VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH KONVOLUČNÍCH NEURONOVÝCH SÍTÍ

EVOLUTIONARY DESIGN OF CONVOLUTIONAL NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JÁN PRISTAŠ**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Pristaš Ján, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Bioinformatika a biocomputing
Název: **Evoluční návrh konvolučních neuronových sítí**
Evolutionary Design of Convolutional Neural Networks
Kategorie: Umělá inteligence
Zadání:

1. Zpracujte studii mapující využití evolučních algoritmů pro návrh či optimalizaci hlubokých neuronových sítí.
2. Seznamte se s knihovnami pro práci s konvolučními neuronovými sítěmi; zaměřte se na TensorFlow.
3. Navrhněte způsob využití evolučního algoritmu při automatizovaném návrhu konvoluční neuronové sítě. Zaměřte se na efektivní zakódování neuronové sítě pro evoluční algoritmus.
4. Navržený způsob implementujte a ověřte jeho přínos na zvolené úloze.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3.

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

Vedoucí práce: **Sekanina Lukáš, prof. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cieľom tejto diplomovej práce je popísať základné techniky evolučného počítania, konvolučných neurónových sietí (CNN) a automatický návrh neurónových sietí pomocou neuroevolúcie (*NAS - Neural Architecture Search*). NAS techniky sú v súčasnej dobe stále viac skúmané, nakoľko zrýchľujú a zjednodušujú zdĺhavý a namáhavý proces návrhu umelých neurónových sietí, a taktiež umožňujú hľadať nekonvenčné architektúry, ktoré by klasickými metódami návrhu nevznikli. Práca obsahuje návrh a implementáciu programu, ktorý je schopný automatického návrhu konvolučných neurónových sietí s využitím open-source knižnice TensorFlow. Program na návrh CNN využíva algoritmus NSGA-II, čo je multikriteriálna varianta genetických algoritmov. Vďaka využití multikriteriálneho optimalizačného algoritmu je program schopný hľadať Pareto množinu optimálnych riešení v závislosti od presnosti sietí a počtu ich parametrov.

Abstract

The aim of this Master's thesis is to describe basic technics of evolutionary computing, convolutional neural networks (CNN), and automated design of neural networks using neuroevolution (*NAS - Neural Architecture Search*). NAS techniques are currently being researched more and more, as they speed up and simplify the lengthy and complicated process of designing artificial neural networks. These techniques are also able to search for unconventional architectures that would not be found by classic methods. The work also contains the design and implementation of software capable of automated development of convolutional neural networks using the open-source library TensorFlow. The program uses a multiobjective NSGA-II algorithm for designing accurate and compact CNNs.

Kľúčové slová

evolučné algoritmy, konvolučné neurónové siete, neuroevolúcia, TensorFlow

Keywords

evolutionary algorithms, convolutional neural networks, neuroevolution, TensorFlow

Citácia

PRISTAŠ, Ján. *Evoluční návrh konvolučních neuronových sítí.* Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Evoluční návrh konvolučních neuronových sítí

Prehlásenie

Prehlasujem, že som túto prácu vypracoval samostatne pod vedením pána prof. Ing. Lukáša Sekaniny, Ph.D. Uviedol som všetky publikácie a literatúru z ktorej som čerpal.

.....
Ján Pristaš
15. mája 2021

Poďakovanie

Rád by som sa poďakoval pánovi prof. Ing. Lukášovi Sekaninovi Ph.D. za jeho odborné rady, konzultácie a vedenie pri písaní tejto práce. Taktiež by som sa rád poďakoval pánovi doc. Ing. Jiřímu Jarošovi Ph.D. za prístup na fakultný výpočtový server a pánovi Ing. Vojtěchovi Mrázkovi za konzultácie a pomocou pri vykonávaní experimentov. Táto práca bola taktiež podporená Ministerstvom školstva, mládeže a telovýchovy Českej republiky prostredníctvom e-INFRA CZ (ID:90140).

Obsah

Kapitola 1

Úvod

Konvolučné neurónové siete – *CNN (Convolutional Neural Networks)* sa dnes tešia stále väčšej popularite a to najmä v úlohách analýzy a rozpoznávania obrazu. Ich hlavná výhoda spočíva v automatizovanej extrakcii dôležitých údajov zo vstupných dát, ktoré sú potrebné pre následnú klasifikáciu. Avšak aby bola konvolučná neurónová sieť schopná pracovať s dostatočne vysokou presnosťou, je potrebný kvalitný návrh architektúry siete. Ten je väčšinou vytváraný manuálne, odborníkmi v danom obore a vychádza prevažne z ľudských skúseností. Architektúra takýchto sietí je obvykle veľmi robustná, čo znamená, že obsahuje veľké množstvo parametrov, ktoré je potrebné trénovať a je vykonávaných veľa výpočtovo náročných operácií *MAC (Multiply-Accumulate)*. Takéto siete vyžadujú množstvo drahých výpočtových zdrojov a ich návrh je obvykle časovo veľmi náročný. Z tohto dôvodu sa stále viac skúmajú možnosti automatizácie návrhu konvolučných neurónových sietí s cieľom minimalizovať dobu hľadania optimálnej architektúry a potrebné výpočtové zdroje. Techniky automatizovaného návrhu umelých neurónových sietí sa nazývajú *NAS (Neural Architecture Search)* [10].

Nakoľko sa pri hľadaní architektúry siete stretávame s niekoľkými konfliktnými kritériami na optimálnu sieť, typicky je to počet trénovacích parametrov vs. presnosť siete, je potrebné využiť algoritmy, ktoré sú navrhnuté špeciálne na tento typ problému. Takéto algoritmy sa nazývajú *multikriteriálne optimalizačné algoritmy* a hľadajú také riešenia, ktoré sú určitým kompromisom v hodnotách jednotlivých kritérií. Výstupom výpočtu preto nie je jedno konkrétne riešenie, ale množina riešení. Takáto množina sa nazýva *Pareto množina vzájomne nedominujúcich riešení.*

V tejto diplomovej práci je uvedený základný prehľad algoritmov inšpirovaných biologickou evolúciou, tzv. *evolučných algoritmov* a základný prehľad konvolučných neurónových sietí. Ďalej sa budeme venovať technikám automatického návrhu umelých neurónových sietí pomocou evolučných algoritmov. Táto oblasť výskumu sa nazýva *neuroevolúcia* [27, 39]. Základná myšlienka tohto prístupu je vo využití postupu obdobnému biologickej evolúcií v návrhu čo najvhodnejšej neurónovej siete. Táto myšlienka je založená na fakte, že biologická evolúcia je "zodpovedná" za vytvorenie najkomplexnejšej neurónovej siete, akú dnes poznáme – *ľudský mozog*. Ďalšia časť práce je zameraná na praktické využitie *neuroevolúcie*. Ako prvé je uvedený návrh implementácie programu, ktorý bude schopný automatického návrhu konvolučných neurónových sietí, pomocou algoritmu *NSGA-II* [7], čo je multikriteriálna varianta genetických algoritmov. V nasledujúcej kapitole je popísaná implementácia

programu v jazyku Python s využitím knižnice TensorFlow, na prácu s umelými neurónovými sieťami. Posledná kapitola je venovaná experimentom, ktoré sú vykonávané na dátovej sade *CIFAR-10*, čo je jedna z najpoužívanejších testovacích sád na hodnotenie kvality algoritmov pre klasifikáciu obrazových dát. Bolo vykonaných niekoľko experimentov, na ktorých sa sleduje kvalita navrhnutých neurónových sietí a možnosť praktického využitia programu. Zdrojové kódy programu, spolu s návodom na použitie sú dostupné v Github repozitáry[1].

[1] https://github.com/xprist06/MasterThesis

Kapitola 2

Evolučné algoritmy

Evolučné algoritmy (EA) sú optimalizačné algoritmy, ktoré hľadajú riešenie problémov nekonvenčnými spôsobmi. Klasické konvenčné (presné) metódy, napr. Backtracking alebo A*, sú na jednej strane schopné nájsť najlepší možný výsledok, avšak negarantujú nájdenie optimálneho riešenia v prijateľnom čase. Existujú rôzne typy úloh, pri ktorých sú tieto metódy neefektívne alebo úplne nepoužiteľné. Takéto typy úloh sú spravidla veľmi rozsiahle a komplikované, a použitím klasických algoritmov nie je možné získať výsledok v reálnom čase. Je preto potrebné znížiť nároky na presnosť a zvoliť heuristické metódy a spôsob výpočtu [38].

Typický inžiniersky prístup k riešeniu problémov je založený na využití určitých pravidiel a na ľudskej skúsenosti [35]. Evolučné algoritmy sa naopak snažia problém previesť na úlohu prehľadávania priestoru S všetkých kandidátnych riešení daného problému, s cieľom nájsť to najlepšie. V prípade jednokriteriálnej optimalizácie to znamená minimalizovať alebo maximalizovať tzv. účelovú funkciu $f(x) : S \to \mathbb{R}$, kde optimálne riešenie je $x_{opt} = arg\ min\ f(x)$, resp. $x_{opt} = arg\ max\ f(x)$.

2.1 História

Myšlienka evolučných algoritmov vychádza z biologického procesu evolúcie, tak ako ho opísal Charles Darwin vo svojej knihe *O pôvode druhov* [6]. Darwin popisuje princíp "prirodzeného výberu" jedincov, ktorý umožňuje silnejším jedincom prežiť, mať viac potomkov ako slabší jedinci a posunúť svoje vlastnosti ďalším generáciám. Ďalší dôležitý objav v oblasti evolúcie urobil Gregor Mendel [12], ktorý popísal princíp dedičnosti vlastností na úrovni génov, z jednej generácie na ďalšiu. Oba prístupy (a mnohé ďalšie modernejšie) dnes integruje pokročilejšia evolučná teória, nazývaná *neodarwinizmus*. Tá pracuje na úrovni DNA, kde popisuje princípy kombinácie a mutácie vlastností jedincov.

Výskum v oblasti evolučný algoritmov sa začal rozširovať až v druhej polovici 20. storočia, pričom termín "evolučné algoritmy" vznikol až na začiatku 90. rokov, kedy spolu začali výskumníci z tejto oblasti spolupracovať [35]. Historicky nezávisle na sebe vznikli štyri základné varianty EA:

- evolučné stratégie [32],

- evolučné programovanie [11],
- genetické algoritmy [15],
- genetické programovanie [19].

Jednotlivé varianty však majú určité spoločné rysy. Medzi hlavné patrí práca s populáciou jedincov, z ktorých vznikajú noví jedinci v snahe nájsť toho najlepšieho. Pri tvorbe novej populácie tieto algoritmy využívajú určité operátory, inšpirované v genetike. Medzi ne patrí výber rodičov (*selekcia*), ich následné kríženie a náhodná mutácia v génoch nového jedinca. U jednotlivých jedincov je vždy vyhodnotená ich kvalita, ktorá určuje pravdepodobnosť ich následného výberu. V tejto práci sa budeme ďalej venovať genetickým algoritmom a ich variante *NSGA-II*.

2.2 Genetické algoritmy

Tvorcom a priekopníkom v oblasti genetických algoritmov sa stal americký teoretický biológ John Holland [15], ktorý študoval elementárne procesy v populáciach, ktoré sú z hľadiska evolúcie nepostrádateľné. Na základe svojho výzkumu neskôr navrhol genetický algoritmus, ako abstrakciu k príslušným biologickým procesom [38].

Genetické algoritmy čerpajú inšpiráciu z evolučného vývoja jedincov, tzn. snažia sa vývoj napodobniť tak, ako prebieha v prírode. Ako teoretický základ je braná Darwinova teória prirodzeného výberu [6] a Mendelova teória prenosu dedičných informácií.

Pri prehľadávaní stavového priestoru sa obvykle využívajú dva rôzne prístupy. Snaha o prehľadanie čo najväčšej časti stavového priestoru (exploration) a snaha o čo najpodrobnejšie preskúmanie sľubných oblastí stavového priestoru (explotation). Tieto prístupy sú navzájom protichodné a pri hľadaní optimálneho riešenia sa odporúča nezameriavať sa iba na jeden, ale zvoliť optimálnu kombináciu oboch. Výhodou GA je, že nastavením vstupných parametrov je možné docieliť ľubovoľné vyváženie medzi týmito dvoma prístupmi.

Základná varianta GA je veľmi všeobecná a umožňuje širokú variabilitu implementácie jednotlivých aspektov algoritmu, v závislosti od riešeného problému. Medzi tieto aspekty patrí napríklad typ kódovania a reprezentácie chromozómu, stratégia výberu rodičov, typ kríženia a mutácie. Základný popis genetických algoritmov v tejto kapitole bol spracovaný podľa [25].

2.2.1 Základné pojmy

V genetických algoritmoch sa taktiež používa terminológia prevzatá z biológie. Pracuje sa s jedincami, ktorí sú často označovaní ako genotypy alebo chromozómy. V biológií majú jedinci bežne niekoľko chromozómov, avšak v GA sa väčšinou pracuje s jedincami s jedným chromozómom, preto je možné tieto pojmy zjednotiť. Chromozómy sú tvorené lineárne usporiadanou postupnosťou génov, ktorých stav je reprezentovaný alelami. Každý jedinec (chromozóm) reprezentuje jedno kandidátne riešenie problému. Jeho kvalita je určená pomocou tzv. fitness hodnoty, ktorá určuje mieru zdatnosti (kvality) daného jedinca.

Gén

V EA predstavujú gény jednotlivé parametre, ktorých optimálne hodnoty sa snažíme nájsť. Platí, že gén je základná stavebná jednotka chromozómu a môže nadobúdať iba hodnoty z definovanej abecedy, napr. binárne alebo celé čísla. Konkrétna hodnota génu sa nazýva alela. V porovnaní s biológiou, je pojem gén v EA značne zjednodušený.

Chromozóm

Chromozóm je lineárna postupnosť génov, kde pre chromozómy rovnakého typu platí, že gén na i-tej pozícií kóduje vždy tú istú vlastnosť. Konkrétna pozícia génu v chromozóme sa nazýva lokus. Návrh optimálneho kódovania chromozómu je jeden z hlavných problémov EA a je vždy závislý od konkrétneho riešeného problému.

Genotyp

Genotyp je súbor všetkých génov jedinca. Bežne je reprezentovaný ako reťazec génov, ktorý kóduje jednotlivé vlastnosti daného jedinca. Z genotypu je následne možné získať konkrétnu podobu kandidátneho riešenia a to jeho vhodnou interpretáciou. V GA sa často nerozlišuje medzi pojmami genotyp a chromozóm, nakoľko jedinci sú bežne reprezentovaní jedným chromozómom. Existujú však aj varianty GA, ktoré uvažujú jedincov s viacerými chromozómami.

Fenotyp

Z biologického hľadiska je fenotyp súbor pozorovateľných znakov a vlastností jedinca v danom prostredí. To znamená, že predstavuje konkrétnu podobu jedinca, ktorého vlastnosti sú kódované v genotype. V evolučných algoritmoch predstavuje fenotyp konkrétnu reprezentáciu genotypu, napr. schému obvodu, ohodnotenie jednotlivých premenných a pod. Konkrétny spôsob reprezentácie genotypu je bežne závislý na riešenom probléme.

Jedinec

Jedinec je jeden predstaviteľ z populácie. Je definovaný genotypom, resp. chromozómom a platí, že každý jedinec predstavuje jedno kandidátne riešenie. Konkrétne kandidátne riešenie môže byť v danej generácií reprezentované viacerými jedincami.

Fitness hodnota

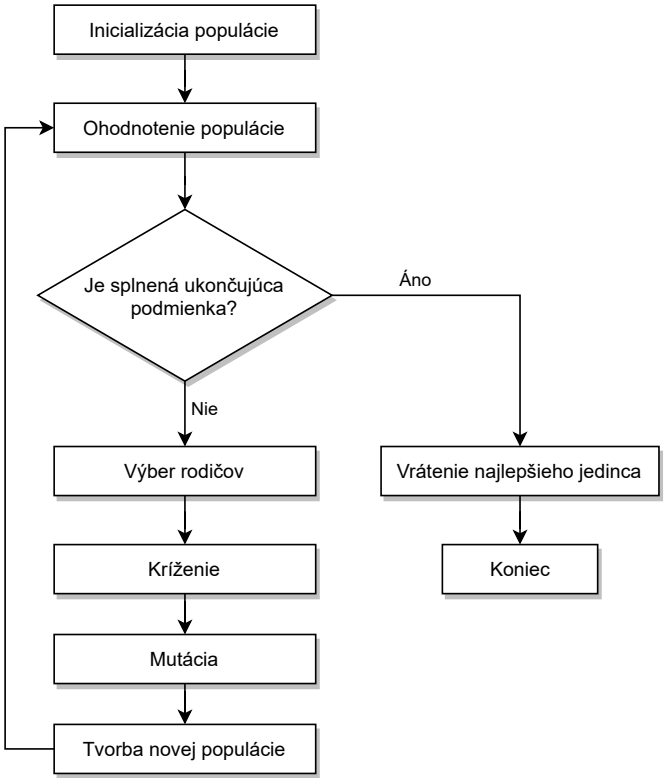
Fitness hodnota je numerické ohodnotenie kvality kandidátneho riešenia, ktoré je reprezentované konkrétnym jedincom. Jej hodnota sa počíta pomocou tzv. fitness funkcie, ktorej implementácia je priamo závislá na riešenom probléme.

Populácia

Populácia je množina obsahujúca všetky kandidátne riešenia daného problému v aktuálnej iterácií výpočtu. Je to štruktúra obsahujúca n chromozómov, kde platí $n \geq 1$ a počas výpočtu sa počet jedincov v populácií nemení. Menia sa iba samotní jedinci naprieč jednotlivým generáciam (iteráciam) výpočtu.

2.2.2 Základná myšlienka

Ako už bolo povedané, základnou myšlienkou genetických algoritmov je evolučný proces v prírode, ktorý je pre účely výpočtu značne zjednodušený. Počas výpočtu sa pracuje s populáciou jedincov, kde každý jedinec je tvorený jedným chromozómom, tzn. jedným kandidátnym riešením problému. Každý jedinec je posudzovaný podľa kvality riešenia, ktoré reprezentuje. Táto miera sa nazýva fitness hodnota, ktorá je vypočítaná pomocou fitness funkcie. Na základe tejto hodnoty sú jedinci vyberaní na vytvorenie novej populácie, pričom platí, že jedinci s vyššou fitness hodnotou majú väčšiu šancu na výber, tzn. platí prirodzená selekcia. Pri tvorbe novej populácie platí, že dvaja jedinci z aktuálnej populácie vytvoria dvoch potomkov. Pri vytváraní potomka dochádza ku kríženiu genetickej informácie z oboch rodičov. Aby sa však do výpočtu vložila aj istá variabilita medzi jedincami, bola ako variačný operátor zavedená mutácia v genotype jedincov. Mutácia je určitá náhodná zmena v génoch nového jedinca. Selekcia, kríženie a mutácia sú spoločne označované ako genetické operátory.



Obr. 2.1: Schéma genetického algoritmu

Základná schéma genetického algoritmu je zobrazená na obrázku 2.1, kde výpočet začína vytvorením počiatočnej populácie, ktorá sa skladá z n náhodne vygenerovaných jedincov. Počet jedincov v populácií je závislý na konkrétnom riešenom probléme, typicky sa však pracuje so stovkami až tisícmi jedincov. Každý jedinec je následne ohodnotený pomocou fitness funkcie a je vypočítaná jeho kvalita. Na základe fitness hodnoty sú postupne vybrané dvojice rodičov, kde ich krížením vznikajú noví jedinci. Títo jedinci môžu byť ešte s určitou pravdepodobnosťou mutovaní. Postup selekcie, kríženia a mutácie sa opakuje, kým nie je vytvorený požadovaný počet nových jedincov, ktorí sú následne ohodnotení fitness funkciou. Novo vytvorená populácia následne nahrádza starú s tým, že v určitých variantách genetických algoritmov sa pracuje s tzv. elitizmom. Ten zabezpečuje, že určitý počet najlepších jedincov z pôvodnej populácie sa dostane do novej. Rovnaký počet najhorších jedincov z novej populácie je následne vyhodený. Výpočet je následne ukončený, ak niektorý z jedincov v novej populácií spĺňa ukončujúcu podmienku, prípadne ak bol dosiahnutý maximálny počet generácií. V takom prípade je ako výsledné riešenie vrátený jedinec s najlepším ohodnotením.

2.2.3 Kódovanie chromozómu

V typickom prípade použitia genetických algoritmov je riešený problém prevedený na postupnosť genetických charakteristík, tzn. parametrov, ktoré sa budú optimalizovať. Napríklad pre funkciu

$$min\ f(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 2)^2,\ \forall x_1, x_2 : -3 \leq x_1 \leq 3 \wedge -8 \leq x_2 \leq 8$$

sú optimalizované parametre x_1 a x_2, pričom obe premenné majú presne definovaný obor hodnôt, ktoré môžu nadobúdať [1].

Pred samotným výpočtom je potrebné určiť akým spôsobom budú jednotlivé chromozómy kódované. Chromozómy sú reprezentované lineárne usporiadanými reťazcami rovnakej dĺžky. Správne kódovanie chromozómov je priamo závislé na konkrétnej úlohe a taktiež od neho závisí úspech, či neúspech výpočtu. Existuje niekoľko bežne používaných variant kódovania, pričom každé má svoje výhody, či nevýhody [38].

Binárne kódovanie

Najbežnejším typom kódovania je binárne kódovanie, kde je každý chromozóm kódovaný ako postupnosť 0 a 1. Pre veľa problémov však toto kódovanie nie je prirodzené a je potrebné vykonať opravy po krížení alebo mutácií [38].

Genetické algoritmy predpokladajú, že malá zmena v chromozóme sa prejaví ako malá zmena vlastnosti jedinca, napríklad zmena spôsobená mutáciou sa prejaví ako malá zmena vo fenotype. Avšak klasické binárne kódovanie tento predpoklad narušuje. Napríklad susedné hodnoty 7 a 8 sú v binárnom kódovaní reprezentované ako 0111 a 1000, tzn. že bolo potrebné zmeniť všetky štyri bity. Tento nedostatok je možné riešiť využitím neštandardného typu kódovania, ako je napr. Grayov kód. V tom sú susedné hodnoty kódované binárne tak, že sa líšia práve v jednom bite.

Príklad binárneho kódovania:

Chromozóm A	11011001010110
Chromozóm B	10011010001010

Kódovanie hodnotou

Kódovanie hodnotou je vhodné použiť na optimalizačné problémy, ako napríklad pri kódovaní váh neurónových sietí. V takomto prípade obsahuje chromozóm reálne čísla, ktoré reprezentujú jednotlivé váhy [1]. Vo všeobecnosti je možné použiť kódovanie hodnotou na problémy, kedy by bolo použitie binárneho kódovania príliš komplikované. Chromozóm je potom reťazec ľubovoľných hodnôt, ktoré sú spojené s problémom. Je ale potrebné prispôsobiť danému kódovaniu aj operácie kríženia a mutácie [25].

Príklady kódovania hodnotou:

Chromozóm A	0.215 0.36 0.84 0.236 0.452 0.195
Chromozóm B	*(up) (right) (down) (left) (down)*

Permutačné kódovanie

Permutačné kódovanie je možné použiť pri problémoch usporiadania prvkov. Medzi takéto typy úloh patrí napríklad problém obchodného cestujúceho (TSP - Traveling Salesman Problem). Každý chromozóm sa skladá z pevného počtu znakov alebo číslic, ktoré určujú poradie spracovania jednotlivých kódovaných vlastností. Každý chromozóm musí obsahovať všetky znaky vstupnej abecedy a je preto potrebné prispôsobiť aj operácie kríženia a mutácie, aby každý nový jedinec spĺňal túto podmienku [1].

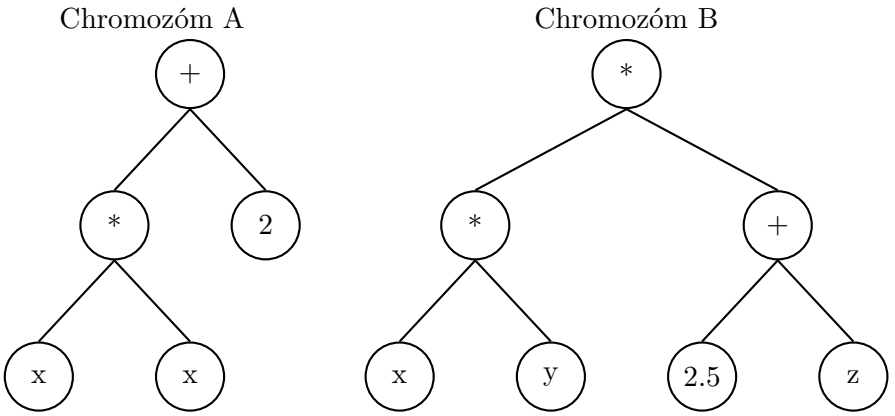
Príklad permutačného kódovania:

Chromozóm A	1 3 5 2 9 8 6 4 7
Chromozóm B	B A D F E G C

Stromové kódovanie

Stromové kódovanie nachádza uplatnenie najmä v genetickom programovaní, kde sú jednotlivé kandidátne riešenia (programy) reprezentované pomocou syntaktických stromov. Pri takomto kódovaní platí, že jedinci tvoria vždy syntakticky správne a spustiteľné programy. Aby bola táto vlastnosť zachovaná, je potrebné prispôsobiť aj genetické operátory kríženia a mutácie. Na rozdiel od ďalších druhov kódovania, pri stromovom platí, že dĺžka chromozómov môže byť premenlivá.

Príklad stromového kódovania:



2.2.4 Postup výpočtu

Vytvorenie počiatočnej populácie

Pri inicializácií výpočtu je vytvorených n jedincov, ktorým sú vygenerované počiatočné hodnoty chromozómov. Inicializácia hodnôt je obvykle náhodná, avšak môžu byť využité určité vlastnosti o riešenom probléme, čo umožní využitie rôznych heuristík a tým generovanie jedincov s hodnotami bližšími optimu. Počet jedincov v populácií je rovnaký počas celého priebehu výpočtu a závisí od riešeného problému. Ich počet je však bežne niekoľko stoviek až tisíc, kvôli zaisteniu dostatočnej genetickej rozmanitosti [2].

Ohodnotenie jedincov

V každej iterácií výpočtu je potrebné ohodnotiť kvalitu všetkých jedincov. Hodnotenie jedincov je vykonávané pomocou tzv. fitness funkcie, ktorá určuje stupeň adaptácie chromozómu s ohľadom na riešený problém. Výsledkom fitness funkcie je fitness hodnota, ktorej hodnotu sa snažíme maximalizovať, kde maximálna hodnota označuje optimálne riešenie problému. Okrem fitness funkcie sa využíva aj tzv. účelová funkcia, ktorá určuje kvalitu kandidátneho jedinca s ohľadom na riešený problém. Jej výsledok môže byť totožný s fitness hodnotou alebo môže byť použitá ako medzivýsledok na výpočet fitness hodnoty [3]. Taktiež je okrem kvality jedincov v každej iterácií počítaná aj kvalita samotnej populácie. Tá je vypočítaná ako priemerná hodnota fitness hodnôt všetkých jedincov z aktuálnej generácie.

Nakoľko je schopnosť genetického algoritmu nájsť správne riešenie silne závislá od správneho návrhu a implementácie fitness funkcie, je potrebné aby boli vždy splnené určité požiadavky:[1]

1. Fitness funkcia by mala byť jasne definovaná, tzn. výpočet fitness hodnoty by mal byť ľahko pochopiteľný.

[1] https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4.

2. Implementácia fitness funkcie by mala byť implementovaná efektívne. Ak by sa výpočet fitness hodnoty stal kritickým bodom algoritmu, celková účinnosť genetického algoritmu by sa znížila.

3. Fitness funkcia by mala vrátiť kvantitatívne ohodnotenie kvality jedinca.

4. Fitness funkcia by mala generovať intuitívne výsledky, tzn. pre najlepších/najhorších jedincov by mala vracať najlepšie/najhoršie ohodnotenie.

Selekcia

Selekcia je výber rodičov (dvojíc jedincov) na tvorbu novej generácie kandidátnych riešení. Obecne v EA platí výber rodičov s preferenciou lepších (silnejších) jedincov, čo simuluje prirodzený výber jedincov v prírode. Do výberu rodičov sa však môžu dostať aj jedinci s horším ohodnotením, aby bola zaručená dostatočná variabilita medzi jedincami a predišlo sa uviaznutiu v lokálnych extrémoch účelovej funkcie. Na zaručenie takéhoto výberu je potrebné využiť niektorý z pravdepodobnostných mechanizmov výberu. Pred samotným výberom je potrebné prepočítať kvalitu jedincov na pravdepodobnosti ich výberu ako rodičov. Táto pravdepodobnosť je prepočítavaná z fitness hodnoty jedinca a z fitness hodnoty zvyšných jedincov v populácií. Po vypočítaní pravdepodobností je možné na výber rodičov použiť niektorý z typicky využívaných mechanizmov selekcie [1]:

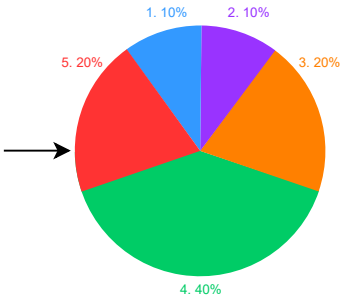
- **Ruleta** je proporcionálny typ selekcie jedincov, kde je na začiatku každému jedincovi priradená pravdepodobnosť jeho výberu na základe jeho fitness hodnoty. Pravdepodobnosť p_i je vypočítaná ako

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2.1}$$

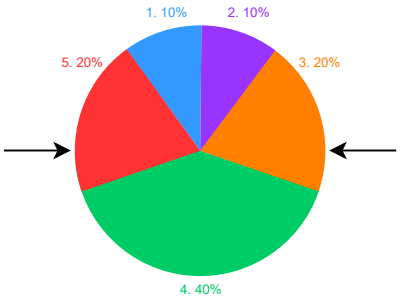
  kde N je veľkosť populácie, $i \in \{1, ..., N\}$ je index jedinca v populácií a f_i je fitness hodnota daného jedinca. Cieľom tohto prístupu je lepším jedincov zvýšiť šancu aby boli vybraní ako rodičia, avšak priradiť určitú, nenulovú pravdepodobnosť aj horším jedincom, aby sa zvýšila diverzita populácie.

  Postup výberu si môžeme predstaviť ako hru ruleta, kde je herné koleso rozdelené na N častí o veľkosti odpovedajúcej pravdepodobnosti jednotlivých jedincov. Následne sa kolesom "zatočí" a vyberie sa jedinec na ktorého segmente sa koleso zastavilo. V praxi sa využíva interval $< 0, 1 >$, ktorý je rozdelený na N segmentov. Príklad rulety je možné vidieť na obrázku 2.2. Následne je náhodne vygenerované číslo a vyberie sa jedinec v ktorého segmente sa číslo nachádza. Výber sa opakuje, kým nie je vybraný požadovaný počet jedincov.

- **Stochastické náhodné vzorkovanie** je obdobou algoritmu rulety. Rovnako ako pri rulete sa vytvorí "koleso" s rozdelením oblastí odpovedajúcim pravdepodobnostiam jedincov, avšak na ruletu v tomto prípade ukazuje toľko ukazovateľov, koľko sa vyberá rodičov, tzn. že všetci rodičia sú vybraní po prvom zatočení. Príklad stochastického náhodného vzorkovania môžte vidieť na obrázku 2.3.



Obr. 2.2: Princíp výberu jedincov podľa rulety



Obr. 2.3: Princíp výberu jedincov na základe stochastického náhodného vzorkovania

- **Turnaj** je spôsob selekcie jedincov, kedy je vytvorená skupina náhodne vybraných m jedincov, pričom platí $2 \leq m \leq n$, kde n je veľkosť populácie. Z tejto skupiny sa následne stáva rodič jedinec s najlepšou fitness hodnotou (víťaz turnaja). Následne sa turnaj opakuje, kým nie je vybraný požadovaný počet rodičov. Tento spôsob simuluje boj medzi jedincami v prírode o právo ďalšej reprodukcie. Výhodou tohto spôsobu je, že umožňuje aj jedincom s nižším fitness aby sa stali rodičmi, čo vnáša do výpočtu väčšiu genetickú variabilitu a umožňuje predchádzať uviaznutiu v lokálnych extrémoch.

Kríženie

Kríženie je jeden z hlavných genetických operátorov, ktorý simuluje náhodnú výmenu informácií medzi rodičmi, ktorú posúvajú na svojho potomka. Typicky sa v genetických algoritmoch zvolia dvaja rodičia, ktorých krížením vzniknú dvaja noví jedinci. V ideálnom prípade by mali byť noví jedinci kvalitnejší ako ich rodičia, môže sa však stať, že ich kvalita bude horšia. To však umožňuje predchádzať uviaznutiu v lokálnych extrémoch účelovej funkcie [38].

Vo všeobecnosti kríženie spočíva vo vzájomnej výmene určitých častí chromozómov. Bežne sa používa niekoľko spôsobov kríženia, pričom sú jednotlivé typy závislé od použitého spôsobu kódovania chromozómov:

- **Jednobodové kríženie** je najjednoduchší spôsob kríženia. Spočíva v náhodnom stanovení bodu kríženia, pričom pre chromozómy dĺžky n môže tento bod nadobúdať hod-

noty 1 ... $n-1$. Potom platí, že prvý potomok vznikne kombináciou prvej časti prvého rodiča s druhou časťou druhého rodiča. Pre druhého potomka je postup opačný:

Rodič A	0 1 1 1 0 1	1 1 0 0
Rodič B	1 1 1 0 0 1	1 0 0 1
1. potomok	0 1 1 1 0 1	1 0 0 1
2. potomok	1 1 1 0 0 1	1 1 0 0

- **Viacbodové kríženie** je rozšírenou variantou jednobodového. Náhodne je vygenerovaných niekoľko bodov kríženia, minimálne aspoň 2. Pre chromozómy dĺžky n môžu nadobúdať hodnôt 1 ... $n-1$. Potomkovia sú následne vytváraný obdobne ako pri jednobodovom krížení s tým, že sa vlastnosti od rodičov dedia striedavo podľa bodov kríženia:

Rodič A	0 1	1 1 0 1	1 1 0	0
Rodič B	1 1	1 0 0 1	1 0 0	1
1. potomok	0 1	1 0 0 1	1 1 0	1
2. potomok	1 1	1 1 0 1	1 0 0	0

- **Uniformné kríženie** využíva pravdepodobnostnú mieru prenosu informácie od jednotlivých rodičov. Na začiatku je pevne stanovená miera pravdepodobnosti, typicky 50% a s tou sa jednotlivé gény prenášajú od rodičov k potomkom:

Rodič A	0 1 1 1 0 1 1 1 0 0
Rodič B	1 1 1 0 0 1 1 0 0 1
1. výber	1 2 1 1 2 1 2 2 2 1
2. výber	2 2 1 2 1 1 2 1 2 2
1. potomok	0 1 1 1 0 1 1 0 0 0
2. potomok	1 1 1 0 0 1 1 1 0 1

- **Aritmetické kríženie** je využívané pri chromozómoch kódovaných reálnymi hodnotami. Hodnoty génov sú vyberané náhodne s uniformným pravdepodobnostným rozložením koeficientu a_i [45]:

$$
\begin{aligned}
g_i^{O_1} &= g_i^{P_1} a_i + g_i^{P_2}(1 - a_i),\ a_i \in [0,1] \\
g_i^{O_2} &= g_i^{P_1}(1 - a_i) + g_i^{P_2} a_i
\end{aligned}
$$

g_i ... hodnota génu i
P_j ... rodič (parent), $j \in \{1,2\}$
O_k ... potomok (offspring), $k \in \{1,2\}$

- **Kríženie v permutačných problémoch** je iný typ kríženia ako predchádzajúce. V permutačných problémoch nie je totiž možné vytvoriť nového jedinca iba kombináciou určitých častí rodičov. V takomto prípade by sa totiž do potomka nemuseli dostať všetky hodnoty a niektoré by sa naopak opakovali. Tento problém je možné riešiť s využitím indexových tabuliek alebo použitím kríženia typu PMX.

Mutácia

Mutácia je variačný operátor, ktorý umožňuje predísť predčasnej konvergencii výpočtu a poskytuje možnosti náhodného prehľadávania v blízkom okolí konvergovanej populácie. Blízke okolie môžeme chápať ako množinu jedincov, ktorí sa od jedincov v populácií líšia v minimálnom počte bitov [38]. Vo všeobecnosti platí, že mutácia je náhodná zmena v génoch chromozómu. V genetických algoritmoch sa s určitou, predom danou pravdepodobnosťou, umelo zmení hodnota náhodného génu a náhodne vybraného potomka. Pravdepodobnosť mutácie sa obvykle volí ako malá hodnota, typicky $1/n$, kde n je počet jedincov v populácií. Maximálne by však hodnota pravdepodobnosti mutácie mala byť aspoň 0,001, pri populáciách menších ako 1000 jedincov. Samotná zmena hodnoty pri mutácií závisí od použitého kódovania chromozómu. Pri binárnom kódovaní je mutáciou invertovaná hodnota náhodne vybraného génu u zvoleného potomka. Pri chromozómoch s kódovaním reálnymi hodnotami sa k vybranému génu pripočíta určitá malá hodnota, tá by však nemala presiahnuť 10% pôvodnej hodnoty s tým, že môže byť aj záporná [45].

Vytvorenie novej generácie

Pri vytváraní novej generácie sa vytvára nová populácia z novovytvorených jedincov a z jedincov starej populácie. Existuje niekoľko nahradzovacích stratégií, ktoré určujú, koľko nových a starých jedincov sa dostane do novej populácie:

- **Generačný model** je najjednoduchší spôsob tvorby novej populácie. Jedinci starej populácie sú vždy jednoducho nahradení svojimi potomkami. Každá populácia preto vždy obsahuje iba nových jedincov.
- **Inkrementálny model** je model s postupným nahradzovaní starých jedincov novými. V každej novej generácií je nahradený jeden, najhorší jedinec starej populácie jedným, najlepším potomkom.
- **Model s prekrytím generácií** je extrémnym prípadom predošlých dvoch modelov. V novej populácií je totižto nahradená vždy iba časť jedincov zo starej populácie. Najčastejšie sa používajú tri spôsoby ako prekryť jedincov z jednotlivých generácií:
  - Je vybraný určitý počet potomkov, spravidla najlepších, ktorý s rovnomerným rozložením pravdepodobnosti nahradia náhodne vybraných rodičov.
  - Zo starej populácie je vybraný určitý počet najlepších rodičov (elita), ktorý v novej populácií nahradia najhoršie ohodnotených potomkov. Táto metóda je vhodná najmä preto, aby sa predišlo strate doteraz najlepšieho nájdeného kandidátneho riešenia.
  - Rodičia aj potomkovia sú zoradení podľa kvality ich kandidátnych riešení a do novej populácie sa vyberie n najlepších jedincov, kde n je veľkosť populácie.

Ukončenie výpočtu

Nakoľko už z podstaty genetického algoritmu vyplýva, že nie sme predom schopní určiť k akému výsledku sa výpočet dostane, je potrebné pred samotným behom algoritmu určiť podmienky, na základe ktorých bude výpočet ukončený. Základnou podmienkou je nájdenie jedinca, ktorý reprezentuje správne (optimálne) riešenie. Ak je takýto jedinec nájdený, výpočet je ukončený. Avšak, ak nie je možné takéhoto jedinca nájsť, je možné výpočet ukončiť z niekoľkých ďalších dôvodov. Napríklad, ak by sa celková kvalita populácie po niekoľko generácií nezvyšovala alebo by bol dosiahnutý vopred stanovený počet generácií. V takomto prípade je ako najoptimálnejšie riešenie vrátené kandidátne riešenie najlepšie hodnoteného jedinca. Pri riešení problémov o ktorých sa vopred vie, že budú časovo veľmi náročné, je možné vopred zvoliť maximálnu dobu behu programu, po ktorej uplynutí je výpočet zastavený a je vrátené kandidátne riešenie najlepšieho jedinca.

2.3 NSGA-II

Klasické evolučné algoritmy uvažujú optimalizáciu na základe jedného kritéria. V praxi je ale typicky potrebné riešiť problémy, kde je potrebné pracovať s niekoľkými konfliktnými kritériami. Takéto problémy sa nazývajú multikriteriálne a je potrebné na ich riešenie využiť multikriteriálne optimalizačné algoritmy *MOEA (Multiobjective Optimization Evolutionary Algorithms)*. V tejto práci budeme ďalej pracovať s algoritmom NSGA-II *Nondominated Sorting Genetic Algorithm - II* [7], čo je multikriterálna varianta genetických algoritmov.

2.3.1 Multikriteriálna optimalizácia

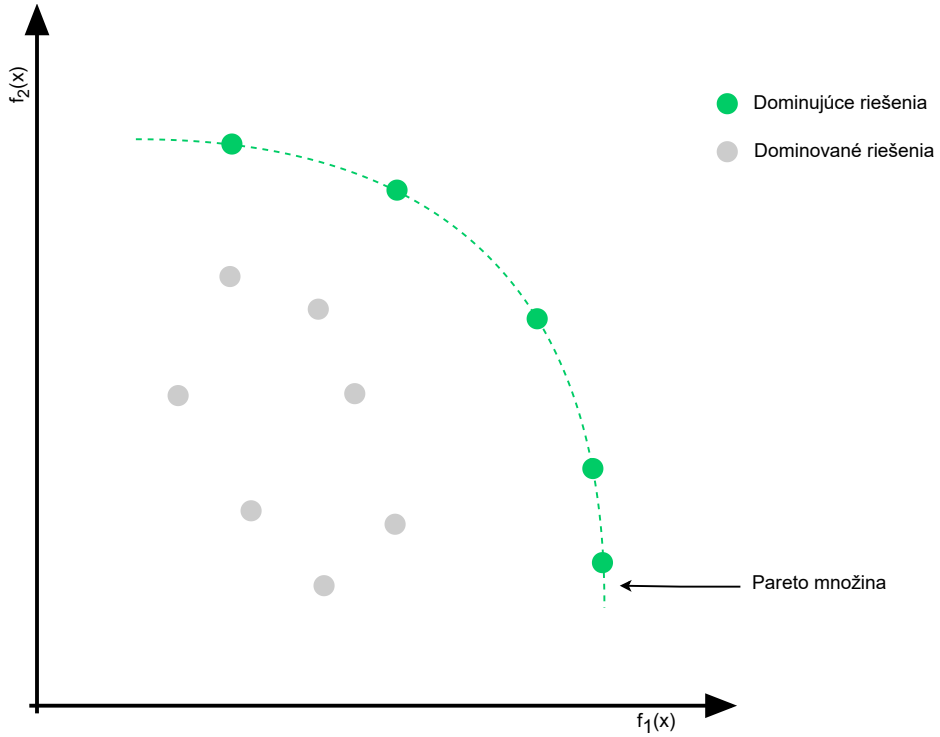
Pri riešení multikriteriálnych optimalizačných problémov sa snažíme optimalizovať niekoľko konfliktných účelových funkcií, pričom platí, že pri zlepšení hodnôt jednej funkcie sa zároveň zhorší hodnota druhej. Nie je teda možné nájsť také riešenie, ktoré by maximalizovalo (resp. minimalizovalo) všetky účelové funkcie. Nakoľko nie je možné nájsť jedno optimálne riešenie, je výstupom výpočtu tzv. *Pareto množina*. Je to množina vzájomne si nedominujúcich riešení, pre ktorú platí

$$P = \{x \in \Omega \mid \nexists y \in \Omega : y \succ x\} \tag{2.2}$$

kde $x, y \in \Omega$ označuje riešenie z množiny riešení Ω a $x \succ y$ je relácia *Pareto dominancie*. Dominujúce riešenie môžeme chápať ako určité riešenie x, ktoré je aspoň rovnako dobré vo všetkých kritériach a lepšie aspoň v jednom kritériu ako nejaké iné riešenie y. Ukážku Pareto množiny môžeme vidieť na obrázku 2.4.

2.3.2 Postup výpočtu

Základná schéma výpočtu algoritmu NSGA-II je rovnaká ako v základnej variante genetických algoritmov. To znamená, že sa na začiatku vytvorí počiatočná populácia kandidátnych



Obr. 2.4: Ukážka Pareto množiny pre problém maximalizácie účelových funkcií

riešení, následne sa vypočíta ich fitness hodnota a vytvorí sa populácia potomkov. Rozdiel je ale vo vytvorení novej populácie. To využíva radenie jedincov podľa ich Pareto dominancie.

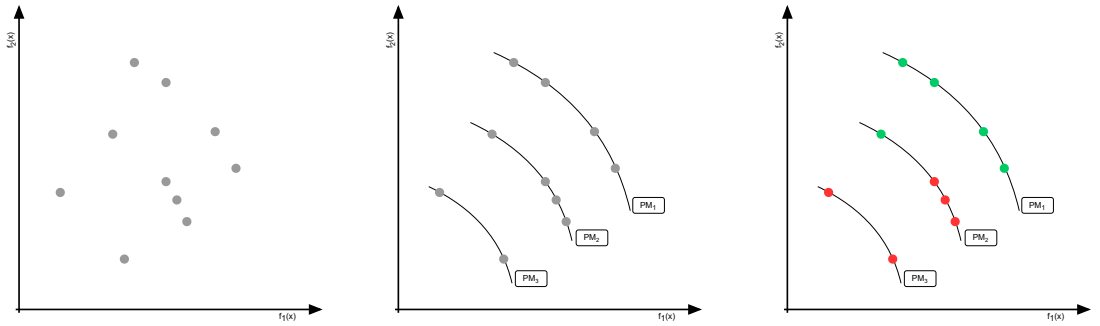
Pri radení jedincov sa na začiatku vytvorí zmiešaná populácia tvorená jedincami starej populácie a ich potomkami. Následne sa pre každého jedinca vyhodnotí či je dominovaný nejakým iným jedincom. Ak dominovaný nie je, vyberie sa z množiny a vloží sa do Pareto množiny. Vybraní jedinci sú odstránení z pomocnej množiny zmiešanej populácie. Takto sa vytvorí niekoľko Pareto množín, kým nie sú priradení všetci jedinci. Pre každú Pareto množinu sa pre každého jedinca v nej vypočíta tzv. zhluková vzdialenosť *(crowding distance)*, ktorá určuje vzdialenosť jedinca od ostatných jedincov v danej Pareto množine. Využíva sa pri výbere jedincov z Pareto množiny, ktorá sa celá nezmestí do novej populácie. Slúži najmä na zvýšenie diverzity medzi jedincami a je počítaná ako

$$D(x_i) = \sum_{j=1}^{N} \frac{d_j^i}{max(f_j) - min(f_j)} \tag{2.3}$$

kde $D(x_i)$ je vzdialenosť jedinca x_i od susedných jedincov x_{i-1} a x_{i+1}, N je počet účelových funkcií a d_j^i je vzdialenosť od susedných jedincov podľa účelovej funkcie f_j. Hodnota d_j^i sa vypočíta ako

$$d_j^i = |f_j(x_{i+1}) - f_j(x_{i-1})| \tag{2.4}$$

Pre krajných jedincov platí, že ich vzdialenosť je rovná ∞, z čoho vyplýva, že sa vždy dostanú do novej populácie. V prípade, že sa nezmestia obaja, je vybraný iba jeden z nich. Po vypočítaní zhlukových vzdialeností pre všetky Pareto množiny sa kandidátne riešenia zoradia podľa úrovne ich Pareto množiny (index iterácie v ktorom bola množina vytvorená) a následne podľa hodnoty ich zhlukovej vzdialenosti v danej úrovni. Po zoradení jedincov je vybraných n najlepších, kde n je veľkosť populácie. Grafickú ukážku tvorby jednotlivých Pareto množín s následnou tvorbou novej populácie môžeme vidieť na obrázku 2.5.



(a) Pomocná populácia (b) Vytvorenie Pareto množín (c) Výber n najlepších jedincov

Obr. 2.5: Tvorba novej populácie z Pareto množín

Kapitola 3

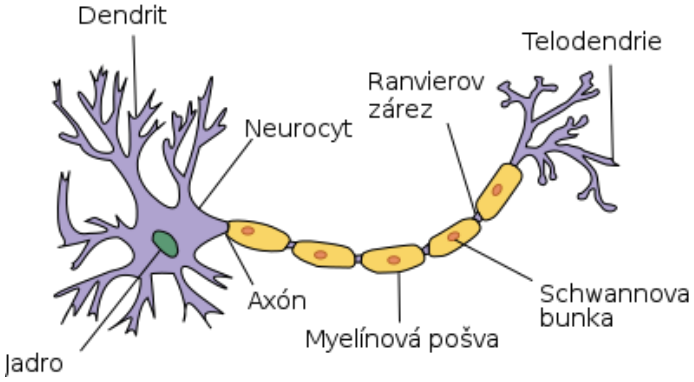
Umelé neurónové siete

Umelé neurónové siete (Artificial Neural Networks) predstavujú jeden z mnohých výpočetných modelov využívaných v oblasti umelej inteligencie. Hlavnú inšpiráciu čerpajú v ľudskom mozgu a v procesoch, ktoré v ňom prebiehajú. Ich základnou zložkou sú umelé neuróny, ktoré sú abstrakciou biologických neurónov. Jednotlivé neuróny sú prepojené do vrstiev a sú schopné reagovať na vonkajšie podnety, a na základe toho upravovať svoj výstup, a odozvu celej siete. Toto chovanie je abstrakciou učenia v mozgu.

Termín "neurónové siete" je prevzatý z práce neurovedcov Warrena McCullocha a Waltera Pittsa [26], ktorí v roku 1943 prišli s prvým konceptuálnym návrhom umelých neurónových sietí. Vo svojej práci popisujú koncept neurónu, inšpirovaného biologickým neurónom, ktorý je organizovaný do hierarchie (siete), ktorá je schopná prijímať vstupy, spracovávať ich a generovať odpovedajúce výstupy.

3.1 Umelý neurón

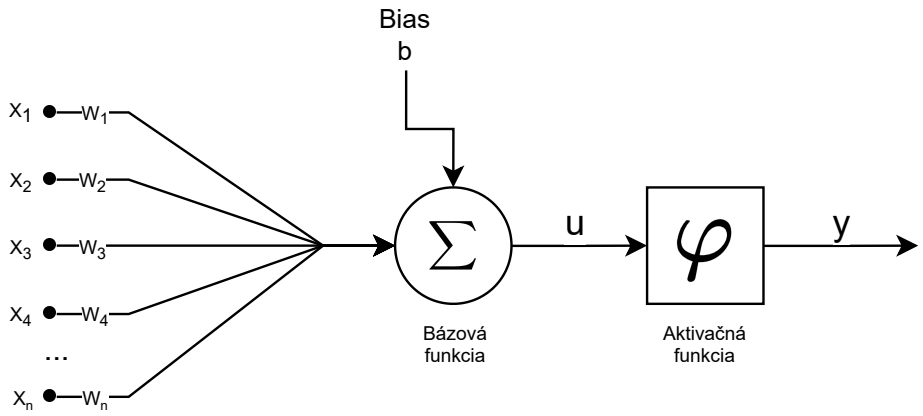
Neurón je základná výpočetná jednotka v neurónovej sieti. Jeho funkcia je veľmi jednoduchá, je to v podstate "spínač", ktorý sa zopne, ak má na vstupe dostatočne veľký impulz. Následne je informácia z neho zaslaná ďalšiemu neurónu alebo na výstup siete.



Obr. 3.1: Biologický neurón, prevzaté z [8]

Model umelého neurónu vychádza z biologického neurónu, ktorý sa skladá z niekoľkých častí. Jeho vyobrazenie je na obrázku 3.1. Hlavnú časť tvorí telo bunky, nazývané tiež soma, obsahujúca jadro okolo ktorého je množstvo malých výbežkov. Tie sa nazývajú dendrity. Slúžia ako vstup neurónu a prijímajú impulzy od iných neurónov. Výstup neurónu sa nazýva axón, ktorý má podobne ako dendrit niekoľko výbežkov, aby mohol byť spojený s viacerými neurónmi. Spojenie s inými neurónmi je zabezpečené pomocou špeciálnych väzieb, nazývaných synaptické väzby. Tie spájajú výstup (axon) jedného neurónu so vstupom (dendritom) ďalšieho. Synapsie sú elektrochemické väzby, ktoré prenášajú impulzy medzi neurónmi. V tele neurónu sú prijaté impulzy následne sčítavané a v prípade prekročenia tzv. vnútorného potenciálu je neurón excitovaný (vybudený). V takom prípade je impulz z tela zaslaný na axón a preposlaný ďalším neurónom. Sila výstupného, resp. vstupného signálu môže byť ešte patrične upravená, podľa synaptickej váhy príslušnej synaptickej väzby. Tá môže byť excitačná alebo inhibičná, tzn. signál môže byť zosilnený alebo zoslabený [20].

Učenie umelej neurónovej siete následne spočíva v nastavovaní jednotlivých synaptických váh. Ako prvý prišiel s týmto konceptom Donald Hebb, ktorý ho v roku 1949 popísal vo svojej knihe *The Organization of Behaviour* [13]. Z tohto poznatku následne vychádzal Frank Rosenblatt [34], ktorý prišiel s vylepšeným modelom pôvodného návrhu neurónu. Tento model nazval Perceptron a jeho hlavné vylepšenie spočívalo v pridaní váh, čo umožňovalo učenie siete [30]. Jeho zobecnenie tvorí základ dnešných neurónových sietí a môžete ho vidieť na obrázku 3.2.



Obr. 3.2: Obecný model umelého neurónu

Neurón obsahuje n rozmerný vstupný vektor $\vec{x} = (x_1, x_2, ..., x_n)$ spolu s n rozmerným váhovým vektorom $\vec{w} = (w_1, w_2, ..., w_n)$, pričom platí, že s každým vstupom x_i je spojená odpovedajúca váha w_i. Jednotlivé vstupy sú následne vynásobené svojimi váhami a sčítané dokopy, tzn. je vypočítaný vnútorný potenciál neurónu u podľa tzv. bázovej funkcie:

$$u = \sum_{i=1}^{n} x_i w_i + b \tag{3.1}$$

Ak hodnota vnútorného potenciálu presiahne prahovú hodnotu, označovanú ako bias, neurón sa následne aktivuje. Neurón môže pre zjednodušenie obsahovať vstup x_0, ktorého hodnota bude 1 a jeho váha bude práh daného neurónu:

$$u = \sum_{i=0}^{n} x_i w_i \tag{3.2}$$

Hodnota výstupu neurónu sa počíta pomocou tzv. aktivačnej funkcie, ktorá je daná na výstup bázovej funkcie. Aktivačná funkcia je nelineárna funkcia a môže mať rôzne podoby. Nelineárna funkcia sa využíva pre lepšiu možnosť učenia siete a na prevod výstupu neurónu do určitého intervalu, typicky od 0 do 1 alebo od -1 do 1.

Kedysi sa využívala najmä tzv. skoková aktivačná funkcia, ktorá môže nadobúdať hodnoty 1 alebo -1, prípadne 1 alebo 0, tzn. výstup je binárny, resp. bipolárny:

$$y = \begin{cases} 1 & \text{pre } u \geq 0 \\ -1 & \text{pre } u < 0 \end{cases} \tag{3.3}$$

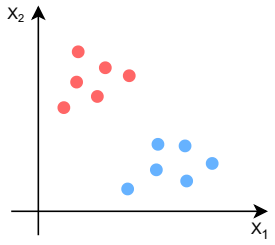
Skokovú aktivačnú funkciu využíval napríklad vyššie spomínaný Pereptron. Nevýhodou tejto funkcie však je, že sa sieť ťažšie učí a malé zmeny váh nemajú na výstup neurónu vplyv. Dnes sa ako aktivačné funkcie používajú najmä spojité funkcie ako Sigmoida (3.4) alebo ReLU (Rectified Linear Unit) (3.5), čo je tzv. usmerňovacia funkcia. V súčasnosti je viac využívaná ReLU funkcia, pretože je jednoduchá na implementáciu a umožňuje rýchle učenie siete.

$$y = \frac{1}{1 + e^{-u}} \tag{3.4}$$

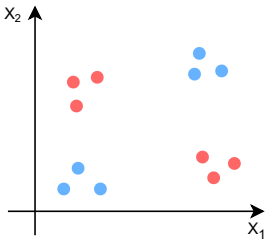
$$y = \begin{cases} u & \text{pre } u \geq 0 \\ 0 & \text{pre } u < 0 \end{cases} \tag{3.5}$$

3.1.1 Trénovanie neurónu

Pri správnom nastavení váh dokáže jeden neurón riešiť lineárne separovateľné problémy, viď obrázok 3.3, avšak nie lineárne neseparovateľné, viď obrázok 3.4. To znamená, že jeden neurón nedokáže vyriešiť ani tak triviálny problém, ako je XOR. Na riešenie takýchto typov problémov je potrebné použiť viacvrstvovú neurónovú sieť.

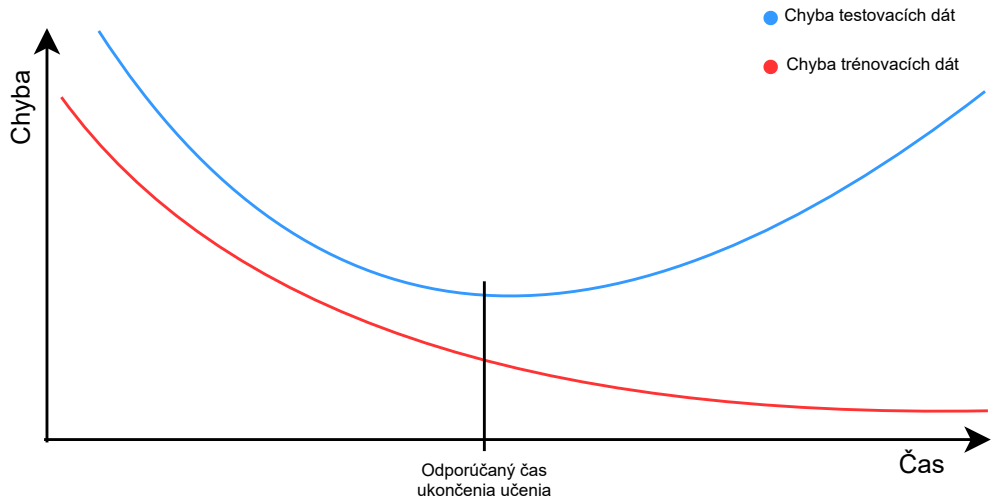


Obr. 3.3: Lineárne separovateľný problém



Obr. 3.4: Lineárne neseparovateľný problém

Aby neurón dával požadované výstupy, je potrebné ho najprv natrénovať. To pozostáva z hľadania správnej kombinácie hodnôt váhového vektoru $\vec{w}$ a hodnoty prahu pre aktiváciu neurónu. Trénovanie neurónu môže byť vykonávané tzv. s učiteľom alebo bez učiteľa. Pri trénovaní s učiteľom sa využívajú tzv. trénovacie dáta, ktoré pozostávajú z dvojice hodnôt - vstupný vektor $\vec{x}$, očakávaný výstup y. Cieľom trénovania je nájsť také hodnoty váh a prahu, pre ktoré bude chyba výstupu čo najmenšia. Súčasne ale nechceme aby došlo k tzv. pretrénovaniu. To znamená, aby bol neurón schopný riešiť aj problémy mimo trénovacej množiny a nebol natrénovaný iba na niekoľko konkrétnych hodnôt. Chceme dosiahnuť, aby bol neurón schopný generalizovať (zobecňovať). Na predídenie pretrénovania existuje niekoľko spôsobov, z ktorých najčastejšie využívaným je včasné ukončenie trénovania. Pri tomto prístupe sa množina trénovacích dát rozdelí na 2 disjunktné podmnožiny - trénovaciu a testovaciu. Trénovacia podmnožina sa použije na trénovanie neurónu a na testovacej sa overuje schopnosť generalizovať naučené vlastnosti. Pri trénovaní sa sleduje miera chybovosti pri testovacích dátach a trénovanie sa ukončí keď sa chybovosť začne zvyšovať, tzn. keď neurón začne byť pretrénovaný [44]. Graf závislosti času trénovania a chybovosti neurónu môžeme vidieť na obrázku 3.5.



Obr. 3.5: Chybovosť neurónu v závislosti od času trénovania

Postup trénovania s učiteľom [36]:

1. Nastavenie $w_0, ..., w_n$ na náhodné, typicky veľmi malé hodnoty
2. Vyberie sa $\vec{x}$ z trénovacej množiny a vypočíta sa y
3. Ak sa výsledok líši od očakávaného, jednotlivé váhy sa nastavia podľa príslušnej adaptačnej metódy, inak sa nerobí nič
4. Kroky 2 a 3 sa opakujú, kým nie je dosiahnutá požadovaná presnosť pre trénovaciu množinu alebo nevyprší čas na trénovanie.

Trénovanie bez učiteľa, označované ako posilňované učenie, sa využíva najmä v problémoch, kde je potrebné sa na základe aktuálneho stavu (situácie) rozhodnúť o ďalšom chovaní systému. Medzi takéto problémy patrí napríklad hranie hier. Pri trénovaní siete

nie je k dispozícií trénovacia množina, ale náhodne inicializovaná neurónová sieť sa nechá pracovať určitý počet cyklov (hier). Sieť sa následne učí z dôsledkov svojich rozhodnutí, tzn. ktoré rozhodnutia boli lepšie, ktoré horšie a ktoré nemali vplyv na výsledok.

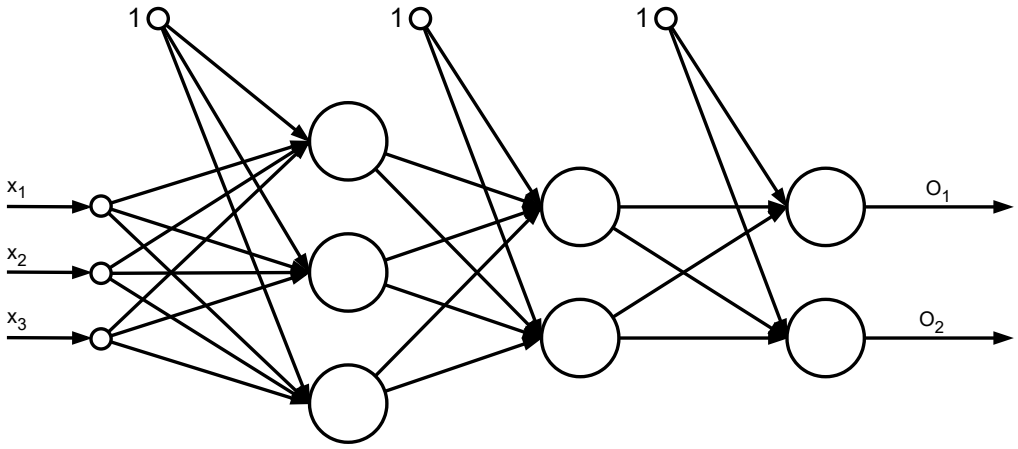
3.2 Viacvrstvová neurónová sieť

Viacvrstvová neurónová sieť je orientovaný graf, kde jednotlivé body grafu sú reprezentované neurónmi siete a cesty v grafe sú väzby medzi neurónmi. Jednotlivé neuróny sú v sieti usporiadané do tzv. vrstiev. Viacvrstvové neurónové siete sa skladajú z troch hlavných častí [37]:

Vstupná vrstva - nie je klasická vrstva neurónovej siete, nakoľko nie je tvorená neurónmi. Hlavným účelom vstupnej vrstvy je prijať vstupné dáta, ktoré budú ďalej spracovávané. Jednotlivé uzly vrstvy sú pasívne, tzn. že nemenia hodnoty dát. Prijaté hodnoty sú iba skopírované na výstupy jednotlivých uzlov.

Skryté vrstvy - sú vnútorné vrstvy siete, v ktorým prebieha výpočet (vyhodnocovanie) riešeného problému. V jednotlivých neurónoch prebieha transformácia dát na základe vstupných hodnôt a váh na jednotlivých prepojeniach. Výstupom každého neurónu je jedna hodnota, ktorá je závislá od použitej aktivačnej funkcie. Neurónové siete môžu mať jednu alebo viac skrytých vrstiev, kde výstup jednotlivých neurónov môže byť vstupom neurónov nasledujúcej skrytej vrstvy alebo výstupnej vrstvy.

Výstupná vrstva - je posledná vrstva siete, ktorá dáva na výstup vypočítané riešenie. Ako vstup prijíma výstupné hodnoty skrytej alebo vstupnej vrstvy. Výstupná hodnota potom závisí od kombinácie výstupov jednotlivých výstupných neurónov. V prípade klasifikačných problémov by mal byť zvyčajne aktívny iba jeden neurón, ktorý reprezentuje správnu triedu pre zadaný vstup.



Obr. 3.6: Dopredná plne prepojená neurónová sieť

Podľa spôsobu prepojenia (architektúry) sa viacvrstvové siete delia na niekoľko typov. V tejto práci sa budeme ďalej venovať dopredným neurónovým sieťam. Je to typ siete, kde existujú väzby iba medzi neurónmi susedných vrstiev. V takejto architektúre neexistuje

rekurzia (cyklus), tzn. že jednotlivé neuróny nemôžu svojim výstupom ovplyvniť neuróny predchádzajúcej alebo rovnakej vrstvy. Príklad takejto siete môžeme vidieť na obrázku 3.6.

Dopredné neurónové siete sa často používajú na problém klasifikácie, tzn. pre zadaný vstup je potrebné určiť do akej triedy patrí. Sieť je následne skonštruovaná tak, že počet neurónov n na výstupnej vrstve je rovný počtu tried do ktorých sieť klasifikuje. To znamená, že každý z výstupných neurónov reprezentuje jednu triedu. Po vyhodnotení vstupu je každému z výstupných neurónov ako výstup priradená hodnota v intervale 0...1, ktorá určuje pravdepodobnosť, že zadaný vstup patrí do triedy reprezentovanej daným neurónom. K tomu je potrebné, aby neuróny vo výstupnej vrstve využívali ako aktivačnú funkciu tzv. Softmax funkciu:

$$y_i = \frac{e^{u_i}}{\sum_{j=1}^{n} e^{u_j}} \tag{3.6}$$

To znamená, že na základe vnútorných potenciálov všetkých n výstupných neurónov funkcia Softmax vypočíta (normalizuje) jednotlivé výstupné hodnoty tak, aby ich súčet bol rovný 1.

3.2.1 Univerzálny aproximačný teorém

Veľkou výhodou dopredných viacvrstvových neurónových sietí je to, že fungujú ako univerzálny aproximátor funkcií. To znamená, že pomocou dostatočne veľkej a dostatočne natrénovanej neurónovej siete je možné aproximovať ľubovoľnú spojitú funkciu $f(\vec{x})$ definovanú v konečnom priestore R^n v inom konečnom priestore R^m s ľubovolne malou, nenulovou chybou [44]. Hlavným dôvodom, prečo sú neurónové siete schopné aproximácie je využitie nelineárneho prvku v ich architektúre. Každý prvok jednotlivých vrstiev je spojený s aktivačnou funkciou, ktorá aplikuje nelineárnu transformáciu na jeho výstup. To znamená, že prvky v jednotlivých vrstvách nepracujú iba s nejakou lineárnou kombináciou výstupov predchádzajúcej vrstvy [4].

Schopnosť neurónových sietí aproximovať ľubovolnú funkciu prvýkrát popísal v roku 1989 George Cybenko [5]. Ten tvrdí, že neurónová sieť obsahujúca aspoň jednu skrytú vrstvu s dostatočným konečným počtom neurónov môže byť pri vhodnej voľbe "squashing" aktivačnej funkcie univerzálnym aproximátorom. "Squashing" aktivačné funkcie sú rastúce aproximačné funkcie, ktoré sú zhora aj zdola obmedzené horizontálnymi asymptotami [44]. V roku 1991 ukázal Kurt Hornik [16], že schopnosť neurónových sietí aproximovať funkcie závisí viac ako od vhodnej aktivačnej funkcie, v jej doprednej viacvrstvovej architektúre.

3.2.2 Učenie neurónovej siete

Učenie (trénovanie) neurónovej siete je proces v ktorom sa snažíme nájsť také hodnoty váh pre jednotlivé neuróny, aby bola chybovosť siete čo najmenšia. Tento proces je výpočetne extrémne náročný. Je preto potrebné využiť taký algoritmus, ktorý proces učenia bude vykonávať čo najefektívnejšie. Najznámejším a najpoužívanejším algoritmom učenia pre dopredné neurónové siete je algoritmus Backpropagation (algoritmus spätného šírenia chyby).

Cieľom tohto algoritmu je minimalizovať tzv. stratovú funkciu, ktorá udáva aktuálnu chybu siete. Pre porovnanie s evolučnými algoritmami, je stratová funkcia ekvivalent fitness funkcie. Algoritmus na minimalizovanie chybovosti využíva metódu gradientného zostupu a je preto potrebné aby mali všetky neuróny diferencovateľné aktivačné funkcie. Ďalej uvedené vzorce platia pri použití sigmoidálnej aktivačnej funkcie a boli prevzaté z materiálov kurzu *Soft computing* na FIT VUT v Brne [44].

Ako stratovú funkciu využíva algoritmus Backpropagation polovicu súčtu kvadratických chýb, ktorá je pre m výstupných neurónov a jeden prvok trénovacej množiny p daná vzťahom 3.7, kde d označuje očakávaný výstup a o skutočný výstup neurónu j vo výstupnej vrstve.

$$E_p = \frac{1}{2}\sum_{j=1}^{m}(d_{pj} - o_{pj})^2 \tag{3.7}$$

Následne sa algoritmus snaží minimalizovať chybu siete E_p metódou gradientného zostupu. Ten upravuje váhový vektor $\vec{w}$ v smere záporného gradientu stratovej funkcie E_p podľa vzťahu 3.8, kde μ označuje tzv. koeficient učenia.

$$\Delta\vec{w} = -\mu\nabla E_p = -\mu\frac{\partial E_p}{\partial\vec{w}} \tag{3.8}$$

Ako vyplýva z názvu algoritmu, výpočet a úprava váh jednotlivých neurónov prebieha od poslednej (výstupnej) vrstvy "dozadu" k vstupnej. Pri výpočte je taktiež potrebné rozlíšiť medzi výstupnou vrstvou a vnútornými vrstvami. Výpočet pre nastavenie jednotlivých váh poslednej vrstvy je daný vzťahom 3.9, kde y_j^L je výstup j-tého neurónu v poslednej vrstve L a δ_j^L je derivácia stratovej funkcie j-tého neurónu v poslednej vrstve.

$$\Delta w_{ji}^L = \mu\delta_j^L x_i^L = \mu(d_j - y_j^L)y_j^L(1 - y_j^L)x_i^L \tag{3.9}$$

Pre zvyšné vrstvy siete sú jednotlivé váhy vypočítané podľa vzťahu 3.10, kde l označuje ľubovolnú (neposlednú) vrstvu siete a n_l počet neurónov v danej vrstve.

$$\Delta w_{ji}^l = \mu\delta_j^l x_i^l = \mu\sum_{k=1}^{n_{l+1}}(\delta_k^{l+1}w_{kj}^{l+1})y_j^l(1 - y_j^l)x_i^l \tag{3.10}$$

Ako už bolo povedané, dopredné neurónové siete sú často využívané na klasifikáciu. Je preto vhodne, aby neuróny poslednej vrstvy využívali funkciu Softmax ako aktivačnú funkciu. Tá určuje pravdepodobnosť náležitosti vstupu do jednotlivých klasifikačných tried, tzn. že v ideálnom prípade je aktivovaný iba jeden výstupný neurón s výstupnou hodnotou rovnou 1. V takomto prípade je vhodné použiť pravdepodobnostnú stratovú funkciu, ktorá je vyjadrená vzťahom 3.11, kde symbol k označuje neurón, reprezentujúci správnu triedu pre zadaný vstup [44].

$$E = -ln(y_k^L) \tag{3.11}$$

Hodnota δ_j^L sa následne vypočíta podľa vzťahu 3.12 a jednotlivé váhy sú upravené podľa vzťahu 3.13.

$$\delta_j^L = \begin{cases} 1 - y_j^L & \text{pre } j = k \\ -y_j^L & \text{pre } j \neq k \end{cases} \tag{3.12}$$

$$\Delta w_{ji}^L = \mu \delta_j^L x_i^L \tag{3.13}$$

Samotné trénovanie siete je možné vykonávať niekoľkými spôsobmi:

- Prvky p sú z trénovacej množiny vyberané náhodne a váhy sú upravené po spracovaní každého prvku
- Váhy sú upravené po spracovaní náhodnej podmnožiny trénovacej množiny o vopred danej veľkosti
- Chyba siete sa počíta pre celú trénovaciu množinu a váhy sú upravené po jej spracovaní

Na rýchlosť konvergencie algoritmu a výslednú kvalitu natrénovania má taktiež veľký vplyv koeficient učenia μ. Jeho hodnota predstavuje veľkosť kroku v smere záporného gradientu a jeho voľba má veľký vplyv na konvergenciu siete. Malá hodnota môže spôsobiť pomalé učenie a príliš veľká hodnota môže mať za následok preskočenie optima a divergenciu riešenia. Je preto potrebné voliť tento parameter opatrne, typicky sa udáva malá hodnota v intervale od 0 do 1. Existujú taktiež rôzne adaptívne metódy, ktoré menia parameter μ počas výpočtu. Medzi najznámejšie patrí napríklad RPROP [33], AdaGrad [9], Adadelta [47] alebo AdaMax [18].

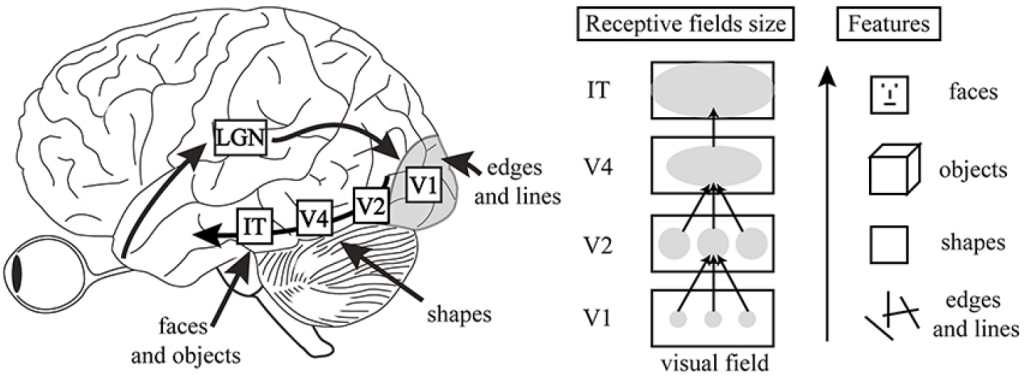
3.3 Konvolučné neurónové siete

Medzi dnes najpoužívanejší typ siete patria tzv. hlboké neurónové siete (Deep Neural Networks). Sú to neurónové siete so stovkami až tisícmi skrytých vrstiev a v súčasnosti patria medzi najúspešnejšiu metódu strojového učenia. Hlboké neurónové siete čerpajú hlavnú inšpiráciu v "stavbe" ľudského mozgu a v schopnosti učiť sa.

Klasické modely na rozpoznávanie a klasifikáciu požadujú aby bola z dát vyextrahovaná tzv. množina príznakov (features). Táto množina je analyzovaná a získaná nejakým nezávislým spôsobom a je potrebná ako prerekvizita na ďalšie spracovanie analyzovaných dát. Extrakcia vhodných príznakov z dát je obvykle komplikovaný proces a nazýva sa *feature engineering*. Výhodou hlbokých neurónových sietí je, že túto množinu sú schopné extrahovať sami. Je to preto, že sú schopné sa naučiť, ktoré informácie zo spracovávaných dát sú dôležité pre ďalšiu analýzu. Táto vlastnosť je však veľmi výpočtovo náročná.

Architektúra a spôsob spracovania dát hlbokými neurónovými sieťami je založená na stavbe ľudského mozgu, ako je zobrazené na obrázku 3.7.

Ako môžeme vidieť, jednotlivé vrstvy siete tvoria určitú hierarchiu, postupnosť neurónov, pričom sa jednotlivé časti špecializujú na extrakciu rôznych príznakov. Pri spracovávaní



Obr. 3.7: Spracovanie obrazu mozgom, prevzaté z [14]

obrazu sú z dát najprv získané elementárne rysy. To je umožnené neurónmi v najspodnejších vrstvách (V1), ktoré sú citlivé na základné vizuálne znaky, ako sú priamky alebo hrany. Z tých sú následne vo vyšších vrstvách (V2, V4) zostavené zložitejšie konštrukcie a tvary až je mozog v najvyššej vrstve (IT) schopný rozpoznať obraz alebo vyhodnotiť situáciu a reagovať na ňu.

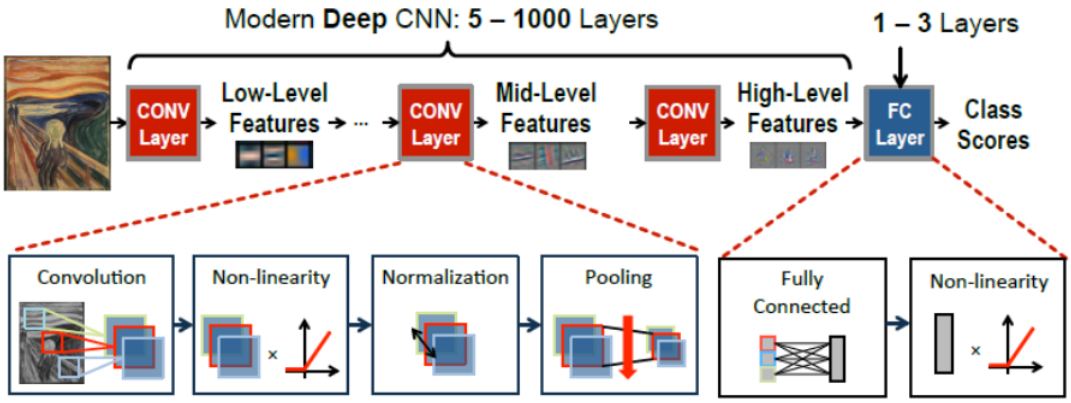
Neuróny sú usporiadané do hierarchie doprednej siete, pričom výstupy jednotlivých vrstiev sú tvorené extrahovanými príznakmi, ktoré slúžia ako vstupy vyšších vrstiev na ďalšiu analýzu. Taktiež platí, že informácie, ktoré sa budú ďalej spracovávať a analyzovať sú plne určené spracovaním v predchádzajúcej fáze, a informácie ktoré neboli zaslané vyšším vrstvám sú nenávratne stratené [14].

V tejto práci sa budeme ďalej venovať tzv. konvolučnými neurónovými sieťami, čo sú hlboké neurónové siete zamerané hlavne na rozpoznávanie a klasifikáciu obrazových dát. Obsahujú tzv. konvolučné vrstvy, ktoré využívajú operáciu konvolúcie na predspracovanie vstupu. Vstupné obrázky sú buď jednokanálové (čiernobiele) alebo trojkanálové (RGB) dáta.

3.3.1 Architektúra CNN

Konvolučné siete sú dopredné neurónové siete, ktoré sa skladajú zo štyroch základných typov vrstiev – *konvolučná vrstva, aktivačná vrstva, zoskupujúca vrstva, plne prepojená vrstva*. Schému architektúry konvolučnej neurónovej siete je možné vidieť na obrázku 3.8.

Rozdiel konvolučných neurónových sietí oproti klasickým dopredným sieťam je v tom, že neuróny sú na konvolučných vrstvách usporiadané do 2D štruktúr nazývaných *mapa príznakov (feature map)*. Tie sú výsledkom operácie konvolúcie nižšej vrstvy. Počet štruktúr v jednotlivých vrstvách závisí od počtu vstupných kanálov. Taktiež to určuje, že hodnota konkrétneho neurónu môže byť daná iba neurónom na rovnakom mieste v štruktúre nižšej vrstvy. To umožňuje predspracovanie vstupu a extrakciu príznakov, potrebných na ďalšiu analýzu doprednou plne prepojenou vrstvou. Dáta sú na jednotlivých vrstvách spracovávané konvolučnými filtrami, ktoré slúžia na extrakciu jednotlivých príznakov z dát. Počet použitých filtrov závisí od počtu hľadaných príznakov a od počtu vstupných kanálov. Využitie



Obr. 3.8: Architektúra konvolučnej neurónovej siete, prevzaté z [43]

operácie konvolúcie umožňuje značne zredukovať počet trénovacích parametrov siete oproti klasickým plne prepojeným dopredným sieťam, nakoľko je potrebné trénovať iba hodnoty jednotlivých filtrov a hodnotu prahu (bias). Napríklad pre RGB vstup o rozmeroch 30x30 by sme pri použití plne prepojenej doprednej siete mali 2700 trénovacích parametrov. Avšak pri použití filtra o veľkosti 6x6 je pre 3-kanálový vstup potrebné trénovať iba 108 parametrov. To má za následok zlepšenie generalizácie a umožňuje predísť pretrénovaniu siete [17].

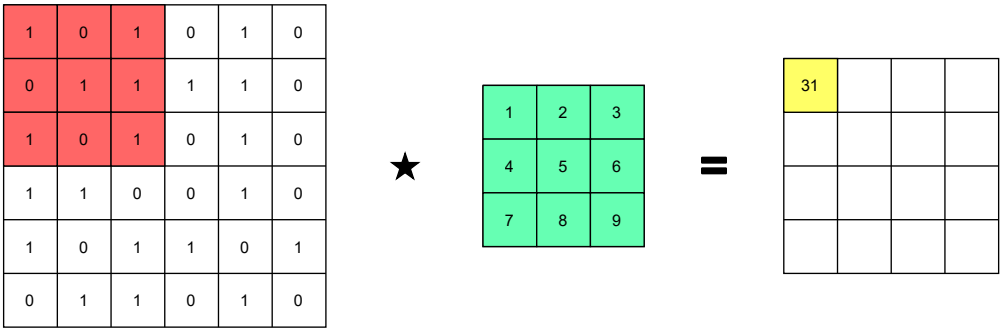
Konvolučná vrstva

Ako už názov siete napovedá, konvolučná vrstva tvorí hlavnú súčasť konvolučných neurónových sietí. Vo vrstve sa využíva matematická operácia konvolúcie, ktorá umožňuje nájsť a extrahovať dôležité príznaky vo vstupných obrázkoch. Počet parametrov, ktoré je potrebné vo vrstve trénovať, závisí od rozmerov a počtu použitých konvolučných filtrov. Počet použitých filtrov závisí od počtu hľadaných príznakov vo vstupe, jednotlivé filtre môžu napríklad slúžiť na detekciu rôzne otočených hrán a od počtu vstupných kanálov, tzn. či je vstup 1 kanálový (čiernobiely) alebo 3-kanálový (RGB). Následne je počet výstupných obrazov závislí od počtu použitých konvolučných filtrov.

Základná myšlienka konvolúcie je aplikácia konvolučných filtrov na celú vstupnú 2D štruktúru (mapu príznakov), čím sa vypočítajú nové hodnoty pre jednotlivé pixely. Operácia konvolúcie je daná vzťahom 3.14, ktorý pre jednotlivé pixely vstupného obrázku počíta jeho novú hodnotu na základe jeho starej hodnoty a hodnôt okolitých pixelov. Vzorec bol prevzatý z [46]. Operácia konvolúcie sa počíta pre všetky mapy príznakov, ktoré sú na vstupe vrstvy a je na ne aplikovaná celá sada filtrov pre danú vrstvu.

$$I(u,v) = \sum_{i=-a}^{b} \sum_{j=-c}^{d} h(i,j) I(u+i, v+j) \tag{3.14}$$

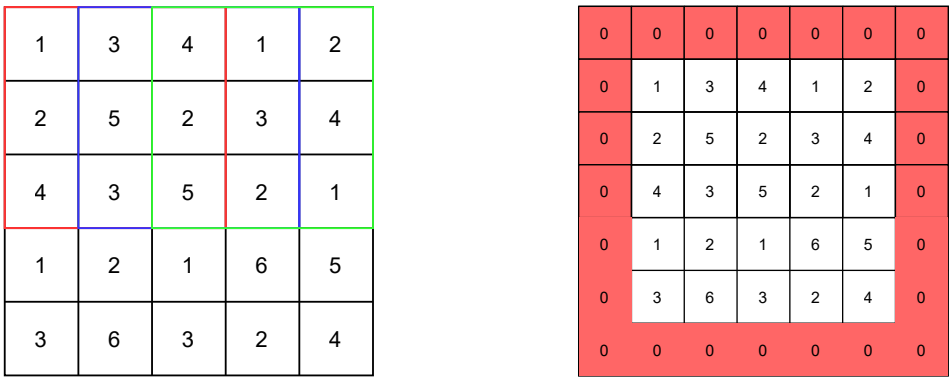
Príklad aplikácie konvolučného filtra na vstupné dáta môžeme vidieť na obrázku 3.9.



Obr. 3.9: Aplikácia operácie konvolúcie na vstupné dáta

Pri konvolúcií pracujeme taktiež s dvomi parametrami, ktorými sme schopní ovplyvniť rozmery výstupu a tým aj kvalitu následnej detekcie príznakov. Týmito parametrami sú krok S (Stride) a výplň P (Padding).

Hodnota parametru S určuje počet pixelov o koľko sa bude posúvať filter pri spracovaní vstupného obrázku. Jeden krok označuje posun filtra o n pixelov v jednom smere. Ak by sme nastavili krok na 1, mali by sme vysokú mieru prekryvov a tým aj vyššiu mieru vybudenia neurónov a s tým spojenú lepšiu extrakciu potrebných príznakov. Pri vyššej hodnote kroku by sme naopak redukovali rozmery výstupu a znížili počet prekrytí, čím by sme redukovali pamäťovú a časovú náročnosť výpočtu. Výplň P určuje mieru vyplnenia okrajov hodnotami 0, čím umožní lepšie spracovať okrajové hodnoty vstupného obrázku [28]. Ukážka parametrov S a P sú na obrázku 3.10a a 3.10b.



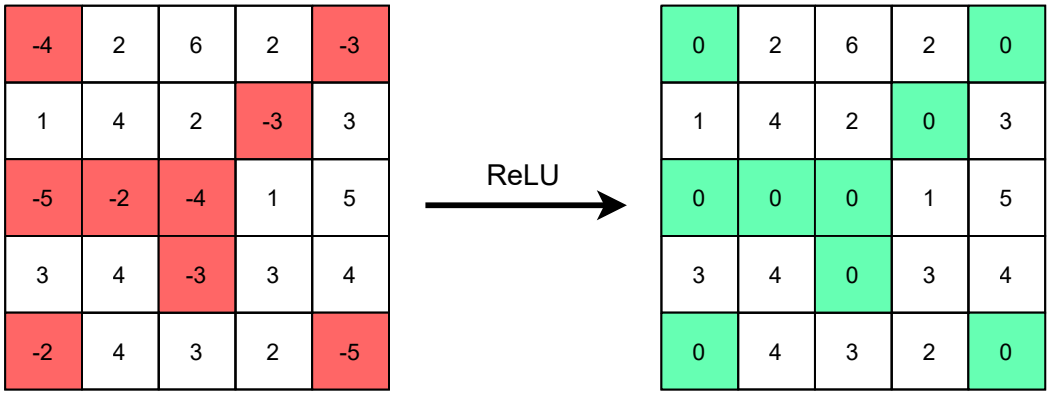
(a) Krok S (Stride) = 1

(b) Výplň P (Padding) = 1

Obr. 3.10: Ukážka parametrov S a P

Aktivačná vrstva

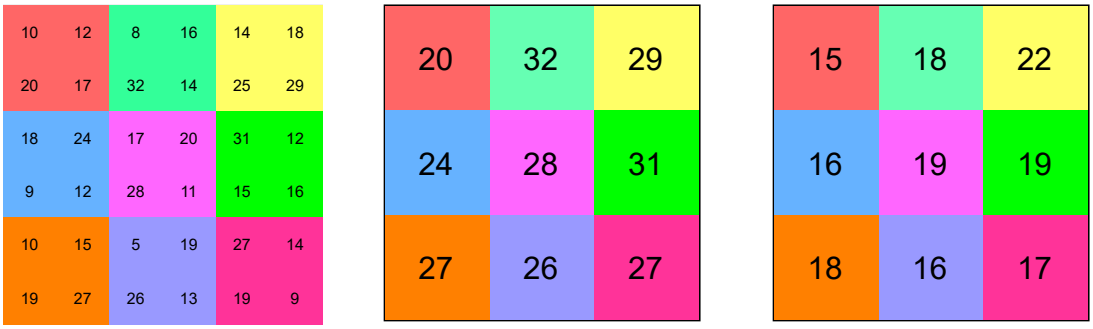
Aktivačná vrstva slúži, rovnako ako pri klasických neurónových sieťach, na výpočet výstupnej hodnoty neurónov na základe ich vnútorných potenciálov. Výpočet je vykonávaný nejakou nelineárnou funkciou. Dnes sa v konvolučných neurónových sieťach najčastejšie používa ReLU aktivačná funkcia. Pre túto funkciu je výstupná hodnota neurónu určená vzťahom 3.5. Príklad výstupu pri použití aktivačnej funkcie ReLU je zobrazený na obrázku 3.11.



Obr. 3.11: Výstup konvolučnej vrstvy pri použití aktivačnej vrstvy ReLU

Zoskupujúca vrstva

Zoskupujúca vrstva slúži na redukciu rozmerov vstupných dát. Obvykle sa používa metóda *max pooling*, čo je výber najvyššej hodnoty zo zadanej oblasti alebo *average pooling*, ktorý udáva výslednú hodnotu ako priemer hodnôt zo zadanej oblasti. Ukážka jednotlivých operácií je zobrazená na obrázku 3.12.



(a) Vstupné dáta (b) Max pooling (c) Average pooling

Obr. 3.12: Redukcia rozmerov vstupných dát

Obdobne ako pri konvolúcií sú parametrami operácie rozmery filtra a veľkosť kroku s tým, že veľkosť kroku sa obvykle rovná rozmerom filtra, aby sa jednotlivé oblasti pri spracovaní neprekrývali. Často sa používajú rozmery filtra 2x2 s posunutím o 2 pixely. Výsledný obraz je potom zredukovaný na polovicu svojej pôvodnej veľkosti, tzn. z n x n na $n/2$ x $n/2$. Výhodou použitia zoskupujúcej vrstvy je, že pri redukcií dát sa zachovávajú dôležité informácie o vstupe a odstraňujú sa nepotrebné, čo má značný vplyv aj na výkon a efektivitu výpočtu.

Plne prepojená vrstva

Plne prepojená vrstva je klasická dopredná neurónová sieť s niekoľkými skrytými vrstvami a používa sa ako posledná časť v konvolučných neurónových sieťach. Ako vstup dostane vektor vyextrahovaných príznakov z predchádzajúcich konvolučných vrstiev. Za týmto účelom

sa pred plne prepojenú vrstvu dáva tzv. zplošťovacia vrstva, ktorá slúži na prevod 2D štruktúry vstupu na 1D vektor, ktorý sa môže ďalej spracovávať doprednou neurónovou sieťou. Počet neurónov na výstupnej vrstve je rovný počtu tried do ktorých sieť klasifikuje s tým, že neuróny používajú ako aktivačnú funkciu funkciu Softmax, ktorá je daná vzťahom 3.6.

Kapitola 4

Neuroevolúcia

Hlboké a konvolučné neurónové siete sú stále viac a viac populárne, a to najmä vďaka ich schopnosti automatickej extrakcie dôležitých príznakov zo surových (raw) dát. Avšak aby tieto siete dávali kvalitné výsledky, je potrebný dobrý návrh ich architektúry a následne danú sieť dostatočne natrénovať. Väčšina dnes využívaných hlbokých neurónových sietí bola navrhnutá "ručne" človekom [41]. Takýto spôsob návrhu neurónových sietí je však veľmi komplikovaný a časovo náročný. Výsledné siete taktiež častokrát obsahujú stovky vrstiev a milióny parametrov, ktoré je potrebné učiť. Preto sa dnes stále viac začína využívať automatizovaný návrh neurónových sietí - *NAS (Neural Architecture Search)*, ktorý umožňuje celý proces návrhu a trénovania značne zrýchliť. Neurónové siete navrhnuté pomocou NAS metód majú taktiež typicky oveľa menej učiacich parametrov, v porovnaní s "ručne" navrhnutými sieťami.

Medzi prístupy automatického návrhu hlbokých neurónových sietí patrí aj oblasť nazývaná *neuroevolúcia*. Tento prístup čerpá inšpiráciu v biologickom procese evolúcie a na návrh neurónových sietí využíva evolučné algoritmy. Počiatky neuroevolúcie siahajú do 80. rokov minulého storočia, kedy sa začal rozširovať výskum v oblasti neurónových sietí a evolučných algoritmov. Na začiatku bola neuroevolúcia zameraná najmä na evolúciu váh vo fixne danej topológii, čo malo slúžiť ako alternatíva k algoritmu backpropagation. Čoskoro sa však začalo uvažovať aj o evolúcií celej topológie siete, čo bolo inšpirované skutočnosťou, že biologická evolúcia vytvorila ľudský mozog, ktorý tvorí najkomplexnejšiu neurónovú sieť, akú dnes poznáme [39].

Prvý väčší úspech v oblasti neuroevolúcie bol v roku 2002 s vytvorením algoritmu *NEAT (Neuroevolution of Augmenting Topologies)* [40]. Ten bol však konkurencieschopný iba pri menších topológiach. Postupom času bol však vytvorený algoritmus AmoebaNet [31], ktorý prehľadával rovnaký priestor ako algoritmus NASNet [48], ale s rýchlejšou konvergenciou k optimálnej sieti v porovnaní s metódami posilňovaného učenia alebo náhodného prehľadávania. Taktiež bol schopný konkurovať iným NAS metódam aj pri väčších topológiach a dával veľmi dobré výsledky na testovacej sade CIFAR-10 [24]. To ukázalo, že výskum v oblasti neuroevolúcie má veľký potenciál a dáva konkurencieschopné výstupy pri automatickom návrhu umelých neurónových sietí.

NAS metódy sú typicky *dvojúrovňové* optimalizačné problémy, kde sa "vnútorná" optimalizácia zameriava na nájdenie čo najlepších hodnôt váh danej siete, zatiaľ čo sa "vonkaj-

šia" optimalizácia zameriava na nájdenie čo najlepšej topológie siete pre zadaný problém a parametre hľadania [23]. V súčasnosti sa výzkum v oblasti neuroevolúcie zameriava viac na automatizovaný návrh topológii, ako na hľadanie váh siete pomocou evolúcie. Tento prístup je najmä preto, že bežne používaný algoritmus backpropagation v kombinácií s metódou gradientného zostupu dáva porovnateľné alebo lepšie výsledky ako evolučný návrh váh. Avšak "ručne" navrhované siete sú bežne omnoho väčšie ako evolučne navrhnuté a od toho sa následne odvíja aj počet trénovacích parametrov a potrebný čas na učenie siete.

NAS metódy nemusia byť použité iba na návrh celej topológie siete, ale môžu byť využité iba na úpravu už existujúcej. Tento prístup sa nazýva *prenosové učenie (transfer learning)* a je využívané najmä v prípadoch kedy máme natrénovanú sieť a potrebujeme ju "vyladiť" na iný typ dát alebo nemáme dostatok výpočtového času na automatizovaný návrh novej siete [41].

4.1 Evolúcia topológie

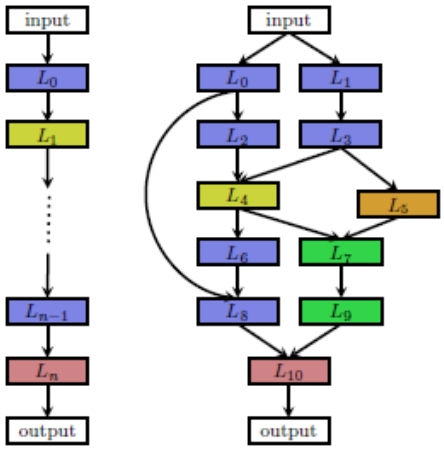
Návrh siete človekom je veľmi náročný a zdĺhavý proces, ktorý môže trvať mesiace alebo roky a je založený prevažne na ľudských skúsenostiach [24]. Využitím NAS techník je možné nájsť topológiu siete pre daný problém v priebehu niekoľkých dní. Avšak aj tento prístup má svoje nevýhody. Medzi tie hlavné patrí to, že automatizovaný návrh siete je výpočtovo veľmi náročný, avšak je to logicky ďalší krok v automatizovanom strojovom učení a už teraz sú evolučne navrhnuté siete konkurencieschopné manuálne navrhnutým [10].

Pri jednotlivých NAS metódach sa sledujú tri hlavné oblasti [10]:

- **Prehľadávaný priestor** vo všeobecnosti určuje aké architektúry môžu byť navrhované daným algoritmom. V závislosti od informácií o danom probléme je možné značne zredukovať prehľadávaný priestor a lepšie usmerniť hľadanie optimálnej architektúry. Napríklad keď vieme, že sieť má riešiť klasifikačný problém, tak môžeme fixne určiť, že neuróny v poslednej vrstve budú využívať aktivačnú funkciu Softmax.

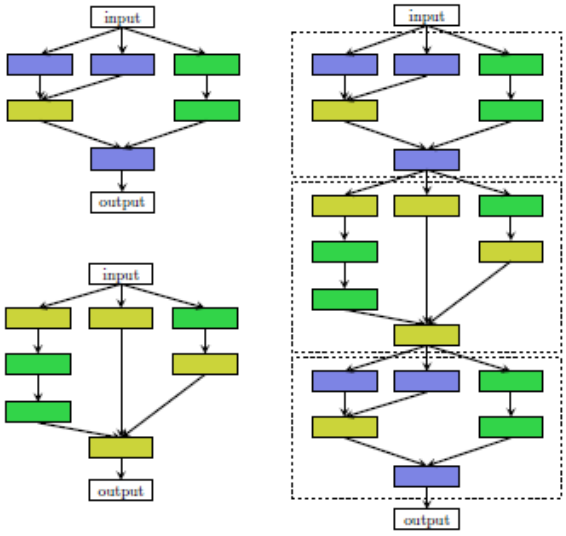
  Taktiež je možné upraviť veľkosť prehľadávaného priestoru na základe spôsobu, akým môžu byť jednotlivé vrstvy siete prepojené. Typicky sa využívajú dva prístupy. Jednoduchší spôsob je využiť zreťazenú postupnosť vrstiev, kedy výstup vrstvy $n-1$ je vstupom vrstvy n. Avšak tento prístup obmedzuje prehľadávaný priestor a kandidátne riešenia sa ťažšie optimalizujú. Druhý spôsob je umožniť vytvorenie rozvetvenej architektúry, v ktorej je umožnené aby sa jednotlivé vrstvy preskakovali alebo vetvili. Typicky takéto architektúry obsahujú na konci nejakú agregačnú vrstvu (*sumarizačnú/konkatenačnú*), ktorá spojí jednotlivé výstupy predošlých vrstiev do jednej. Tento prístup značne rozširuje prehľadávaný priestor a umožňuje vytvoriť širokú škálu kandidátnych riešení. Porovnanie oboch prístupov môžeme vidieť na obrázku 4.1.

  Existujú však aj prístupy, ktoré kombinujú zreťazený návrh s vetvením. V takýchto prípadoch sa sieť skladá zo zreťazenej postupnosti blokov, kde každý blok obsahuje určitú rozvetvenú architektúru vytvorenú z jednotlivých vrstiev. Samotné bloky majú vždy jeden vstup a výstup, kde platí, že výstup $n-1$ bloku je vstupom n bloku. Architektúry v jednotlivých blokoch môžu byť navrhované dynamicky počas výpočtu pre každý blok alebo môže byť preddefinovaná určitá množina architektúr, ktorá je



Obr. 4.1: Porovnanie zreťazenej *(vľavo)* a rozvetvenej *(vpravo)* architektúry, prevzaté z [10]

následne zreťazená počas návrhu samotnej siete. Ukážku návrhu architektúry kombinujúceho zreťazený aj vetvený prístup môžeme vidieť na obrázku 4.2.



Obr. 4.2: Návrh architektúry kombináciou zreťazeného a vetveného prístupu, prevzaté z [10]

- **Stratégia prehľadávania** popisuje spôsob, akým sa bude prehľadávať prehľadávaný priestor. Zahŕňa vyváženie medzi exploráciou a exploitáciou, nakoľko je na jednej strane žiadané nájsť čo najlepšiu architektúru čo možno najrýchlejšie, avšak taktiež sa chceme vyhnúť uviaznutiu v lokálnych extrémoch.
- **Stratégia odhadu výkonu** je spôsob ohodnotenia jednotlivých kandidátnych riešení. Nakoľko je hlavným cieľom NAS algoritmov nájsť čo najefektívnejšiu sieť, je potrebné zistiť kvalitu jednotlivých kandidátnych riešení. Najjednoduchší spôsob je natrénovať a odtestovať jednotlivé architektúry, avšak tento spôsob je výpočetne veľmi náročný a značne obmedzuje počet kandidátnych riešení, ktoré môžu byť preskúmané. V poslednej dobe sa výzkum v oblasti NAS algoritmov sústredí na nájdenie metód,

ktoré by umožňovali znížiť výpočtovú cenu odhadu výkonu nájdených architektúr a tým umožnili zefektívniť automatizáciu hľadania optimálnych neurónových sietí pre zadané problémy.

Pri evolučnom návrhu topológie je potrebné zakódovať predpis kandidátnej siete do chromozómu algoritmu. Na kódovanie je možné použiť dva prístupy - priame a nepriame. Priame kódovanie je prístup kedy chromozómy obsahujú priamo počet a prepojenia medzi jednotlivými neurónmi. Tento prístup však nie je dobre škálovateľný. Nepriame kódovanie na druhú stranu obsahuje iba "predpis" na vytvorenie daného kandidátneho riešenia. Tento prístup umožňuje zredukovať dĺžky chromozómov a rozumne tak kódovať aj veľké topológie.

Aj keď tradičné prístupy na automatizovanú optimalizáciu počtu neurónov a prepojení v neurónových sieťach pracovali na nižšej úrovni abstrakcie, tzn. na úrovni neurónov, dnešné hlboké neurónové siete pozostávajú z rôznych typov neurónov a vrstiev, ako napr. konvolučné, zoskupujúce a pod. Optimalizácia takého počtu parametrov v rozumnom čase je veľmi náročná a preto sa pri návrhu topológii pracuje na úrovni celých vrstiev siete. To znamená, že sa hľadá ideálna kombinácia a prepojenie rôznych typov vrstiev, aby výsledná sieť bola čo najefektívnejšia [42].

4.2 Evolúcia váh

V súčasnosti sa vedci v oblasti výskumu hlbokých neurónových sietí zameriavajú prevažne na *hlboké* a *hlboké posilňované učenie*. V tejto oblasti je dominantným prístupom algoritmus backpropagation s využitím metódy gradientného zostupu. Ten má však tiež určité obmedzenia. Nie je ho možné napríklad použiť na učenie siete, ktorej neuróny nevyužívajú diferencovateľné aktivačné funkcie. Taktiež nie je schopný sa dostať z lokálneho extrému účelovej funkcie. Alternatívny prístup je hľadanie optimálnych váh siete pomocou evolučných algoritmov, nakoľko nájdenie ideálnej kombinácie váh je optimalizačný problém. Výhodou tohoto prístupu je, že pracuje s množinou kandidátnych riešení a výpočet (trénovanie) siete je možné efektívne paralelizovať [39].

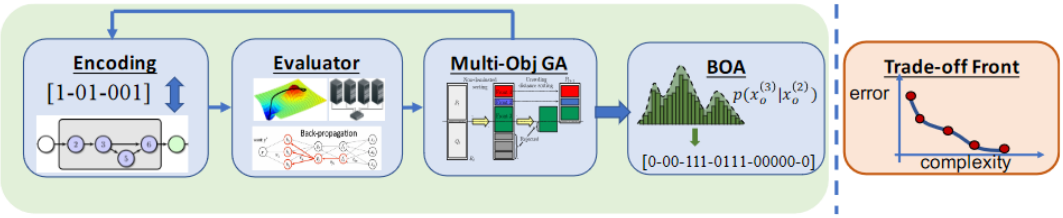
Pri evolučnom návrhu váh siete sa postupuje obdobne ako pri klasických evolučných algoritmoch. To znamená, že sa na začiatku vytvorí populácia kandidátnych jedincov, kde každý reprezentuje váhový vektor, pre jednotlivé neuróny v sieti. Následne je každý jedinec ohodnotený na základe kvality výstupu siete s jeho váhovými hodnotami. Jedinci sú potom na základe svojich fitness hodnôt vyberaní ako rodičia na vytvorenie novej populácie, kým nie je nájdený najlepší váhový vektor alebo nie je dosiahnutý predom daný počet iterácií.

4.3 Využitie GA v neuroevolúcii

Mnohé NAS metódy založené na metóde gradientného zostupu, ako napríklad DARTS [22], sa zameriavajú výhradne na minimalizáciu chyby siete, avšak nie je jednoduché ich prispôsobiť na hľadanie kandidátnych riešení na základe viacerých konfliktných kritérií. Existujú taktiež metódy, ktoré využívajú iba jeden výpočtový blok a opakujú ho ľubovolný počet krát na vytvorenie celej siete. Jednou z hlavných požiadaviek na neurónové siete

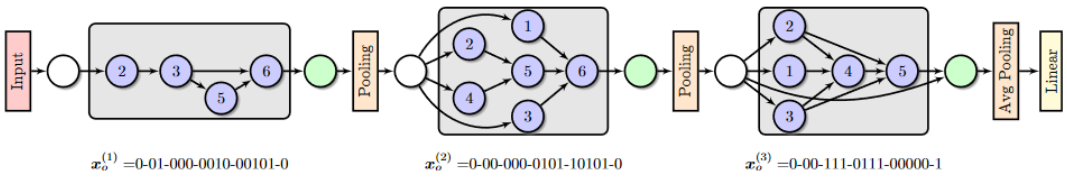
navrhnuté pomocou NAS metód je, aby boli čo najmenšie a pracovali čo najrýchlejšie. To má však za cenu určité zníženie presnosti. Snažíme sa preto nájsť kompromis medzi presnosťou a veľkosťou siete, aby sme dostávali dostatočne presné výsledky, ale taktiež aby s nimi bolo možné pracovať aj na bežných počítačoch s obmedzenými výpočtovými zdrojmi, ako sú energetická spotreba, potrebná operačná pamäť alebo MAC (Multiply-Accumulate) operácie. O riešenie podobných obmedzení sa snaží algoritmus NSGA-Net, uvedený v práci [24]. Tento algoritmus vychádza z multikriteriálneho genetického algoritmu NSGA-II a je prispôsobený na evolučný návrh architektúr umelých neurónových sietí pri optimalizácií chyby siete vs. počet FLOPs operácií v jednej inferencii (jednom prechode sieťou). Výsledkom je Pareto množina vzájomne nedominujúcich riešení.

Postup výpočtu je obdobný ako pri algoritme NSGA-II, ktorého popis je uvedený v kapitole 2.3. Na začiatku je vytvorená množina kandidátnych riešení, ktorá vytvorí populáciu potomkov a je vytvorená pomocná populácia obsahujúca starých aj nových jedincov. Tí sú postupne ohodnotení a priradení do jednotlivých Pareto množín, podľa dominancie s inými jedincami. N najlepších jedincov je vždy vybraných, aby tvorili novú populáciu, až kým nie je dosiahnutý vopred daný počet generácií. Potom nasleduje fáza exploitácie, kde je využitá Bayesovská optimalizácia [29], s cieľom prehľadať okolie nájdených riešení a optimalizovať ich. Po skončení výpočtu je vrátená najlepšia Pareto množina. Proces výpočtu môžeme vidieť na obrázku 4.3.



Obr. 4.3: Postup evolučného návrhu neurónovej siete v NSGA-Net, prevzaté z [24]

Počas výpočtu sú jednotlivé kandidátne riešenia kódované ako postupnosť blokov, kde každý blok môže obsahovať niekoľko uzlov. Každý uzol v bloku reprezentuje určitý typ vrstvy, pričom platí, že každý uzol môže byť v bloku použitý práve raz. Za každým blokom nasleduje pevne daná zoskupovacia vrstva a celá sieť je ukončená plne prepojenou doprednou vrstvou, ktorá slúži na klasifikáciu vyextrahovaných príznakov. Jedinci sú kódovaní ako postupnosť binárnych postupností, ktoré určujú prepojenie uzlov v jednotlivých blokoch. Ukážku kódovania jedinca môžeme vidieť na obrázku 4.4.



Obr. 4.4: Kódovanie kandidátnych neurónových sietí v NSGA-Net, prevzaté z [24]

Kapitola 5

Návrh riešenia

Cieľom tejto kapitoly je popísať návrh implementácie programu na automatický návrh konvolučných neurónových sietí pomocou genetických algoritmov. Nakoľko vytvorené neurónové siete sú hodnotené na základe viacerých protichodných kritérií, ako evolučný algoritmus bola zvolená multikriteriálna varianta genetického algoritmu - NSGA-II, ktorého popis je uvedený v kapitole 2.3. Na prácu s neurónovými sieťami je použitá knižnica TensorFlow[1], ktorá umožňuje jednoduchú prácu s neurónovými sieťami a taktiež umožňuje akcelerovať výpočet na GPU.

Spôsob kódovania neurónových sietí a ich evolučný návrh bol inšpirovaný článkom *NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm* [24].

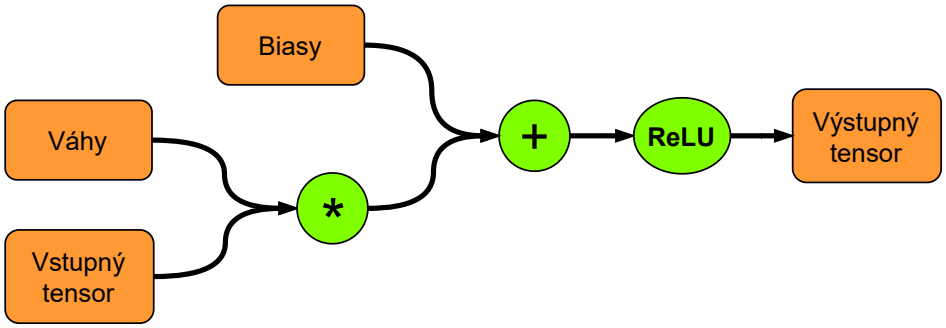
5.1 TensorFlow

TensorFlow je open-source knižnica vyvinutá spoločnosťou Google, ktorá sa využíva prevažne na prácu s umelými neurónovými sieťami. Umožňuje vytvoriť grafovú reprezentáciu neurónovej siete, ktorá abstrahuje jej funkcionalitu. Jednotlivé uzly grafu následne predstavujú rôzne matematické operácie, ktoré sa majú vykonať. Spojenia medzi uzlami reprezentujú n-rozmerné polia hodnôt (vstupy/výstupy), nazývané tiež *tensory*. Samotnú prácu s knižnicou je možné vykonávať v jazyku Python, avšak jednotlivé matematické výpočty sú napísané v jazyku C++, čo značne zvyšuje výpočtový výkon.

Vďaka abstrakcií na úroveň orientového grafu je možné pomocou knižnice TensorFlow vytvoriť množinu blokov (vrstiev), z ktorých budú jednotlivé kandidátne siete zostavené. Ukážku implementácie plne prepojenej vrstvy môžeme vidieť na obrázku 5.1.

TensorFlow taktiež umožňuje trénovanie sietí na dátovej sade *CIFAR-10*, ktorá sa bežne používa na trénovanie a testovanie konvolučných neurónových sietí. Táto sada obsahuje 60000 farebných obrazových dát (50000 trénovacích, 10000 testovacích) o rozmeroch 32x32 pixelov, pričom jednotlivé obrázky je možné klasifikovať do 10 rozličných tried. Samotné trénovanie siete je následne vykonávané pomocou metódy backpropagation.

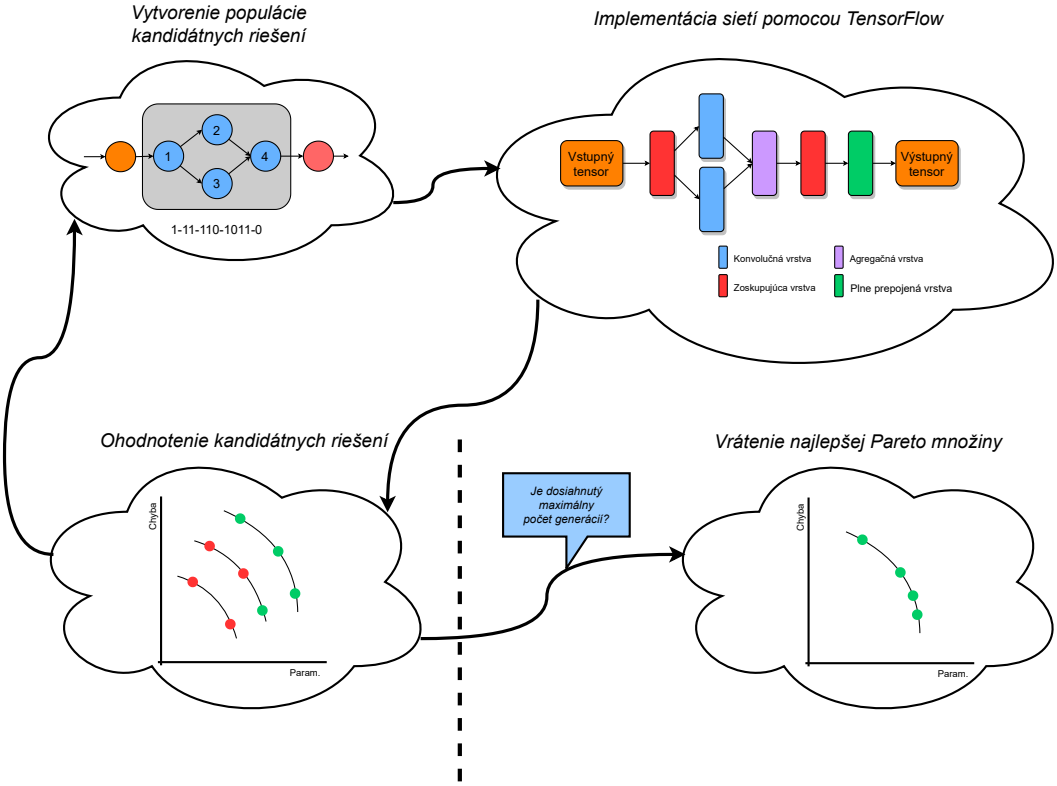
[1] https://www.tensorflow.org/



Obr. 5.1: Implementácia plne prepojenej vrstvy pomocou knižnice TensorFlow

5.2 Návrh CNN pomocou GA

Ako už bolo povedané, nakoľko hľadáme optimálnu sieť na základe viacerých protichodných kritérií, ako evolučný algoritmus je použitý NSGA-II, ktorý po skončení výpočtu vráti tzv. Pareto množinu, ktorá predstavuje množinu vzájomne nedominujúcich riešení. Postup výpočtu je možné vidieť na obrázku 5.2.



Obr. 5.2: Postup evolučného návrhu neurónovej siete

Na začiatku výpočtu je vytvorená množina kandidátnych neurónových sietí, ktoré sú následne implementované pomocou knižnice TensorFlow. Siete sú tvorené rôznymi kombináciami rôznych typov vrstiev, ktorých implementácia je vytvorená v knižnici TensorFlow. Vytvorené siete je potrebné natrénovať a otestovať. Trénovanie aj testovania je vykonávané

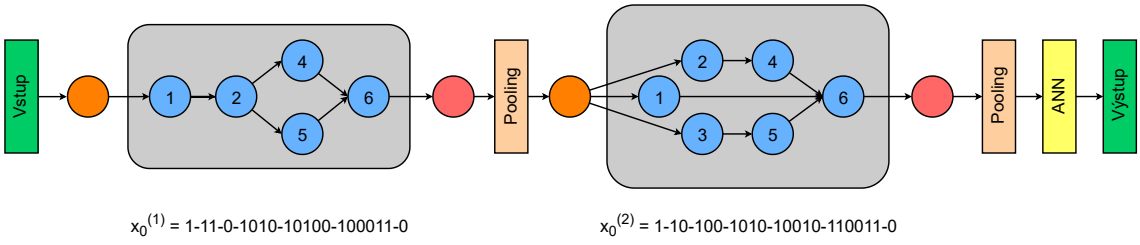
na klasifikačnom probléme CIFAR-10, pričom pri trénovaní je využívaný algoritmus Backpropagation. Pri ohodnotení sietí sa vytvárajú Pareto množiny z kandidátnych riešení na základe chybovosti siete a počtu trénovacích parametrov. Trénovanie a testovanie jednotlivých sietí je výpočtovo najnáročnejšia časť algoritmu. Preto sa pri optimalizácií výkonu programu budeme zameriavať práve na túto časť. Po ohodnotení jedincov je vytvorená nová populácia, ak nie je dosiahnutý predom daný maximálny počet iterácií. V opačnom prípade je vrátená Pareto množina konvolučných neurónových sietí.

5.2.1 Kódovanie jedincov

Ako bolo povedané v kapitole 2, evolučné algoritmy pracujú s fenotypmi a genotypmi, kde genotypy sú reťazce, ktoré kódujú jednotlivé kandidátne riešenia. Fenotypy sú následne konkrétne podoby genotypov, v našom prípade konkrétna podoba kandidátnej neurónovej siete. Aby bolo kódovanie genotypu efektívnejšie, je možné určité komponenty siete pridať "fixne" každému jedincovi a zredukovať tak dĺžku výsledného genotypu.

Kandidátne neurónové siete sú rozdelené na určité výpočtové bloky, tzv. fázy, za ktorými nasledujú pevne nastavené komponenty siete. Každá fáza môže obsahovať maximálne n uzlov, kde každý uzol predstavuje určitú komponentu siete, napr. konvolučnú vrstvu, zoskupovaciu vrstvu alebo vopred stanovenú postupnosť operácií. Genotypy sú reprezentované ako binárna postupnosť $x_i = (x_i^{(1)}, x_i^{(2)}, ..., x_i^{(p)})$, kde p je počet fáz a $x_i^{(j)}$ predstavuje prepojenie uzlov vo fáze j. Každý prvok $x_i^{(j)}$ je binárna postupnosť, ktorá kóduje orientovaný acyklický graf v danej fáze. Každý graf môže obsahovať maximálne n uzlov a ich vzájomné prepojenie je dané konkrétnou binárnou postupnosťou.

Aby sme zväčšili prehľadávaný priestor kandidátnych riešení, kódovanie jedincov je navrhnuté tak, aby umožňovalo v každom bloku vytvoriť rozvetvenú architektúru z jednotlivých vrstiev, tzn. prepojenia medzi uzlami bloku sa môžu preskakovať alebo vetviť. Avšak je potrebné aby po každej fáze nasledovala agregačná vrstva, ktorá vypočíta jeden spoločný výstup daného bloku. Na výstupe jednotlivých agregačných vrstiev nasledujú zoskupovacie vrstvy a po poslednej fáze nasleduje plne prepojená sieť, ktorá klasifikuje vyextrahované príznaky z kandidátnej konvolučnej siete.



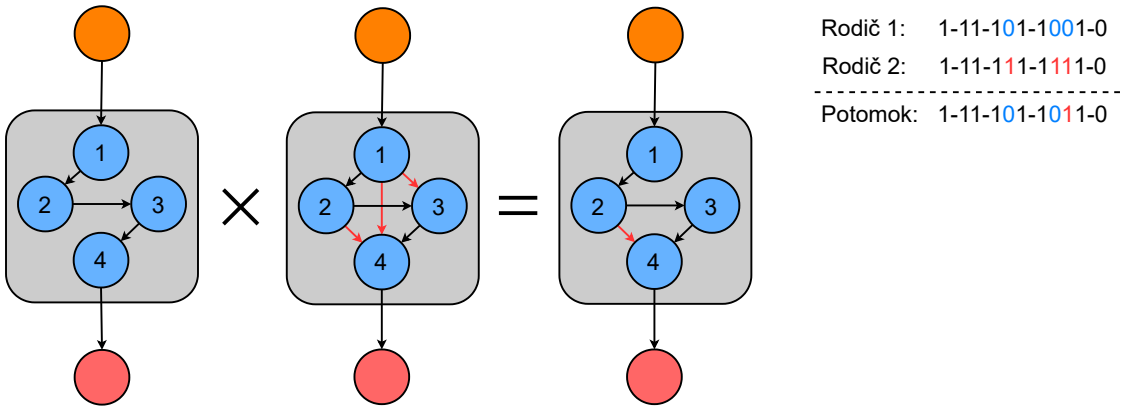
Obr. 5.3: Kódovanie kandidátnych neurónových sietí

Na obrázku 5.3 môžeme vidieť ukážku kódovania v jednotlivých fázach. Pre lepšiu čitateľnosť je postupnosť rozdelená na niekoľko častí pomocou pomlčiek, kde každá časť označuje prepojenie konkrétneho uzla s predchádzajúcimi. Na koniec postupnosti je taktiež pridaný príznak, ktorý určuje, či je vstup fázy priamo prepojený na výstup. V kódovaní jednotlivých častí vždy prvý bit určuje či je uzol využitý. Zvyšné bity následne označujú,

ktoré výstupy predchádzajúcich uzlov má na vstupe. Ak daný uzol nemá na vstupe žiadny výstup z predchádzajúcich uzlov, na jeho vstup ide priamo vstup danej fázy. To znamená, že napr. reťazec 10100 označuje, že sa využíva uzol číslo 5 a na svojom vstupe má výstup uzla číslo 2.

5.2.2 Kríženie

Kríženie medzi vybranými jedincami (rodičmi) pozostáva vo vzájomnej kombinácií bitov v ich genotypoch. Počas kríženia sa môžu kombinovať bloky a prepojenia vždy iba medzi fázami na rovnakej pozícií j. Výsledkom kríženia sú vždy dvaja noví jedinci, ktorí v jednotlivých fázach zdieľajú rovnaké bloky a prepojenia ako zdieľajú ich rodičia a tvoria novú kombináciu častí v ktorých sa jednotliví rodičia líšia. Ukážku kríženia jedincov môžeme vidieť na obrázku 5.4.



Obr. 5.4: Kríženie kandidátnych neurónových sietí

5.2.3 Mutácia

Na zvýšenie diverzity medzi kandidátnymi riešeniami a na zníženie rizika uviaznutia v lokálnom extréme účelovej funkcie je využívaný operátor mutácie. Nakoľko sú genotypy jedincov reprezentované ako binárne reťazce, je možné využiť mutáciu inverzie bitu, ktorá s určitou malou pravdepodobnosťou invertuje náhodný bit v genotype nového jedinca. Vďaka povahe daných genotypov umožňuje aj malá zmena v postupnosti vytvoriť úplne nové riešenie a tým zvýšiť diverzitu populácie.

5.2.4 Ohodnotenie jedincov

Pri ohodnocovaní jedincov sa postupuje tak, ako je to popísané v kapitole 2.3. To znamená, že sa vytvorí pomocná populácia, ktorá obsahuje jedincov starej populácie spolu s ich potomkami. Pre každého jedinca v populácií sa následne vyhodnotí, či je dominovaný iným alebo nie a postupne sa vytvoria jednotlivé Pareto množiny. Pre tie sa vyhodnotia zhlukové vzdialenosti jedincov, ktorí do danej množiny patria. Fitness hodnota jedincov bude následne vypočítaná na základe indexu ich Pareto množiny a hodnoty zhlukovej vzdialenosti

od ostatných jedincov. V našom prípade sa snažíme optimalizovať podľa dvoch protichodných kritérií, ktorými sú presnosť siete a počet parametrov, ktoré je potrebné trénovať.

5.2.5 Tvorba novej populácie

Po vypočítaní všetkých fitness hodnôt sa jedinci zoradia podľa svojho ohodnotenia a vyberie sa n najlepších, ktorí budú tvoriť novú populáciu. Celý proces sa následne opakuje v ďalších iteráciach, kým nie je dosiahnutý predom daný počet generácií. Po skončení výpočtu sa ako najlepší výsledok vráti Pareto množina s najnižším indexom.

Kapitola 6

Implementácia

Cieľom tejto kapitoly je popísať implementáciu a použitie programu, ktorého návrh je popísaný v kapitole 5. Program je implementovaný v jazyku Python s využitím knižnice TensorFlow na prácu s neurónovými sieťami. Súčasťou programu je tiež grafické užívateľské rozhranie, ktoré umožňuje jednoduché nastavenie vstupných parametrov výpočtu a bolo vytvorené pomocou frameworku PyQt.

Implementácia algoritmu NSGA-II bola inšpirovaná projektom *NSGA-II*[1]. Zdrojové kódy programu, spolu s návodom na použitie sú dostupné v Github repozitáry[2].

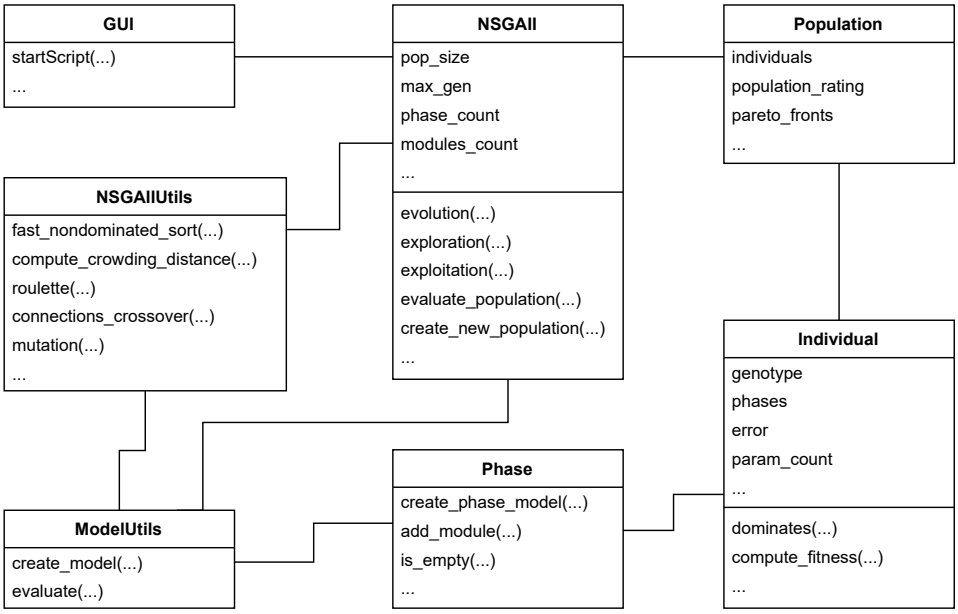
6.1 Štruktúra programu

Program je napísaný s využitím objektovo orientovaného prístupu a obsahuje niekoľko hlavných a pomocných tried. Hlavnou triedou je `NSGAII`, ktorá riadi celý proces evolúcie. Obsahuje jednotlivé parametre výpočtu a riadi jeho priebeh pomocou metód genetických operátorov (selekcia, kríženie, mutácia), ktoré sú implementované v pomocnej triede `NSGAIIUtils`. Ďalšou z hlavných tried je trieda `Population`, ktorá obsahuje jednotlivé objekty jedincov. Jedinci sú reprezentovaný triedou `Individual`, ktorá obsahuje genotyp a fenotyp daného jedinca. Každý jedinec má taktiež vypočítanú svoju fitness hodnotu, spolu s chybou a počtom parametrov svojej kandidátnej siete. Základnú hierarchiu jednotlivých tried môžeme vidieť na obrázku 6.1.

6.1.1 Reprezentácia jedinca

Jedinci sú reprezentovaní triedou `Individual`, ktorá obsahuje genotyp daného jedinca a informácie o jeho kandidátnej CNN (presnosť, chyba, počet parametrov). Genotyp jedinca je reprezentovaný ako viacrozmerné pole, obsahujúce jednotlivé fázy siete. Každá fáza následne obsahuje *n+2* polí, kde *n* je počet modulov v každej fáze. Polia *1* až *n* obsahujú hodnoty *0* alebo *1*, kde prvá hodnota označuje, či je modul použitý a zvyšné hodnoty označujú vstupy predchádzajúcich modulov, tak ako to bolo popísané v kapitole 5. Pole *n+1*

[1] https://github.com/baopng/NSGA-II
[2] https://github.com/xprist06/MasterThesis



Obr. 6.1: Grafické zobrazenie základnej štruktúry programu

je pridané oproti pôvodnému návrhu a reprezentuje výstupný (agregačný) uzol fázy. Obsahuje n hodnôt *0* alebo *1*, kde každá hodnota určuje, či je výstup modulu s odpovedajúcim indexom vstupom výstupného uzlu. Toto pole bolo potrebné pridať kvôli zväčšeniu prehľadávaného priestoru, nakoľko pôvodný návrh umožňoval iba poslednému modulu, aby bol súčasne aj výstupný. Posledné pole *n+2* obsahuje iba jednu hodnotu *0* alebo *1* a označuje, či ide vstup fázy priamo aj na výstup.

6.1.2 Reprezentácia CNN

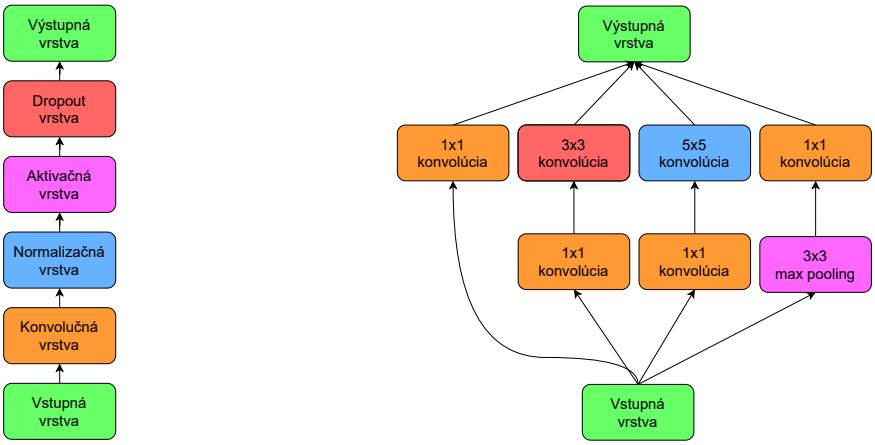
Jednotlivé kandidátne siete sa skladajú z niekoľkých fáz, kde každá obsahuje určitý počet modulov, tak ako je to popísané v kapitole 5. Jednotlivé moduly použité v sieťach, sú definované v súbore `layers.py` v poli `phase_layers`, kde index každého modulu určuje číslo jeho uzlu vo fáze. Na tvorbu kandidátnych sietí sú použité tri typy modulov:

- **Konvolučný modul**, ktorý sa skladá zo štyroch po sebe idúcich vrstiev:
  1. konvolučná vrstva,
  2. normalizačná vrstva,
  3. aktivačná vrstva,
  4. dropout vrstva,

  kde jednotlivé konvolučné moduly obsahujú rôzne veľkosti a počty filtrov. Krok výpočtu (Stride) je prednastavený na 1 a je použitá **ReLU** aktivačná funkcia. Hodnota pravdepodobnosti vynechania neurónu (Dropout) je nastavená je 10%. Schému konvolučného modulu môžeme vidieť na obrázku 6.2a.
- **Inception modul** je založený na paralelnom využití niekoľkých rôznych konvolučných modulov s tým, že výstupy sú následne konkatenované (agregované) do jedného

výstupu. Jednotlivé konvolučné moduly mali v rámci jedného Inception modulu rovnaký počet kanálov. Schému inception modulu môžeme vidieť na obrázku 6.2b.

- **Zoskupujúci modul** sa skladá z jednej zoskupujúcej vrstvy, ktorá vykonáva redukcia vstupných dát na základe maximálnej hodnoty (*max_pooling*) alebo na základe priemeru hodnôt (*average_pooling*), tak ako je to zobrazené na obrázku 3.12. Veľkosť filtra je 3x3 a krok výpočtu je prednastavený na 1.



(a) Konvolučný modul (b) Inception modul

Obr. 6.2: Konvolučný a Inception modul

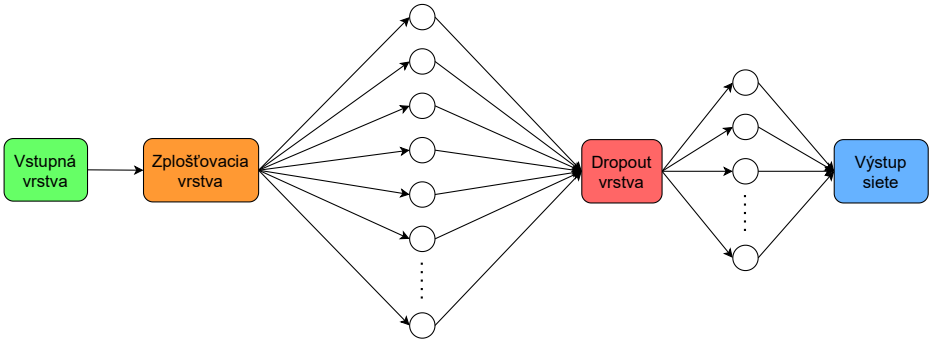
Okrem dynamicky pridávaných častí, obsahuje každá kandidátna CNN aj niekoľko fixných. Prvou je pevne pridaný konvolučný modul s veľkosťou filtra 3x3 a 32 kanálmi. Modul je pridaný pred prvú fázu, aby bolo zaručené, že sieť vykoná aspoň určité spracovanie vstupu, pre prípad, že by boli vytvorené prázdne fázy. Druhá časť je pridanie dropout vrstvy za každú fázu. Hodnota pravdepodobnosti "dropout" je rovná 10-násobku indexu fázy v sieti. To znamená, že dropout za druhou fázou bude mať pravdepodobnosť 20%. Posledná pevná súčasť sietí je modul, reprezentujúci plne prepojenú sieť. Ten sa nachádza na konci každej kandidátnej CNN a obsahuje štyri vrstvy:

1. splošťovaciu vrstvu,
2. plne prepojenú skrytú vrstvu obsahujúcu 512 neurónov s **ReLU** aktivačnou funkciou,
3. dropout vrstvu s pravdepodobnosťou 25%,
4. výstupnu vrstvu obsahujúcu 10 neurónov so Softmax aktivačnou funkciou.

Ukážku plne prepojenej vrstvy môžeme vidieť na obrázku 6.3.

6.2 Priebeh evolúcie

Priebeh evolučného hľadania kandidátnych CNN je implementovaný podľa algoritmu NSGA-II, popísaného v kapitole 2.3.



Obr. 6.3: Plne prepojená vrstva

Na začiatku výpočtu je vytvorená počiatočná populácia, kde je postupne pre každého jedinca vytvorený jeho model siete, ktorý je následne natrénovaný a je vyhodnotená presnosť a počet parametrov siete. Trénovanie jedincov je počas evolúcie vykonávané na podmnožine trénovacej dátovej sady, ktorá je náhodne vybraná. Veľkosť "evolučnej" dátovej sady je 25% pôvodnej veľkosti. Trénovacia sada je taktiež počas trénovania rozdelená na trénovaciu a validačnú. Miera rozdelenia dát je daná vstupným parametrom programu. Po natrénovaní siete je presnosť každého jedinca vyhodnotená na pôvodnej testovacej sade daného klasifikačného problému.

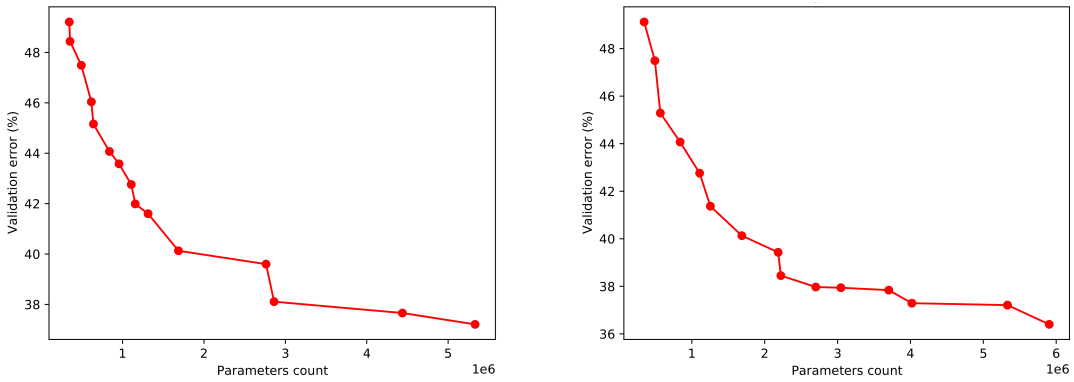
Po ohodnotení populácie je vytvorená populácia potomkov. Jedinci sú vytváraní zo svojich rodičov pomocou kríženia a náhodnej mutácie. Spôsob výberu rodičov je možné zvoliť medzi ruletov a turnajom. Pri samotnom krížení je taktiež možné zvoliť medzi dvoma spôsobmi. Prvým typom kríženia je kríženie na úrovní génov, tak ako je to znázornené na obrázku 5.4 v kapitole 5.2.2. Druhým typom je kríženie na úrovni modulov, kedy sú medzi sebou kombinované celé moduly jedincov v jednotlivých fázach, so svojími vstupmi aj výstupmi. Po samotnom krížení a mutácií je každý genotyp následne zvalidovaný, tak aby reprezentoval validnú sieť. Prvým krokom validácie je kontrola vstupov z nepoužitých modulov. To znamená, že každý jedinec musí mať použitý každý modul, ktorý je vstupom iného modulu, resp. žiadny modul nesmie mať vstup z nepoužitého modulu. Ak je takýto modul použitý na vstupe, tak je v rámci validácie daný vstup v genotype nastavený na hodnotu *0*. Druhým krokom validácie je ošetrenie "mŕtvych" uzlov. To znamená ošetriť, aby bol použitý výstup každého modulu. Ak nie je výstup použitý iným modulom, je nastavený ako vstup výstupného (agregačného) uzlu fázy. Keď sú genotypy zvalidované, je skontrolované, či už rovnaký genotyp nebol v predchádzajúcich generáciach vytvorený. Ak áno, genotyp sa znova nepoužije a je vytvorený nový.

Keď je vytvorená celá populácia potomkov, jedinci sú natrénovaní a ohodnotení, a je vytvorená pomocná populácia jedincov, pozostávajúca z rodičov a potomkov. Jedinci sú následne zoradení do Pareto množín a z nich je vytvorená nová populácia, tak ako je to popísané v kapitole 2.3.

Celý výpočet je rozdelený na 3 hlavné časti:

1. **Explorácia prehľadávaného priestoru**, kedy je vykonávaný proces evolúcie podľa algoritmu NSGA-II.

2. **Exploitácia prehľadávaného priestoru** je snaha o nájdenie lepších jedincov, ktorý sa nachádzajú v okolí populácie, ktorá bola výsledkom evolúcie. Tento proces sa vykonáva po skončení evolúcie a počet generácií je rovný polovici počtu generácií vo fáze explorácie. V každej generácii je vytvorená nová populácia kandidátnych jedincov, z ktorej je v kombinácii s populáciou rodičov vytvorená nová generácia jedincov, tak ako pri algoritme NSGA-II. Avšak oproti evolúcií je operátorom selekcie vybraný vždy iba jeden rodič z ktorého je nový jedinec vytvorený náhodnou mutáciou v genotype. Na obrázkoch 6.4a a 6.4b môžeme vidieť porovnanie v kvalite jedincov po skončení explorácie a exploitácie v experimente na dátovej sade *CIFAR-10*. Z grafov je vidieť zlepšenie presnosti jedincov, ktorá bola po skončení exploitácie v priemere vyššia o 2%.



(a) Populácia po skončení explorácie (b) Populácia po skončení exploitácie

Obr. 6.4: Porovnanie Pareto množín medzi exploráciou a exploitáciou

3. **Validácia jedincov** je dotrénovanie a vyhodnotenie kvality jedincov, ktorí boli vrátení ako najlepší v poslednej populácii. Toto trénovanie, oproti evolúcii, prebieha na celej dátovej sade. Jedinci v populácii sú po skočení fáze validácie opäť zoradení do Pareto množín, podľa algoritmu NSGA-II a ako výsledok výpočtu je vrátená množina najlepších jedincov.

6.3 Výstup programu

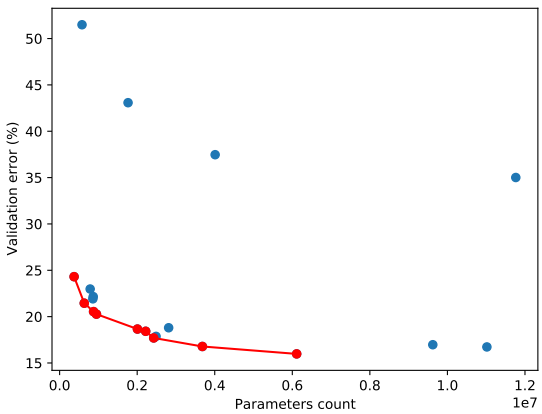
Pre začiatkom výpočtu je vytvorená zložka, do ktorej sú postupne ukladané priebežné výstupy programu, spolu s celkovým výsledkom, ktorý je uložený nakoniec. Názov zložky je totožný s časom vytvorenia za ktorý je pridané unikátne id, aby nedošlo ku konfliktu v prípade paralelného spustenia. Výstup programu obsahuje niekoľko častí:

1. **Výstupný log evolúcie**, ktorý obsahuje základné textové informácie o výpočte a o jeho priebehu. Medzi informácie patria hodnoty jednotlivých vstupných parametrov programu a hodnoty jedincov (chyba, počet parametrov, genotyp) v populáciach naprieč generáciami. Log je uložený v súbore `output.log`.

2. **Grafické zobrazenie evolúcie**, ktoré pozostáva z dvoch súborov. Prvým súborom je `pareto_graph.pdf`, ktorý obsahuje zobrazenie výslednej populácie, spolu so zvýraznením najlepšej Pareto množiny. Druhým súborom je *pareto_evolution.gif*, ktorý obsahuje animáciu zmien v populácii počas evolúcie. Animácia je zostavená z jednotlivých grafov populácií, ktoré sú vytvárané po každej generácii. Na vytvorenie grafických zobrazení jednotlivých populácií bola využitá knižnica *Matplotlib*[3]. Ukážku grafu môžeme vidieť na obrázku 6.5.

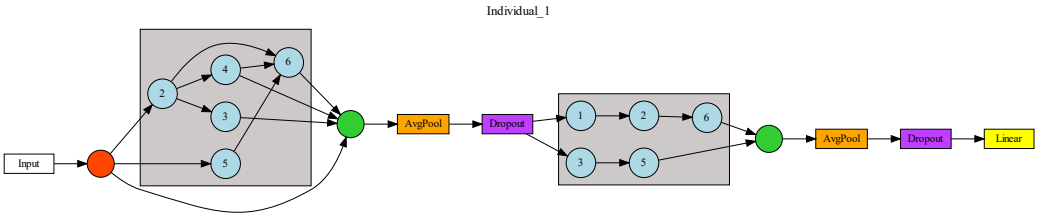
3. **Informácie o jednotlivcoch** z najlepšej Pareto množiny. Pre každého jedinca je vytvorená samostatná zložka `individual_x`, kde x označuje index jedinca z množiny. Jednotlivé zložky obsahujú niekoľko súborov:

   - `genotype.txt` je súbor obsahujúci textovú podobu genotypu, spolu s hodnotou presnosti, chyby, počtu parametrov a čas trénovania daného modelu,
   - `genotype_graph.pdf` je graf zobrazujúci prepojenie modulov v jednotlivých fázach genotypu. Graf je vygenerovaný pomocou knižnice *Graphviz*[4]. Implementácia generovania grafu bola inšpirovaná z projektu *NSGA-Net*[5]. Ukážku grafu môžeme vidieť na obrázku 6.6,
   - `cnn_structure.png` je graf zobrazujúci kandidátnu CNN. Na generovania je využitá metóda `plot_model()`[6] frameworku Keras,
   - `model.json` je štruktúra modelu vo formáte *JSON*. Model je možné opätovne zostaviť pomocou metódy `model_from_json()`[7] frameworku Keras.



Obr. 6.5: Výstupný Pareto graf

[3] https://matplotlib.org/
[4] https://graphviz.org/
[5] https://github.com/ianwhale/nsga-net
[6] https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model
[7] https://www.tensorflow.org/api_docs/python/tf/keras/models/model_from_json



Obr. 6.6: Graf prepojenia modulov v genotype

6.4 Práca s programom

S programom je možné pracovať priamo cez príkazový riadok alebo cez jednoduché grafické užívateľské rozhranie (GUI), ktoré bolo vytvorené pomocou knižnice PyQt5.

Aby bolo možné s programom pracovať na rôznych zariadeniach a nebolo nutné inštalovať požadované závislosti, bolo vytvorené virtuálne prostredie `tf-gpu-2.4.1-pristas` pomocou nástroja *Anaconda*. Prostredie je uložené v súbore `tf-gpu-2.4.1.yml`. Na vytvorenie prostredia je potrebné mať nainštalovaný program *Anaconda* a následne spustiť príkaz:

- `conda env create -f tf-gpu-2.4.1.yml`

Prostredie je možné aktivovať príkazom:

- `source activate tf-gpu-2.4.1-pristas`

Program je následne možné spustiť dvoma spôsobmi:

1. `python3 main.py -p 15 -g 15 -m 0.15 --phases 3 --modules 6`
2. `python3 gui.py`

Prvá možnosť znázorňuje ukážku spustenia cez príkazový riadok, kedy je možné pomocou vstupných parametrov nastaviť počet jedincov (`-p`), počet generácii (`-g`), pravdepodobnosť mutácie (`-m`), počet fáz (`--phases`) a počet modulov (`--modules`). Jednotlivé vstupné argumenty nie sú povinné a je možné si ich hodnoty prednastaviť v súbore `main.py`. Program umožňuje nastavenie väčšieho množstva vstupných parametrov, ale kvôli zjednodušeniu je možné pomocou vstupných argumentov nastaviť iba týchto päť, ktoré boli vybrané ako najdôležitejšie. Zvyšné parametre je možné prednastaviť v súbore `main.py`. Druhá možnosť spustí GUI, ktoré umožňuje nastavenie všetkých vstupných parametrov programu.

V prípade, že je k dispozícií viac GPU, je možné zvoliť, ktoré sa použije pre výpočet príkazom:

- `CUDA_VISIBLE_DEVICES=0`

kde `0` označuje `id` GPU, ktoré sa použije. V prípade, že nie je zvolené GPU a je ich k dispozícií viac, program bude na výpočet používať prvé voľné a zvyšné nechá k dispozícií pre ďalšie výpočty.

6.4.1 Grafické užívateľské rozhranie

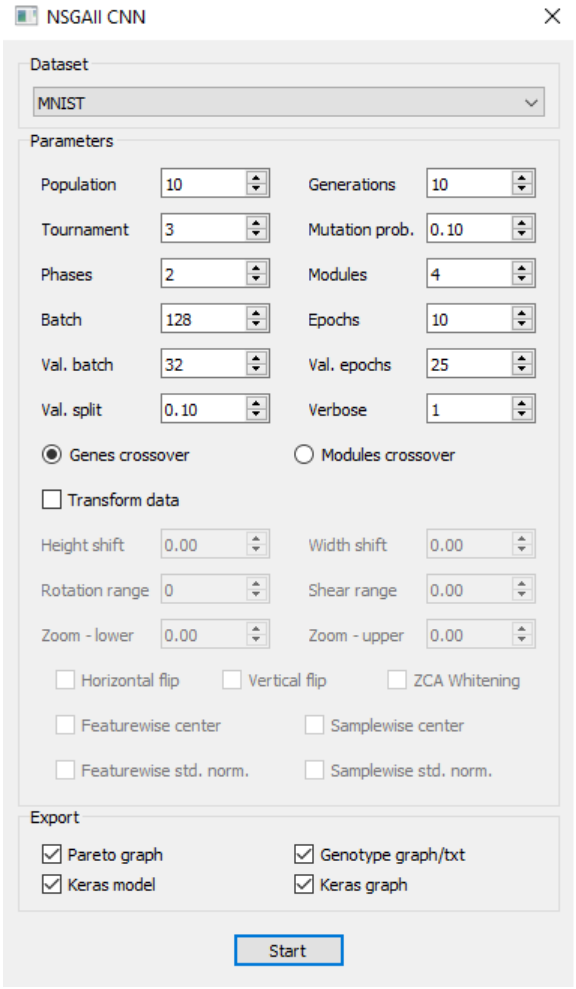
Ako môžeme vidieť na obrázku 6.7, GUI je veľmi jednoduché a slúži na nastavenie vstupných parametrov programu a jeho následné spustenie. Po spustení výpočtu sa zatvorí a ďalej s ním nie je možné pracovať. Samotné GUI obsahuje niekoľko častí:

1. **Výber datasetu**, ktorý sa nachádza na začiatku GUI. Je možné si vybrať z piatich rôznych datasetov, ktoré sa bežne používajú pre problém klasifikácie obrazu. Okrem datasetu *CIFAR-10* boli pridané dátové sady *MNIST, Fashion MNIST, SVHN* a *CIFAR-100*. Datasety boli pridané ako rozšírenie programu. Avšak v rámci experimentov neboli použité.

2. **Nastavenie parametrov** sa nachádza v strednej časti GUI. Na začiatku je možné nastaviť základné parametre výpočtu, ako počet jedincov v populácii, počet generácii a pod. Následne je možné povoliť predspracovanie vstupných dát a nastaviť rôzne náhodné transformácie obrazových dát, čo môže viesť k zlepšeniu natrénovania siete.

3. **Export výsledku** sa nachádza v spodnej časti GUI a umožňuje nastaviť dáta, ktoré sa budú ukladať počas a po výpočte. V rámci exportu nie je možné vypnúť logovanie, ktoré je vytvorené pri každom výpočte a uložené do súboru `output.log`.

Po nastavení jednotlivých parametrov je možné výpočet spustiť tlačítkom `Start`.



Obr. 6.7: Grafické užívateľské rozhranie

Kapitola 7

Experimenty

V tejto kapitole budú popísané experimenty vykonané na dátovej sade *CIFAR-10* s cieľom vyhodnotiť kvalitu navrhnutého a implementovaného programu pre automatický návrh konvolučných neurónových sietí.

7.1 CIFAR-10

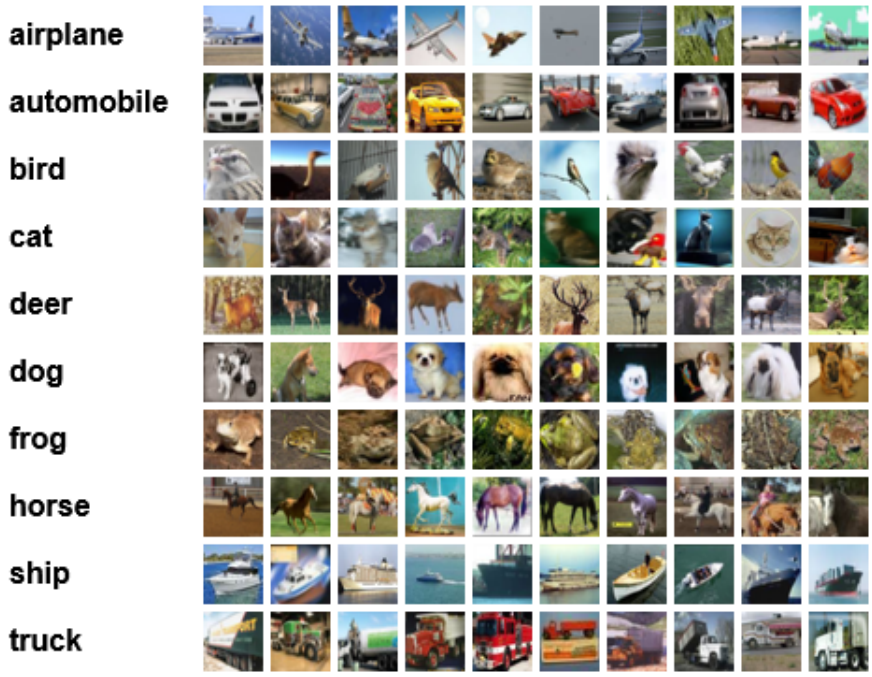
Dátová sada *CIFAR-10* sa skladá zo 60000 RGB obrázkov o rozmeroch 32x32 pixelov, ktoré je možné rozdeliť do 10 tried so 6000 obrázkami pre každú triedu. Dataset je taktiež rozdelený na testovaciu a trénovaciu množinu v pomere 5:1. Ako bolo povedané v kapitole 6, na trénovanie kandidátnych sietí bola použitá podmnožina trénovacej dátovej sady, ktorá bola následne rozdelená na trénovaciu a validačnú. Na testovanie sietí bola použitá celá testovacia sada, tzn. 10000 vzoriek. Celá trénovacia sada bola použitá až na validáciu najlepších jedincov, po skončení evolúcie. Ukážku datasetu, spolu s triedami pre jednotlivé vzorky môžeme vidieť na obrázku 7.1.

7.2 Nastavenie experimentov

Testovanie programu bolo vykonávané na fakultnom servery SC-GPU1, ktorý obsahuje grafické karty NVIDIA GTX 1080 (Pascal), 8GB RAM. Ostré experimenty boli vykonávané na superpočítači Barbora s grafickými kartami NVIDIA V100, 32 GB RAM, ktorý umožňoval niekoľkonásobné zrýchlenie výpočtov a vďaka tomu bolo možné vykonať niekoľko behov programu pre každý experiment.

Celkovo boli vykonané štyri rôzne experimenty, každý po osem behov. Z každého experimentu bol následne vybraný najlepší beh, ktorého výsledok je popísaný v nasledujúcej podkapitole. Z jednotlivých behov boli taktiež vytvorené krabicové diagramy, ktoré znázorňujú presnosť sietí pre jednotlivé experimenty.

Všetky experimenty obsahovali niekoľko fixných parametrov a niekoľko dynamicky menených pre každý experiment. Hodnoty spoločných parametre pre jednotlivé experimenty



Obr. 7.1: Dátová sada CIFAR-10, prevzaté z [21]

sú uvedené v tabuľke 7.1. Pre jednotlivé experimenty boli menené dva hlavné parametre. Prvým je počet fáz, z ktorých sa skladajú kandidátne CNN, čo má veľký vplyv na veľkosti jednotlivých sietí a s tým aj počet trénovaných parametrov. Druhým meneným parametrom bol počet modulov z ktorých sa skladajú jednotlivé fázy. Vyšší počet modulov umožňoval viac možných kombinácií jednotlivých typov operácií a s tým aj možnosť lepšieho spracovania vstupu. Na druhej strane, nižší počet modulov umožňoval znížiť počet parametrov kandidátnych CNN.

V rámci experimentov bolo použitých 9 rôznych modulov. Prvé tri boli konvolučné moduly s veľkosťou filtra 3x3 a počtom kanálov 32, 64 a 128. Moduly štyri a päť reprezentovali inception moduly s počtom kanálov 32 a 64. Šiesty modul reprezentoval zoskupujúcu vrstvu s operáciou priemeru a veľkosťou filtra 3x3. Posledné tri moduly boli opäť konvolučné moduly s veľkosťou filtra 5x5 a počtom kanálov 32, 64 a 128. Experimenty 1 a 2 pracovali s dvomi fázami, kde prvý experimenty využíval šesť modulov a druhý experimenty využíval deväť modulov. Experimenty 3 a 4 pracovali s tromi fázami, kde obdobne ako pri prvých dvoch experimentov využíval tretí experiment šesť modulov a štvrtý experiment deväť modulov.

V rámci ohodnocovania sietí boli jednotlivé kandidátne siete trénované s využitím optimalizačného algoritmu *Adam* a ako stratová funkcia bola využitá kategorická krížová entropia (*Sparse Categorical Crossentropy*). Táto kombinácia bola zvolená, nakoľko dávala najlepšie výsledky počas prvotného testovania programu.

Tabuľka 7.1: Základné nastavenie experimentov

Názov parametra	Hodnota parametra
Veľkosť populácie	15
Počet generácií	20
Pravdepodobnosť mutácie	0.15
Výber jedincov	Ruleta
Typ kríženia	Génové kríženie
Dávka (Batch)	128
Počet epoch	10
Validačná dávka	64
Validačný počet epoch	50
Validačné rozdelenie (Val. split)	0.2
Verbose (Logovanie trénovania)	1
Transformácia dát	Nie

7.3 Výsledky experimentov

Pre každý experiment bol vybraný najlepší beh, ktorého výsledky sú zobrazené vo forme tabuľky. Výsledky pre jednotlivé experimenty sú zobrazené v tabuľkách 7.3, 7.4, 7.5 a 7.6. Pre každý experiment sú zobrazení jedinci z najlepšej Pareto množiny spolu s ich presnosťou, počtom parametrov a časom trénovania vo fáze validácie. Pre každý experiment bol z jednotlivých behov programu vypočítaný priemerný čas výpočtu, ktorý je zobrazený v tabuľke 7.2. Experimenty boli spúšťané na superpočítači Barbora, vždy na jednom GPU NVIDIA V100, 32GB RAM. Z jednotlivých behov boli taktiež vytvorené krabicové diagramy, ktoré znázorňujú presnosti najlepších jedincov pre jednotlivé experimenty. Diagramy môžeme vidieť na obrázku 7.2, kde je vidieť postupne sa zlepšujúca kvalita jedincov v experimentoch.

Jedinec s najvyššou presnosťou bol nájdený vo štvrtom experimente. Dosahoval 86.01% presnosť klasifikácie s počtom parametrov 5884650. Na obrázkoch 7.3a a 7.3b môžeme vidieť porovnanie Pareto množín v tomto experimente a na obrázku 7.4 môžeme vidieť vizualizáciu genotypu jedinca s najvyššou presnosťou. Pre porovnanie veľkosti kandidátnych CNN je na obrázku 7.5 zobrazený jedinec s najmenším počtom parametrov z rovnakého behu štvrtého experimentu. Jedinec dosahoval presnosť 71.43% s počtom parametrov 294442.

Vizualizácie Pareto množín a genotypy najlepších jedincov pre experimenty 1, 2 a 3 sú zobrazené v prílohách A, B a C.

Tabuľka 7.2: Priemerná doba behu jednotlivých experimentov na GPU NVIDIA V100

Doba výpočtu experimentov	
Experiment	**Doba výpočtu (hh:mm:ss)**
Experiment 1 - 2 fázy, 6 modulov	06:38:26
Experiment 2 - 2 fázy, 9 modulov	09:55:29
Experiment 3 - 3 fázy, 6 modulov	07:50:41
Experiment 4 - 3 fázy, 9 modulov	09:54:16

Tabuľka 7.3: Experiment 1: 2 fázy, 6 modulov

Experiment 1			
Jedinec	**Presnosť**	**Počet parametrov**	**Doba trénovania (mm:ss)**
Individial_490	82.27%	7588202	16:42
Individial_609	81.63%	2402058	08:35
Individial_658	80.05%	1465034	13:05
Individial_57	79.73%	1233482	07:43
Individial_685	57.80%	1196426	00:19
Individial_75	55.94%	1120074	00:10
Individial_435	54.72%	1083018	00:07
Individial_518	52.06%	1064490	00:06
Individial_629	50.67%	1055178	00:05

Tabuľka 7.4: Experiment 2: 2 fázy, 9 modulov

Experiment 2			
Jedinec	**Presnosť**	**Počet parametrov**	**Doba trénovania (mm:ss)**
Individial_122	84.03%	6043242	14:35
Individial_494	82.85%	5328522	22:26
Individial_146	82.81%	3286986	09:50
Individial_296	82.20%	2845738	09:56
Individial_572	80.11%	2488426	05:16

Tabuľka 7.5: Experiment 3: 3 fázy, 6 modulov

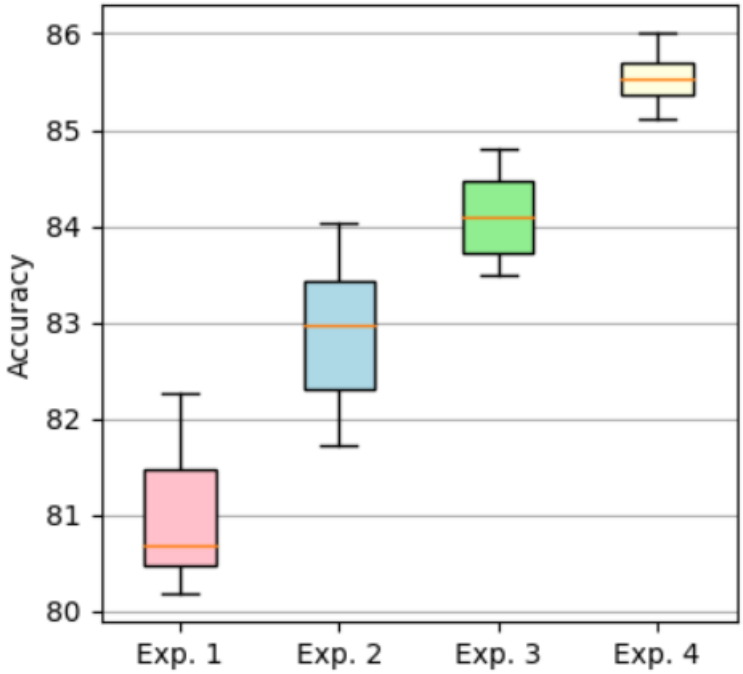
Experiment 3			
Jedinec	**Presnosť**	**Počet parametrov**	**Doba trénovania (mm:ss)**
Individial_541	84.79%	5504682	22:01
Individial_511	84.27%	3084970	20:15
Individial_488	83.99%	1494570	14:22
Individial_430	83.10%	1374762	14:14
Individial_477	80.45%	677642	09:40
Individial_208	77.95%	409802	05:22
Individial_504	76.19%	379914	04:05
Individial_397	75.23%	333450	02:24
Individial_404	74.45%	305802	02:14
Individial_265	69.98%	287274	01:36
Individial_515	68.56%	278058	01:28

7.4 Zhodnotenie experimentov

S programom boli vykonané štyri experimenty, každý po osem behov, kde najpresnejšie riešenie sa podarilo nájsť v štvrtom experimente, ktorý pracoval s tromi fázami a deviatimi modulmi. Nájdené riešenie dosahovalo presnosť 86.01% s počtom parametrov 5.8M. V sú-

Tabuľka 7.6: Experiment 4: 3 fázy, 9 modulov

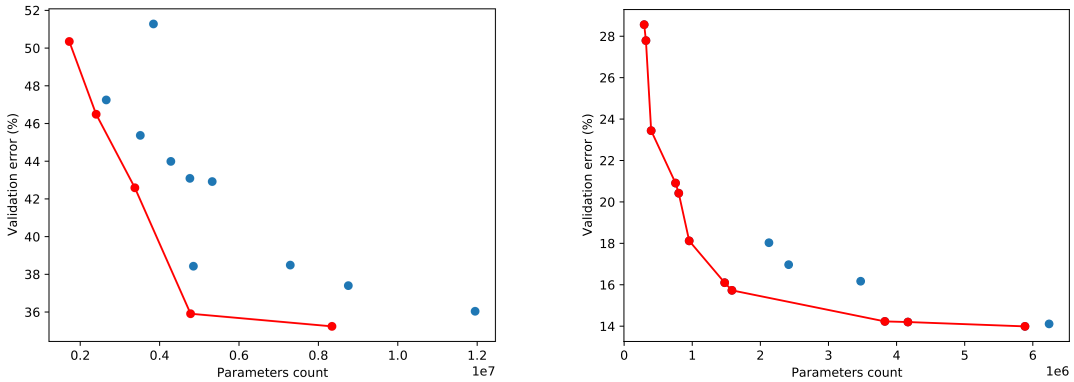
Experiment 4			
Jedinec	**Presnosť**	**Počet parametrov**	**Doba trénovania (mm:ss)**
Individial_596	86.01%	5884650	15:55
Individial_455	85.79%	4166090	15:08
Individial_389	85.76%	3827978	13:48
Individial_516	84.27%	1582602	12:48
Individial_381	83.89%	1476010	12:08
Individial_442	81.87%	955338	11:22
Individial_327	79.57%	801738	10:42
Individial_152	79.08%	754698	04:48
Individial_532	76.56%	397034	03:21
Individial_568	72.21%	320042	01:45
Individial_550	71.43%	294442	01:38



Obr. 7.2: Diagram presnosti jedincov v experimentoch

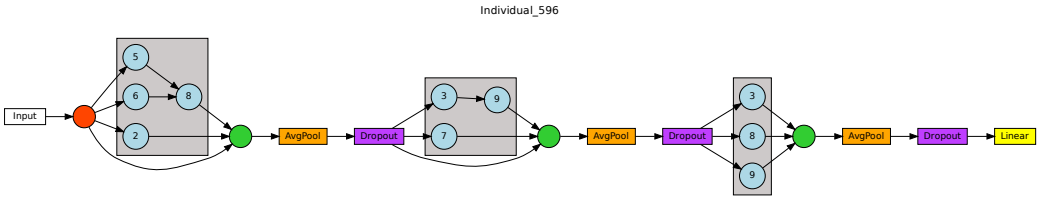
časnosti dosahujú najlepšie riešenia presnosť cez 98% s desiatkami miliónov parametrov[1]. Aj keď sa nájdené riešenia presnosťou neblížia najpresnejším architektúram, z výsledkov jednotlivých experimentov je možné povedať, že implementovaný program je schopný nájsť nekonvenčné architektúry konvolučných neurónových sietí, dosahujúcich vysokú mieru presnosti. Program taktiež umožňuje užívateľovi si vybrať z nájdených riešení takú sieť, ktorá najviac vyhovuje jeho potrebám, v závislosti od počtu parametrov alebo požadovanej presnosti.

[1] https://paperswithcode.com/sota/image-classification-on-cifar-10

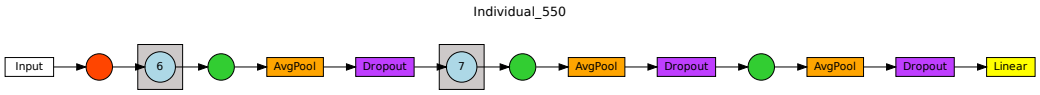


(a) Pareto množina po 1. generácii (b) Pareto množina po výpočte

Obr. 7.3: Porovnanie Pareto množín medzi generáciami



Obr. 7.4: Genotyp jedinca s najvyššou presnosťou



Obr. 7.5: Genotyp jedinca s najmenším počtom parametrov

Výhodou taktiež je, že program nie je navrhnutý na hľadanie konvolučných neurónových sietí iba pre jeden dataset, ale je možné pomocou neho hľadať architektúry sietí pre rôzne datasety pre problém klasifikácie obrázkov. Pre porovnanie boli s programom vykonané ďalšie štyri experimenty po jednom behu pre klasifikačné datasety *MNIST, Fashion MNIST* a *SVHN*. Experimenty mali rovnaké spoločné nastavenia parametrov ako pri experimentoch s datasetom *CIFAR-10* a pracovali s tromi fázami a šiestimi modulmi. Z výsledkov experimentov, zobrazených v tabuľke 7.7 je vidieť, že program bol schopný nájsť riešenia dosahujúce vyššiu presnosť s menším počtom parametrov pre ľahšie datasety, v porovaní s ťažšími.

Tabuľka 7.7: Porovnanie riešení pre rôzne datasety

Porovnanie datasetov		
Dataset	**Presnosť**	**Počet parametrov**
MNIST	99.48%	375914
SVHN	95.24%	1323850
Fashion MNIST	93.72%	2864842
CIFAR-10	84.79%	5504682

7.5 Pokračovanie práce

S programom bolo vykonaných niekoľko experimentov ktoré dokázali, že program je pre zadaný dataset schopný nájsť architektúry konvolučných neurónových sietí dosahujúcich vysokú mieru presnosti. V budúcnosti by s programom bolo vhodné vykonať viac experimentov, ktoré by dôkladnejšie preverili jeho schopnosti nájsť efektívne architektúry CNN a ktoré kvôli časovej náročnosti nebolo možné vykonať v rámci tejto práce. V rámci experimentov by bolo vhodné taktiež využiť možnosť augmentácie vstupných dát, využiť iný spôsob výberu rodičov, ako aj ich následné kríženie. V budúcnosti by bolo taktiež možné program rozšíriť o návrh konvolučných neurónových sietí na iné problémy ako klasifikácia obrazových dát.

Kapitola 8

Záver

Cieľom tejto diplomovej práce bolo popísať techniky využívané k automatickému návrhu konvolučných neurónových sietí pomocou evolučných algoritmov a navrhnúť, a implementovať program, ktorý bude takéhoto návrhu schopný.

Práca je rozdelená do niekoľkých kapitol. V kapitole 2 je uvedený teoretický základ evolučných algoritmov so zameraním na genetické algoritmy a na jeho multikriteriálnu variantu - NSGA-II. Kapitola 3 popisuje umelé neurónové a konvolučné neurónové siete, ich učenie a architektúru. Posledná teoretická kapitola je kapitola 4, kde je uvedený popis automatického návrhu neurónových sietí pomocou evolučných algoritmov, so zameraním na algoritmus NSGA-Net, uvedeným v práci [24], ktorý ako hlavný prvok využíva algoritmus NSGA-II. Kapitola 5 následne obsahuje návrh implementácie programu na automatizovaný návrh konvolučných neurónových sietí. V kapitole je uvedený základný popis knižnice TensorFlow, ktorá slúži na prácu s neurónovými sieťami. Taktiež je uvedený spôsob kódovania kandidátnych neurónových sietí a proces "evolúcie" čo najlepšej siete.

Navrhnutý program bol implementovaný a jeho popis, spolu s popisom ako s programom pracovať, je uvedený v kapitole 6. Implementovaný program bol následne otestovaný a bolo s ním vykonaných niekoľko experimentov, ktoré sú spísané v kapitole 7. Z dosiahnutých výsledkov je vidieť, že program je schopný automatického návrhu konvolučných neurónových sietí, ktoré sú schopné dosiahnuť rozumnú mieru presnosti na testovacej dátovej sade. Experimenty s programom boli vykonávané na dátovej sade *CIFAR-10*, ktorá patrí medzi najpoužívanejšie dátové sady na testovanie kvality CNN pre úlohu rozpoznávania a klasifikácie obrazu. V rámci experimentov sa podarilo nájsť riešenie dosahujúce 86.01% presnosť klasifikácie s počtom parametrov 5.8M. Zdrojové kódy programu, spolu s návodom na použitie sú dostupné v Github repozitáry[1].

[1] https://github.com/xprist06/MasterThesis

Literatúra

[1] Ajith, A. *Evolutionary computation, in Handbook of Measuring System Design.* John Wiley and Sons, 2005. ISBN 9780470021439.

[2] Ashlock, D. *Evolutionary Computation for Modeling and Optimization.* Springer Science and Business Media, 2006. ISBN 9780387319094.

[3] Bidlo, M. *Úvod do evolučních algoritmů, Fakulta informačných technológií, VUT Brno.* Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FEVO-IT%2Flectures%2F03-UvodEA_EP_ES.pdf&cid=13232.

[4] Clark, T. H. *Why Neural Nets Can Approximate Any Function.* Dostupné z: https://towardsdatascience.com/why-neural-nets-can-approximate-any-function-a878768502f0.

[5] Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems.* New York: Springer-Verlag. 1989, zv. 2, č. 4, s. 303–314. ISSN 0932-4194.

[6] Darwin, .-., Murray, .-., Tatham, E. a Lacaita, .-. *On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life.* John Murray, Albemarle Street, 1859. ISBN 0659910217. Dostupné z: https://www.biodiversitylibrary.org/item/135954.

[7] Deb, K., Pratap, A., Agarwal, S. a Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation.* 2002, zv. 6, č. 2, s. 182–197. DOI: 10.1109/4235.996017.

[8] Dhp1080. Neuron. Wikimedia Commons. 2007. Dostupné z: https://commons.wikimedia.org/wiki/File:Neuron_slk.svg.

[9] Duchi, J., Hazan, E. a Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* JMLR.org. júl 2011, zv. 12, null, s. 2121–2159. ISSN 1532-4435.

[10] Elsken, T., Metzen, J. H. a Hutter, F. *Neural Architecture Search: A Survey.* 2019. ArXiv:1808.05377.

[11] Fogel, L., Owens, A. a Walsh, M. *Artificial intelligence through simulated evolution.* Chichester, WS, UK: Wiley, 1966. DOI: 10.1109/9780470544600.ch7.

[12] Gregor, M. *Experiments in plant hybridization*. Verhandlungen des naturforschenden Vereins Brünn, 1865. 672 s. ISBN 978-1148616858.

[13] Hebb, D. O. *The Organization of Behavior*. Wiley, 1949. ISBN 978-0805843002.

[14] Herzog, M. H. a Clarke, A. M. Why vision is not both hierarchical and feedforward. *Frontiers in Computational Neuroscience*. 2014, zv. 8, s. 135. DOI: 10.3389/fncom.2014.00135. ISSN 1662-5188. Dostupné z: https://www.frontiersin.org/article/10.3389/fncom.2014.00135.

[15] Holland, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975. 232 s. ISBN 9780262082136. Second edition, 1992.

[16] Hornik, K. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Netw.* GBR: Elsevier Science Ltd. marec 1991, zv. 4, č. 2, s. 251–257. DOI: 10.1016/0893-6080(91)90009-T. ISSN 0893-6080. Dostupné z: https://doi.org/10.1016/0893-6080(91)90009-T.

[17] Indolia, S., Goswami, A. K., Mishra, S. a Asopa, P. Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach. *Procedia computer science*. Elsevier B.V. 2018, zv. 132, s. 679–688. ISSN 1877-0509.

[18] Kingma, D. P. a Ba, J. Adam: A Method for Stochastic Optimization. *CoRR*. 2017. arXiv:1412.6980.

[19] Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992. ISBN 0262111705.

[20] Kriesel, D. *A Brief Introduction to Neural Networks*. 2007. Dostupné z: http://www.dkriesel.com.

[21] Krizhevsky, A. Cifar-10. 2009. Dostupné z: https://www.cs.toronto.edu/~kriz/cifar.html.

[22] Liu, H., Simonyan, K. a Yang, Y. DARTS: Differentiable Architecture Search. In: *International Conference on Learning Representations*. 2019. Dostupné z: https://openreview.net/forum?id=S1eYHoC5FX.

[23] Lu, Z., Deb, K., Goodman, E., Banzhaf, W. a Naresh Boddeti, V. NSGANetV2: Evolutionary Multi-Objective Surrogate-Assisted Neural Architecture Search. *ArXiv e-prints*. júl 2020. arXiv:2007.10396. Dostupné z: https://ui.adsabs.harvard.edu/abs/2020arXiv200710396L.

[24] Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K. et al. NSGA-Net: Neural Architecture Search Using Multi-Objective Genetic Algorithm. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2019, s. 419–427. GECCO '19. DOI: 10.1145/3321707.3321729. ISBN 9781450361118. Dostupné z: https://doi.org/10.1145/3321707.3321729.

[25] Masárová, M. *Genetické algoritmy*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2017. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/114512.

[26] McCulloch, W. S. a Pitts, W. A Logical Calculus of the Idea Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*. 1943, zv. 5, s. 115–133. DOI: 10.1007/bf02478259.

[27] Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D. et al. Evolving Deep Neural Networks. In: Kozma, R., Alippi, C., Choe, Y. a Morabito, F. C., ed. *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Amsterdam: Elsevier, 2018. Dostupné z: http://nn.cs.utexas.edu/?miikkulainen:chapter18.

[28] O'Shea, K. a Nash, R. An Introduction to Convolutional Neural Networks. *ArXiv e-prints*. november 2015. arXiv:1511.08458.

[29] Pelikan, M., Goldberg, D. E. a Cantú Paz, E. BOA: The Bayesian Optimization Algorithm. In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, s. 525–532. GECCO'99. ISBN 1558606114.

[30] Piňos, M. *Evoluční návrh konvolučních neuronových sítí*. Brno, CZ, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/21369/.

[31] Real, E., Aggarwal, A., Huang, Y. a Le, Q. V. Regularized Evolution for Image Classifier Architecture Search. *Proceedings of the AAAI Conference on Artificial Intelligence*. Jul. 2019, zv. 33, č. 01, s. 4780–4789. DOI: 10.1609/aaai.v33i01.33014780. Dostupné z: https://ojs.aaai.org/index.php/AAAI/article/view/4405.

[32] Rechenberg, I. *Evolutionsstrategie; Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Mit einem Nachwort von Manfred Eigen*. Frommann-Holzboog [Stuttgart-Bad Cannstatt], 1973. 170 p. s. ISBN 3772803733.

[33] Riedmiller, M. a Braun, H. *RPROP - A Fast Adaptive Learning Algorithm*. Proc. of ISCIS VII), Universitat, 1992.

[34] Rosenblatt, F. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*. 1958, s. 65–386.

[35] Sekanina, L. *Evoluční design, Fakulta informačných technológií, VUT Brno*. Dostupné z: https://www.fit.vutbr.cz/study/courses/BIN/private/prednasky/bin2020_p03.pdf.

[36] Sekanina, L. *Neuronové sítě, neuroevoluce a neuropočítače, Fakulta informačných technológií, VUT Brno*. Dostupné z: https://www.fit.vutbr.cz/study/courses/BIN/private/prednasky/bin2020_p09.pdf.

[37] Sharma, S. *Artificial Neural Network (ANN) in Machine Learning*. Dostupné z: https://www.datasciencecentral.com/profiles/blogs/artificial-neural-network-ann-in-machine-learning.

[38] Snaselova, P. a Zboril, F. Genetic Algorithm using Theory of Chaos. *Procedia computer science*. Elsevier B.V. 2015, zv. 51, C, s. 316–325. ISSN 1877-0509.

[39] Stanley, K., Clune, J., Lehman, J. a Miikkulainen, R. Designing neural networks through neuroevolution. *Nature Machine Intelligence*. Január 2019, zv. 1. DOI: 10.1038/s42256-018-0006-z.

[40] Stanley, K. O. a Miikkulainen, R. Evolving Neural Networks through Augmenting Topologies. *Evol. Comput.* Cambridge, MA, USA: MIT Press. jún 2002, zv. 10, č. 2, s. 99–127. DOI: 10.1162/106365602320169811. ISSN 1063-6560. Dostupné z: https://doi.org/10.1162/106365602320169811.

[41] Stein, B. van, Wang, H. a Bäck, T. *Neural Network Design: Learning from Neural Architecture Search*. 2020. ArXiv:2011.00521.

[42] Suganuma, M., Shirakawa, S. a Nagao, T. A Genetic Programming Approach to Designing Convolutional Neural Network Architectures. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, Júl 2018, s. 5369–5373. DOI: 10.24963/ijcai.2018/755. Dostupné z: https://doi.org/10.24963/ijcai.2018/755.

[43] Sze, V., Chen, Y., Yang, T.-J. a Emer, J. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE*. 2017, zv. 105, s. 2295–2329.

[44] Zbořil, F. V. *Acyklické a dopředné neuronové sítě. Algoritmus backpropagation, Fakulta informačných technológií, VUT Brno*. Dostupné z: https://www.fit.vutbr.cz/study/courses/SFC/private/20sfc_2.pdf.

[45] Zbořil, F. V. *Genetické algoritmy, Fakulta informačných technológií, VUT Brno*. Dostupné z: https://www.fit.vutbr.cz/study/courses/SFC/private/19sfc_7.pdf.

[46] Zbořil, F. V. *Neocognitron a konvoluční neuronové sítě, Fakulta informačných technológií, VUT Brno*. Dostupné z: https://www.fit.vutbr.cz/study/courses/SFC/private/20sfc_4.pdf.

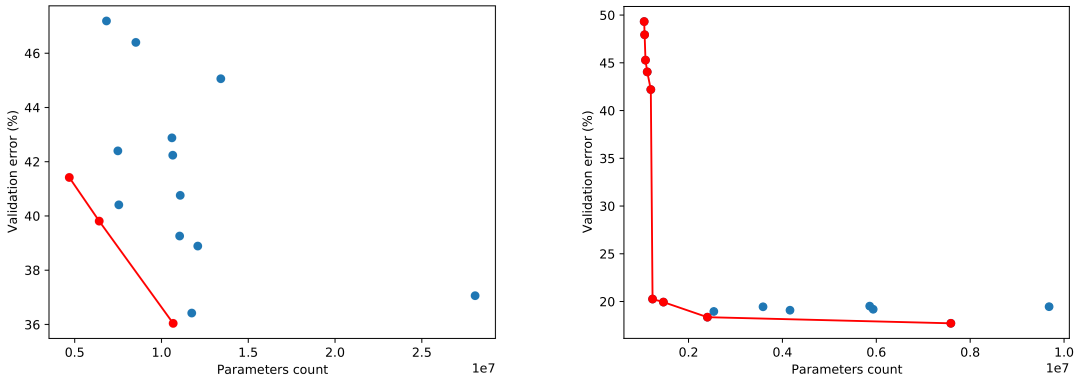
[47] Zeiler, M. D. ADADELTA: An Adaptive Learning Rate Method. *ArXiv e-prints*. december 2012. arXiv:1212.5701.

[48] Zoph, B., Vasudevan, V., Shlens, J. a Le, Q. V. Learning Transferable Architectures for Scalable Image Recognition. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, Jun 2018, s. 8697–8710. DOI: 10.1109/CVPR.2018.00907. Dostupné z: https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00907.

Príloha A

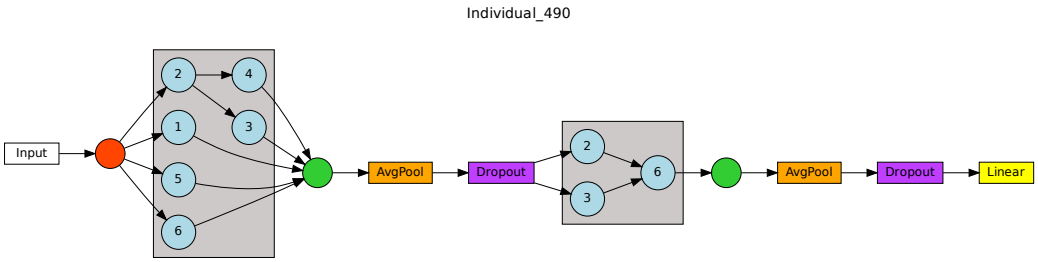
Vizualizácia experimentu 1

V tejto prílohe je zobrazený výsledok najlepšieho behu pre prvý experiment. Na obrázkoch A.1a a A.1b môžeme vidieť porovnanie Pareto množín po prvej generácii a po skončení výpočtu. Na obrázku A.2 môžeme vidieť vizualizáciu genotypu jedinca s najvyššou presnosťou.



(a) Pareto množina po 1. generácii (b) Pareto množina po výpočte

Obr. A.1: Porovnanie Pareto množín medzi generáciami

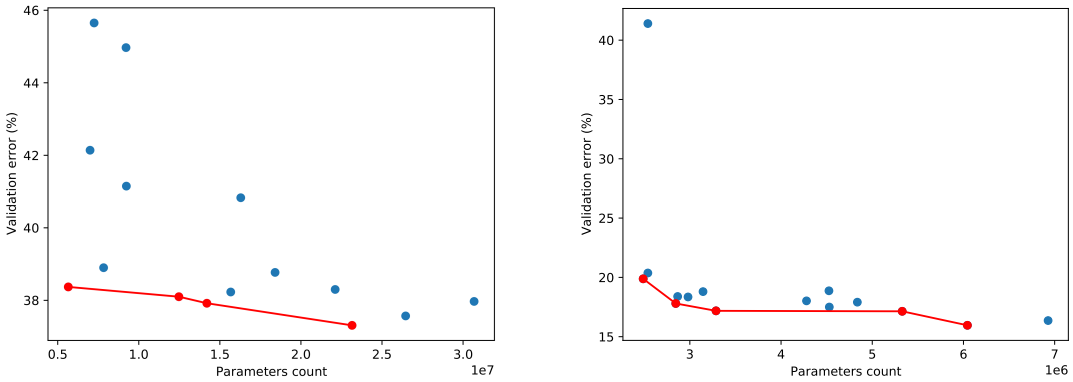


Obr. A.2: Genotyp jedinca s najvyššou presnosťou

Príloha B

Vizualizácia experimentu 2

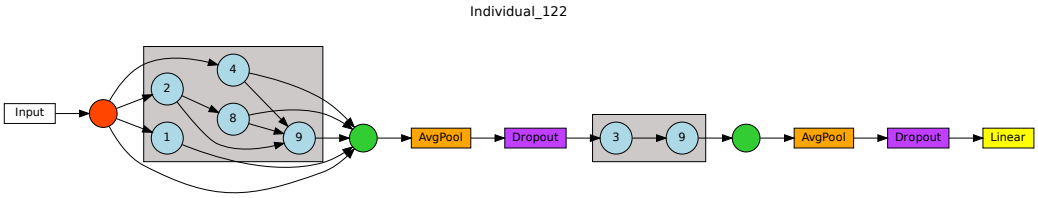
V tejto prílohe je zobrazený výsledok najlepšieho behu pre prvý experiment. Na obrázkoch B.1a a B.1b môžeme vidieť porovnanie Pareto množín po prvej generácii a po skončení výpočtu. Na obrázku B.2 môžeme vidieť vizualizáciu genotypu jedinca s najvyššou presnosťou.



(a) Pareto množina po 1. generácii

(b) Pareto množina po výpočte

Obr. B.1: Porovnanie Pareto množín medzi generáciami

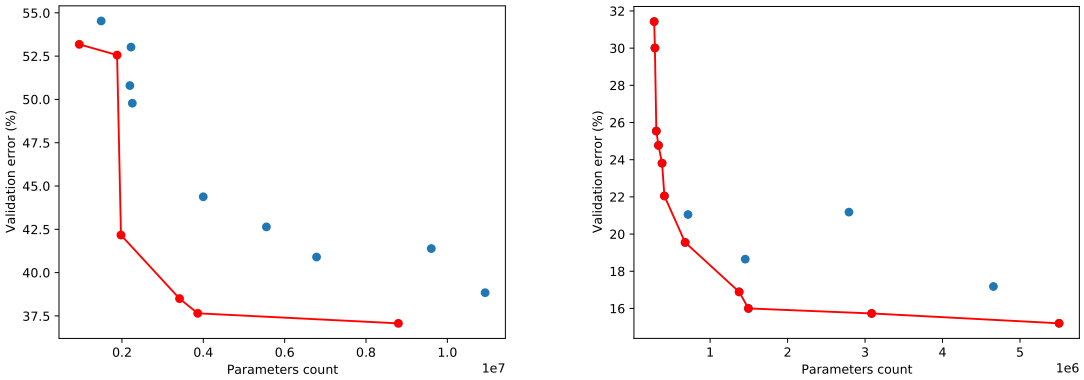


Obr. B.2: Genotyp jedinca s najvyššou presnosťou

Príloha C

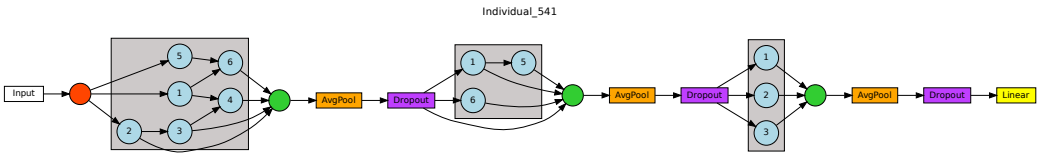
Vizualizácia experimentu 3

V tejto prílohe je zobrazený výsledok najlepšieho behu pre prvý experiment. Na obrázkoch C.1a a C.1b môžeme vidieť porovnanie Pareto množín po prvej generácii a po skončení výpočtu. Na obrázku C.2 môžeme vidieť vizualizáciu genotypu jedinca s najvyššou presnosťou.



(a) Pareto množina po 1. generácii (b) Pareto množina po výpočte

Obr. C.1: Porovnanie Pareto množín medzi generáciami



Obr. C.2: Genotyp jedinca s najvyššou presnosťou

Príloha D

Štruktúra súborov na pamäťovom médiu

Na pamäťovom médiu priloženému k tejto diplomovej práci sú uložené zdrojové kódy programu, výsledky jednotlivých experimentov, návod na prácu s programom a zdrojové kódy tejto práce v jazyku LaTeX. Zdrojové kódy sú uložené v zložke `src`, experimenty sú uložené v zložke `experiments`, návod na prácu s programom je uložený v súbore `README.md` a zdrojové kódy práce spolu s `.pdf` verziou sú uložené v zložke `thesis`.