka 7.2: Výsledky Shapiro-Wilkova testu pro míry příspěvků aproximativních Pareto front skupin I až IV

Skupina	W	P-hodnota
I	0.8988	0.1083
II	0.8979	0.1053
III	0.6313	7.81×10^{-5}
IV	0.9226	0.2393

závislé na jeho hodnotě a $R_i = \sum_{j=1}^{N_i} R_{ij}$, pak lze testovací statistiku popsat rovnicí 7.6

$$S_{KW} = \frac{\sum_{i=1}^K N_i \left(\frac{R_i}{N_i} - \frac{N+1}{2} \right)^2}{\frac{1}{N-1} \sum_{i=1}^K \sum_{j=1}^{N_i} \left(R_{ij} - \frac{N+1}{2} \right)^2}, \quad (7.6)$$

kde N značí celkový počet prvků ze všech skupin [9]. Statistika S_{KW} má asymptotické rozdělení chi-kvadrát s $K-1$ stupni volnosti. V případě, že prvky neobsahují vázané hodnoty lze výpočet definovat rovnicí 7.7 [9].

$$S_{KW} = \frac{12}{N(N+1)} \sum_{i=1}^K N_i \left(\frac{R_i}{N_i} - \frac{N+1}{2} \right)^2. \quad (7.7)$$

Výsledky Kruskal-Wallisova testu naznačují, že mezi skupinami neexistuje statisticky významný rozdíl. Výsledná **p-hodnota** = 0.857 překračuje stanovenou hranici statistické významnosti $\alpha = 0.05$. Z toho vyplývá, že na základě dat získaných z experimentů není možné zamítnout nulovou hypotézu, která tvrdí, že průměrné hodnoty ohodnocení prvků skupin jsou stejné.

7.5 Zhodnocení výsledků

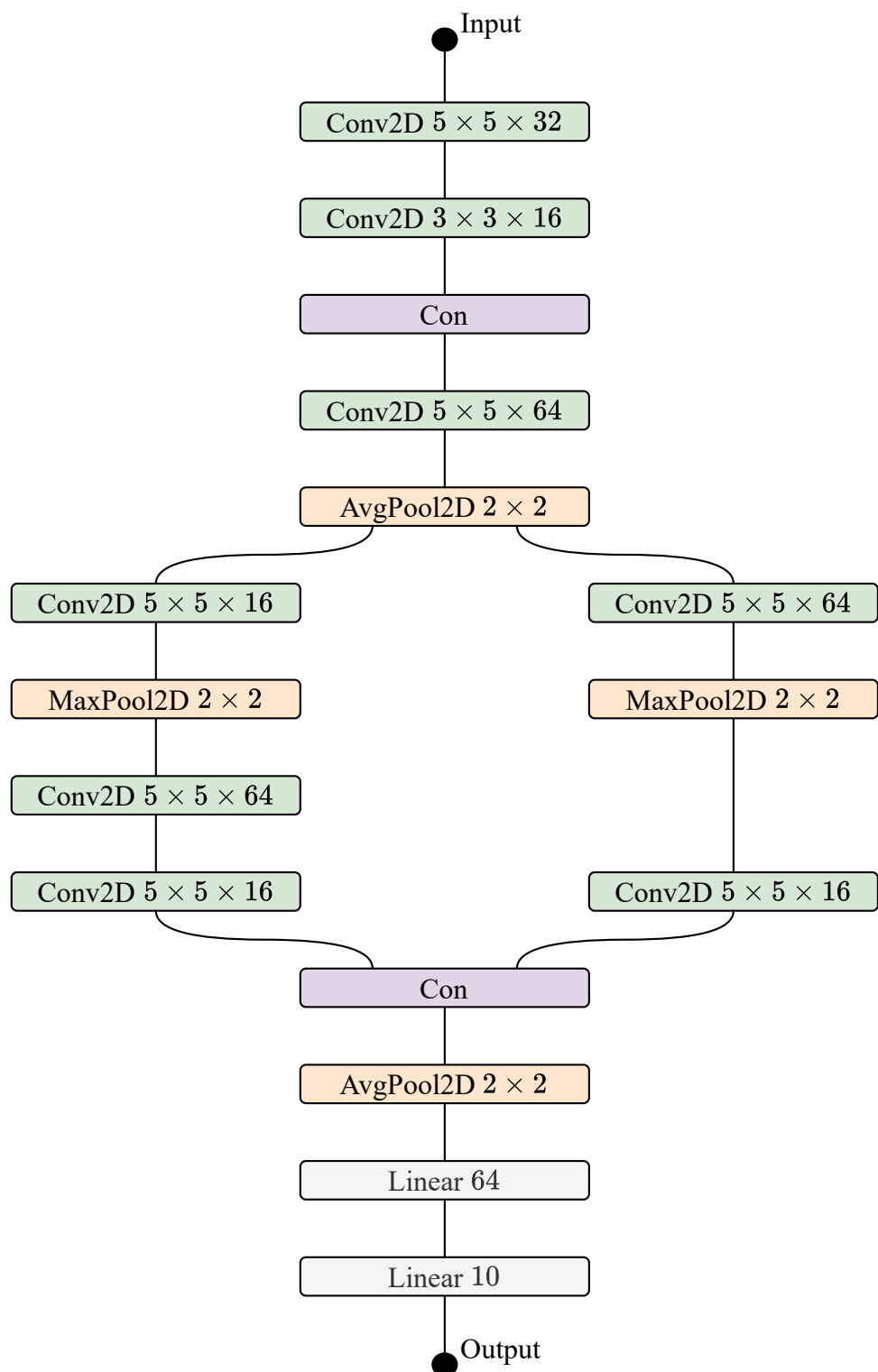
Pomocí Kruskal-Wallisova testu byl zkoumán vliv velikosti populace modifikovaného NSGA-II, použitého pro automatický návrh architektury CNN, na kvalitu výsledného řešení. Výsledky však naznačují, že rozdíly mezi zkoumanými skupinami I až IV, které odpovídají výstupním hodnotám implementovaného programu s parametry uvedenými v tabulce 7.1, nejsou statisticky významné. Toto zjištění je v souladu s předpokladem nulové hypotézy, kterou nebylo možné na základě výsledků testu zamítnout. Nemožnost zamítnutí nulové hypotézy na základě statistických testů nevyvrací možnost existence alternativní hypotézy ve skutečnosti a při provedení testů s větším vzorkem by mohlo dojít k jejímu přijetí.

I přesto, že analýza dat neprokázala statisticky významné rozdíly, tato práce přináší základ v podobě navrženého a implementovaného programu, který by mohl sloužit jako základ pro další výzkum v oblasti této problematiky. Implementovaný program přináší také význam z praktického hlediska, kdy při vhodně stanoveném omezení na prohledávací prostor architektur je schopna nalézt řešení, které je schopno z pohledu kvality klasifikace konkurovat již etablovaným architekturám. Tabulka 7.3 prezentuje 20 nejlepších jedinců z hlediska kvality klasifikace vybraných ze všech pozorovaných skupin I až IV.

Nejlepší výsledek byl dosažen jedincem uvedeným v tabulce 7.3, který byl následně podroben dalšímu tréninku na trénovací sadě obsahující 60,000 vzorků po dobu 10 epoch. Poté byl testován na validační sadě, která obsahovala 10,000 vzorků, a dosáhl kvality klasifikace ve výši 99.45 %. Toto nalezené řešení, s 314,186 učiteli se parametry, se svou přesností blíží jedné z nejlepších současných architektur **SimpleNetv1**, která je inspirována klasickou architekturou neuronových sítí a disponuje 5.48 miliony parametrů a dosahuje přesnosti 99.75 % [13]. Architektura sítě nejlepšího nalezeného řešení z hlediska kvality klasifikace je ilustrována obrázkem 7.12.

7.6 Pokračování práce

Z nalezených architektur je patrné, že při pečlivě zvolených parametrech omezujících prohledávací prostor je metoda schopna nalézt řešení, které se mohou vyrovnat již etablovaným



Obrázek 7.12: Architektura nejlepší nalezené sítě z hlediska kvality klasifikace, která po dodatečném trénování dosáhla přesnosti 99.45 %.

Tabulka 7.3: Nejlepší jedinci podle úspěšnosti klasifikace na testovací sadě

Přesnost (%)	Počet Parametrů	Skupina	Pareto Fronta
99.33	314 186	I	2
99.31	175 482	II	10
99.28	688 170	I	10
99.26	142 538	III	5
99.26	151 034	II	4
99.26	173 866	IV	5
99.23	130 394	III	5
99.23	158 314	I	3
99.23	227 402	III	10
99.22	64 874	II	1
99.21	148 778	I	1
99.21	173 546	II	5
99.21	419 418	IV	3
99.21	554 602	I	10
99.20	133 850	II	13
99.20	144 442	III	12
99.19	142 378	II	3
99.18	198 618	IV	11
99.17	208 410	IV	10
99.16	671 050	IV	14

modelům. V tomto kontextu představuje tato práce základ pro další zkoumání vlivu různých parametrů na kvalitu výsledného řešení s cílem hlouběji porozumět procesu evolučního návrhu konvolučních neuronových sítí.

Z praktického hlediska lze pomocí automatického návrhu architektury CNN nalézt nestandardní řešení, která by ručním návrhem podle doporučených praktik pravděpodobně nevznikla. V této práci byl důraz kladen na architektury CNN, avšak automatický návrh může být obecně rozšířen pro hledání architektur neuronových sítí. Vícekriteriální optimalizace, která byla v tomto článku zaměřena na počet parametrů a kvalitu klasifikace neuronové sítě, může být použita i k jiným aspektům jako je například doba trénování neuronové sítě nebo počet operací násobení a sčítání. V rámci neuronových sítí lze využít různé formy jejich regularizace nebo optimalizace učení, které by podopřily proces neuroevoluce [11].

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a implementovat metodu pro automatizovaný návrh architektury neuronových sítí a na základě experimentů, prováděných za různých konfigurací této implementace, vyhodnotit kvalitu nalezených neuronových sítí a účinnost aplikovaných evolučních algoritmů.

Úloha automatického návrhu konvolučních neuronových sítí byla výpočetně náročná, a proto byl proces učení neuronových sítí optimalizován sdílením učících se parametrů mezi rodiči a potomky. Evoluční algoritmus byl omezen v prohledávaném prostoru za účelem dosažení rychlejší konvergence a nalezení optimálních řešení. Při tréninku sítí se objevil problém mizejícího gradientu, který byl řešen pomocí metody Batch Normalization.

V rámci experimentů bylo provedeno 56 běhů evolučního algoritmu na datové sadě MNIST. Tyto běhy byly rozděleny do čtyř skupin podle různých nastavení implementace, zaměřující se na porozumění vlivu velikosti populace na kvalitu výsledného řešení. Pro porovnání skupin byla použita metrika hypervolume, která byla spočítána pro výsledné Pareto fronty. Získané hodnoty hypervolume byly testovány na normalitu, avšak z výsledků bylo patrné, že data pravděpodobně nepochází z normálního rozdělení. Proto byl pro analýzu použit Kruskal-Wallisův test, který nepotvrdil statisticky významný rozdíl mezi skupinami. Z experimentů vzešlo velké množství architektur, u kterých přesnost na validační sadě přesahovala 99 %, a nejlepší řešení, které po dotrénování dosáhlo přesnosti 99.45 % a obsahovalo 314 186 parametrů, se svou přesností blíží jedné z nejlepších architektur dnešní doby SimpleNetv1, která dosahuje přesnosti 99.75 %.

Tato práce představuje základ pro další výzkum vlivu nastavení implementace na kvalitu výsledného řešení. Z praktického hlediska může implementovaná metoda nalézt řešení konkurenční etablovaným architekturám a optimalizaci lze rozšířit i na jiné aspekty, než je počet parametrů a kvalita klasifikace CNN. Metoda je obecně rozšiřitelná na automatizovaný návrh neuronových sítí.

Literatura

- [1] BARTZ BEIELSTEIN, THOMAS, BRANKE, JÜRGEN, MEHNEN et al. Evolutionary Algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*. 2014, sv. 4, č. 3, s. 178–195. ISSN 1942-4795.
- [2] BLOCK, H. D. The Perceptron: A Model for Brain Functioning. I. *Rev. Mod. Phys.* American Physical Society. January 1962, sv. 34, s. 123–135.
- [3] CAO, Y., SMUCKER, B. J. a ROBINSON, T. J. On using the hypervolume indicator to compare Pareto fronts: Applications to multi-criteria optimal experimental design. *Journal of Statistical Planning and Inference*. Elsevier. 2015, sv. 160, s. 60–74.
- [4] DAS, K. R. a IMON, A. A brief review of tests for normality. *American Journal of Theoretical and Applied Statistics*. 2016, sv. 5, č. 1, s. 5–12.
- [5] DEB, K., PRATAP, A., AGARWAL, S. a MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*. IEEE. 2002, sv. 6, č. 2, s. 182–197.
- [6] DENG, L. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*. 2012, sv. 29, č. 6, s. 141–142. DOI: 10.1109/MSP.2012.2211477.
- [7] EIBEN, A. a SMITH, J. *Introduction to Evolutionary Computing*. 2. vyd. Heidelberg: Springer Berlin, Heidelberg, July 2015. Natural Computing Series. ISBN 978-3-662-44874-8.
- [8] EIBEN, A. E. Multi-parent recombination. *Evolutionary computation*. 1997, sv. 1, s. 289–307.
- [9] FAN, C. a ZHANG, D. A note on power and sample size calculations for the Kruskal-Wallis test for ordered categorical data. *J Biopharm Stat.* 2012, sv. 22, č. 6, s. 1162–1173. DOI: 10.1080/10543406.2011.578313.
- [10] FRANCIS, K. *Charles Darwin and The Origin of Species*. London, UK: Bloomsbury Publishing, 2006. Greenwood Guides to Historic Events 1500-1900. ISBN 9781573567947.
- [11] GOODFELLOW, I., BENGIO, Y. a COURVILLE, A. *Deep Learning*. Cambridge, MA: MIT Press, 2016. ISBN 9780262035613.
- [12] HARDESTY, L. Explained: neural networks. *MIT News*. 2017, sv. 14. Dostupné z: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>.

- [13] HASANPOUR, S. H., ROUHANI, M., FAYYAZ, M. a SABOKROU, M. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. *ArXiv preprint arXiv:1608.06037*. 2016.
- [14] KUNDU, S., MOSTAFA, H., SRIDHAR, S. N. a SUNDARESAN, S. Attention-based image upsampling. *ArXiv preprint arXiv:2012.09904*. 2020.
- [15] LV, Z., SONG, X., FENG, Y., OU, Y., SUN, Y. et al. Evolutionary Neural Network Architecture Search. In: *Handbook of Evolutionary Machine Learning*. Springer, 2023, s. 247–281.
- [16] MCCULLOCH, W. S. a PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*. Springer. 1943, sv. 5, s. 115–133.
- [17] MILLER, J. a TURNER, A. Cartesian Genetic Programming. In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: Association for Computing Machinery, 2015, s. 179–198. GECCO Companion '15. ISBN 9781450334884.
- [18] MILLER, J. F. a SMITH, S. L. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on evolutionary computation*. IEEE. 2006, sv. 10, č. 2, s. 167–174.
- [19] MILLER, J. F., THOMSON, P., FOGARTY, T. et al. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study. *Genetic algorithms and evolution strategies in engineering and computer science*. Wiley Chechester, UK. 1997, s. 105–131.
- [20] MUKHOPADHYAY, A., MAULIK, U., BANDYOPADHYAY, S. a COELLO, C. A. C. A survey of multiobjective evolutionary algorithms for data mining: Part I. *IEEE Transactions on Evolutionary Computation*. IEEE. 2013, sv. 18, č. 1, s. 4–19. ISSN 1941-0026.
- [21] O'SHEA, K. a NASH, R. An introduction to convolutional neural networks. *ArXiv preprint arXiv:1511.08458*. 2015.
- [22] PRODHON, C. a PRINS, C. *Metaheuristics*. Cham, Switzerland: Springer International Publishing, 2016. ISBN 978-3-319-45403-0.
- [23] ROSENBLATT, F. et al. *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*. Spartan books Washington, DC, 1962.
- [24] SEKANINA, L. Neural architecture search and hardware accelerator co-search: A survey. *IEEE access*. IEEE. 2021, sv. 9, s. 151337–151362.
- [25] SETTE, S. a BOULLART, L. Genetic programming: principles and applications. *Engineering applications of artificial intelligence*. Brno: Elsevier. 2001, sv. 14, č. 6, s. 727–736. ISSN 1234-5678.
- [26] SUBRAMANIAN, V. *Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch*. Packt Publishing Ltd, 2018.

- [27] SUGANUMA, M., KOBAYASHI, M., SHIRAKAWA, S. a NAGAO, T. Evolution of deep convolutional neural networks using cartesian genetic programming. *Evolutionary computation*. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info 2020, sv. 28, č. 1, s. 141–163.
- [28] SUGANUMA, M., SHIRAKAWA, S. a NAGAO, T. A genetic programming approach to designing convolutional neural network architectures. In: *Proceedings of the genetic and evolutionary computation conference*. 2017, s. 497–504.
- [29] WILSON, D. G., CUSSAT BLANC, S., LUGA, H. a MILLER, J. F. Evolving simple programs for playing Atari games. In: *Proceedings of the genetic and evolutionary computation conference*. 2018, s. 229–236.
- [30] WOLFFE, A. *Chromatin: structure and function*. 3. vyd. London, UK: Academic press, 1998. ISBN 0-12-761915-1.
- [31] YAO, X. Evolving artificial neural networks. *Proceedings of the IEEE*. IEEE. 1999, sv. 87, č. 9, s. 1423–1447.
- [32] ZOU, J., HAN, Y. a SO, S.-S. Overview of artificial neural networks. *Artificial neural networks: methods and applications*. Springer. 2009, s. 14–22.