VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ ALGORITMY V NÁVRHU KONVOLUČNÍCH NEURONOVÝCH SÍTÍ

EVOLUTIONARY ALGORITHMS IN CONVOLUTIONAL NEURAL NETWORK DESIGN

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. FILIP BADÁŇ**
AUTHOR

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

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Badáň Filip, Bc.**
Program: Informační technologie Obor: Bioinformatika a biocomputing
Název: **Evoluční algoritmy v návrhu konvolučních neuronových sítí**
Evolutionary Algorithms in Convolutional Neural Network Design
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s problematikou neuronových sítí, konvolučních neuronových sítí a s využitím evolučních algoritmů v návrhu neuronových sítí.
2. Seznamte se s knihovnami pro práci s konvolučními neuronovými sítěmi.
3. Navrhněte způsob využití evolučního algoritmu při návrhu konvoluční neuronové sítě.
4. Navržený způsob implementujte a ověřte jeho přínos ve zvolené případové studii.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Sekanina Lukáš, prof. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 26. října 2018

Abstrakt

Táto práca sa zaoberá možnosťami automatizácie návrhu neurónových sietí pomocou neuroevolúcie, t. j. využitia evolučných algoritmov pri konštruovaní umelých neurónových sietí alebo optimalizovaní ich parametrov. Cieľom práce je navrhnúť a implementovať evolučný algoritmus v podobe frameworku slúžiaceho na automatizáciu a optimalizáciu návrhu topológií konvolučných neurónových sietí. Účinnosť frameworku bola následne experimentálne vyhodnotená na úlohách klasifikácie obrazu na datasetoch MNIST a CIFAR10. Výsledky ukázali, že neuroevolúcia má potenciál hľadať úspešné a efektívne architektúry konvolučných neurónových sietí.

Abstract

This work focuses on automatization of neural network design via the so-called neuroevolution, which employs evolutionary algorithms to construct artificial neural networks or optimise their parameters. The goal of the project is to design and implement an evolutionary algorithm which can be used in the process of designing and optimizing topologies of convolutional neural networks. The effectiveness of the proposed framework was experimentally evaluated on tasks of image classification on datasets MNIST and CIFAR10 and compared with relevant solutions. The results showed that neuroevolution has a potential to successfully find accurate and effective convolutional neural network architectures.

Kľúčové slová

evolučné algoritmy, genetické algoritmy, konvolučné neurónové siete, neuroevolúcia

Keywords

evolutionary algorithms, genetic algorithms, convolutional neural network, neuroevolution

Citácia

BADÁŇ, Filip. *Evoluční algoritmy v návrhu konvolučních neuronových sítí.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Lukáš Sekanina, Ph.D.

Evoluční algoritmy v návrhu konvolučních neuronových sítí

Prehlásenie

Prehlasujem, že som túto prácu vypracoval samostatne pod vedením pána prof. Ing. Lukáša Sekaninu, Ph.D. Uviedol som všetky literárne zdroje a publikácie, z ktorých som čerpal.

.....
Filip Badáň
21. mája 2019

Poďakovanie

Rád by som sa poďakoval vedúcemu tejto práce prof. Ing. Lukášovi Sekaninovi, Ph.D. za trpezlivosť, ochotu a cenné odborné rady pri vypracovávaní tohto textu. Taktiež by som sa chcel poďakovať doc. Ing. Jiřímu Jarošovi, Ph.D. za ochotu pri poskytovaní výpočtového času na superpočítačoch Anselm a Salomon.

Táto práca bola podporená Ministerstvom školstva, mládeže a telovýchovy z podpory veľkých infraštruktúr pre výskum, experimentálny vývoj a inovácie v rámci projektu „IT4Innovations národní superpočítačové centrum - LM2015070“.

Obsah

Kapitola 1

Úvod

Strojové učenie (angl. *Machine Learning*) zaznamenalo v posledných rokoch obrovský úspech a našlo svoje využitie v mnohých technologických odvetviach. Medzi najrozšírenejšie algoritmy strojového učenia patria najmä systémy inšpirované fungovaním biologického mozgu, tzv. neurónové siete.

S rastúcim výpočtovým výkonom sa neurónové siete stali účinným prostriedkom na riešenie veľmi náročných a komplexných problémov. Výrazný podiel na tom má špeciálne oblasť tzv. hlbokých neurónových sietí (DNN, angl. *Deep Neural Networks*). Medzi najsilnejšie domény DNN patrí najmä spracovanie reči a prirodzeného jazyka či rozpoznávanie tvárí a objektov. V týchto oblastiach dokázali aplikácie DNN prekonať aj presnosť, ktorú dosahuje človek [16, 24]. Náročnosť a rôznorodosť problémov, ktoré neurónové siete riešia, sa však postupom času neustále zvyšuje, s čím priamo úmerne rastie aj komplexnosť neurónových sietí v podobe veľkého počtu vrstiev a celkových parametrov. Rovnako sa zvyšuje aj potreba nasadenia DNN v stále širšom spektre zariadení s veľkými nárokmi na efektivitu (hlavne na energetickú náročnosť) takýchto aplikácií. Práve zväčšujúci sa počet vrstiev a parametrov využívaných v najmodernejších hlbokých neurónových sieťach znamená aj čoraz väčšiu výzvu pre návrhárov modelov takýchto DNN.

Cieľom tejto práce je navrhnúť a implementovať framework slúžiaci na automatizáciu procesu návrhu hlbokých neurónových sietí, a to so zameraním na špeciálny druh DNN – konvolučné neurónové siete (CNN, angl. *Convolutional Neural Networks*). Výsledný framework sa preto sústredí na úlohy klasifikácie obrazu, ktoré patria medzi hlavný typ problémov riešených pomocou konvolučných neurónových sietí. Takisto je tento typ úlohy dobre spracovaný v odbornej literatúre a výskum v tejto oblasti je na vysokej úrovni, čo umožňuje kvalitné porovnanie výstupov tejto práce s najnovšími výsledkami. Na návrh štruktúry DNN využíva framework princípy tzv. evolučných algoritmov (EA, angl. *Evolutionary Algorithms*), ktoré sa v súčasnosti ukazujú ako veľmi účinné v riešení rôznych druhov optimalizačných problémov [7]. Tieto algoritmy sú inšpirované jednoduchým konceptom darwinovskej evolúcie, podľa ktorého tzv. *fitness* hodnota jedinca určuje, s akou pravdepodobnosťou tento jedinec prežije (t. j. dostane sa do ďalšej generácie) alebo sa dokáže rozmnožiť. Účelom frameworku je hľadať modely konvolučných neurónových sietí, ktoré dosiahnu čo najpresnejšie výsledky, ale zároveň minimalizovať ich výpočtové nároky.

Spojenie prístupu evolučných algoritmov a neurónových sietí sa v odbornej literatúre nazýva neuroevolúcia (angl. *neuroevolution*) a s narastajúcim množstvom dostupných výpočtových prostriedkov získava popularitu. Úspešné využitie evolučných algoritmov bolo v literatúre zaznamenané pri návrhu štruktúry a parametrov neurónových sietí [46, 9, 37], ale aj ako alternatívny prístup k trénovaniu DNN [47]. Vysoké výpočtové nároky, obzvlášť

pri návrhu štruktúry DNN, sú však stále najväčším problémom neuroevolučných aplikácií. Z toho dôvodu bolo jednou z najväčších výziev tejto práce navrhnúť evolučný algoritmus a kódovanie kandidátnych riešení tak, aby bol proces evolúcie vysoko efektívny a účinný.

Ďalšou náplňou tejto práce bolo aj zhodnotenie aktuálneho stavu výskumu v oblasti neuroevolúcie, analýza dostupných technológií a knižníc určených na prácu s konvolučnými neurónovými sieťami a implementácia navrhnutého evolučného algoritmu. Posledným krokom bolo experimentálne overenie jeho výstupov na dobre známych a preskúmaných úlohách klasifikácie obrazu. Ako datasety na vyhodnotenie navrhnutého frameworku boli zvolené voľne dostupné databázy MNIST a CIFAR10, ktoré sú v odbornej literatúre veľmi často využívané ako vzorové klasifikačné úlohy. Cieľom týchto experimentov bola aj snaha o dosiahnutie čo najlepších výsledkov, a to formou postupného vylepšovania a ladenia navrhnutého algoritmu pomocou empiricky nadobudnutých poznatkov. Súčasťou práce je aj porovnanie získaných výsledkov s najaktuálnejšími riešeniami rovnakých úloh zaznamenaných v odbornej literatúre a následné celkové zhodnotenie účinnosti implementovaného frameworku.

Členenie textu v tejto práci je nasledovné: kapitoly 2 a 3 zhrňujú základné teoretické znalosti z prostredia neurónových sietí a evolučných algoritmov, potrebné na správne pochopenie ďalšieho textu práce. Kapitola 4 sumarizuje aktuálne výsledky z dostupnej odbornej literatúry, ktorá sa týka neuroevolúcie, a popisuje postupný vývoj v tejto oblasti. Návrh všetkých aspektov frameworku, ktorý predstavuje hlavný výstup tejto práce, je popísaný v kapitole 5. V kapitole 6 sa nachádza konkrétny a podrobný popis implementácie frameworku a použitých technológií. Experimenty a zhodnotenie výsledkov spolu s porovnaním s výsledkami zistenými v odbornej literatúre sú obsiahnuté v kapitole 7. Táto kapitola obsahuje aj kompletný súhrn dosiahnutých výstupov s vyvodením vyplývajúcich záverov a návrhy ďalších možných pokračovaní tejto práce.

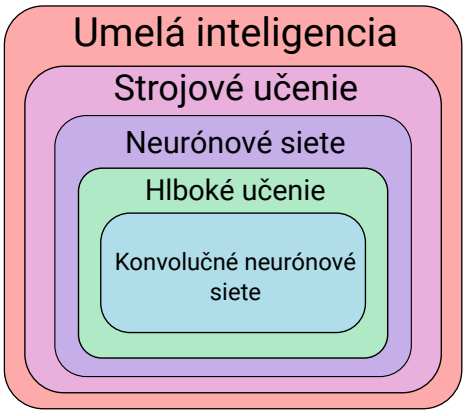
Kapitola 2

Umelé neurónové siete

Oblasť umelej inteligencie získava v súčasnej dobe veľké množstvo pozornosti nielen v akademickej sfére, ale aj v komerčnom sektore. Vo veľkej miere za to vďačí odvetviu strojového učenia, ktoré prinieslo do tejto oblasti revolučnú myšlienku: dať strojom (programom) schopnosť učiť sa na základe dostupných dát, pomocou procesu nazvaného tréning, a dospieť tak k riešeniu určitého problému.

Neurónové siete tvoria širokú časť rodiny algoritmov strojového učenia. Sú čiastočne inšpirované procesom učenia odohrávajúceho sa v biologickom mozgu a podobne ako mozog pozostávajú z poprepájaných elementov nazvaných **neuróny**. Tento základný princíp bude popísaný ďalej v podkapitole 2.1. Po spojení mnohých vrstiev takýchto neurónov vznikne komplexný model siete patriaci do rodiny takzvaného hlbokého učenia (angl. *deep learning*). Práve pomocou hlbokých neurónových sietí sa v súčasnosti riešia mnohé problémy z rôznych technologických oblastí.

Táto práca sa venuje najmä konkrétnej triede rodiny hlbokého učenia – **konvolučným neurónovým sieťam**. Tieto siete využívajú matematickú operáciu konvolúcie, ktorá redukuje celkovú výpočtovú komplexnosť siete a zároveň stále dosahuje veľmi dobré výsledky v riešení mnohých typov problémov. Vzťahy od konvolučných neurónových sietí až po celkovú umelú inteligenciu znázorňuje obrázok 2.1. Hlboké učenie a konvolučné siete, vrátane najaktuálnejších trendov v tejto oblasti, budú popísané v podkapitolách 2.2 a 2.2.1.

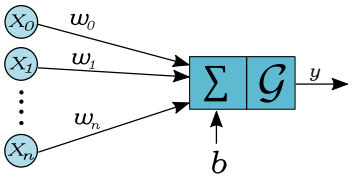


Obr. 2.1: Zobrazenie postavenia konvolučných neurónových sietí vo vzťahu k celkovej umelej inteligencii.

2.1 Stručné základy neurónových sietí

Ľudský mozog je stále považovaný za jeden z najefektívnejších výpočtových systémov súčasnosti. Idea spracovávania informácií spôsobom, akým sa to deje vo vnútri mozgu, sprevádza vývoj moderných počítačov už od počiatku. Základy neurónových sietí, tak ako sa vnímajú dnes, položili už Warren McCulloch a Walter Pitts v 40. rokoch minulého storočia, keď popísali model umelého neurónu [30]. Takisto ukázali, že siete umelých neurónov majú potenciál, teoreticky, vypočítať akúkoľvek logickú alebo aritmetickú operáciu. Ďalším výrazným krokom v tejto oblasti sa stal model *perceptronu* a základného učiaceho algoritmu v roku 1958 [38]. Postupom času však výskum neurónových sietí upadol, najmä kvôli nedostatku výpočtovej sily potrebnej na uskutočnenie experimentov. Svoje znovuzrodenie zažili až v 80. rokoch objavom algoritmu nazvaného *backpropagation* slúžiaceho na trénovanie viacvrstvových sietí perceptronov.

Základná schéma umelého neurónu je zobrazená na obrázku 2.2. Neurón v prvom kroku spočíta váženú sumu vstupných hodnôt X_0 až X_n, kde **váhy** sú reprezentované hodnotami w_0 až w_n. Hodnota b predstavuje prahovaciu hodnotu nazývanú tiež *bias*. Tento postup však nie je kompletný, pretože vzhľadom na linearitu operácie suma by výsledok kaskády prepojených neurónov bol stále len jednoduchá lineárna funkcia. Z toho dôvodu je ďalším prvkom v modeli umelého neurónu tzv. aktivačná funkcia $\mathcal{G}(x) : \mathbb{R} \to \mathbb{R}$, ktorá pridáva nelinearitu a mapuje výsledok do požadovaného intervalu na základe prahu. Súhrnne sa dá povedať, že princíp spočíva v aplikácii nelineárnej funkcie na váženú sumu vstupných hodnôt. Tento výpočet sa dá zapísať rovnicou 2.1. Váhy a biasy sa spoločne označujú ako **parametre siete**.



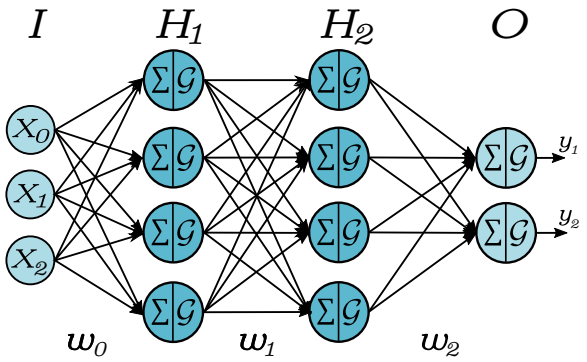
Obr. 2.2: Model umelého neurónu. Jedná sa o váženú sumu vstupov X_0 až X_n, ku ktorej sa ešte pričíta bias b. Následne je na túto sumu aplikovaná nelineárna funkcia $\mathcal{G}$.

$$y = \mathcal{G}(\sum_{0}^{n} X_i w_i + b) \tag{2.1}$$

Prepojením viacerých umelých neurónov dostávame viacvrstvovú neurónovú sieť schopnú riešiť zložitejšie problémy. Príklad takejto siete je zobrazený na obrázku 2.3.

2.1.1 Proces učenia

Proces učenia môže prebiehať buď ako *učenie s učiteľom* (angl. *supervised learning*), ako *učenie bez učiteľa* (angl. *unsupervised learning*), alebo ako *učenie posilňovaním* (angl. *reinforcement learning*). Vzhľadom na to, že zameraním tejto práce sú konvolučné neurónové siete a úlohy spojené s rozpoznávaním a klasifikáciou, bude ďalej popísané iba učenie s učiteľom, ktoré je pre tento typ úloh najvhodnejšie. Pre tento typ učenia je typické, že existuje tzv. označená tréningová sada dát, pričom je pri každej trénovacej vzorke dané, do ktorej klasifikačnej triedy patrí.



Obr. 2.3: Umelá neurónová sieť ako prepojenie viacerých umelých neurónov. I predstavuje vstupnú vrstvu siete, H_1 a H_2 sú tzv. skryté vrstvy (angl. *hidden layers*) a O predstavuje výstupnú vrstvu siete. Počet neurónov vo výstupnej vrstve odpovedá počtu rôznych výstupov celej úlohy, ktorú má neurónová sieť riešiť.

Učenie (alebo aj trénovanie) neurónovej siete spočíva v procese nastavovania hodnôt váh tak, aby tzv. stratová funkcia (angl. *loss function/cost function*), t. j. funkcia predstavujúca rozdiel medzi aktuálnym a očakávaným výstupom, bola čo najmenšia pre celú trénovaciu sadu dát. Váhy sú upravované optimalizačným procesom *gradientný zostup* (angl. *gradient descent*). Cieľom je teda minimalizovať hodnotu váh pomocou zápornej hodnoty gradientu stratovej funkcie, čo je vlastne vektor parciálnych derivácií straty L vzhľadom na jednotlivé váhy (parametre). Táto úprava je popísaná rovnicou 2.2, kde α je koeficient učenia (alebo aj krok učenia).

$$w_i = w_i - \alpha \frac{\delta L}{\delta w_i} \tag{2.2}$$

Na výpočet gradientov potrebných na úpravu jednotlivých váh sa v súčasnosti najviac používa **algoritmus spätného šírenia chyby** (angl. *backpropagation*). Tento algoritmus využíva reťazové pravidlo derivácie a postupuje smerom od výstupnej vrstvy siete až po jej vstupnú vrstvu, aby spočítal, ako každý parameter ovplyvňuje výslednú stratovú funkciu.

Keďže je nutné vypočítať gradienty postupne pre každý prvok trénovacej množiny dát, je najväčším problémom tohto algoritmu jeho vysoká výpočtová náročnosť. V rámci optimalizácie sa preto v súčasnosti často používa dávkové spracovanie tréningovej sady, pri ktorom sa sada rozdelí na malé množiny vzoriek – *dávok* (angl. *mini-batch*) a výsledná stratová funkcia sa počíta z celej takejto množiny. Modifikácia váh teda neprebieha pre každý prvok zvlášť, ale len toľkokrát, koľko existuje jednotlivých dávok. [14, 20, 13].

2.2 Hlboké neurónové siete

Medzi hlboké neurónové siete sa radia siete, ktoré pozostávajú z mnohých vrstiev a riešia komplexné problémy. Medzi výskumníkmi z tejto oblasti nepanuje zhoda v presnom počte vrstiev nutných na to, aby sa sieť považovala ze hlbokú [40], ale všeobecne sa pod týmto pojmom uvažujú siete s počtom skrytých vrstiev väčších než jedna [49]. V súčasnosti sa počet vrstiev v hlbokých neurónových sieťach pohybuje od piatich až do niekoľko tisíc. Architektúra takejto siete teda pozostáva z mnohých nelineárnych elementov, ktoré môžu reprezentovať aj extrémne zložité funkcie jej vstupov. Typickým príkladom takejto funkcie

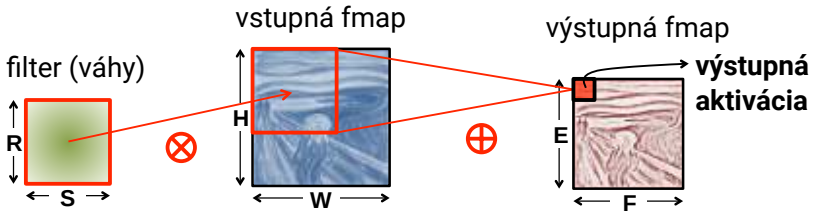
je schopnosť hlbokej architektúry spracovať veľké obrazové dáta a nachádzať v nich rôzne vysokoúrovňové príznaky.

Medzi dva najpoužívanejšie typy hlbokých neurónových sietí patria *rekurentné neurónové siete* (RNN, angl. *recurrent neural networks*) a *konvolučné neurónové siete* (CNN, angl. *convonutiolnal neural networks*). Táto práca sa primárne zameriava práve na konvolučné neurónové siete a tie budú ďalej popísané v nasledujúcej podkapitole.

2.2.1 Konvolučné neurónové siete

Teoretické poznatky pre túto podkapitolu sú čerpané z [14, 24, 49]. Hlavnou motiváciou pre vznik konvolučných neurónových sietí bolo výrazne zredukovať počet parametrov siete, ktorých počet môže byť v prípade DNN veľmi vysoký, a tým jej umožniť, aby bola hlbšia bez exponenciálneho nárastu výpočtových nárokov.

Základným prvkom architektúry CNN je **konvolučná vrstva**, v ktorej neuróny nie sú prepojené každý s každým, ako to bolo znázornené na obrázku 2.3, ale využíva sa v nich operácia konvolúcie. Táto operácia počíta váženú sumu pre každú výstupnú aktiváciu len z malého okolia vstupnej aktivácie, vďaka tomu výrazne redukuje výpočtové nároky výslednej siete. Priebeh konvolúcie v kontexte spracovania obrazu si môžeme predstaviť ako posúvanie malého okna hodnôt, označovaného ako konvolučné jadro alebo aj filter, cez celý vstupný obrázok. Pri každom posunutí tohto okna sa vynásobia prekrývajúce sa hodnoty a výsledky sa sčítajú (akumulujú) do jedinej výslednej hodnoty. Výstupom konvolučnej vrstvy je potom matica hodnôt v kontexte konvolučných neurónových sietí označovaná ako *mapa aktivácií* alebo aj *mapa príznakov* (angl. *feature map*). Filtre sú v ponímaní neurónových sietí súborom váh, čo znamená, že ich hodnota sa získava procesom učenia. Najčastejším algoritmom používaným na učenie v CNN je *backpropagation*. Priebeh konvolúcie v konvolučných neurónových sieťach je graficky znázornený na obrázku 2.4.



Obr. 2.4: Princíp konvolúcie v konvolučnej vrstve. Pohyblivé okno hodnôt, nazývané aj filter, sa posúva cez celý vstupný obrázok. V každej pozícii sú vynásobené a následne sčítané všetky prekrývajúce sa hodnoty. Výsledkom tohto sčítania je potom jeden prvok vo výslednej mape príznakov. Prevzaté z [49].

Cieľom konvolúcie v CNN je extrahovať lokálne príznaky z predchádzajúcich vrstiev a vytvoriť tak čoraz vyššiu abstrakciu vstupných dát. Na obrázku 2.5 je zobrazený celkový výpočet prebiehajúci v konvolučnej vrstve. Typicky sa v každej konvolučnej vrstve aplikuje niekoľko filtrov extrahujúcich rôzne príznaky. Vytvára sa tak sada máp príznakov, kde sa každá z nich nazýva *kanál* (angl. *channel*) a ich počet sa rovná počtu použitých filtrov v každej vrstve. Celý výsledný kanál, t. j. celá jedna mapa príznakov, zdieľa práve jeden filter. Tento princíp sa nazýva zdieľanie váh (angl. *weight sharing*). Je tiež dôležité zdôrazniť, že počet kanálov vstupnej vrstvy určuje aj veľkosť použitých filtrov. Pre predstavu, farebný (RGB) vstupný obrázok pozostáva z troch kanálov (každý pre jednu farbu) a v prvej konvolučnej vrstve je použitých šesť filtrov. Každý filter teda pozostáva z troch matíc

o veľkosti typicky 3×3 alebo 5×5. Počet výstupných kanálov bude teda pri aplikovaní šiestich filtrov takisto šesť.

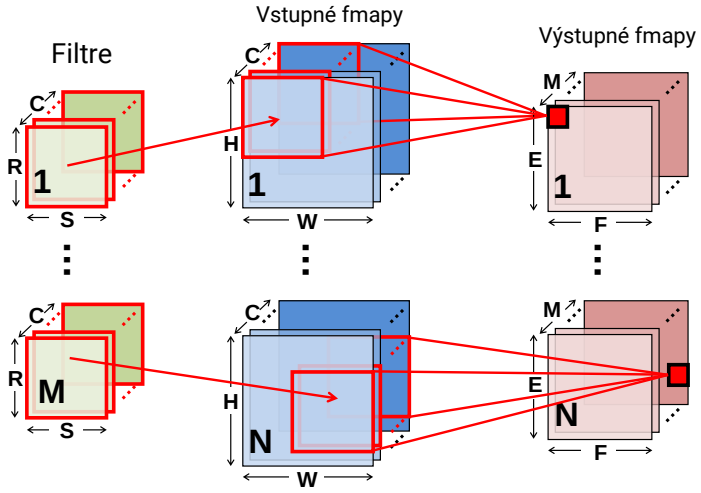
Ďalším prvkom v architektúre konvolučných neurónových sietí je **nelinearita**. Podobne ako u plne prepojených neurónových sietí popísaných vyššie sa na výsledok konvolúcie aplikuje aktivačná nelineárna funkcia. Najčastejšie používanou nelineárnou funkciou je ReLU (Rectified Linear Unit), ktorá je definovaná ako $y = max(0, x)$.

Kvôli optimalizácii rýchlosti trénovania a zvýšeniu presnosti sa v mnohých modeloch pridávajú ešte rôzne normalizačné prvky, ako napríklad *BN* (*Batch Normalization*) [18] alebo *LRN* (*Local Response Normalization*) [14].

Okrem konvolučných vrstiev sa v typických konvolučných neurónových sieťach využíva aj tzv. **pool vrstva**. Táto vrstva slúži na redukciu dimenzionality výstupných kanálov. Cieľom je zlúčiť sémanticky podobné príznaky a zároveň tým zredukovať celkovú výpočtovú náročnosť. Takáto redukcia sa nazýva aj *podvzorkovanie* (angl. *downsampling*). Operácie, ktorými sa podvzorkovanie realizuje, sú najčastejšie *priemer* alebo *maximum*. Operácia sa vykonáva na malej oblasti susedných prvkov a opakovane pre každú mapu príznakov. Veľkosť tejto oblasti sa nazýva *stride*.

Ako finálny prvok v architektúre CNN sa obvykle používa jedna alebo viac (< 3) plne prepojených vrstiev. Výsledný počet neurónov v poslednej plne prepojenej vrstve predstavuje počet možných výstupov riešeného problému.

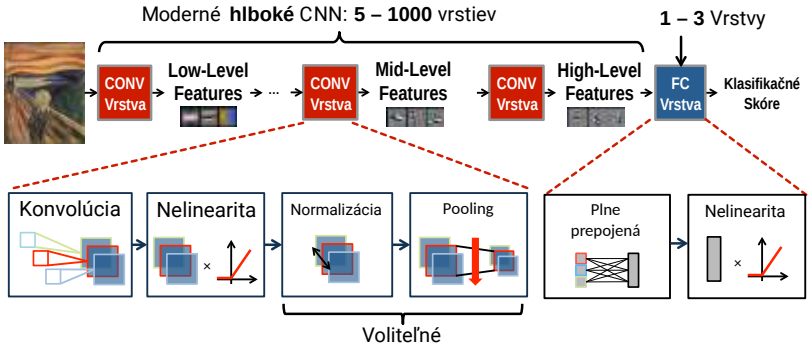
Typická vzorová architektúra konvolučnej neurónovej siete je zobrazená na obrázku 2.6.



Obr. 2.5: Viacrozmerná konvolúcia prebiehajúca v konvolučnej vrstve. **R** a **S** sú výška, respektíve šírka aplikovaných filtrov, **H** a **W** sú výška, respektíve šírka vstupných máp, **E** a **F** sú výška, respektíve šírka výstupných máp, **C** je počet vstupných kanálov, **M** je počet filtrov (a zároveň počet výstupných kanálov) a **N** je počet vzoriek v jednej dávke. Prevzaté z [14].

Populárne modely CNN

Princíp konvolučnej neurónovej siete bol navrhnutý už na začiatku 80. rokov minulého storočia [12], avšak tento koncept nebol dlho využívaný z dôvodu nedostatočnej výpočtovej kapacity. Prvý výrazný prielom v tejto oblasti nastal až v roku 1989 vďaka francúzskemu výskumníkovi menom Yann LeCun, ktorý úspešne využil algoritmus backpropagation v ob-



Obr. 2.6: Vzorová architektúra CNN. Vstupom siete je obrázok, ktorý má sieť za úlohu rozpoznať alebo klasifikovať. Sieť sa skladá z niekoľkých vrstiev vyhodnocujúcich operáciu konvolúcie nasledovanými nelineárnou aktiváciou. Cieľom je extrahovať abstraktné príznaky zo vstupného obrazu. V prípade potreby môžu aktivácie v konvolučných vrstvách nasledovať rôzne normalizačné prvky alebo vrstvy (*pooling* vrstvy), ktorých úlohou je podvzorkovať (zmenšiť rozmery generalizáciou lokálnych príznakov) výstupné mapy príznakov. Na konci architektúry sa typicky nachádza plne prepojená vrstva, ktorej výstupy predstavujú zároveň výsledné klasifikačné triedy. Prevzaté z [49].

lasti rozpoznávania ručne písaných číslic [25]. Tieto princípy sa stali základom pre všetky najznámejšie konvolučné neurónové siete.

- **LeNet** [26] (1998) Sieť architektúry nazvanej *LeNet* bola navrhnutá už v roku 1998 Yannom Lecunom a spol. Táto práca nadväzovala na jeho predošlú úspešnú aplikáciu algoritmu backpropagation na rozpoznávanie ručne písaných číslic a ešte zlepšila dosiahnuté výsledky. Jej schopnosť rozpoznávať bola demonštrovaná na databáze MNIST[1], kde táto architektúra dosiahla chybovosť 0,95%. Najznámejšia verzia tejto architektúry je *LeNet*. Sieť *LeNet-5* sa skladá zo siedmich vrstiev: dve konvolučné vrstvy nasledované podvzorkovacou vrstvou, následne posledná konvolučná vrstva a dve plne prepojené vrstvy. Ako finálna aktivácia sa používa funkcia *softmax*, ktorá slúži ako klasifikátor. Táto sieť sa stala základom pre všetky nasledujúce CNN.

- **AlexNet** [23] (2012) Architektúra siete *AlexNet* znamenala obrovský prielom v oblasti CNN. Jej úspešnosť bola prezentovaná na databáze *ImageNet*, ktorá sa neskôr stala základným porovnávajúcim benchmarkom pre väčšinu konvolučných sietí. Top-5[2] chybovosť tejto siete na úlohu klasifikácie pre túto databázu bola 15,3% a vďaka tomu vyhrala Large Scale Visual Recognition Challenge (ILSVRC) v roku 2012 ako prvá konvolučná neurónová sieť. Model siete pozostáva z piatich konvolučných a troch plne prepojených vrstiev. Táto sieť bola trénovaná na kartách GPU a to jej umožnilo spracovávať aj oveľa väčšie vstupné obrázky ako dokázali jej predchodcovia.

- **VGGNet** [44] (2014) Architektúra VGG priniesla do oblasti CNN ešte väčšiu hĺbku. Počet konvolučných vrstiev sa pre rôzne verzie tejto architektúry pohyboval od 16 do 19 vrstiev. Ako kompenzáciu výpočtovej náročnosti takejto hlbokej siete autori zmenšili veľkosť použitých filtrov z doteraz používaných 5x5 na 3x3. Architektúra

[1]MNIST – Modified National Institute of Standards and Technology, najpoužívanejšia voľne dostupná databáza ručne písaných číslic.

[2]Ako *Top-5* chybovosť sa označuje percentuálne skóre, pre ktoré správna klasifikačná trieda NEBOLA v piatich najpravdepodobnejších odhadoch daného modelu.

bola veľmi úspešná a v ILSVRC2014 získala v lokalizácii a klasifikácii prvé, respektíve druhé miesto s top-5 chybovosťou 25,3% pre lokalizáciu a 7,3% pre klasifikáciu.

- **GoogLeNet** [50] (2014) Ďalšou úspešnou architektúrou v ILSVRC2014 sa stala *GoogLeNet*. Jej zvláštnosťou je zavedenie modulu, ktorý spája sériu paralelných prepojení. Tieto moduly nazvali *inception* moduly a ich cieľom je kombinovať príznaky z rôzne veľkých filtrov a podvzorkovacích operácií. Modul pozostáva z konkatenácie výstupov 1x1, 3x3, 5x5 konvolúcií a pooling operácie. Zároveň sa autori snažili výrazne znížiť výpočtové nároky výraznou redukciou počtu parametrov. Toto dosiahli zavedením 1x1 *bottleneck* konvolúcií, ktoré zachovávajú rozmery vstupu, avšak redukujú hĺbku výsledných máp príznakov.

- **ResNet** [15] (2015) Veľký pokrok v klasifikácií pomocou CNN dosiahla aj architektúra *ResNet* (alebo aj Residual Net), ktorá v roku 2015 pokorila hranicu piatich percent top-5 chybovosti v databáze ImageNet v ILSVRC2015. Zároveň veľmi výrazným spôsobom zvýšila počet vrstiev oproti predchádzajúcim architektúram až na 152. Kľúčovým prvkom v tejto architektúre boli tzv. reziduálne prepojenia. Princíp týchto prepojení sa dá zapísať rovnicou 2.3, kde $\mathcal{H}(x)$ je výstup reziduálneho bloku, x je vstup a $\mathcal{F}(x)$ je výstup niekoľkých konvolučných vrstiev v tomto bloku.

$$\mathcal{H}(x) = \mathcal{F}(x) + x \tag{2.3}$$

  Výstup tohto bloku je kombinácia niekoľkých vrstiev konvolúcie a vstupu (*identita*). Motiváciou takéhoto prepojenia bolo vyriešenie problému, ktorý nastával vo veľmi hlbokých sieťach. V teórii by mala veľmi hlboká sieť dosahovať aspoň také výsledky ako ich plytkejšie verzie, ale v praxi sa ukázalo, že to neplatí. Dôvodom je, že optimalizačné metódy v procese učenia pri veľkom počte vrstiev a parametrov strácajú schopnosť efektívne modifikovať váhy aj na prvých vrstvách siete. Reziduálne prepojenie tento problém rieši, pretože sieť sa nemusí učiť celú funkciu $\mathcal{H}(x)$, ale len jej reziduálne mapovanie $\mathcal{F}(x) = \mathcal{H}(x) - x$. V prípadoch, že je teda výstupná funkcia bližšie k identite, je pre sieť ľahšie nájsť len daný rozdiel ako celú funkciu. V ostatných prípadoch sieť nie je nijako znevýhodnená oproti klasickým architektúram.

2.3 Knižnice na prácu s CNN

Vzhľadom na veľkú popularitu a úspešnosť neurónových sietí vo všetkých oblastiach nie je prekvapivé, že existuje mnoho voľne dostupných knižníc a frameworkov slúžiacich na prácu s nimi. V tejto podkapitole sú zhrnuté niektoré najznámejšie knižnice a popísané ich vlastnosti.

2.3.1 TensorFlow

Framework *TensorFlow* [1], vyvinutý spoločnosťou Google v roku 2013, je jeden z najpoužívanejších voľne dostupných framework na prácu so strojovým učením. Poskytuje rozhranie pre veľké množstvo jazykov ako C++, Python, R, Java a pod. Vďaka tomu je možné ho integrovať do veľkej časti vyvíjaných aplikácií využívajúcich algoritmy strojového učenia. Obsahuje všetky najaktuálnejšie a najmodernejšie techniky strojového učenia popísané v odbornej literatúre. Vďake veľkej popularite je pre *TensorFlow* dostupná aj veľká komunita užívateľov a vďaka tomu jeho vývoj stále napreduje.

2.3.2 Caffe

Ďalším veľmi populárnym a rozšíreným frameworkom na prácu s hlbokými neurónovými sieťami je *Caffe* [19]. Bol vyvinutý ako open-source projekt tímom Berkeley AI Research (BAIR) v roku 2014. Poskytuje rozhranie pre jazyky C, C++, Python a MATLAB. Popularitu si tento framework získal najmä pre jeho vysokú rýchlosť a efektivitu. Podporuje výpočet na CPU aj GPU a masívny paralelizmus. Úspech získal najmä v oblastiach obrazového rozpoznávania dát. V rámci frameworku *Caffe* existuje aj veľká databáza predtrénovaných neurónových sietí nazvaná *Caffe Model Zoo*. Nevýhodou frameworku *Caffe* je nedostatočná podpora rekurentných modelov.

2.3.3 Microsoft Cognitive Toolkit/CNTK

Framework *Microsoft Cognitive Toolkit* [42], predtým známy aj ako CNTK, bol vyvinutý v roku 2016 spoločnosťou Microsoft. Je dostupný ako open-source a podporuje rozhrania pre jazyky C++ a Python. Jeho hlavnými výhodami sú jednoduchá kombinovateľnosť viacerých typov hlbokých neurónových sietí a tiež vysoká možnosť paralelizácie naprieč viacerými výpočtovými servermi. Nevýhodou frameworku je nedostatočná podpora pre ARM architektúry, a je preto obmedzené jeho použitie v aplikáciách určených pre mobilné zariadenia.

2.3.4 TinyDNN

Framework *TinyDNN* [33] bol implementovaný v roku 2016 a je voľne dostupný pod licenciou BSD. Podporuje rozhranie do jazyka C++. Je naimplementovaný len v hlavičkových súboroch a neobsahuje takmer žiadne externé závislosti. Je preto veľmi prenositeľný a vďaka tomu je vhodný najmä pre embedded zariadenia. Podporuje tiež paralelný výpočet a akceleráciu na GPU. Jeho nevýhodou je pomerne malá komunita a absencia niektorých najaktuálnejších typov modelov hlbokých neurónových sietí.

Kapitola 3

Evolučné algoritmy

Pod pojmom evolučné algoritmy (EA, angl. *evolutionary algorithms*) sa v súčasnosti rozumie široká rodina heuristických algoritmov inšpirovaných procesom evolúcie odohrávajúcej sa v prírode. Tieto algoritmy vznikali vo väčšine prípadov nezávisle od seba, avšak mnohé princípy a postupy sú veľmi podobné, v niektorých prípadoch až rovnaké. Nie je prekvapením, že práve biologická evolúcia sa stala podkladom pre množstvo algoritmov, keďže práve evolúcia dokázala vytvoriť jeden z najefektívnejších „počítačov“ aký poznáme – ľudský mozog.

Aj keď myšlienka využitia princípov biologickej evolúcie pri riešení optimalizačných problémov sa začala objavovať v odbornej literatúre už od 50. rokov minulého storočia, za počiatok evolučného počítania sa vo všeobecnosti považuje až takmer súbežný vznik troch na sebe nezávislých smerov evolučných algoritmov v 60. a 70. rokoch [5]. V roku 1975 navrhol Holland prvú verziu genetického algoritmu (GA, angl. *genethic algorithm*) [17], kým v USA Fogel a spol. položili základy evolučnému programovaniu (EP, angl. *evolutinary programming*) [11] a nemeckí výskumníci Rechenberg a Schwefel predstavili metódu nazvanú evolučne stratégie (ES, angl. *evolution strategies*) [36, 41]. Koncom 80. rokov pribudol štvrtý základný smer EA a to genetické programovanie (GP, angl. *genetic programming*) preslávené najmä Kozom [21].

Evolučné algoritmy patria do množiny stochastických prehľadávacích algoritmov. Keďže sa vo svojom výpočte opierajú o množstvo náhodných faktorov, je matematicky veľmi náročné dokázať, za akých podmienok budú konvergovať ku globálnemu optimu (riešeniu problému) [8]. Napriek tomu sa však v praxi ukazuje, že evolučné algoritmy dokážu riešiť problémy v širokej škále rôznych oblastí, ako napríklad evolučný návrh hardvéru [43], spracovanie obrázkov, astronómia, robotika [6, 29] ale aj mnohých ďalších. Všeobecne sa evolučné algoritmy používajú najmä tam, kde je problém dostatočne zložitý a klasické matematické optimalizačné metódy sú tým pádom veľmi neefektívne.

Všetky evolučné algoritmy vychádzajú z jedného hlavného princípu evolúcie – *prežitie „najlepších“* (angl. *survival of the fittest*). Tento princíp spočíva v tom, že v istom prostredí s obmedzenými zdrojmi, zvyšujú jedinci, ktorí sú na dané prostredie najlepšie adaptovaní (majú najväčšiu *fitness*[1]), svoju šancu sa rozmnožiť a preniesť tak svoje gény do ďalších generácií (t. j. prírodná selekcia). V kontexte EA sú jedincami kandidátne riešenia problému, ktorý je aktuálne predmetom evolúcie a za najviac „fit“ riešenie sa považuje to, ktoré dokáže daný problém riešiť optimálne. V odbornej literatúre sa kandidátne riešenia označujú aj

[1] Z pohľadu darwinovskej evolúcie byť najviac *fit* neznamená nutne, že jedinci musia byť fyzicky najsilnejší, najmúdrejší alebo najrýchlejší, ale že dokážu obmedzené zdroje v danom prostredí čerpať najefektívnejšie.

ako **fenotypy**. Na vyhodnotenie kvality kandidátneho riešenia sa používa tzv. **fitness funkcia**, ktorá dokáže túto kvalitu zmerať. Merateľnosť kvality riešení pre konkrétny problém je teda aj jedným z hlavných predpokladov nasadenia akéhokoľvek evolučného algoritmu.

Ďalej v tejto kapitole budú popísané základné prvky spoločné pre všetky typy EA, avšak vzhľadom na to, že praktická časť tejto práce vychádza z genetických algoritmov, bude kladený dôraz práve na prvky, ktoré sa využívajú v tejto verzii EA. Následne budú popísané najznámejšie a najpoužívanejšie verzie EA a budú vysvetlené hlavné rozdiely medzi nimi.

3.1 Základné princípy a komponenty EA

Napriek tomu, že väčšina typov evolučných algoritmov vznikala takmer súčasne a nezávisle na sebe, všetky zdieľajú rovnaké základy. Dôvodom je, že hlavná podstata evolúcie je principiálne jednoduchá, ale zároveň stále veľmi účinná. Algoritmus 1 popisuje základný priebeh všeobecného evolučného algoritmu. V zásade ide o jednoduchý iteratívny proces, kde sa v každom kroku aplikujú isté zmeny (pomocou tzv. **genetických operátorov**) na populáciu jedincov. V nasledujúcich podkapitolách budú vysvetlené jednotlivé kroky a prvky tohto algoritmu. Teoretické poznatky popísané v tejto podkapitole boli prevzaté z [7, 10, 35][2].

Algoritmus 1 Pseudokód všeobecného evolučného algoritmu. Vytvorené na základe [7].

```
1:  procedure EA
2:      inicializácia populácie EA
3:      evaluácia každého jedinca z populácie
4:      while splnená podmienka ukončenia do
5:          výber rodičov
6:          kríženie / mutácia
7:          evaluácia nových jedincov
8:          výber jedincov pre novú generáciu
9:      end while
10: end procedure
```

3.1.1 Reprezentácia problému

Hlavným a prvotným problémom aplikácie akéhokoľvek EA je počítačová reprezentácia kandidátnych riešení problému tak, aby ich bolo možné efektívne evaluovať a prevádzať nad nimi evolučné operácie. Inými slovami sa pod reprezentáciou myslí mapovanie (alebo kódovanie) fenotypov na vhodnú dátovú reprezentáciu (reťazec bitov), s ktorým dokáže pracovať počítač. Takýto reťazec sa nazýva aj **genotyp** alebo **chromozóm**. Vhodná reprezentácia problému často rozhoduje o účinnosti a rýchlosti celého algoritmu, a preto je potrebné jej venovať dostatok času pri návrhu algoritmu. Vzhľadom na to, že v mnohých prípadoch nie je riešený problém dostatočne preskúmaný alebo je príliš komplexný, vybrať vhodnú reprezentáciu nie je ľahká úloha.

Najpoužívanejšie reprezentácie používané v EA sú nasledovné:

[2] Je nutné tiež dodať, že nižšie popísané techniky a metódy nie sú (a ani sa nesnažia byť) vyčerpávajúce. Pre viac informácií a podobnejších popisov odporúčam zmienenú literatúru.

- *Binárna reprezentácia* – chromozómy sú zložené z postupnosti bitov a genetické operátory nad nimi vykonávajú bitové operácie.
- *Celočíselná reprezentácia* – chromozómy sú zložené zo série celých čísel – evolučné operátory menia hodnoty týchto čísel jednoduchou aritmetikou.
- *Reprezentácia pomocou reálnych čísel* – chromozóm pozostáva z niekoľkých čísel s pohyblivou rádovou čiarkou. Evolučná zmena prebieha pomocou matematických operácií s reálnymi číslami.
- *Stromová reprezentácia* – chromozóm predstavuje hierarchickú stromovú štruktúru. Genetické operátory v tomto prípade manipulujú s uzlami stromu alebo s celými podstromami.
- *Reprezentácia špecifická pre riešený problém* – v mnohých prípadoch je chromozóm špecifický pre daný problém.

3.1.2 Populácia

Všetky „žijúce“ (aktívne) genotypy kandidátnych riešení sa združujú v populácii, ktorá sa časom mení a vyvíja. Veľkosť populácie je jeden zo základných parametrov každého EA a jeho veľkosť ovplyvňuje rýchlosť a efektívnosť konvergencie k optimálnemu riešeniu. Príliš veľká populácia môže byť až neúnosne náročná na výpočtové zdroje, zatiaľ čo príliš malá populácia neposkytuje dostatočnú diverzitu[3] potrebnú na nájdenie globálneho optima.

V súvislosti s populáciou sa okrem jej veľkosti rieši ešte aj otázka jej počiatočnej inicializácie. V drvivej väčšine prípadov sa považuje za dostatočné inicializovať populáciu náhodnými (ale dostatočne rozdielnymi) jedincami. V niektorých prípadoch však môže byť zvlášť užitočné inicializovať populáciu predpripravenými jedincami, o ktorých vieme, že budú dosahovať vyššiu fitness. Je však otázka, nakoľko to procesu evolúcie pomôže – vždy záleží výhradne na konkrétnom probléme.

3.1.3 Selekcia jedincov

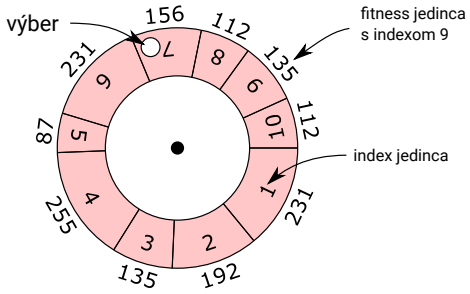
Aby populácia nestagnovala a jej kvalita sa neustále zlepšovala, zatiaľ čo jej veľkosť zostáva nemenná, je nutné zaviesť nástroje, ktoré budú populáciu udržiavať. Jedným z týchto nástrojov je aj selekcia jedincov, ktorí budú základom novej generácie, buď ako *rodičia* nových potomkov, alebo to budú „preživší“ zo súčasnej populácie.

Na výber *rodičov*, čiže chromozómov, na ktoré sa aplikuje operácia kríženia, aby vznikol nový jedinec – **potomok** – sa používa niekoľko techník:

Proporčná selekcia Výber jedincov je uskutočnený na základe ich fitness hodnoty v porovnaní s fitness hodnotami zvyšku populácie. Pravdepodobnosť, že jedinec bude zvolený ako rodič, sa teda dá vyjadriť ako podiel jeho fitness k súčtu hodnôt fitness všetkých ostatných aktívnych jedincov v populácii. Takýto postup sa zvyčajne implementuje algoritmom kolesa rulety (angl. *roulette wheel algorithm*). Princíp tohto algoritmu je zobrazený na obrázku 3.1.

Nevýhodou tohto prístupu je však veľká pravdepodobnosť, že gény jedného jedinca ovládnu celú populáciu (ak je jeho fitness príliš vysoká oproti iným riešeniam) alebo že

[3] Diverzita - počet rôznych jedincov v populácii v jednom okamihu



Obr. 3.1: Reprezentácia proporčného výberu ruletovým algoritmom. Pravdepodobnosť vybratia každého jedinca veľkostne úmerná hodnote jeho fitness v porovnaní s ostatnými jedincami v populácii. Prevzaté z [35].

ako následok stochastických efektov nebudú niektorí kvalitní jedinci vyberaní vôbec a populácia uviazne v lokálnom minime (ak je fitness hodnota všetkých jedincov veľmi podobná).

Výber založený na poradí Tento výber je veľmi podobný proporčnému výberu a v niektorých odborných literatúrach sa uvádza ako jeho varianta [35]. Hlavný rozdiel spočíva v tom, že pre pravdepodobnosť výberu nezáleží, aký veľký je rozdiel vo fitness medzi jednotlivými riešeniami, ale len na jeho celkovom poradí v rebríčku najlepších riešení. Týmto je možné zamedziť problémom popísaným pri proporčnom výbere.

Výber turnajom Výber turnajom spočíva v určení podmnožiny k jedincov, ktorí medzi sebou budú súťažiť. Jedinec s najvyššou fitness v rámci tejto podmnožiny bude vybratý ako rodič. Táto metóda zachováva dobrý kompromis medzi genetickou diverzitou a výberom tých najlepších riešení. Veľkosť hodnoty k závisí od konfigurácie algoritmu a býva predmetom experimentácie.

Okrem výberu rodičov je nutné vyberať aj jedincov, ktorí prežijú do ďalšej generácie, resp. iterácie evolučného algoritmu. Teoreticky je možné použiť akúkoľvek z vyššie uvedených metód, ale v praxi sa používajú zvyčajne jednoduchšie metódy. Tieto metódy sa zakladajú buď na *veku* jedinca (t. j. ako dlho sa už konkrétny jedinec nachádza v populácii), alebo na základe *hodnoty fitness*.

Pri metódach založených na veku je najdominantnejším spôsobom *generačná výmena*, čo znamená, že všetkých súčasných jedincov v generácii nahradia noví jedinci. V takomto prípade je počet nových jedincov rovný počtu jedincov v populácii. V prípadoch v ktorých je počet vygenerovaných potomkov menší, sa rodičia, ktorí budú nahradení, môžu vyberať na základe ich veku alebo náhodne. Takáto výmena sa nazýva aj *postupná výmena* (angl. *steady-state selection*).

V metódach, ktoré pracujú na základe fitness hodnoty jedincov, sa jedinci vyberajú z tých najlepších z množiny všetkých nových jedincov (v prípadoch, keď je vygenerovaných potomkov viac ako je veľkosť populácie), prípadne z množiny všetkých potomkov a rodičov spolu. Dôležitým pojmom pri selekcii na základe hodnoty fitness je aj **elitizmus**. Tento pojem označuje techniku, pri ktorej sa v rámci každej novej generácii uchová jeden alebo skupina najlepších jedincov, ktorí majú garantované prežitie do novej generácie. Týmto spôsobom je možné zaručiť, že neprídeme o najlepšie doteraz dosiahnuté riešenia. Elitizmus je možné ďalej kombinovať s inými spôsobmi selekcie.

3.1.4 Genetické operátory (kríženie a mutácia)

Aby sa mohla populácia vyvíjať a dospievať tak k stále lepším riešeniam, je nutné zaviesť operácie, ktoré ju budú istým spôsobom obmeňovať. Na tento účel slúžia genetické operátory, špeciálne mutácia a kríženie. Rôzne verzie evolučných algoritmov môžu jeden alebo druhý operátor vynechať, ak sa ukážu neefektívne.

Myšlienka, ktorá stojí za operátorom **kríženie**, spočíva v predpoklade, že existuje veľká pravdepodobnosť, že vygenerovaný potomok bude ťažiť z genetickej kombinácie dvoch kvalitných kandidátnych riešení, ktoré sú reprezentované jeho rodičmi. Inými slovami je šanca, že nové riešenie bude kombináciou tých častí rodičov, ktoré sa podieľajú na ich vyššej fitness hodnote, a tým pádom bude toto nové riešenie dosahovať ešte vyššiu fitness. Toto však nie je zaručené, a preto je pravdepodobnosť, s akou kríženie nastane, takmer vždy predmetom empirických experimentov. Proces kríženia veľmi úzko súvisí s genetickou reprezentáciou problému, a nie je preto možné zostaviť postup, ktorý bude univerzálny pre všetky typy problémov. Rodičia sú vyberaní buď náhodne, alebo pomocou techník selekcie popísaných vyššie.

Princíp **mutácie** spočíva v náhodnej modifikácii niektorých častí chromozómu s cieľom dosiahnuť väčšiu diverzitu v populácii a prehľadávať tak čo najširšiu časť stavového priestoru riešení. Mutácia tak napomáha, aby populácia neuviazla v lokálnom minime. Je však dôležité zachovať pravdepodobnosť mutácie na rozumných hodnotách, aby sa neznehodnocovali najkvalitnejšie riešenia. Podobne ako u kríženia aj u mutácie je postup a efektívna pravdepodobnosť úzko spätá s riešeným problémom. Pri každom probléme je vhodné, aby bola vyhodnotená sila efektu mutácie a na základe toho adekvátne nastavené mutačné parametre ako pravdepodobnosť a rozsah aplikácie. Jedinci, ktorí sa stanú predmetom mutácie, sú takmer výhradne vyberaní náhodným procesom, pretože je nemožné zistiť, ktoré zmeny povedú k lepším výsledkom.

3.1.5 Ukončovacia podmienka

Posledným prvkom evolučného algoritmu je ukončovacia podmienka. Pri väčšine problémov nie je jednoduché rozhodnúť, kedy je nájdené riešenie dostatočné kvalitné a či je šanca získať riešenia, ktoré budú ešte lepšie. Typicky sa ako ukončovacia podmienka používajú nasledovné možnosti:

1. doba celkového výpočtu,
2. počet vykonaných cyklov v rámci EA,
3. obdobie, po aké sa už fitness hodnota viac nezlepšuje,
4. diverzita populácie už nie je dostatočná.

V prípadoch, keď poznáme maximálnu hodnotu fitness pre daný problém (globálne optimum), môžeme túto hodnotu zapracovať do ukončovacej podmienky. Avšak vzhľadom na to, že nie je zaručené nájdenie takéhoto riešenia, nikdy by táto hodnota nemala byť v rámci ukončovacej podmienky použitá ako jediná.

3.2 Varianty evolučných algoritmov

V tejto podkapitole sú popísané štyri základné a najpoužívanejšie varianty evolučných algoritmov. Napriek tomu, že princíp je veľmi podobný, každý z nich používa rôzne implemen-

tačné detaily a rôzne techniky z tých popísaných vyššie. Je však nutné dodať, že praktické aplikácie nemusia striktne dodržiavať postupy z jedného variantu a často sú praktické aplikácie výsledkom kombinácie prístupov a techník zozbieraných z viacerých verzií.

Medzi najrozšírenejšie evolučné algoritmy vôbec patria jednoznačne **Genetické algoritmy (GA)**. Základný genetický algoritmus je veľmi jednoduchý na implementáciu, a to je aj dôvod, prečo je taký rozšírený. Typicky využíva binárnu reprezentáciu, generačnú výmenu a kladie veľký dôraz na kríženie. Na výber rodičov zvyčajne využíva proporčnú selekciu pomocou ruletového algoritmu, ale v poslednej dobe je častý aj výber turnajom alebo výber založený na poradí. Mutácia máva malú pravdepodobnosť a prebieha ako náhodné invertovanie bitov na rôznych pozíciach chromozómu. [7, 4]

Evolučné stratégie (ES) využívajú genotypy zakódované ako vektor reálnych čísel. Na rozdiel od GA však kladú hlavný dôraz na mutáciu, ktorá prebieha ako pripočítanie náhodného čísla (generované Gaussovým rozložením) ku každému prvku celého chromozómu. Dôležitým aspektom však je postupná zmena sily mutácie pomocou rozptylu použitého ako parameter pre Gaussovo rozloženie. V kontexte ES sa rozptylu hovorí aj mutačný krok. Spôsob, akým sa tento parameter mení, môže byť rôzny. Typicky sa používa známe pravidlo *1/5*, zavedené Rachenbergom [36], ktoré spočíva v zmene sily mutácie zvýšením alebo znížením na základe počtu úspešných mutácií. Ďalšou možnosťou môže byť aj tzv. *samoadaptácia* (angl. *self-adaptation*), t. j. zakódovanie mutačného kroku priamo do chromozómu, a teda podriadenie ho evolučným zmenám. Tento prístup sa stal úspešným a to nielen pri reprezentácii reálnymi číslami, ale aj pri binárnej a celočíselnej reprezentácii [3]. Evolučné stratégie využívajú aj kríženie podobným spôsobom ako v GA, avšak oproti GA je možné, že potomok má aj viac ako dvoch rodičov, ktorí sú vyberaní náhodne. [4]

Veľmi blízke ES je **evolučné programovanie (EP)**. Podobne ako EA aj EP využíva reprezentáciu pomocou reálnych čísel a adaptívnu mutáciu. Hlavným rozdielom je nevyužívanie kríženia a výber preživších pomocou turnaja. [7]

Jedným z najmladších variantov EA je **genetické programovanie (GP)**. Tento typ EA využíva stromovú reprezentáciu, a teda mutácia aj kríženie sú operácie nad stromami. Výsledkom evolučného procesu teda nemusí byť len sada vstupov, ale aj celý model. Z tohto dôvodu aj situácie, v ktorých je výhodné GP nasadiť, sú rôzne. Často sa využíva napríklad ako variant strojového učenia alebo na evolučný návrh počítačových programov. [7]

Kapitola 4

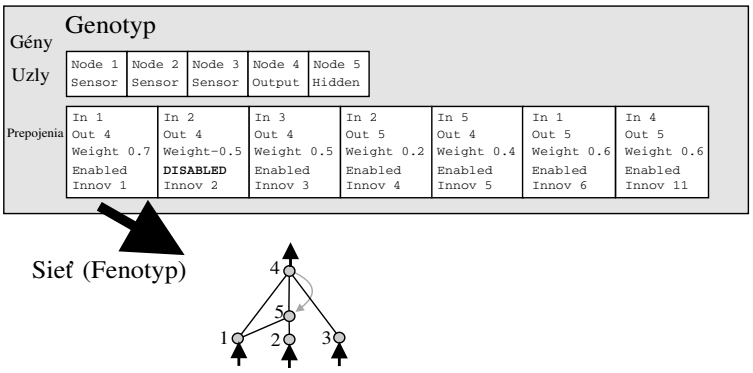
Neuroevolúcia

Biologická evolúcia dosiahla svoj významný úspech, keď sa jej podarilo vyvinúť jedno z najvýznamnejších a najvýkonnejších výpočtových „zariadení“, akým ľudský mozog nepochybne je. Nie je preto ťažké si predstaviť využitie algoritmov, ktoré sú inšpirované procesmi odohrávajúcimi sa pri biologickej evolúcii na návrh neurónových sietí, ktoré zase vychádzajú práve z fungovania biologického mozgu. Proces biologickej evolúcie však trval milióny rokov a takisto aj v praktických aplikáciách je najväčším problémom výpočtová náročnosť. Mnohé najaktuálnejšie hlboké neurónové siete sú samy o sebe výpočtovo náročné a efektívna evolúcia niekoľko desiatok až tisícov takýchto modelov predstavuje najväčšiu výzvu v oblasti neuroevolúcie.

Počiatky využitia EA pre návrh neurónových sietí siahajú až do konca 80. rokov [32]. V odbornej literatúre z oblasti neuroevolúcie sa vo vzťahu k umelým neurónovým sieťam podarilo evolučné algoritmy úspešne aplikovať na rôznych úrovniach. Jednou z prvých a najjednoduchších aplikácii je optimalizácia parametrov neurónových sietí zvyčajne využitím ES ako doplnku či alternatíve ku algoritmu backpropagation [34, 39, 47].

Z pohľadu tejto práce je však zaujímavé najmä využitie EA na návrh celých topológií neurónových sietí. Priekopníkom v tomto smere sa stal výskumník Kenneth O. Stanley s jeho algoritmom **NEAT** (NeuroEvolution of Augmenting Topologies) z roku 2002 [46]. Táto práca sa stala základom pre mnohé ďalšie práce z oblasti evolučného návrhu topológií neurónových sietí. Genotyp v algoritme NEAT je lineárna reprezentácia prepojení siete. Každý genotyp v sebe obsahuje dva typy génov: gény predstavujúce uzly (neuróny) v sieti a gény predstavujúce jednotlivé prepojenia medzi týmito uzlami vrátane ich váh. Na základe týchto informácií je potom možné jednoducho zostrojiť fenotyp, ako je to zobrazené na obrázku 4.1. Toto kódovanie sa nazýva *priame kódovanie*, čo znamená, že v každom genotype sú zakódované všetky konkrétne prepojenia v sieti a je možné na jeho základe zostrojiť fenotyp (výslednú sieť). Algoritmus NEAT využíva v procese evolúcie dva druhy mutácií (pridaj uzol/pridaj prepojenie) a kríženie.

Autori NEATu tiež upozorňujú na problém, že zmena topológie siete zo začiatku výrazne znižuje jej fitness oproti menším sieťam, pretože trvá istý čas, kým sa nové váhy stihnú optimalizovať. Týmto sú nové siete znevýhodnené a zabraňuje to ďalším inováciám. Ako riešenie autori navrhujú zaraďovanie jedincov s podobnou architektúrou do jedného druhu, kde celý druh zdieľa fitness hodnotu, ktorá je stále nižšia s čoraz väčším počtom jedincov patriacich do daného druhu. Toto umožňuje inovatívnejším riešeniam získať čas na optimalizáciu svojich váh a konkurovať tak ostatným architektúram. Podobnosť rôznych architektúr sa v algoritme NEAT počíta pomocou tzv. inovačného čísla, ktoré sa inkrementačne priraďuje každému novému prepojeniu nachádzajúcemu sa v populácii. Na základe



Obr. 4.1: Príklad mapovania genotypu na fenotyp v algoritme NEAT. Genotyp obsahuje tri vstupné (input), jeden skrytý (hidden) a jeden výstupný (output) uzol. Ďalej obsahuje 7 prepojení medzi týmito uzlami s informáciami o týchto prepojeniach, ako vstup a výstup (In, Out), váha (weight), inovačné číslo (innov) a či je prepojenie aktívne (enabled) alebo neaktívne (disabled). Prevzaté z [46].

týchto hodnôt je potom možné dohľadať historické korene každého genotypu a určiť tak podobnosť dvoch rôznych jedincov.

Problémom priameho kódovania, a teda aj celého algoritmu NEAT, je nedostatočná škálovateľnosť. Napriek tomu, že NEAT dosahuje dobré výsledky na malých architektúrach, pri komplexnejších problémoch algoritmus zlyháva najmä z pohľadu výpočtovej náročnosti. Jedným z riešení, ako sa vyrovnať s čoraz väčšou hĺbkou siete potrebnej na riešenie praktických problémov, je využitie *nepriameho kódovania*. Na rozdiel od priameho kódovania sa do genotypu nekóduje priamo štruktúra siete, ale len akýsi návod, resp. proces, na základe ktorého je potom zostrojený konkrétny fenotyp (architektúra siete) napríklad pomocou gramatických pravidiel [28].

Týchto problémov si bol vedomý aj Stanley a preto v roku 2009 navrhol vlastný spôsob nepriameho kódovania a to pomocou CPPN (Compositional Pattern Producing Networks) a využil ho v algoritme nazvanom **HyperNEAT** (Hypercube-based NeuroEvolution of Augmenting Topologies) [45]. CPPN sa dá chápať ako komplexná funkcia zložená z niekoľkých jednoduchších funkcií, ktorá má ako parametre súradnice dvoch bodov (uzlov, ktoré chceme prepojiť) a jej výstupom je „sila“ (váha), s akou sú tieto dva uzly prepojené. Pokiaľ túto funkciu zavoláme pre každé dva body z vopred definovanej sady bodov rozmiestnených v priestore, dostaneme výslednú schému prepojení a váh medzi týmito bodmi. V algoritme HyperNEAT sú práve CPPN predmetom evolúcie, ktorá v princípoch vychádza z algoritmu NEAT. Tento prístup už netrpí problémami popísanými pri priamom kódovaní a môže potenciálne navrhovať aj pomerne zložité topológie neurónových sietí.

Iný prístup pre riešenie obmedzení algoritmu NEAT zvolili Miikkulainen a spol., keď v roku 2017 navrhli algoritmy **DeepNEAT** a **CoDeepNEAT** [31]. Základná a principiálne jednoduchá idea stojaca za algoritmom DeepNEAT je, že genotyp v sebe nekóduje priamo jednotlivé neuróny, ale celé vrstvy siete. Predmetom evolúcie sa tak stávajú prepojenia medzi týmito vrstvami a ich parametre, ktoré sú takisto zakódované v genotype. Cieľom algoritmu CoDeepNEAT bolo naviazať na DeepNEAT a zaviesť do návrhu ešte aj vysokú modularitu, z ktorej ťažia mnohé manuálne navrhované hlboké neurónové siete. Tento algoritmu už dokázal generovať veľmi hlboké neurónové siete schopné konkurovať aj najmodernejším architektúram.

4.1 Neuroevolúcia v konvolučných neurónových sieťach

Vzhľadom na to, že táto práca sa sústredí primárne na evolučný návrh CNN a úlohám spracovania vizuálnych dát, budú v tejto podkapitole popísané niektoré najaktuálnejšie a najúspešnejšie výsledky dosiahnuté v tejto oblasti.

DPPN V roku 2016 výskumníci z tímu Google DeepMind navrhli **DPPN** (Differentiable Pattern Producing Networks) vychádzajúce z CPPN popísanými pri algoritme HyperNEAT [9]. Na rozdiel od CPPN, váhy v tejto práci nepodliehajú evolúcii, ale sú predmetom učenia. Výsledky tejto práce ukazujú, že kombinácia učenia založeného na gradientnom zostupe a evolúcie architektúry siete, môže byť veľmi účinným spôsobom, ako pristúpiť k neuroevolučným aplikáciám. Riešenie testovali na datasete MNIST na úlohe rekonštrukcie obrazu a výsledky ukázali veľkú efektívnosť takto navrhnutých modelov.

Large-scale evolution Vlastnú verziu evolučného algoritmu prispôsobenú špecificky na konvolučné neurónové siete navrhli v roku 2017 Real a spol. [37]. Podobne ako DeepNEAT využili kódovanie celých vrstiev namiesto jednotlivých neurónov a celé architektúry uložili vo forme grafu do štruktúry, ktorú nazvali DNA[1]. Navrhli tiež sadu mutácií špecifickú pre CNN architektúry, zatiaľ čo experimenty s krížením autorom nepriniesli výrazné zlepšenie, preto sa vo finálnej verzii nepoužíva. Aj v tejto práci sa autori rozhodli ťažiť z kombinácie evolučných algoritmov a klasického učenia. Váhy v jednotlivých vrstvách sú dedené z rodiča na potomka a ich optimalizácia prebieha klasickým učiacim algoritmom založeným na zostupe po gradiente. Algoritmus bol vyvíjaný s ohľadom na masívnu paralelizáciu s cieľom výrazne urýchliť celý evolučný proces. Táto práca bola pravdepodobne prvá, kde boli automaticky navrhnuté modely porovnané na datasetoch CIFAR-10 a CIFAR-100, ktoré sa považujú za benchmarkové datasety aj pre najnovšie CNN architektúry a dosiahli na nich výsledky porovnateľné s najlepšími zaznamenanými v literatúre.

CNN-GA Úspech práce Large-scale evolution dokonca predčili Sun a spol. so svojím algoritmom **CNN-GA** z roku 2018 [48], ktorý sa podľa mojich vedomostí aktuálne dá považovať za najaktuálnejšie v tejto oblasti. Autori využili svoje poznatky z manuálneho návrhu CNN a spojili to so svojou verziou GA. Genotyp je zakódovaný ako séria základných blokov, ktoré predstavujú vrstvy. V algoritme využívajú celkovo dva typy vrstiev a to *skip* vrstvu a *pool* vrstvu. Skip vrstva je zložená z dvoch konvolučných vrstiev a jedného reziduálneho prepojenia, ktoré sa ukázalo ako veľmi účinné pri hlbokých konvolučných sieťach. Pool vrstva funguje klasicky, ako je známe z moderných CNN modelov. Ako genetické operátory autori navrhli sadu mutácií a vlastný operátor kríženia, a tieto operátory využívajú v evolučnom procese spôsobom typickým pre GA. Pri evaluácii jedincov je každý jedinec najprv natrénovaný pomocou klasického backpropagation algoritmu a následne je vyhodnotená jeho fitness. K tomu využívajú GPU karty a špeciálnu štruktúru nazvanú *Cache*, do ktorej ukladajú identifikátor a fitness už vyhodnotených jedincov a tieto informácie využívajú k tomu, aby nebolo nutné rovnakých jedincov vyhodnocovať viac krát (ak prežijú do ďalšej generácie). Toto výrazne optimalizuje celý proces a umožňuje to autorom hľadať komplexnejšie riešenia. Výsledky CNN-GA autori porovnali s najnovšími CNN architektúrami a takisto aj s inými metódami na automatizovaný návrh CNN na datasetoch

[1] DNA v tomto kontexte, môže byť chápané ako genotyp, ale autori striktne nepoužívajú terminológiu všeobecne zaužívanú v EA.

CIFAR-10 a CIFAR-100. Výsledky, ktoré dosiahli, sa dajú považovať za výrazný úspech pre túto oblasť. Architektúry navrhnuté algoritmom CNN-GA dosahovali nie len veľmi vysokú presnosť, ale boli aj výrazne lepšie optimalizované z pohľadu počtu parametrov ako konkurenčné modely.

Tieto, ale aj ďalšie práce, sa dajú považovať za veľký úspech v oblasti neuroevolúcie a dokazujú, že evolučné techniky môžu byť dobrým spôsobom na návrh umelých neurónových sietí. Táto práca sa snaží na ich úspech nadviazať a ďalej rozvíjať možnosti využitia evolučných algoritmov v tejto oblasti.

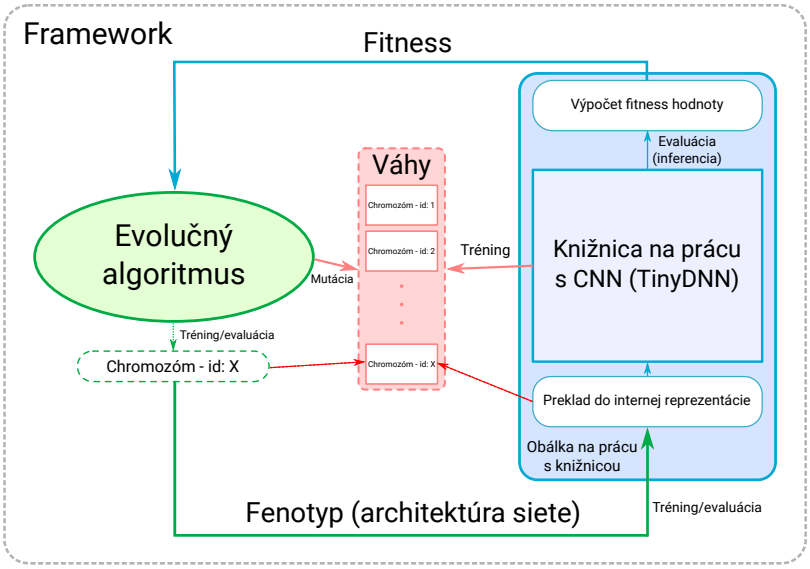
Kapitola 5

Návrh frameworku

Proces návrhu hlbokých konvolučných neurónových sietí je veľmi náročný a vyžaduje si značné skúsenosti zo strany návrhárov. Účelom frameworku, ktorý je predmetom tejto práce, je preto automatizovať tento proces. Framework je navrhnutý s ohľadom na efektívny proces prehľadávania stavového priestoru s účelom nájsť efektívne architektúry. Navrhovaný framework môžeme rozdeliť na dve základné časti:

- evolučný algoritmus zodpovedný za návrh topológií sietí,
- knižnica na prácu s konvolučnými neurónovými sieťami.

Obrázok 5.1 schematicky zobrazuje jednotlivé komponenty frameworku a prepojenia medzi nimi.



Obr. 5.1: Schéma frameworku. Evolučný algoritmus generuje jednotlivé chromozómy, ktoré sa následne v obálke prevedú na internú štruktúru použitej knižnice v podobe modelu neurónovej siete. Táto sieť môže byť pomocou knižnice natrénovaná alebo vyhodnotená. Výslednú fitness potom obálka posiela naspäť evolučnému algoritmu. Váhy každej vrstvy všetkých jedincov sú uložené externe a môžu byť zmenené trénovaním pomocou CNN knižnice alebo mutáciami zo strany EA.

Evolučný algoritmus pracuje nezávisle na použitej knižnici a komunikuje s ňou pomocou implementovanej obálky, ktorej úlohou je mapovať architektúru siete reprezentovanej chromozómom na internú formu špecifickú pre konkrétnu knižnicu. Knižnica potom zabezpečuje trénovanie podporovanými metódami (typicky backprogation) a inferenciu (výpočet na testovacích dátach) skonštruovanej siete. Váhy sa dedia z generácie na generáciu (bude detailnejšie vysvetlené nižšie) a sú teda uložené v externých súboroch mimo evolučného algoritmu. Každý chromozóm (presnejšie povedané každá vrstva) má pridelené a uložené vlastné váhy, ktoré sú ovplyvňované buď CNN knižnicou (učením), alebo pomocou špeciálnych typov mutácií z prostredia evolučného algoritmu.

Táto práca využíva ako knižnicu na prácu s CNN **TinyDNN** [33], ktorá vyniká svojou kompaktnosťou a rýchlosťou a bude ďalej popísaná v implementačnej časti tejto práce. Ako knižnica na prácu s CNN môže byť však použité akékoľvek riešenie, ktoré podporuje všetky operácie vyžadované evolučným algoritmom. Algoritmus, ktorý riadi proces evolúcie, je však na konkrétnej implementácii nezávislý.

5.1 Evolučný algoritmus

Navrhnutý evolučný algoritmus vychádza svojím princípom z genetických algoritmov a algoritmu prezentovaného v [37]. V tejto podkapitole sú popísané všetky komponenty a ich navrhované realizácie. Algoritmus je otestovaný na úlohách rozpoznávania obrazových dát na datasetoch MNIST a CIFAR-10 a porovnaný s relevantnými riešeniami. Dosiahnuté výsledky sú spracované a vyhodnotené v kapitole 7. Algoritmus 2 popisuje priebeh evolučného algoritmu.

Algoritmus 2 Pseudo-kód navrhnutého EA.

```
1:  procedure EA
2:      inicializuj počiatočnú populáciu
3:      vyhodnoť počiatočnú populáciu
4:      while (Aktuálna generácia < Počet generácií) do
5:          vyber turnajovým algoritmom n rodičov, kde n = 2
6:          if (PRNG(0, 1)[1] < P_k) then
7:              vygeneruj dvoch potomkov krížením rodičov
8:          else
9:              potomkami sa stávajú kópie rodičov
10:         end if
11:         for (pre každého potomka) do
12:             if (PRNG(0, 1) < P_m) then
13:                 aplikuj na potomka operátor mutácie
14:             end if
15:         end for
16:         natrénuj každého jedinca na trénovacom datasete
17:         vyhodnoť potomkov pomocou testovacieho datasetu
18:         vyber jedincov do ďalšej generácie
19:     end while
20: end procedure
```

[1] PRNG(x, y) – funkcia vygeneruje (uniformne) pseudo-náhodné reálne číslo z intervalu x až y.

Trénovanie sietí prebieha každú generáciu pre každého nového jedinca (potomka). Trénovanie kandidátnych jedincov každý evolučný cyklus až po plný potenciál siete, ktorú jedinec prezentuje, by bol veľmi náročný proces z pohľadu časových a výpočtových nárokov. Aby sa výrazne zredukoval čas potrebný na celý evolučný proces, nie je teda trénovanie jedincov kompletné, ale len čiastočné. Už natrénované váhy sa potom dedia z rodičov na ich potomkov, aby už naučené vlastnosti zostali v čo najvyššej miere zachované. Váhy sú uložené v externých súboroch a pre každú vrstvu zvlášť. V prípade zmeny parametrov alebo štruktúry vrstvy sa framework snaží o zachovanie čo najväčšieho počtu váh (orezaním, resp. inicializáciou). Tento prístup bol inšpirovaný technikou *weight inheritence* popísanou v [37].

Pred každým trénovaním sú trénovacie dáta náhodne premiešané, aby bolo učenie všeobecné, a teda predchádzalo pretrénovaniu[2]. Následkom takéhoto postupu nie je evolučný proces len o zmene štruktúry siete, ale aj o hľadaní správneho poradia výberu vstupných dát v rámci tréningu. Náhodné miešanie je navyše obvykle veľmi účinné pri dávkovom spracovaní tréningových dát [2, 18].

5.1.1 Kódovanie neurónových sietí

Podobne ako bolo navrhnuté v [31, 37], algoritmus využíva priame kódovanie spôsobom, v ktorom sú do chromozómov kódované celé vrstvy a nie jednotlivé neuróny. Vzhľadom na to, že vo väčšine voľne dostupných frameworkov na prácu s CNN funguje konštrukcia neurónových sietí na podobnej úrovni abstrakcie, je výber takéhoto kódovania veľmi intuitívny a je vďaka nemu možné odizolovať evolučný algoritmus od konkrétnej implementácie CNN.

Každý chromozóm obsahuje hlavičku so základnými informáciami spoločnými pre celú sieť, ako je *id*, *vek* (bude vysvetlené neskôr), *rozmery vstupných dát* a *koeficient učenia*. Nasleduje telo chromozómu, ktoré pozostáva z niekoľkých po sebe idúcich vrstiev a parametrov špecifických pre túto vrstvu. Ako bolo ukázané v [48], môže byť výhodné mať čo najmenší počet typov vrstiev, aby bolo umožnené efektívne prehľadávanie stavového priestoru, a preto sú navrhované vrstvy dve nasledovné:

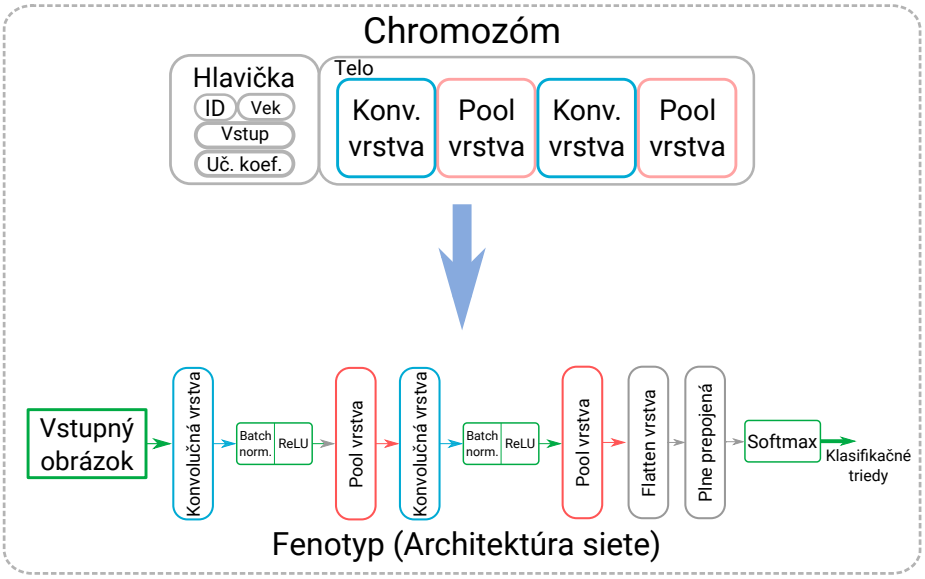
- **Konvolučná vrstva** - základný stavebný blok CNN, parametre: *veľkosť kernelu (kernelSize)*, *počet filtrov (outChannels)*, *veľkosť kroku (stride)* a *výplň (padding)*.
- **Pool vrstva** - zmenšuje rozmery výstupných máp príznakov, parametre: *veľkosť kroku (stride)*, *veľkosť podvzorkovania* (poolSize) a *typ podvzorkovania (poolType)*.

Za každou konvolučnou vrstvou ešte nasleduje dávková (batch) normalizácia a ReLU aktivácia. Fenotyp môže byť na základe chromozómu skonštruovaný priamočiaro intuitívnym spôsobom – poskladaním jednotlivých vrstiev podľa ich parametrov za sebou. Na záver architektúry sa vždy pridáva ešte konvolučná tzv. „zoštíhľovacia“ (*angl. flatenning*) vrstva slúžiaca na redukciu dimenzií výstupu konvolučných vrstiev na lineárny vektor a jedna plne prepojená vrstva. Za touto vrstvou nasleduje *softmax* aktivácia, ktorá slúži ako klasifikátor. Príklad chromozómu a jeho mapovanie na konvolučnú neurónovú sieť je zobrazený na obrázku 5.2.

5.1.2 Selekcia

V rámci evolučného algoritmu sú navrhnuté dva druhy mutácie. Najprv sa uskutoční výber rodičov, z ktorých sú následne vygenerovaní potomkovia. Potom nastáva environmentálna

[2]Pretrénovanie (*angl. overfitting*) – jav, pri ktorom rastie úspešnosť na siete na trénovacom datasete, avšak klesá na testovacom datasete, inými slovami – model sa stal príliš špecializovaný na trénovacie dáta.



Obr. 5.2: Príklad mapovania genotypu (chromozómu) na fenotyp (architektúru siete). Za každú konvolučnú vrstvu sa pridáva ešte dávková (batch) normalizácia a ReLU aktivácia, na záver sa pridáva softmax klasifikátor. Pre jednoduchosť sú v obrázku opomenuté parametre jednotlivých vrstiev, ktoré sú taktiež zakódované v chromozóme.

selekcia jedincov, ktorí sa dostanú (prežijú) do ďalšej generácie. Ostatní jedinci sú zahodení (zabití). Takáto selekcia má za úlohu vytvárať tlak na kandidátne riešenia a v tomto kontexte by mohla prispieť k znižovaniu počtu cyklov potrebných na plné natrénovanie siete – a teda hľadať čo najoptimálnejšie siete.

Selekcia rodičov prebieha klasickým turnajovým algoritmom, pri ktorom veľkosť turnaja k podlieha konfigurácii evolučného algoritmu. Rodičia sa vyberajú postupne, až kým nie je naplnený potrebný počet ich potomkov, ktorý sa rovná veľkosti celej populácie.

Environmentálna selekcia prebieha ako selekcia z $(\mu + \lambda)$, pričom μ je veľkosť populácie a λ je počet vygenerovaných potomkov. Selekcia $(\mu + \lambda)$ potom znamená, že do ďalšej generácie je vybratých μ jedincov z celej množiny rodičov a ich potomkov. Takáto selekcia umožňuje rodičom, ktorí sú už plne natrénovaní (t.j. ďalšie trénovanie by viedlo k zníženiu presnosti na testovacom datasete – pretrénovaniu), prežiť do ďalšej generácie.

Rozdeľovanie do druhov

Výber μ jedincov pre ďalšiu generáciu môže v navrhnutom frameworku prebiehať dvoma spôsobmi. Prvým je intuitívny výber μ najlepších jedincov z množiny $(\mu + \lambda)$. Problémom tohto typu selekcie však môže byť znevýhodňovanie inovatívnejších a potenciálne lepších, ale nedostatočne pretrénovaných jedincov. Je predpoklad, že zásadné zmeny ako kríženie a niektoré typy mutácií môžu výrazným spôsobom negatívne ovplyvniť presnosť sietí. Ďalším problémom môže byť aj pomalšia učiaca krivka hlbších sietí. Je preto dôležité dať takýmto jedincom dostatočný čas na optimalizáciu svojich váh pomocou učiaceho procesu. Ako riešenie tohto problému je v algoritme navrhnutý koncept rozdeľovania do druhov na princípe podobnom ako v [46]. V tejto práci sú navrhnuté a otestované dva spôsoby rozdeľovania do druhov – na základe *trénovacieho veku* alebo *veľkosti (alebo aj hĺbky) siete*. Mechanizmus

selekcie spolu s rozdeľovaním do druhov je popísaný algoritmom 3. Druhom sa v tomto kontexte myslia všetci jedinci s rovnakými vlastnosťami (veľkosť alebo trénovací vek).

Prvým spôsobom rozdeľovania do druhov je inovatívny koncept **trénovacieho veku**, ktorý rastie spolu s počtom trénovacích procesov pre daného jedinca. Naopak vek sa zresetuje na počiatočnú hodnotu v prípadoch, v ktorých bol jedinec výrazným spôsobom (t.j. negatívne vplývajúcim na presnosť) zmenený. Medzi takéto zmeny patrí pridanie/odstránenie vrstvy, kríženie a pod. Algoritmus potom v rámci selekcie porovnáva iba jedincov s rovnakým vekom, aby dal novým jedincom čas na dotrénovanie. V praxi to znamená, že pre každý vekový level je zvolených k najlepších jedincov, ako je znázornené v algoritme 3. Zároveň je možné nastaviť tzv. hranicu dospelosti, od ktorej sú už všetci jedinci porovnávaní spoločne. Dôvodom tejto hranice je zachovanie selekčného tlaku pri vysokých úrovniach trénovacieho veku, na ktorých už ďalšie trénovanie nemá na presnosť veľký vplyv.

Druhým spôsobom je delenie **podľa veľkosti danej siete**. Pri každom jedincovi sa teda zaznamenáva veľkosť siete, ktorú reprezentuje. Pri selekcii sa potom vyberajú najlepší jedinci pre každý veľkostný level. Vďaka tomuto prístupu majú hlbšie siete viac času na optimalizáciu svojich váh. Aj v rámci tohto mechanizmu je možné definovať hranicu "dospelosti", avšak v tomto prípade ide o hraničnú hĺbku. Všetky siete hlbšie ako táto hranica budú porovnávané spoločne, podobne ako to bolo popísané pri rozdeľovaní podľa veku.

Algoritmus 3 Pseudo-kód selekčného mechanizmu s rozdelením do druhov.

```
1: while (jedinci != veľkosť populácie) do
2:     for (pre každý druh) do
3:         k = PRNG_INT(0, t)[3]
4:         vyber k najlepších z daného druhu
5:     end for
6: end while
```

5.1.3 Mutácia a kríženie

Ako genetické operátory sú v rámci tejto práce navrhnuté rôzne typy mutácie a operácia kríženia. Ich účinnosť je experimentálne vyhodnotená v kapitole 7.

Na každý vybratý pár rodičov sa s pravdepodobnosťou P_k aplikuje operátor kríženia. Každý takýto pár potom vygeneruje práve dvoch potomkov, a to nasledovným spôsobom. Každý rodič sa rozdelí v náhodnom bode na dve časti. Prvý potomok potom vznikne spojením prvej časti prvého rodiča a druhej časti druhého rodiča. Druhý potomok vznikne analogicky spojením prvej časti druhého rodiča a druhej časti prvého rodiča. V prípade, že operácia kríženia nenastane, stávajú sa potomkami kópie dvoch vybratých rodičov.

Na vzniknutých potomkov sa následne môže ešte s pravdepodobnosťou P_m aplikovať operátor mutácie, ktorý náhodne mení štruktúru alebo parametre chromozómu. Medzi mutačné operácie patria teda základné a intuitívne operácie so štruktúrou siete. Ich celkový výpis je nasledovný:

- **resetuj váhy pre vrstvu** – odstráni súbor uchovávajúci váhy pre danú vrstvu,
- **pridaj náhodnú vrstvu** – pridá náhodne vygenerovanú vrstvu na náhodný index,

[3] PRNG_INT(x,y) – funkcia vygeneruje pseudo-náhodné celé číslo uniformným rozložením z intervalu x až y.

- **odober vrstvu** – odstráni vrstvu z náhodne vybraného indexu,
- **modifikuj vrstvu** – náhodne modifikuje niektoré parametre vrstvy na náhodne vybranej vrstve,
- **uprav počet parametrov pre plne prepojenú vrstvu** – zvýši alebo zníži počet prepojení v poslednej plne prepojenej vrstve,
- **uprav koeficient učenia** – zvýši alebo zníži koeficient učenia (angl. learning rate).

Každá mutácia má pridelenú pravdepodobnosť, s ktorou sa v rámci operátora mutácie vykoná. Nastavenie týchto pravdepodobností a tiež pravdepodobnosti P_k a P_m sú súčasťou nastavenia parametrov evolučného algoritmu.

5.1.4 Fitness funkcia

Ako fitness hodnota kandidátnych jedincov slúži presnosť siete reprezentovanej týmto jedincom na testovacom datasete. Presnosť siete je možné definovať ako pomer správne klasifikovaných vzoriek k počtu všetkých vzoriek testovacieho datasetu. Fitness hodnota jedincov je vypočítaná v rámci knižnice na prácu s CNN a následne odoslaná evolučnému algoritmu.

Oproti riešeniam zaznamenaným v literatúre sa do hodnoty fitness nemusí započítavať len úspešnosť siete a v riešení daného problému, ale aj počet parametrov p, ktoré daná architektúra obsahuje. Cieľom je hľadať architektúry, ktoré nie sú len najlepšie, ale aj najefektívnejšie v riešení daného problému. Silu jednotlivých prvkov fitness hodnoty je možné ovplyvňovať koeficientom k. V prípade, že je koeficient k nastavený na 0, presnosť siete sa stáva priamo aj fitness hodnotou a počet parametrov nemá žiadny vplyv. So zvyšujúcim sa koeficientom k sa zvyšuje aj vplyv počtu parametrov siete na celkovú fitness.

Výpočet fitness sa teda dá zapísať rovnicou 5.1. Počet parametrov je škálovaný logaritmicky, aby bol vplyv počtu parametrov dostatočný. Grafy 5.3a a 5.3b znázorňujú vplyv počtu parametrov a hodnôt koeficientu k na celkovú fitness.

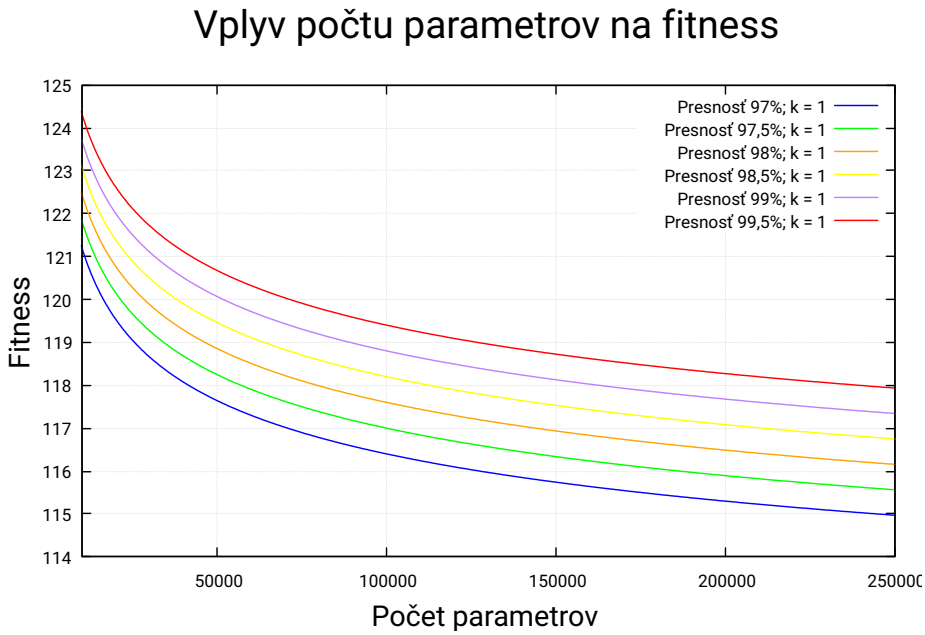
$$f_x = a * (k \ * \frac{1}{\log(p) + 1} + 1) \tag{5.1}$$

5.1.5 Inicializácia populácie

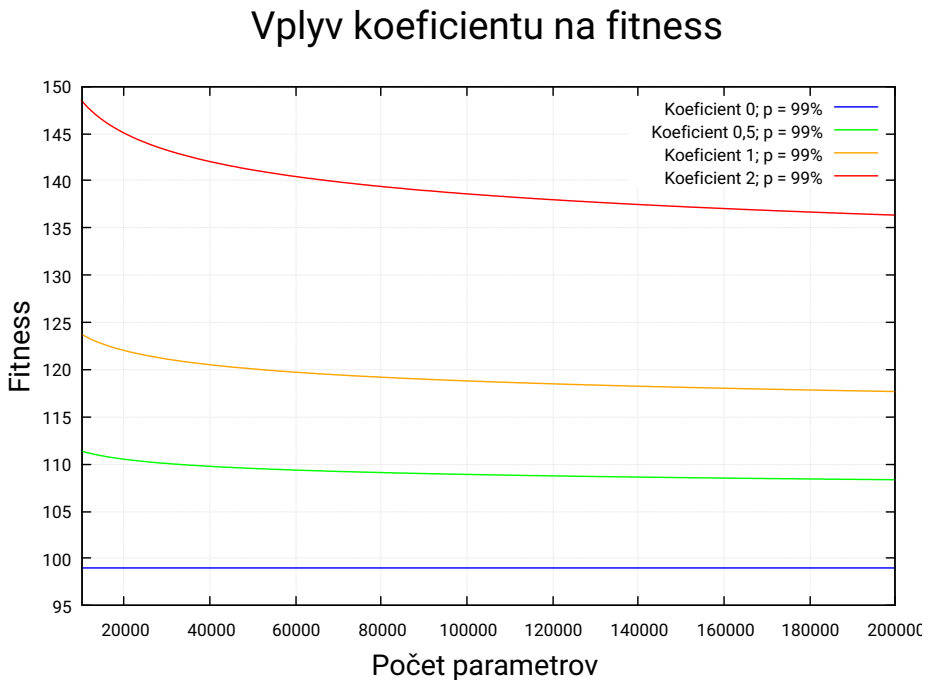
Počiatočná populácia môže byť inicializovaná dvoma spôsobmi:

1. Všetci počiatoční jedinci sú inicializovaní náhodne vygenerovanými sieťami o náhodnej veľkosti.
2. Všetci jedinci sú inicializovaní predom definovanou sieťou.

Pri zvolení druhej možnosti inicializácie funguje navrhnutý framework ako optimalizátor tejto siete. Podľa nastavenia koeficientu počtu parametrov sa snaží hľadať riešenia, ktoré sú efektívnejšie, ale len za malú stratu presnosti, alebo riešenia, ktoré sa presnosť dosiahnutú touto sieťou snažia ešte zlepšiť. Znižovanie výpočtových nárokov siete aj za cenu miernej straty presnosti, môže byť výhodné napríklad v embedded aplikáciách, v ktorých nám primárne záleží na výkone a menej na presnosti daného riešenia.



(a) Vplyv počtu parametrov na výslednú fitness.



(b) Vplyv koeficientu na výslednú fitness

Obr. 5.3: Grafy znázorňujúce vlastnosti fitness funkcie. Prvý graf zobrazuje vplyv počtu parametrov na fitnes hodnotu pre rôzne úrovne presnosti siete. Druhý graf zase zobrazuje vplyv koeficientu k na hodnotu fitness pri sieti dosahujúcej presnosť 99% a meniacom sa počte parametrov.

Kapitola 6

Implementácia a použitie

V tejto kapitole budú postupne popísané prostriedky, ktoré boli na implementáciu frameworku využité, a vysvetlené všetky dôležité časti implementácie. Okrem toho bude popísaný aj spôsob použitia a možnej konfigurácie výsledného frameworku. Celá práca bola implementovaná v jazyku *C++* s využitím funkcionality poskytovanej v rámci štandardných knižníc *C++14*.

6.1 TinyDNN

Ako knižnica na prácu s konvolučnými neurónovými sieťami bola v rámci tejto práce zvolená knižnica *TinyDNN* [33]. Táto knižnica je napísaná v jazyku *C++* a pozostáva iba z hlavičkových súborov, čo umožňuje veľmi jednoduchú integráciu a zabezpečuje pohodlnú prenositeľnosť. Z pohľadu tejto práce sú tieto vlastnosti výhodné najmä kvôli testovaniu na rozličných zariadeniach. Knižnica je takisto pomerne rýchla, čo je nesmierne dôležité vzhľadom na nutnosť počítania veľkého množstva zložitejších neurónových sietí v rámci priebehu evolučného algoritmu. TinyDNN takisto podporuje paralelizáciu pomocou *OpenMP* pre trénovanie a testovanie neurónových sietí. V rámci knižnice sú tiež plne implementované a dostupné všetky základné prvky konvolučných neurónových sietí popísaných v literatúre.

6.2 Obálka na prácu s CNN knižnicou

Aby bolo možné oddeliť implementáciu evolučného algoritmu od použitej CNN knižnice, je implementovaná abstraktná šablónová trieda `LibraryInterface`, ktorá slúži ako obálka nad CNN frameworkom. Táto trieda slúži ako *rozhranie* a obsahuje virtuálne metódy pre prácu s konvolučnými neurónovými sieťami – prevod chromozómu do internej reprezentácie CNN knižnice, trénovanie a testovanie siete, príprava datasetu a tiež načítavanie a ukladanie váh do externých súborov. Ako šablónový parameter prijíma táto trieda dátový typ siete implementovanej v konkrétnom frameworku, prípadne jej abstrakciu. Pre každý použitý CNN framework je následne nutné implementovať triedu dediacu z `LibraryInterface` a doimplementovať všetky metódy určené na prácu s konvolučnými neurónovými sieťami z pohľadu evolučného algoritmu.

V rámci tejto práce bola implementovaná obálka pre prácu s knižnicou *TinyDNN* popísanej vyššie. Túto obálku predstavuje trieda `TinyDNNInterface`, ktorej šablónovým parametrom je dátový typ `tiny_dnn::sequential`. Tento typ reprezentuje model siete v knižnici *TinyDNN*. Trieda `TinyDNNInterface` implementuje všetky metódy rozhrania

`LibraryInterface` a slúži tak ako prepojenie evolučného algoritmu s frameworkom *TinyDNN*.

6.2.1 Načítavanie a ukladanie parametrov

Načítavanie a ukladanie parametrov jednotlivých vrstiev do externých súborov je dôležitou súčasťou evolučného procesu navrhnutého frameworku, pretože zabezpečuje možnosť dedenia váh z rodiča na potomka. Táto vlastnosť je preto prerekvizitou pri výbere vhodnej knižnice na prácu s konvolučnými neurónovými sieťami. Knižnica TinyDNN prácu s váhami umožňuje pomocou metód `save(..)` a `load(..)` implementovaných v triede `layer`. Tieto metódy sú potom využívané metódami `saveWeights(..)` a `loadWeights(..)` triedy `TinyDNNInterface`. Okrem načítavania parametrov siete (váh a biasov) je úlohou metódy `loadWeights(..)` aj kontrola parametrov ovplyvňujúcich štruktúru načítavanej vrstvy a v prípade zmeny zabezpečuje táto metóda aj prispôsobenie načítaných váh a biasov novej štruktúre. Aby bolo možné pri načítavaní uchovať čo najväčší počet parametrov, využíva sa metóda `reshapeWeights(..)`, ktorá z načítaných parametrov (váhy a biasy) vyberie len tie, ktoré zostali po zmene štruktúry zachované, prípadne pridá nové inicializované parametre.

6.3 Evolučný algoritmus

Samotný evolučný algoritmus je implementovaný s dôrazom na prehľadnosť a jednoduchú rozšíriteľnosť. Tieto vlastnosti sú dôležité najmä z dôvodu nutnosti experimentovania s rôznymi nastaveniami a implementovanými mechanizmami.

Z pohľadu implementácie je najdôležitejšia trieda `Chromosome` reprezentujúca chromozóm a teda abstrakciu neurónovej siete. Obsahuje všetky parametre dôležité pre evolučný proces. Vrstvy siete sú uložené ako vektor objektov triedy `Layer`, predstavujúcej abstrakciu vrstiev, ktoré sú predmetom evolúcie (konvolučná vrstva a podvzorkovacia vrstva). Trieda `Chromosome` obsahuje štyri druhy konštruktorov slúžiacich na rôzne účely – načítanie chromozómu, vytvorenie chromozómu s inicializovanou sieťou, kopírovanie chromozómu a tiež konštruktor slúžiaci ako evolučný operátor kríženia. Konštruktor kríženia prijíma ako argumenty dva chromozómy, z ktorých následne vygeneruje potomka. Ako evolučný operátor mutácie zase slúži metóda `mutate()`, ktorá nad chromozómom vykoná náhodnú mutáciu (podľa nastavených váh mutácií). Pomocou šablónovej metódy `evaluateFitness()` je možné vypočítať fitness hodnotu s využitím CNN knižnice, ktorej obálka je špecifikovaná ako šablónový parameter tejto metódy. Trieda `Chromosome` okrem toho obsahuje aj niekoľko ďalších podporných metód.

6.4 Konfigurácia a použitie

Konfigurácia parametrov evolučného algoritmu je obsiahnutá v súbore `config.h`. V zdrojovom kóde 6.1 je zobrazený príklad nastavenia priebehu evolučného algoritmu. Parametre v súbore `config.h` sa dajú rozdeliť do dvoch skupín:

- **parametre evolučného algoritmu** – ide o veľkosť populácie, počet generácií, veľkosť turnaja, hranicu dospelosti a pravdepodobnosti mutačných operátorov,
- **parametre konvolučných neurónových sietí** – zahŕňa počiatočné hodnoty veľkosti vygenerovaných sietí, veľkosti podvzorkovania, hĺbku plne prepojenej siete, učiaceho koeficientu, trénovacích epoch a veľkosti dávky (angl. batch).

```
// evolucne nastavenia
// zakladne parametre evolucie
const int POPULATION_SIZE = 5;
const int GENERATIONS = 50;

// parametre selekcie
// velkost turnaja
const int TOURNAMENT_SIZE = 2;
// hranica dospelosti
const int ADULT_AGE = 4;

// pravdepodobnosti evolucnych operatorov
const double MUTATION_PROBABILITY = 0.4;
const double CROSSOVER_PROBABILITY = 0.3;

// vahy jednotlivych mutacii:
// Pridaj vrstvu|Odober vrstvu|Resetuj vahy|Modifikuj vrstvu|Uprav hlbku FC|Uprav uciaci koeficient
const std::initializer_list<double> MUTATION_WEIGHTS = {40, 10, 5, 10, 15, 15};

// parametre CNN
// maximalna dlzka nahodne generovanej siete
const int MAX_START_SIZE = 6;
// maximalna velkost podvzorkovania
const int MAX_POOL_SIZE = 4;

// parametre ucenia
const double START_LEARNING_RATE = 0.1;
const double LEARNING_STEP = 0.01;

// pociatocne hodnoty pre nove vrstvy
// pociatocny pocet filtrov
const int START_FILTER_COUNT = 10;
// pociatocna hlbka plne prepojenej vrstvy
const int START_FC_DEPTH = 15;

// treningove parametre
// velkost davky (batch)
const int TRAIN_BATCH_SIZE = 32;
// pocet treningovych cyklov
const int TRAINING_EPOCHS = 1;
```

Výpis 6.1: Zdrojový kód predstavujúci príklad konfigurácie frameworku.

Projekt je možné preložiť jednoducho pomocou programu *CMake*[1]. Na preklad je tiež potrebné, aby cieľový systém obsahoval *C++* prekladač, ktorý podporuje štandard *C++14*. Priložený *CMakeList.txt* obsahuje všetky nastavenia potrebné na správne vygenerovanie súboru *Makefile*, ktorý slúži na preklad frameworku. Dôležitou súčasťou konfigurácie je najmä možnosť vypnutia, resp. zapnutia paralelizácie trénovania a testovania CNN v rámci knižnice *TinyDNN* pomocou *OpenMP* (parameter `USE_OMP`) a vektorových inštrukcií SSE (parameter `USE_SSE`). Takisto je možné pomocou parametra `OMP_NUM_THREADS` nastaviť maximálny počet vlákien alokovaných pre výpočet v prípade využitia *OpenMP*. V rámci *CMakeList.txt* je možné upraviť aj niektoré ďalšie nastavenia *TinyDNN*, ale z pohľadu tejto práce nie je odporúčané implicitné nastavenie meniť.

[1] CMake – open-source, multiplatformový nástroj určený na prekladanie, zostavovanie, testovanie a balíkovanie softvéru.

Preložený framework ako vstupné parametre akceptuje zvolený dataset (MNIST alebo CIFAR10) a cestu k binárnym súborom datasetu. Všetky ostatné parametre je potrebné nastaviť v súbore *config.h*. Príklad spustenia je:

```
$ ./evolve --dataset "mnist" --data_path "../data/"
```

Kapitola 7

Experimenty a výsledky

V tejto kapitole sú popísané a vyhodnotené experimenty, ktorých úlohou bolo zmerať a zhodnotiť účinnosť frameworku navrhnutého a implementovaného v rámci tejto práce. Ďalej sú v tejto kapitole uvedené všetky prostriedky a zdroje použité v rámci vykonávania experimentov a tiež datasety, nad ktorými boli experimenty realizované. Všetky namerané výsledky zobrazené v tejto kapitole sú priemerom z n priebehov s rovnakým nastavením. Kvôli obmedzeným výpočtovým zdrojom bolo na nižšie popísané experimenty n zvolené na hodnotu 5, pokiaľ nie je explicitne uvedené inak. Nastavenia parametrov konvolučných neurónových sietí sú vypísané v tabuľke 7.1. Tabuľka 7.2 zase ukazuje nastavenia váh jednotlivých mutácií pre oba datasety. Tieto parametre boli použité v rámci každého experimentu popísaného nižšie. Ich hodnoty boli nastavené empiricky na základe niekoľkých pokusných behov EA. Najvhodnejšie nastavenie týchto parametrov by mohlo byť predmetom ďalšieho výskumu.

Vo všetkých experimentoch, pokiaľ nie je uvedené inak, bol tiež nastavený **koeficient počtu parametrov v hodnote fitness na 0**. Fitness hodnota jedinca a presnosť siete teda majú rovnaký význam – sú zameniteľné. Dôvodom tohto nastavenia je cieľ, otestovať navrhnuté vylepšenia v evolučnom algoritme, ktorý hľadá čo najpresnejšie siete bez obmedzenia. Schopnosť algoritmu hľadať efektívne siete s ohľadom na počet parametrov je otestovaný zvlášť v experimente popísanom v podkapitole 7.6.

Nastavanie parametrov CNN		
Parametre/Dataset	**MNIST**	**CIFAR10**
Učiaci koeficient	0.1	0.1
Maximálna hĺbka plne prepojenej vrstvy	50	70
Maximálny počet filtrov	12	20
Krok učenia	0.01	0.01
Maximálna veľkosť pool vrstvy	4	4
Veľkosť dávky (batch)	32	32
Počet epoch trénovania	1	1
Maximálna veľkosť počiatočnej populácie	6	8

Tabuľka 7.1: Tabuľka zobrazuje základné nastavenie parametrov CNN pre navrhnutý framework. Parametre sú nastavené rôzne pre datasety MNIST a CIFAR10. Hodnoty v tabuľkách pre parametre učiaci koeficient, hĺbka plne prepojenej vrstvy a počiatočný počet filtrov predstavujú len počiatočné nastavenie, ale môžu podliehať zmene v rámci evolučného procesu. Ostatné parametre sú fixné a EA nemá možnosť ich zmeniť.

Nastavanie váh jednotlivých mutácií EA		
Váhy/Dataset	**MNIST**	**CIFAR10**
Pridaj vrsvu	50	60
Odober vrstvu	10	10
Resetuj vahy	5	5
Modifikuj vrstvu	40	40
Uprav hĺbku plne prepojenej vrstvy	15	15
Uprav učiaci koeficient	15	15

Tabuľka 7.2: Tabuľka zobrazuje nastavenie váh podľa ktorých sa aplikujú jednotlivé mutácie na datasetoch MNIST a CIFAR10. Váhy v tomto kontexte slúžia na určovanie pravdepodobnosti s akou sa aplikuje daná mutácia. Pravdepodobnosť, že sa aplikuje niektorá mutácie je teda hodnota jej váhy podelená súčtom váh všetkých mutácií.

7.1 Datasety

Na vyhodnotenie navrhnutého frameworku boli použité dva štandardné testovacie datasety z oblasti klasifikácie obrázkov – MNIST a CIFAR10. Oba sú voľne dostupné pre široké využitie a používajú sa ako benchmarkové a testovacie datasety vo veľkom množstve publikovanej literatúry. Sú preto vhodnými referenčnými datasetmi.

7.1.1 MNIST

Databáza MNIST[27] je zložená zo 60 000 vzoriek obrázkov ručne písaných číslic od 0 do 9. Vznikla ako podmnožina databázy NIST. Všetky obrázky sú čiernobiele o veľkosti 28×28 pixelov. Príklad dát z databázy MNIST je zobrazený na obrázku 7.1. Databáza je voľne dostupná a rýchlo sa stala základným benchmarkovým datasetom pre nové metódy klasifikácie obrazu. Pozostáva zo štyroch binárnych súborov, ktoré je nutné spracovať jednoduchým algoritmom:

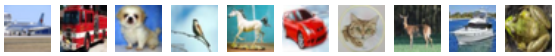
- train-images-idx3-ubyte.gz – trénovacia sada obrázkov (9912422 bytov)
- train-labels-idx1-ubyte.gz – popisky k trénovacej sade obrázkov (28881 bytov)
- t10k-images-idx3-ubyte.gz – testovacia sada obrázkov (1648877 bytov)
- t10k-labels-idx1-ubyte.gzz – popisky k testovacej sade obrázkov (4542 bytov)

7.1.2 CIFAR10

Databáza CIFAR10 [22] pozostáva zo 60 000 obrázkov veľkosti 32×32, ktoré zozbierali Alex Krizhevsky, Vinod Nair a Geoffrey Hinton. Každý z obrázkov patrí do jednej z 10 tried, ako napríklad lode, lietadlá, autá, mačky alebo vtáky.

Celý dataset je rozdelený na päť tréningových sád, ktoré spolu obsahujú 50 000 obrázkov (5 000 obrázkov z každej triedy) a jednu testovaciu sadu s 10 000 obrázkami (1 000 obrázkov z každej triedy). Na obrázku 7.2 je zobrazený príklad dát z databázy CIFAR10.

Obr. 7.1: Príklad normalizovaných dát z databázy MNIST. Zdroj: [26].



Obr. 7.2: Príklady obrázkov z každej triedy v databáze CIFAR10. Zdroj: [22].

7.2 Výpočtové zdroje

Všetky experimenty boli realizované na počítačoch Anselm a Salomon patriacich pod Národné superpočítačové centrum IT4Innovations. Superpočítač Anselm pozostáva z 209 výpočtových uzlov, čo predstavuje 3 344 procesorových výpočtových uzlov, a teda celkovo 24 192 procesorových jadier a 129 TB RAM. Každý experimentálny priebeh bol vykonaný na jednom výpočtovom uzle. Experimenty na datasete MNIST prebiehali na počítači Anselm, zatiaľ čo experimenty na datasete CIFAR10 prebiehali na počítači Salomon. Parametre týchto uzlov sú zobrazené na tabuľke 7.3.

Parametre uzla/ Superpočítač	Anselm	Salomon
Procesor	2x Intel Sandy Bridge E5-2665	2x Intel Xeon E5-2680v3
Frekvencia	2,4 GHz	2,5 GHz
Počet vlákien	16	24
RAM	64 GB	128 GB
HDD	500GB SATA 2,5” 7,2 krpm	zdieľaný

Tabuľka 7.3: Parametre výpočtových uzlov superočítačov IT4I Anselm a Salomon.

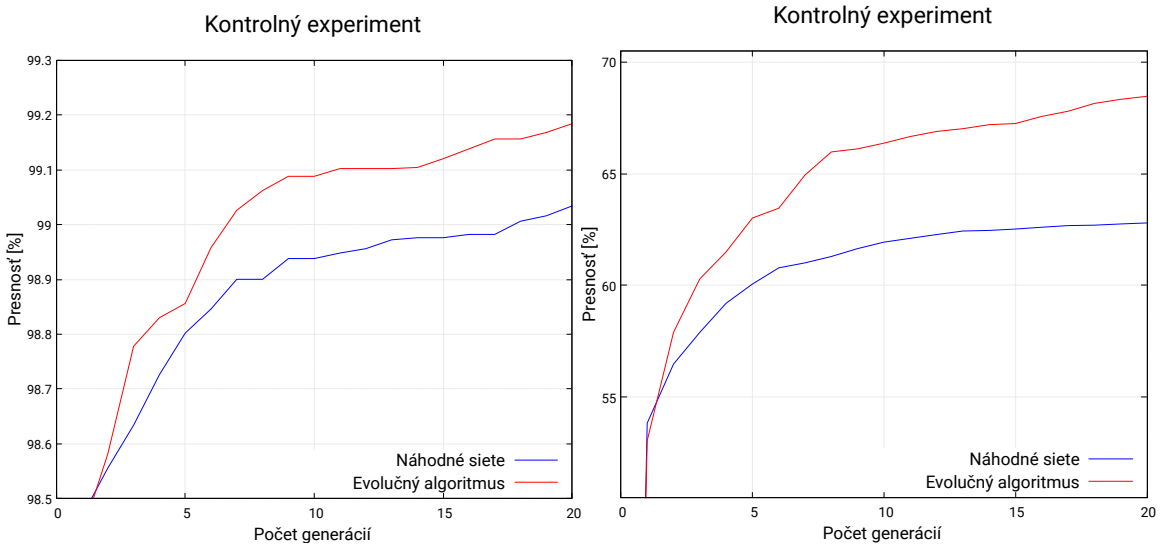
7.3 Kontrolný experiment

Ako kontrolný experiment bola porovnaná najlepšia nameraná konfigurácia evolučného algoritmu voči plne natrénovaným náhodne vygenerovaným sieťam bez selekcie a ich postupného vylepšovania. Tento experiment bol navrhnutý ako kontrola všeobecnej účinnosti evolučného algoritmu oproti čisto náhodnému prístupu. Experiment bol realizovaný na datasetoch MNIST a CIFAR10.

Nastavenie evolučného algoritmu je zobrazené v tabuľke 7.4. Počiatočná populácia bola inicializovaná náhodne. Náhodný beh spočíval vo vygenerovaní ôsmich alebo dvanástich náhodných sietí (podľa počtu jedincov v populácii) a ich plnom natrénovaní až po 20 epoch (ako 20 generácií v ponímaní EA). Experiment teda do najvyššej možnej miery simuluje priebeh EA, avšak bez selekcie a evolučných operátorov.

Nastavanie evolučného algoritmu		
Parametre/Dataset	**MNIST**	**CIFAR10**
Počet generácií	20	20
Veľkosť populácie	8	12
Hranica dospelosti	6	8
Veľkosť turnaja	2	2
Pravdepodobnosť kríženia	35%	35%
Pravdepodobnosť mutácie	50%	50%

Tabuľka 7.4: Tabuľka zobrazuje základné nastavenie evolučného algoritmu pre datasety MNIST a CIFAR10.



(a) Dateset MNIST.

(b) Dataset CIFAR10.

Obr. 7.3: Grafy zobrazujúce priebehy evolučného algoritmu voči náhodne vygenerovaným sieťam a ich postupnému trénovaniu. Osa x zobrazuje počet generácií EA – resp. počet epoch pre náhodný algoritmus – a osa y zobrazuje presnosť najlepšej siete. (Priemer z piatich priebehov.)

Graf 7.3a zobrazuje priebeh evolučného algoritmu a priebeh trénovania náhodných sietí na datasete MNIST. Evolučný algoritmus dosiahol za 20 generácií presnosť na testovacích dátach o 0,15 percentuálneho bodu vyššiu ako náhodný beh. Hlavnou príčinou, prečo rozdiel nie je významnejší, je jednoduchosť datasetu MNIST. Pri tomto datasete sa ukazuje, že aj náhodne vygenerované siete majú pomerne veľkú šancu byť presné. Napriek tomu však mal evolučný algoritmus lepší priebeh a získaná presnosť bola nezanedbateľne vyššia. Priemerná doba výpočtu bola mierne v prospech náhodného behu (300,4 procesoro-hodín oproti 449,5 procesoro-hodín), čo vzbudzuje otázku využiteľnosti takéhoto algoritmu na tak malom datasete. Zaujímavou vlastnosťou evolučného algoritmu však bola efektívnosť nájdených sietí v porovnaní so sieťami vygenerovanými náhodne. Priemerný počet parametrov náhodných sietí bol 220 846,2, zatiaľ čo priemerná veľkosť sietí nájdených evolučným algoritmom bola len 58 300,6. Dataset MNIST je možné kvôli jeho jednoduchosti pomerne rýchlo natrénovať len pomocou plne prepojenej vrstvy, a preto niektoré náhodne nájdené architektúry

obsahovali len veľmi málo (1-2) konvolučných vrstiev. Počet parametrov v plne prepojenej vrstve bol preto pomerne vysoký. Príčinou tohto efektu je pravdepodobne tiež selekčný tlak algoritmu spojený s dedičnosťou váh. Trénovať siete s veľkým počtom parametrov trvá obvykle oveľa dlhšie než menej zložité siete. Keďže EA zahadzuje menej úspešné siete, vzniká selekčný tlak na to, aby boli siete čo najpresnejšie za čo najmenší počet trénovacích cyklov. Vďaka tejto vlastnosti sú výsledné siete podstatne optimálnejšie v porovnaní s náhodnými sieťami, ktoré majú na natrénovanie dostatok času. Po spojení s evolučnými operátormi, ktorých úlohou je rozširovať prehľadávaný stavový priestor, sú teda siete nájdené evolučným algoritmom nezanedbateľne presnejšie a z pohľadu počtu parametrov aj podstatne efektívnejšie než siete vygenerované náhodne.

Graf 7.3b ukazuje priebehy evolučného algoritmu a náhodných sietí pre dataset CIFAR10. Tento graf potvrdzuje a zvýrazňuje úspešnosť algoritmu hľadať presnejšie siete oproti náhodne vygenerovaním sieťam. Rozdiel medzi evolučným algoritmom a náhodným behom sa vyšplhal na testovacom datasete až na 5,7% za 20 generácií. Na grafe je tiež vidieť potenciál EA ďalej hľadať presnejšie siete aj po ďalšie generácie, zatiaľ čo náhodný algoritmus začína výrazne stagnovať. Vlastnosť algoritmu hľadať (z pohľadu počtu parametrov) efektívnejšie siete oproti náhodne vygenerovaným sieťam sa však na datasete CIFAR10 nepotvrdila (173 106 parametrov pre EA oproti 127 646,6 pre náhodný algoritmus). Siete nájdené pomocou EA sú však výrazne presnejšie, a preto nie je úplne možné porovnávať ich optimálnosť. Ďalšie experimenty sú nutné, aby sa zistilo, či bol tento efekt na datasete MNIST čisto náhodný, alebo je to zapríčinené zložitosťou testovaných datasetov.

Najlepšie siete zo všetkých spustených priebehov náhodného algoritmu boli 99,14% pre dataset MNIST a 65,96% pre dataset CIFAR10. Aj v tomto ohľade dokázal byť evolučný algoritmus výrazne úspešnejší oproti náhodnému algoritmu. Najlepšia sieť zo všetkých behov evolučného algoritmu na datasete CIFAR10 bola 73,05% a na datasete MNIST 99,3%. Navrhnutý evolučný algoritmus sa teda preukázal ako pomerne úspešný spôsob automatického vyhľadávania celkom presných architektúr konvolučných neurónových sietí a môže slúžiť ako základ pre ďalší výskum.

7.4 Účinnosť navrhnutých operátorov

Na vyhodnotenie účinnosti predstavených evolučných operátorov mutácie a kríženia bola navrhnutá a prevedená sada troch experimentov na datasetoch MNIST a CIFAR10.

V prvom experimente bola nameraná fitness dosiahnutá bez použitia akýchkoľvek evolučných operátorov. V praxi tento experiment predstavuje trénovanie náhodne vygenerovaných sietí na začiatku procesu a hľadanie optimálnych váh pomocou procesu trénovania. Rozdielom oproti kontrolnému experimentu je schopnosť algoritmu na základe selekcie hľadať optimálne poradie výberu tréningových dát. Tréningový proces môže mať výrazne rozdielne výsledky pri rôznych náhodných usporiadaniach vstupných dát, a to najmä z dôvodu dávkového spracovania. Evolučný algoritmus sa teda v tomto experimente zúžil len na hľadanie správneho poradia premiešania dát, namiesto zmeny štruktúry danej siete.

Druhý experiment spočíval v evolučnom priebehu so zapnutým evolučným operátorom mutácie a následne tretí experiment testoval účinnosť algoritmu po pridaní operátora kríženie. Na tabuľke 7.5 je zobrazené základné nastavenie evolučného algoritmu pre datasety MNIST a CIFAR10 v rámci overovania účinnosti evolučných operátorov. Tabuľka 7.6 zobrazuje nastavenie pravdepodobností mutácie a kríženia pre všetky tri experimenty.

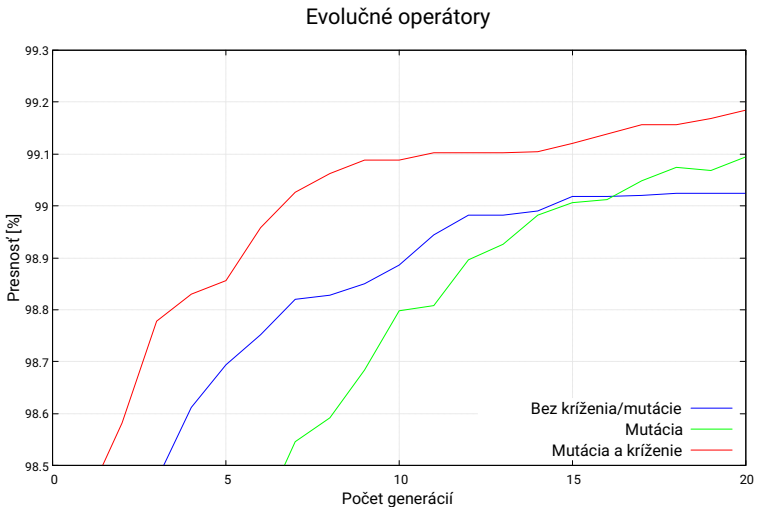
Na grafe 7.4 je možné vidieť efektivitu navrhnutých evolučných operátorov na datasete MNIST. Napriek tomu, že rozdiely medzi jednotlivými priebehmi nie sú veľmi veľké, na

Nastavanie evolučného algoritmu		
Parametre/Dataset	**MNIST**	**CIFAR10**
Počet generácií	20	20
Veľkosť populácie	8	12
Selekcia preživších	podľa veku	podľa veku
Hranica dospelosti	6	8
Veľkosť turnaja	2	2

Tabuľka 7.5: Tabuľka zobrazuje základné nastavenie evolučného algoritmu pre datasety MNIST a CIFAR10.

Pravdepodobnosti evolučných operátorov			
Parametre/Experimenty	**Exp. č. 1**	**Exp. č. 2**	**Exp. č. 3**
Pravdepodobnosť mutácie	0%	80%	50%
Pravdepodobnosť kríženia	0%	0%	35%

Tabuľka 7.6: Tabuľka zobrazuje pravdepodobnosti aplikácie evolučných operátorov pre všetky tri experimenty.

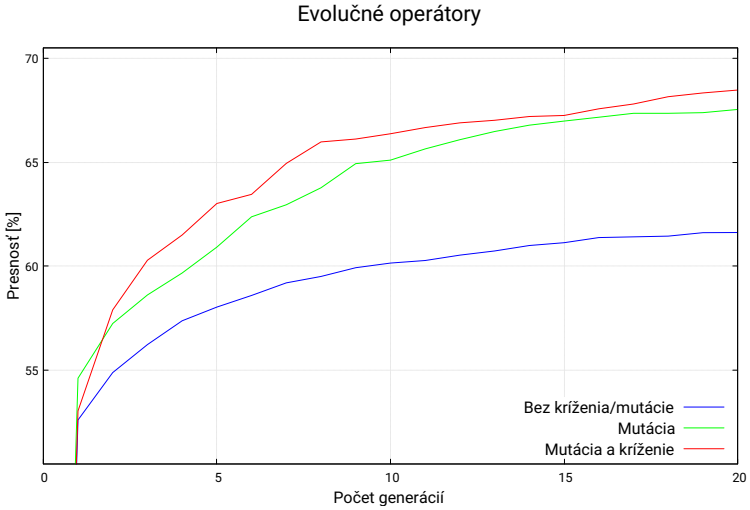


Obr. 7.4: Graf zobrazujúci účinnosť jednotlivých evolučných operátorov na datasete MNIST. Osa x zobrazuje počet generácií EA a osa y zobrazuje presnosť najlepšej siete. (Priemer z piatich priebehov.)

tak malom datasete je možné ich považovať za úspech. Evolučný algoritmus je s presnosťou 99,02% na testovacom datasete bez evolučných operátorov na úrovni náhodného generovania. Zaujímavosťou je pomerné nízka priemerná hĺbka nájdených sietí – 3,4 vrstvy bez výslednej zoštíhľujúcej a plne prepojenej vrstvy. Príčinou je pravdepodobne neschopnosť algoritmu prehľadávať úspešnejšie a hlbšie siete bez aplikovania evolučných operátorov.

Úspešnosť EA len s operátorom mutácie sa ukázala takmer o 0,1 percentuálneho bodu vyššia s priemernou hĺbkou 4,6 vrstiev, a to aj napriek horšej východiskovej presnosti (horšia úspešnosť počiatočnej, náhodne vygenerovanej generácie). Priebeh EA s oboma navrhnutými operátormi sa ukázal najpresnejší s priemernou úspešnosťou na testovacích dátach až

99,19%. Oba navrhnuté evolučné operátory sa teda ukázali pomerne prospešné v hľadaní presných architektúr pomocou EA.



Obr. 7.5: Graf zobrazujúci účinnosť jednotlivých evolučných operátorov na datasete CIFAR10. Osa x zobrazuje počet generácií EA a osa y zobrazuje presnosť najlepšej siete. (Priemer z piatich priebehov.)

Graf 7.5 zobrazuje výsledky experimentov pre dataset CIFAR10. Graf potvrdzuje a ešte zvýrazňuje výsledky experimentov na datasete MNIST. EA bez evolučných operátorov dosahuje výsledky opäť len na úrovni náhodného algoritmu. Po pridaní operátora mutácie dosahuje evolučný algoritmus oveľa vyššiu presnosť – celkovo až o 5,9 percentuálneho bodu v rámci 20 generácií. Evolučný algoritmus so zapnutým operátorom kríženia dosiahol presnosť až 68,46%. Tento výsledok je oproti EA používajúcemu len operátor mutácie ešte o 0,804 percentuálneho bodu vyšší a potvrdzuje tak miernu úspešnosť kríženia v rámci evolučného algoritmu.

Celkovo je možné zhodnotiť, že oba navrhnuté evolučné operátory predstavujú prínos k evolučnému algoritmu.

7.5 Účinnosť rozdeľovania do druhov

Aby bola overená účinnosť konceptu rozdeľovania do druhov, boli navrhnuté nasledovné experimenty:

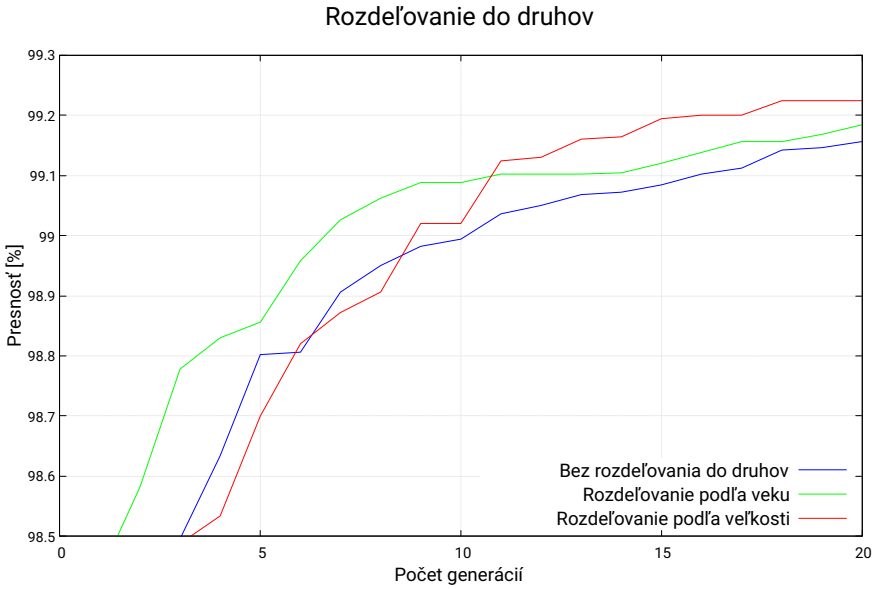
- Experiment č. 1: selekcia bez rozdeľovania do druhov – výber k najlepších jedincov z množiny rodičov a potomkov
- Experiment č. 2: rozdeľovanie do druhov na základe veku
- Experiment č. 3: rozdeľovanie do druhov na základe veľkosti

Všetky tri experimenty boli vyhodnotené na datasetoch MNIST a CIFAR10. Základná konfigurácia EA je zobrazená v tabuľke 7.7.

Výsledky uvedených experimentov zobrazené na grafe 7.6 pre dataset MNIST ukazujú len mierne vyššiu úspešnosť mechanizmov rozdeľovania do druhov oproti klasickej selekcii.

Nastavanie evolučného algoritmu		
Parametre/Dataset	**MNIST**	**CIFAR10**
Počet generácií	20	20
Veľkosť populácie	8	12
Hranica dospelosti	6	8
Veľkosť turnaja	2	2
Pravdepodobnosť kríženia	35%	35%
Pravdepodobnosť mutácie	50%	50%

Tabuľka 7.7: Tabuľka zobrazuje základné nastavenie evolučného algoritmu pre datasety MNIST a CIFAR10.



Obr. 7.6: Graf zobrazujúci účinnosť jednotlivých mechanizmov rozdeľovania do druhov na datasete MNIST. Osa x zobrazuje počet generácií EA a osa y zobrazuje presnosť najlepšej siete. (Priemer z piatich priebehov.)

Rozdeľovanie do druhov podľa veku malo v rámci datasetu MNIST najrýchlejšiu konvergenciu k vysokým úspešnostiam, avšak pomalší rast v neskorších fázach evolučného priebehu. V posledných generáciách sa ukázalo ako výhodné rozdeľovanie do druhov podľa veľkosti s priemernou úspešnosťou 99,22%. Úspechom tohto mechanizmu bola najlepšia nájdená sieť s presnosťou 99,36%. Táto sieť bola najúspešnejšia aj v rámci všetkých vykonaných experimentov na datasete MNIST.

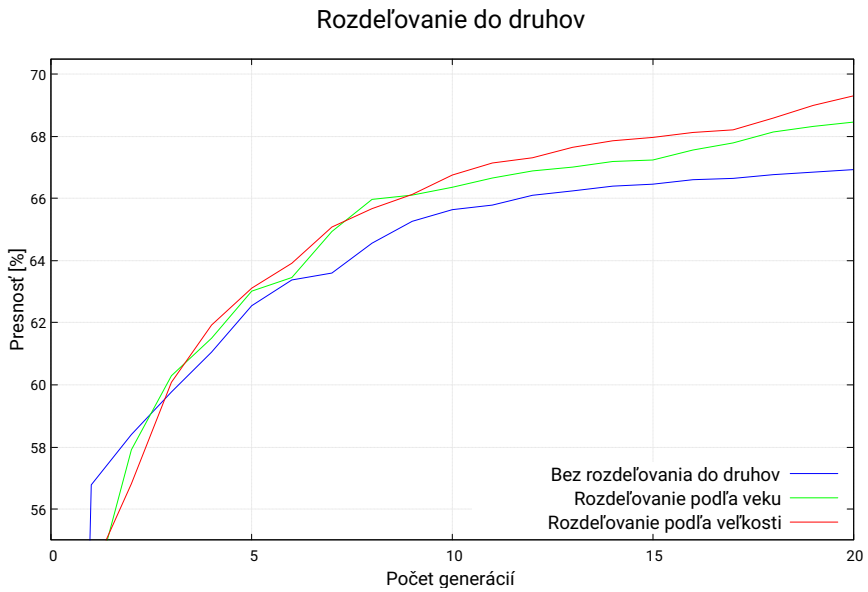
Tabuľka 7.8 zobrazuje prehľad výsledkov zo všetkých troch experimentov na datasete MNIST. Napriek miernemu náskoku rozdeľovania do druhov je rozdiel medzi všetkými typmi selekcie pomerne zanedbateľný. Prekvapivé sú aj veľmi podobné vlastnosti najlepších nájdených sietí v rámci všetkých experimentov. Vysvetlením tohto javu je pravdepodobne opäť jednoduchosť datasetu MNIST – evolučný algoritmus veľmi neťaží zo zvýhodňovania hlbších sietí, pretože daný problém dokážu riešiť aj pomerne jednoduché siete. Rozdiel medzi presnosťami najlepších nájdených sietí zo všetkých behov pre každý druh selekcie ukazuje,

že klasická selekcia je najmenej stabilná a ako jediná nedokázala nájsť vo všetkých behoch sieť presnú aspoň 99%.

Obe metódy rozdeľovania do druhov mali teda celkovo veľmi porovnateľné výsledky a na tak malom datasete ako MNIST je ich efekt takmer zhodný.

Úspešnosť rozdeľovania do druhov na datasete MNIST			
Vlastnosti/Experimenty	**Exp. č. 1**	**Exp. č. 2**	**Exp. č. 3**
Priemerná úspešnosť	99,156%	99,184%	99,124%
Priemerná hĺbka	4,8	4,4	4,4
Priemerný počet parametrov	45 288	58 300,6	58 316,4
Priemerná doba výpočtu [procesoro-hodiny]	448	449,6	451,2
Rozptyl presnosti	98,99%-99,36%	99,11%-99,24%	99,09%-99,36%

Tabuľka 7.8: Tabuľka zobrazuje vlastnosti najlepších nájdených sietí v rámci vyhodnocovania efektívnosti rôznych mechanizmov rozdeľovania do druhov na datasete MNIST. Experiment č. 1 predstavuje priebeh s klasickou selekciou, experiment č. 2 predstavuje rozdeľovanie podľa veku a experiment č. 3 reprezentuje rozdeľovanie podľa veľkosti.



Obr. 7.7: Graf zobrazujúci účinnosť jednotlivých mechanizmov rozdeľovania do druhov na datasete CIFAR10. Osa x zobrazuje počet generácií EA a osa y zobrazuje presnosť najlepšej siete.

Výsledky experimentu na datasete CIFAR10 – zobrazené na grafe 7.7 – potvrdzujú závery vyvodené z výsledkov pre dataset MNIST. Rozdeľovanie do druhov na základe hĺbky siete malo opäť najvyššiu úspešnosť. Len mierne menej úspešné sa opäť preukázalo rozdeľovanie podľa veku, avšak pre tento dataset nepreukázalo rýchlejšiu konvergenciu v počiatočných fázach. Tradičný prístup k selekcii bez použitia mechanizmu rozdeľovania do druhov mal najhorší priebeh. Rozdiel nie je výrazný, avšak na grafe je vidieť nastupujúca stagnácia klasickej selekcie v posledných generáciách. Je preto na mieste predpoklad, že po vykonaní experimentov na väčší počet generácií by sa rozdiel stále zvyšoval. Vlastnosti nájdených

sietí pre jednotlivé mechanizmy selekcie na datasete CIFAR10 zobrazuje tabuľka 7.9. Ukázalo sa, že rozdeľovanie podľa hĺbky má na väčšom datasete očakávaný efekt a nájdené najlepšie siete sú skutočne aj výrazne hlbšie oproti ostatným prístupom. Negatívom tohto prístupu je však značne vyšší výpočtový čas. Rozptyl presností najlepších sietí ukazuje, že najstabilnejším mechanizmom je rozdeľovanie podľa hĺbky – najlepšie siete zo všetkých behov majú aspoň 67% – avšak najúspešnejšia sieť celkovo bola v tomto prípade (oproti datasetu MNIST) nájdená pomocou rozdeľovania podľa veku. Najväčší rozptyl zaznamenala – takisto ako pri datasete MNIST – klasická selekcia bez rozdeľovania do druhov, ktorá ako jediná v niektorých behoch nedokázala prekonať hranicu 60%. Výsledky teda potvrdzujú prínos oboch mechanizmov rozdeľovania do druhov.

Úspešnosť rozdeľovania do druhov na datasete CIFAR10			
Vlastnosti/Experimenty	**Exp. č. 1**	**Exp. č. 2**	**Exp. č. 3**
Priemerná úspešnosť	66,93%	68,46%	69,304%
Priemerná hĺbka	3,6	4,6	5,6
Priemerný počet parametrov	114 108,2	173 106	168 961,4
Priemerná doba výpočtu [procesoro-hodiny]	1152	1142,4	1315,2
Rozptyl presnosti	59,93%-72,96%	62,02%-73.05%	67,48%-71,21%

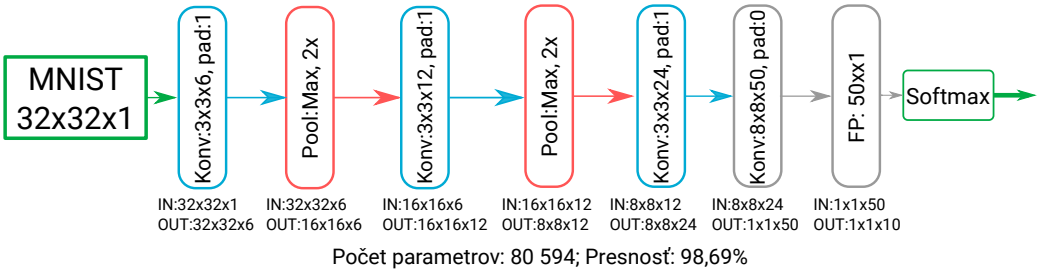
Tabuľka 7.9: Tabuľka zobrazuje vlastnosti najlepších nájdených sietí v rámci vyhodnocovania efektívnosti rôznych mechanizmov rozdeľovania do druhov na datasete CIFAR10. Experiment č. 1 predstavuje priebeh s klasickou selekciou, experiment č. 2 predstavuje rozdeľovanie podľa veku a experiment č. 3 reprezentuje rozdeľovanie podľa veľkosti.

Výsledky navrhnutých experimentov, ktorých účelom bolo zhodnotenie účinnosti mechanizmov rozdeľovania do druhov navrhnutých v tejto práci, preukázali prínos týchto mechanizmov oproti klasickej selekcii, najmä čo sa týka stability jednotlivých behov. Metóda rozdeľovania podľa veku sa ukázala ako najpresnejšia pre oba testované datasety. Avšak vzhľadom na to, že obe dve metódy rozdeľovania do druhov vykazovali rozdielne vlastnosti pre rôzne datasety a ich výsledky boli porovnateľné (rozdiely v úspešnostiach boli len veľmi malé), je nutné vykonať viac experimentov, aby bolo možné spoľahlivo určiť, ktorý z nich je pre evolučný algoritmus prínosnejší.

7.6 Optimalizácia parametrov pomocou EA

Navrhnutý framework, ktorý je predmetom tejto práce, môže slúžiť aj ako optimalizátor vopred definovaných sietí. V tomto prípade je možné do fitness hodnoty započítať okrem presnosti danej siete aj počet parametrov, ktoré daná sieť obsahuje. Výsledkom sú potom siete, ktoré sú presné, ale zároveň aj čo najefektívnejšie.

V rámci experimentu, ktorého úlohou bolo vyhodnotiť efektivitu popísanej vlastnosti, bola počiatočná populácia inicializovaná vopred definovanou a predtrénovanou sieťou. Jej štruktúra a parametre sú zobrazené na obrázku 7.8. Táto sieť spolu po počiatočnom natrénovaní na päť epoch dosahovala na datasete MNIST presnosť 98,69% a počet jej parametrov bol 80 594. Následne bol spustený evolučný algoritmus s koeficientom počtu parametrov nastaveným na hodnotu 1. Tabuľka 7.10 zobrazuje ostatné nastavenia evolučného algoritmu a tabuľka 7.11 zobrazuje nastavenie váh jednotlivých mutácií. Celý experiment bol realizovaný na datasete MNIST.



Obr. 7.8: Architektúra siete, ktorou bola inicializovaná počiatočná populácia v rámci optimalizačného experimentu. IN a OUT predstavujú rozmery vstupných, resp. výstupných dát. Parametre konvolučnej vrstvy sú popísané vo formáte: *výška filtra*×*šírka filtra*×*počet filtrov*, *aplikácia výplne (padding):* 1 – áno, 0 – nie. Parametre pool vrstvy sú popísané vo formáte: *Typ podvzorkovania, veľkosť podvzorkovania.*

Nastavanie evolučného algoritmu	
Parametre/Dataset	**MNIST**
Počet generácií	20
Veľkosť populácie	5
Hranica dospelosti	5
Veľkosť turnaja	2
Selekcia preživších	podľa veku
Pravdepodobnosť kríženia	35%
Pravdepodobnosť mutácie	50%
Koeficient počtu parametrov	1

Tabuľka 7.10: Tabuľka zobrazuje základné nastavenie evolučného algoritmu pre datasety MNIST pre optimalizačný experiment.

Nastavanie váh jednotlivých mutácií EA	
Váhy/Dataset	**MNIST**
Pridaj vrsvu	30
Odober vrstvu	15
Resetuj vahy	5
Modifikuj vrstvu	40
Uprav hĺbku plne prepojenej vrstvy	15
Uprav učiaci koeficient	15

Tabuľka 7.11: Tabuľka zobrazuje nastavenie váh, podľa ktorých sa aplikujú jednotlivé mutácie na datasetoch MNIST pre optimalizačný experiment. Váhy v tomto kontexte slúžia na určovanie pravdepodobnosti, s akou sa aplikuje daná mutácia. Pravdepodobnosť, že sa aplikuje niektorá mutácia, je teda hodnota jej váhy podelená súčtom váh všetkých mutácií.

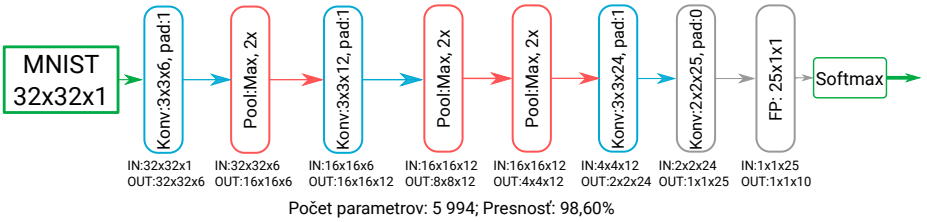
Výsledky experimentu sú popísané v tabuľke 7.12 pre päť evolučných behov s rovnakým počiatočným nastavením a rovnakou architektúrou inicializačnej siete. Namerané výsledky ukazujú značný potenciál navrhnutého algoritmu hľadať efektívnejšie verzie vopred definovaných a predtrénovaných sietí. Priemerná úspora parametrov zo všetkých piatich behov bola až 90,15% za cenu straty presnosti len 0,32 percentuálneho bodu. Najlepšia nájdená

sieť pritom dosiahla úsporu parametrov až 92,62%, pričom stratila 0,09 percentuálneho bodu z počiatočnej presnosti. Dosiahnuté výsledky sa teda dajú považovať za úspech a prinášajú sľubný základ pre ďalší výskum v tejto oblasti.

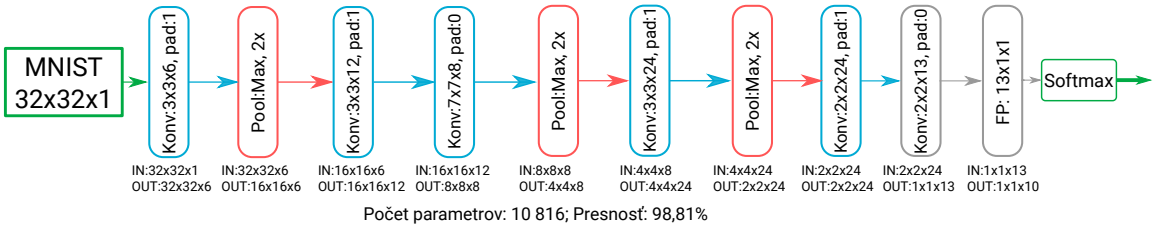
Výsledky optimalizačného experimentu						
Parametre/Experimenty	**Beh 1**	**Beh 2**	**Beh 3**	**Beh 4**	**Beh 5**	**Priemer**
Fitness hodnota	124,73	124,04	123,56	123,30	123,28	123,78
Počet vrstiev	6	8	9	8	7	7,6
Presnosť	98,6%	98,53%	97,12%	98,81%	98,81%	98,37
Počet parametrov	5 944	7 298	4 720	10 816	10 898	7 935,2
Úspora počtu parametrov	92,62%	90,94%	94,14%	86,15%	86,48%	90,15%

Tabuľka 7.12: Tabuľka zobrazuje pravdepodobnosti aplikácie evolučných operátorov pre všetky tri experimenty.

Na obrázkoch 7.9a a 7.9b sú zobrazené dve z nájdených architektúr v rámci optimalizačného experimentu. Z architektúr je možné pozorovať schopnosť EA aplikovať podvzorkovacie vrstvy na správnych miestach, a tým docieliť zníženie počtu parametrov bez veľkej straty presnosti. Nájdené siete sú tiež pomerne hlboké (priemerne až 7,6 vrstiev bez zoštíhľujúcej a plne prepojenej vrstvy), čo ukazuje pomerne úspešné použitie evolučných operátorov.



(a) Beh 1



(b) Beh 4

Obr. 7.9: Príklady niektorých architektúr nájdených v rámci optimalizačného experimentu. IN a OUT predstavujú rozmery vstupných resp. výstupných dát. Parametre konvolučnej vrstvy sú popísané vo formáte: *výška filtra*×*šírka filtra*×*počet filtrov*, *aplikácia výplne (padding):* 1 – áno, 0 – nie. Parametre pool vrstvy sú popísané vo formáte: *Typ podvzorkovania, veľkosť podvzorkovania.*

7.7 Zhodnotenie výsledkov

Najlepší výsledok navrhnutého riešenia na datasete MNIST je úspešnosť *99,36%*, zatiaľ čo najlepšie riešenie zaznamenané v literatúre je *99,79%*. Z uvedeného výsledku vyplýva, že

framework má schopnosť hľadať riešenia blížiace sa aktuálnym riešeniam na menších datasetoch. Najlepší zaznamenaný výsledok na datasete CIFAR10 je *96,53%*, s čím sa najlepší výsledok frameworku *73,05%* zatiaľ nemôže porovnávať. Experimenty však ukázali potenciál, že nameraný výsledok nie je konečný a po ďalších komplexných experimentoch by bolo s veľkou pravdepodobnosťou možné ho ešte navýšiť.

Z pohľadu aktuálnych výsledkov z oblasti neuroevolúcie je možné prácu konceptuálne porovnávať s riešeniami *Large-scale evolution* [37] a CNN-GA [48]. Je však nutné poznamenať, že obe z uvedených prác boli realizované na masívnych paralelných GPU systémoch, a preto sa dá očakávať, že experimenty navrhnuté v tejto práci sú významne menšieho rozsahu. Výsledky preto môžu byť porovnávané len približne alebo len z konceptuálneho hľadiska. Uvedené práce realizovali svoje experimenty nad datasetom CIFAR10. Práca [37] dosiahla úspešnosť *94,66%*, kde počet parametrov najlepšej siete dosahoval 5,6 milióna a výpočet trval 2 750 GPU dní. Výsledky v [48] boli významne lepšie, keď dosiahli úspešnosť *95,22%* pri 2,9 milióna parametrov po výpočte trvajúcom 35 GPU dní. Najlepšia sieť na datasete CIFAR10 v tejto práci dosiahla presnosť len *73,05%* s počtom parametrov 85 365. Na nájdenie tejto architektúry pomocou EA navrhnutého v tejto práci bolo potrebných približne 1 152 procesoro-hodín. Napriek veľkému rozdielu v úspešnosti tejto práce a aktuálne najlepšími riešeniami v tejto oblasti, výsledky dosiahnuté v rámci obmedzených výpočtových možností dostupných pri realizácii experimentov ukazujú veľký potenciál navrhnutého riešenia. Nájdené riešenia sú pomerne presné a ich nájdenie bolo dosiahnuté za relatívne krátky výpočtový čas. Na potvrdenie tejto hypotézy a relevantné porovnanie s aktuálnymi výsledkami je však nutné vykonať experimenty porovnateľného rozsahu ako v uvedených prácach.

V tejto práci bol tiež navrhnutý koncept optimalizácie parametrov pomocou neuroevolúcie, ktorý bol následne experimentálne vyhodnotený. Tento koncept je v rámci neuroevolúcie značne inovatívny a v rámci dostupnej literatúry zameranej na neuroevolúciu sa nepodarilo nájsť prácu s podobným zameraním. Z tohto dôvodu nie je možné porovnanie s aktuálnymi výsledkami. Výsledky experimentu však ukázali, že navrhnutý EA je schopný – pomocou evolučných operátorov a špeciálne navrhnutej fitness funkcie – hľadať (z pohľadu počtu parametrov) efektívnejšie varianty vopred definovaných sietí.

7.7.1 Navrhované pokračovanie

Výsledky experimentov popísaných v tejto kapitole ukázali, že neuroevolučné prístupy majú potenciál automatizovať proces návrhu konvolučných neurónových sietí. K preskúmaniu plného potenciálu riešenia navrhnutého v tejto práci by však bolo nutné uskutočniť ešte mnoho ďalších a komplexnejších experimentov, ktoré by svojím objemom aj výpočtovými nárokmi výrazne prekročili rozsah tejto práce.

Veľkým prínosom pre navrhnutý EA by mohlo byť dôkladnejšie preskúmanie vplyvu evolučných parametrov (veľkosť populácie, veľkosť turnaja, pravdepodobnosti evolučných operátorov a pod.) na celkovú účinnosť evolučného algoritmu. Tieto experimenty si však vzhľadom na obrovskú výpočtovú náročnosť vyžadujú veľké množstvo prostriedkov a z tohto dôvodu neboli kompletne preskúmané v rámci tejto práce. Vzhľadom na veľkú závislosť navrhnutého evolučného algoritmu na vygenerovaných riešeniach môže byť napríklad veľmi výhodné používať početnejšie populácie, aby sa predišlo skĺznutiu do lokálneho maxima. Z toho istého dôvodu by mohlo byť prínosné rozšíriť navrhnuté evolučné operátory o ďalšiu funkcionalitu a tiež dôslednejšie vyladiť pravdepodobnosti ich aplikácie. Príkladom môže byť zavedenie reziduálneho prepojenia v rámci mutačných operátorov, ktoré sa zdá byť

účinné v hlbokých neurónových sieťach, ako bolo ukázané napríklad aj v [48]. Z pohľadu dedenia váh a rýchlosti celého evolučného procesu by mohlo mať výrazne pozitívny vplyv upravenie operátorov tak, aby boli čo najmenej invazívne z pohľadu presnosti kandidátnej siete. Takýmto spôsobom by mohla byť zvýšená účinnosť dedenia váh a tým znížený počet potrebných trénovacích cyklov v rámci evolúcie. V tejto oblasti je však potrebný ďalší výskum, aby bol dôkladne preskúmaný efekt možných úprav siete na ich presnosť.

Najväčším problémom neuroevolúcie však zostáva veľmi vysoká výpočtová náročnosť a tým aj otázka jej využiteľnosti. Veľký dôraz na paralelizáciu a GPU výpočet by mohol byť čiastočným riešením tohto problému, avšak nové techniky podobné alebo nadväzujúce na dedenie váh majú potenciál byť veľmi úspešné.

Možným smerom vývoja oblasti neuroevolúcie by mohlo byť aj výraznejšie využitie vlastnosti evolučných algoritmov hľadať – vďaka selekčnému tlaku a špeciálne navrhnutým fitness funkciám – veľmi efektívne siete najmä s ohľadom na počet parametrov, rýchlosť trénovania alebo rýchlosť inferencie. Neuroevolúcia by tak mohla slúžiť ako doplnkový nástroj pri návrhu veľmi komplexných hlbokých neurónových sietí.

Kapitola 8

Záver

Cieľom práce bolo spracovať teoretické poznatky z oblasti prepojenia evolučných algoritmov a neurónových sietí, súhrnne označovaného ako neuroevolúcia, a následne navrhnúť a implementovať využitie evolučného algoritmu na návrh konvolučných neurónových sietí. Stanovené ciele boli v rámci tejto práce naplnené.

V prvej časti práce boli podrobne zhrnuté základné teoretické poznatky z oblasti konvolučných neurónových sietí a evolučných algoritmov, potrebné na úplné pochopenie celej práce. Následne bol popísaný vývoj v oblasti neuroevolúcie spolu s jeho najaktuálnejšími výsledkami. Hlavným prínosom práce bol návrh a implementácia frameworku určeného na automatizáciu návrhu architektúr konvolučných neurónových sietí. Framework pozostáva z dvoch základných komponentov: modifikovaného evolučného algoritmu a obálky na prácu s CNN knižnicou. Evolučný algoritmus využíva špeciálne navrhnuté evolučné operátory mutácie a kríženia slúžiace na prehľadávanie stavového priestoru. Takisto boli predstavené nové metódy selekcie preživších, a to pomocou rozdeľovania do druhov na základe spoločných vlastností.

Všetky spomenuté princípy boli experimentálne vyhodnotené na datasetoch MNIST a CIFAR10 – v odbornej literatúre dobre spracovaných úlohách klasifikácie obrazu. Dosiahnuté výsledky potvrdzujú potenciál neuroevolúcie hľadať úspešné a pomerne efektívne konvolučné neurónové siete. Najlepšia sieť nájdená evolučným algoritmom na datasete MNIST dosiahla presnosť 99,36%, zatiaľ čo najlepšia architektúra na datasete CIFAR10 dosiahla presnosť 73,05%. Úspešnosť navrhnutých evolučných operátorov a mechanizmov bola experimentálne overená. Oba operátory sa ukázali byť pomerne prospešné v hľadaní úspešných architektúr CNN. Navrhnuté mechanizmy rozdeľovania do druhov sa zdajú byť efektívne najmä pri väčších datasetoch, ako je napríklad CIFAR10. Pri datasete MNIST ich vplyv nie je tak výrazný.

V práci bol tiež predstavený inovatívny spôsob optimalizácie (z pohľadu počtu parametrov výslednej siete) vopred definovaných architektúr CNN pomocou evolučného algoritmu. Základom takejto optimalizácie je zahrnutie počtu parametrov – t.j. naučených váh a biasov – do celkovej fitness hodnoty jedinca. V rámci experimentu, ktorého úlohou bolo overiť účinnosť EA v optimalizácii parametrov, bola počiatočná populácia inicializovaná vopred definovanou sieťou. Úlohou evolučného algoritmu bolo potom nájsť z pohľadu počtu parametrov efektívnejšie varianty inicializačnej siete bez veľkej straty presnosti. Priemerná redukcia počtu parametrov z piatich evolučných behov dosahovala až 90,15% za stratu presnosti len 0,32 percentuálneho bodu.

Možné pokračovania práce spočívajú v hľadaní optimálnej konfigurácie frameworku na odhalenie plného potenciálu riešenia implementovaného v rámci tejto práce. Na nájde-

nie takéhoto nastavenia je nutné ďalšie empirické ladenie parametrov pomocou mnohých ďalších komplexných experimentálnych priebehov. Nevýhodou je však vysoká výpočtová náročnosť, ktorá je spoločnou vlastnosťou väčšiny neuroevolučných prístupov. Nové experimenty si teda vyžadujú veľké množstvo výpočtových prostriedkov. Možným pokračovaním tejto práce teda môže byť aj rozširovanie alebo návrh nových techník na zrýchlenie evolučného procesu. Nová funkcionalita v rámci evolučných operátorov, ako napríklad reziduálne spojenia, môže zase pomôcť v unikaní z lokálneho maxima a prehľadávaní hlbších a komplexnejších architektúr.

Literatúra

[1] Abadi, M.; Agarwal, A.; Barham, P.; aj.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, software available from tensorflow.org.
URL https://www.tensorflow.org/

[2] Bengio, Y.: Practical recommendations for gradient-based training of deep architectures. *CoRR*, ročník abs/1206.5533, 2012, 1206.5533.
URL http://arxiv.org/abs/1206.5533

[3] Bäck, T.: Self-Adaptation in Genetic Algorithms. In *Proceedings of the First European Conference on Artificial Life*, MIT Press, 1992, s. 263–271.

[4] Corne, D. W.; Lones, M. A.: Evolutionary Algorithms. *CoRR*, ročník abs/1805.11014, 2018, 1805.11014.
URL http://arxiv.org/abs/1805.11014

[5] De Jong, K.; Fogel, D.; Schwefel, H.-P.: *A history of evolutionary computation*. 01 1997, s. A2.3:1–12.

[6] Dianati, M.; Song, I.; Treiber, M.: An Introduction to Genetic Algorithms and Evolution. 2002.

[7] Eiben, A. E.; Smith, J. E.: *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, druhé vydanie, 2015, ISBN 3662448734, 9783662448731.

[8] Eiben, E.; Rudolph, G.: Theory of evolutionary algorithms: a bird's eye view. *Theoretical Computer Science*, ročník 229, č. 1, 1999: s. 3 – 9, ISSN 0304-3975, doi:https://doi.org/10.1016/S0304-3975(99)00089-4.

[9] Fernando, C.; Banarse, D.; Reynolds, M.; aj.: Convolution by Evolution: Differentiable Pattern Producing Networks. *CoRR*, ročník abs/1606.02580, 2016, 1606.02580.
URL http://arxiv.org/abs/1606.02580

[10] Floreano, D.; Mattiussi, C.: *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. The MIT Press, 2008, ISBN 0262062712, 9780262062718.

[11] Fogel, L.; Owens, A.; Walsh, M.: Artificial Intelligence Through Simulated Evolution. 1966.
URL https://books.google.cz/books?id=QMLaAAAAMAAJ

[12] Fukushima, K.: Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, ročník 36, 1980: s. 193–202.

[13] Gurney, K.: *An Introduction to Neural Networks*. Bristol, PA, USA: Taylor & Francis, Inc., 1997, ISBN 1857286731.

[14] Habibi Aghdam, E., Hamed; Jahani Heravi: *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. Cham: Springer, 2017, ISBN 9783319575490.

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

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

[17] Holland, J.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. 1975.
URL https://books.google.cz/books?id=JE5RAAAAMAAJ

[18] Ioffe, S.; Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, ročník abs/1502.03167, 2015, 1502.03167.
URL http://arxiv.org/abs/1502.03167

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

[20] Keller, D. F. D. B., James M.; Liu: *Fundamentals of Computational Intelligence : Neural Networks, Fuzzy Systems, and Evolutionary Computation*. Hoboken: John Wiley & Sons, Incorporated, 2016, ISBN 9781119214403.

[21] Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992, ISBN 0-262-11170-5.

[22] Krizhevsky, A.; Nair, V.; Hinton, G.: CIFAR-10 (Canadian Institute for Advanced Research).
URL http://www.cs.toronto.edu/~kriz/cifar.html

[23] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, editácia F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger, Curran Associates, Inc., 2012, s. 1097–1105.
URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[24] Lecun, Y.; Bengio, Y.; Hinton, G.: Deep learning. *Nature*, ročník 521, č. 7553, 2015: s. 436–444, ISSN 1476-4687.
URL http://search.proquest.com/docview/1684430894/

[25] LeCun, Y.; Boser, B. E.; Denker, J. S.; aj.: Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, ročník 1, č. 4, 1989: s. 541–551, doi:10.1162/neco.1989.1.4.541.
URL https://doi.org/10.1162/neco.1989.1.4.541

[26] Lecun, Y.; Bottou, L.; Bengio, Y.; aj.: Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998, s. 2278–2324.

[27] LeCun, Y.; Cortes, C.: MNIST handwritten digit database. 2010.
URL http://yann.lecun.com/exdb/mnist/

[28] Lehman, J.; Miikkulainen, R.: Neuroevolution. *Scholarpedia*, ročník 8, č. 6, 2013: str. 30977, doi:10.4249/scholarpedia.30977, revision #137053.

[29] Marczyk, A.: Genetic algorithms and evolutionary computation. *The Talk Origins Archive: http://www. talkorigins/faqs/genalg/genalg. html*, 2004.

[30] McCulloch, W. S.; Pitts, W.: Neurocomputing: Foundations of Research. kapitola A Logical Calculus of the Ideas Immanent in Nervous Activity, Cambridge, MA, USA: MIT Press, 1988, ISBN 0-262-01097-6, s. 15–27.

[31] Miikkulainen, R.; Liang, J. Z.; Meyerson, E.; aj.: Evolving Deep Neural Networks. 2017, 1703.00548.
URL http://arxiv.org/abs/1703.00548

[32] Miller, G. F.: Designing Neural Networks using Genetic Algorithms. In *ICGA*, 1989.

[33] Nomi, T.: TinyDNN. https://github.com/tiny-dnn/tiny-dnn, 2016.

[34] Parra, J.; Trujillo, L.; Melin, P.: Hybrid Back-propagation Training with Evolutionary Strategies. *Soft Comput.*, ročník 18, č. 8, August 2014: s. 1603–1614, ISSN 1432-7643, doi:10.1007/s00500-013-1166-8.
URL http://dx.doi.org/10.1007/s00500-013-1166-8

[35] Petrowski, A.; Ben-Hamida, S.: *Evolutionary Algorithms*. Computer Engineering: Metaheuristics, Wiley, 2017, ISBN 9781848218048.
URL https://books.google.cz/books?id=fvRRCgAAQBAJ

[36] Rachenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. 1973.
URL https://books.google.de/books?id=-WAQAQAAMAAJ

[37] Real, E.; Moore, S.; Selle, A.; aj.: Large-Scale Evolution of Image Classifiers. *arXiv e-prints*, Marec 2017: arXiv:1703.01041, 1703.01041.

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

[39] Salimans, T.; Ho, J.; Chen, X.; aj.: Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[40] Schmidhuber, J.: Deep Learning in Neural Networks. *Neural Netw.*, ročník 61, č. C, Január 2015: s. 85–117, ISSN 0893-6080, doi:10.1016/j.neunet.2014.09.003.
URL http://dx.doi.org/10.1016/j.neunet.2014.09.003

[41] Schwefel, H.: Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: Mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie. 1976.
URL https://books.google.cz/books?id=AP5ZAAAACAAJ

[42] Seide, F.; Agarwal, A.: CNTK: Microsoft's Open-Source Deep-Learning Toolkit. 08 2016, s. 2135–2135, doi:10.1145/2939672.2945397.

[43] Sekanina, L.: Evolutionary hardware design. In *VLSI Circuits and Systems v*, ročník 8067, International Society for Optics and Photonics, 2011, str. 806702.

[44] Simonyan, K.; Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, ročník abs/1409.1556, 2014, 1409.1556.
URL http://arxiv.org/abs/1409.1556

[45] Stanley, K. O.; D'Ambrosio, D. B.; Gauci, J.: A Hypercube-based Encoding for Evolving Large-scale Neural Networks. *Artif. Life*, ročník 15, č. 2, Apríl 2009: s. 185–212, ISSN 1064-5462, doi:10.1162/artl.2009.15.2.15202.
URL http://dx.doi.org/10.1162/artl.2009.15.2.15202

[46] Stanley, K. O.; Miikkulainen, R.: Evolving Neural Networks Through Augmenting Topologies. *Evol. Comput.*, ročník 10, č. 2, Jún 2002: s. 99–127, ISSN 1063-6560, doi:10.1162/106365602320169811.
URL http://dx.doi.org/10.1162/106365602320169811

[47] Such, F. P.; Madhavan, V.; Conti, E.; aj.: Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *CoRR*, ročník abs/1712.06567, 2017, 1712.06567.
URL http://arxiv.org/abs/1712.06567

[48] Sun, Y.; Xue, B.; Zhang, M.; aj.: Automatically Designing CNN Architectures Using Genetic Algorithm for Image Classification. *CoRR*, ročník abs/1808.03818, 2018, 1808.03818.
URL http://arxiv.org/abs/1808.03818

[49] Sze, V.; Chen, Y.; Yang, T.; aj.: Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *CoRR*, ročník abs/1703.09039, 2017, 1703.09039.
URL http://arxiv.org/abs/1703.09039

[50] Szegedy, C.; Liu, W.; Jia, Y.; aj.: Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.
URL http://arxiv.org/abs/1409.4842

Príloha A

Obsah priloženého pamäťového média

Pamäťové médium priložené k tejto práci obsahuje všetky zdrojové kódy implementované v rámci tejto práce, potrebné knižnice a tento text spolu s jeho zdrojovými súbormi.

Obsah média je nasledovný:

- `Text/` – text tejto práce
  - `badan.pdf` – text práce vo formáte pdf
  - `latex-src/` – zdrojové kódy práce v latexu
- `Src/` – zdrojové kódy práce
  - `evolve/` – vlastné zdrojové kódy frameworku implementovaného v rámci tejto práce
  - `tiny-dnn/` – knižnica TinyDNN
  - `README.md` – stručný manuál k prekladu a spusteniu implementovaného riešenia
- `results.ods` – všetky namerané výsledky