BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2020
http://excel.fit.vutbr.cz

Evoluční návrh konvolučních neuronových sítí

Michal Piňos

Abstrakt
Cílem této práce je návrh a implementace programu pro automatizovaný návrh konvolučních neuronových sítí (*CNN*) s využitím evolučních výpočetních technik. Z praktického hlediska tento přístup redukuje potřebu lidského faktoru při tvorbě CNN, a tak eliminuje zdlouhavý a namáhavý proces návrhu. Tato práce využívá speciální formu genetického programování nazývanou kartézské genetické programování, které pro zakódování řešeného problému využívá grafovou reprezentaci. Tato technika umožňuje uživateli parametrizovat proces hledání CNN, a tak se zaměřit na architektury zajímavé z pohledu použitých výpočetních jednotek, přesnosti či počtu parametrů. Navrhovaný přístup byl otestován na standardizované datové sadě CIFAR-10, která je často využívána výzkumníky pro srovnání výkonnosti jejich CNN. Prvotní experimenty ukázaly, že vytvořená implementace (využívající GPU akceleraci) je schopna vytvořit či vylepšit přesnost CNN. Výsledkem experimentů, kdy bylo pro trénování k dispozici pouze několik epoch, byla řešení s přesností 64.5 % a počtem parametrů 146K při využití základních vrstev a řešení s přesností 74.5 % s počtem parametrů 475K při využití reziduálních vrstev. Záměrem těchto experimentů bylo dokázat funkčnost implementovaného programu a *proof-of-concept* navržené metody.

Klíčová slova: Neuroevoluce — Neuronové sítě — Evoluční výpočetní techniky

Přiložené materiály: N/A

*xpinos03@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

1. Úvod

Návrh architektury konvolučních neuronových sítí dnes hraje důležitou roli ve výzkumu umělé inteligence (konkrétně v oboru rozpoznávání obrázků). Návrh nových architektur CNN vyžaduje velké úsilí, mnoho zkušeností a pokusů. Tento proces je tak velmi namáhavý a ne vždy efektivní. Z tohoto důvodu je nutné přemýšlet o nových způsobech tvorby těchto sítí, které by nebyly tak složité a náročné, a přitom poskytovaly výsledky minimálně srovnatelné s ručně navrženými sítěmi. Technika automatizovaného hledání architektur umělých neuronových sítí, nazývaná NAS (Neural Architecture Search) [1], je v dnešní době velmi populární [2, 3] a již dala vzniknout hlubokým neuronovým sítím překonávajícím ty odborně navržené [4, 5, 6].

Tvorba nových architektur CNN rovněž obnáší problém optimalizace navrhované sítě z hlediska počtu parametrů, hyperparametrů (počtu vrstev, skrytých neuronů, ...) a potřebného výpočetního výkonu. Tento požadavek je dán tím, že velikost výsledné sítě (ať už co se týká počtu parametrů nebo hyperparametrů) hraje důležitou roli z pohledu požadavků na zdroje zařízení, na kterém poběží. V dnešní době mobilních a vestavěných zařízení s omezenými zdroji je tento požadavek zcela pochopitelný. Najít optimalizační metodu, která by byla schopna tento úkol zcela vyřešit, je ale prakticky nemožné, a tak je často nutné uchýlit se k různým heuristickým metodám. Ty jsou schopny na úkor přesnosti, úplnosti či efektivity nalézt takové řešení, které se co nejvíce blíží zadaným požadavkům. Jednu skupinu takovýchto metod tvoří evoluční výpočetní techniky [7], založené na poznatcích ze světa biologické evoluce.

Cílem této práce je využití evolučních výpočetních technik jako optimalizační metody při automatizovaném návrhu nových architektur konvolučních neuronových sítí. Pro tyto účely byla zvolena speciální technika nazývaná kartézské genetické programování [8]. S využitím této techniky se lze zaměřit na hledání

zajímavých architektur z pohledu přesnosti, počtu parametrů či technických požadavků na výslednou síť.

Výsledkem této práce je implementace programu pro automatizovaný návrh konvolučních neuronových sítí, využívající uživatelem definované výpočetní prostředky (v našem případě GPU), určených k řešení specifických problémů.

2. Teoretické základy

2.1 Konvoluční neuronové sítě

Konvoluční neuronové sítě (Convolutional Neural Networks, zkráceně *CNN*) tvoří skupinu hlubokých neuronových sítí, které se specializují na zpracování obrazových dat. Moderní konvoluční neuronové sítě se skládají ze čtyř základních typů vrstev – *konvoluční*, *aktivační*, *seskupující* a *plně propojené*. Jejich možné umístění v architektuře CNN je uvedeno na obrázku 1.

Konvoluční vrstvy jsou zodpovědné za extrakci užitečných rysů ze vstupních obrázků. K tomu využívají operaci konvoluce, která realizuje aplikaci filtrů (označovaných též jako jádro či kernel) na vstupní obrázek. Ten má zpravidla několik kanálů (v případě RGB obrázku 3), přičemž konvoluce je prováděna zvlášť pro každý z nich.

Aktivační vrstvy plní důležitou roli, jelikož do modelu umělých neuronových sítí vnáší prvek nelinearity. Bez aktivačních vrstev by sebesložitější vícevrstvá neuronová sítí představovala jen o něco složitější, jednovrstvou neuronovou síť. Aktivační vrstvy jsou zpravidla umisťovány za lineární operace jako je konvoluce či bázová funkce perceptronu. Nejpoužívanější aktivační funkcí je usměrněná lineární jednotka (Rectified Linear Unit, zkráceně *ReLU*) [9].

Účelem seskupujících vrstev je zachovat pouze důležité informace z obrázku a zbavit se nepodstatných detailů, jako je například umístění či natočení extrahovaných rysů. Seskupující vrstvy rovněž redukují prostorové rozměry vnitřní reprezentace vstupního obrázku (*downsampling*). Tato vrstva se nejčastěji umisťuje za konvoluční, přičemž pracuje zvlášť pro každý vstupní kanál.

Poslední část klasifikačních CNN tvoří plně propojená neuronová vrstva, určená ke klasifikaci vstupních obrázků do výstupních tříd. Před tuto část je zpravidla nutné vložit zplošťující vrstvu, která transformuje vnitřní tří dimenzionální reprezentaci obrázku (s dimenzemi výška, šířka a počet kanálů) do podoby vhodné pro zpracování plně propojenou vrstvou.

2.2 Evoluční algoritmy

Evoluční algoritmy (zkráceně *EA*) jsou stochastické, heuristicky založené algoritmy inspirované biologickou evolucí. Jejich primárním úkolem je optimalizace problémů, pro které neexistuje žádný efektivní algoritmus[1]. Základní premisou konvergence EA je, že jedinci, kteří splňují určité požadavky (jsou nadprůměrně zdatní), budou mít potomky, a tak budou moci předat své užitečné vlastnosti do další generace.

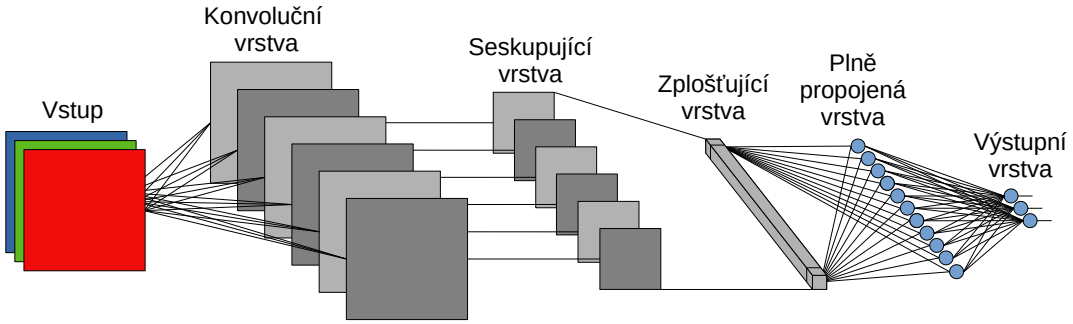
EA pracují nad množinou *jedinců*, kteří představují nějaká validní řešení daného problému. Evoluce probíhá postupně po iteracích, nazývaných *generace*, do té doby, dokud nejsou splněny *ukončující podmínky*. Mezi ty patří například nalezení řešení s dostatečnou kvalitou nebo vyčerpání maximálního počtu generací. Množina jedinců, se kterými algoritmus pracuje v rámci jedné generace, se nazývá *populace*. V každé generaci je s využitím operátoru *selekce* vybráno několik jedinců, kteří se budou podílet na tvorbě nové populace. Z vybraných jedinců jsou pomocí operátorů *křížení* a *mutace* vyrobeni *potomci*, kteří tvoří populaci následující generace. Jedinci jsou obvykle uloženi v lineární struktuře nazývané *genotyp*. Jednotlivé části genotypu se nazývají *geny*, přičemž konkrétní forma genu se nazývá *alela*. Skutečná forma jedince se nazývá *fenotyp*.

2.3 Kartézské genetické programování

Kartézské genetické programování (Cartesian Genetic Programming, zkráceně *CGP*) [8] je speciální forma genetického programování, ve které jednotliví jedinci představují programy reprezentované acyklickými orientovanými grafy (Directed Acyclic Graph, zkráceně *DAG*). Uzly DAG jsou tvořeny výpočetními jednotkami, uspořádanými ve dvoudimenzionální mřížce (odtud název *"kartézské"*).

CGP má tři hlavní, uživatelem definované, parametry, počet řádků r, počet sloupců c a levels-back l. První dva parametry udávají maximální počet výpočetních uzlů, který je $r \times c$. Nastavením parametru l uživatel udává, z kolika předchozích sloupců může daný uzel brát své vstupy. Tento parametr tak ukládá omezení při tvorbě propojení jednotlivých uzlů. Pokud $l = 1$, tak může každý uzel brát vstup z výstupů uzlů z předchozího sloupce nebo primárních vstupů. Při $l = 2$ může uzel brát svoje vstupy z výstupů uzlů ve dvou z levé strany sousedících sloupců nebo z primárních vstupů. Změnou těchto parametrů lze docílit různých topologií grafu. Speciálním případem je situace, kdy $r = 1$ a $l = c$. Při tomto nastavení může vzniknout libovolně propojený DAG, omezený pouze počtem uzlů, který je roven c.

[1] Efektivním algoritmem se z pohledu teorie složitosti rozumí algoritmus, který je schopný poskytnout řešení v polynomiálním čase.



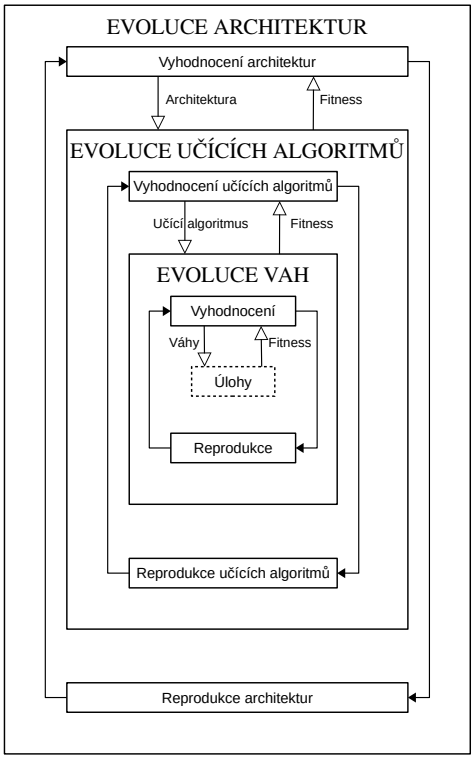
Obrázek 1. Ukázka jednoduché architektury CNN obsahující jednu konvoluční vrstvu, následovanou seskupující vrstvou, zplošťovací vrstvou, plně propojenou vrstvou a výstupní vrstvou.

Uživatel dále definuje počet vstupů a výstupů výpočetní mřížky, která se skládá z funkčních uzlů, přičemž funkce uzlu je vybrána z uživatelem definované množiny funkcí. Genotyp jedince má konstantní velikost a skládá se z genů jednotlivých uzlů a výstupních genů. Geny uzlů jsou dále rozděleny na funkční geny (udávající funkci uzlu) a propojovací geny, které stanovují, odkud bude uzel brát svůj vstup. Výstupní geny pak určují, z výstupů kterých uzlů bude tvořen výstup grafu.

Proces evoluce je u CGP řízen algoritmem inspirovaným evoluční strategií $(\mu + \lambda)$, nejčastěji ve formě $(1+4)$ [8]. V tomto jednoduchém algoritmu značí μ počet nejlepších jedinců, kteří jsou v každé generaci vybráni jako rodiče. Z nich je následně pomocí mutace vytvořeno λ potomků. V drtivé většině případů používá CGP pouze operaci mutace a nejčastěji pracuje s jedním rodičem a λ potomky v každé generaci.

2.4 Neuroevoluce

Neuroevoluce se zabývá využitím evolučních výpočetních technik při návrhu umělých neuronových sítí. Evolučně navržené neuronové sítě se označují jako EANN (Evolutionary Artificial Neural Networks) [10]. Proces evoluce je v kontextu tvorby umělých neuronových sítí využit v několika instancích, přičemž asi nejčastější je využití evoluce pro návrh architektury neuronové sítě. Zbylé dvě instance se zabývají evolucí vah nebo ještě obecněji návrhem učícího algoritmu. Evoluční algoritmy určené k evoluci architektur a vah neuronových sítí se označují jako TWEANN (Topology and Weight Evolving Artificial Neural Network). Nejznámějším takovýmto algoritmem je genetický algoritmus NEAT (NeuroEvolution of Augmenting Topologies), představený autorem Kennethem O. Stanleym v roce 2002 [11]. TWEANN algoritmy se nejčastěji dělí na konstruktivní a destruktivní [10]. Konstruktivní algoritmy začínají s minimální funkční architekturou a v průběhu evoluce přidávají další prvky (uzly nebo spojení) do výsledné architektury. Destruktivní algoritmy naopak začínají s architekturou s maximálním množstvím prvků a v průběhu evoluce náhodné prvky eliminují. Obecné schéma neuroevoluce je uvedeno na obrázku 2.



Obrázek 2. Univerzální schéma neuroevoluce. Převzato z [10].

3. Související práce

Tato sekce obsahuje krátké shrnutí a porovnání již existujících prací, zabývajících se tematikou evolučního návrhu konvolučních neuronových sítí. Zvláštní pozornost je zde věnována řešením zaměřujícím se na evoluční návrh architektur CNN s využitím CGP.

Využití CGP při návrhu umělých neuronových sítí se označuje jako *CGPANN* (Cartesian Genetic Programming of Artificial Neural Networks) [12]. Jedná se o neuroevoluční algoritmus, který využívá přímého zakódování architektury, vah a funkcí do genotypu.

CGPANN tak představuje TWEANN algoritmus, který při procesu evoluce využívá výhradně operátoru mutace. Na rozdíl od genetických TWEANN algoritmů je CGPANN konstruktivní i destruktivní algoritmus.

Příklad úspěšného využití CGP při návrhu CNN je uveden v práci [13]. Její autoři vytvořili metodu nazvanou CGP-CNN, která používá CGP kódování jedinců, schopné reprezentovat architekturu a propojení výsledné CNN. Výhodnou tohoto kódování je jeho flexibilita, jelikož umožňuje reprezentaci různě hlubokých sítí i využití *skip* propojení. Mimo jednoduché výpočetní uzly CGP mřížky, jako jsou konvoluční a seskupující vrstva, autoři uvedli i složitější moduly, nazvané *ConvBlock* a *ResBlock*. Tyto moduly představují komplexnější výpočetní uzly vytvořené z jednodušších operací.

ConvBlock se skládá z konvoluce s krokem 1 a ReLU aktivační funkce. Parametry tohoto modulu jsou velikost filtru (3×3 nebo 5×5) a počet výstupních kanálů (32, 64 nebo 128).

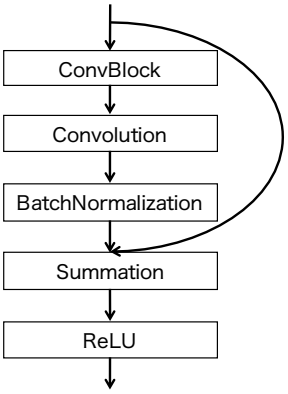
ResBlock je složen z konvoluce, konkatenace a ReLU aktivační funkce. *Skip* propojení přeskakuje konvoluční část tohoto modulu a přivádí nezměněný vstup modulu do konkatenačního uzlu, kde dochází ke konkatenaci s výsledkem konvoluce. Na výsledek konkatenace je následně použita aktivační funkce ReLU. Tento modul obsahuje parametry shodné jako u modulu ConvBlock. Architektura modulu ResBlock je uvedena na obrázku 3.

Experimenty s touto metodou její autoři rozdělili na dva případy, podle toho, zda byly použity ConvBlock nebo ResBlock moduly. Experimenty byly prováděny na datasetu CIFAR-10. Nejlepší vytvořená architektura v případě použití modulů ConvBlock dosáhla přesnosti 93.25 % s 1.52M parametrů. Architektura vytvořená z modulů ResBlock dosáhla přesnosti až 94.02 % s 1.68M parametrů. Provedené experimenty ukázaly, že metoda CGP-CNN je schopna vytvořit architektury porovnatelné s těmi ručně navrženými. V případě využití ResBlock modulů výsledná síť překonala i jedny z nejlepších ručně navržených sítí [13].

3.1 Srovnání existujících řešení

Tabulka 1 obsahuje srovnání přesnosti a počtu parametrů konvolučních neuronových sítí na datasetu CIFAR-10, navržených různými evolučními algoritmy. Pro srovnání obsahuje tabulka i současnou *state-of-the-art* odborně navrženou architekturu CNN nazvanou DenseNet [14].

Model NAS [4] využívá rekurentní neuronovou síť LSTM s posilovaným učením pro generování architektur CNN. Model nazvaný CNN-GA [15] je založen na využití genetického programování pro návrh architektur CNN. Skvělých výsledků CNN-GA dosáhlo díky využití takzvaných *skip* propojení, které přeskakují některé vrstvy v architektuře. Díky tomu dochází k lepší zpětné propagaci chyby do počátečních vrstev, a tak k rychlejší konvergenci. Posledním zkoumaným modelem v této sekci je CGP-CNN [13], využívající metodu CGP pro automatické hledání optimálních architektur CNN.



Obrázek 3. Architektura modulu ResBlock. BatchNormalization provádí normalizaci hodnot po procesu konvoluce. Převzato z [13].

Srovnání algoritmů pro CIFAR-10			
Název modelu	Přesnost (%)	Počet parametrů	Doba běhu (GPU dny)
DenseNet [14]	96.54	25.6M	–
NAS [4]	93.99	2.5M	22 400
CGP-CNN [13]	94.02	1.68M	27
CNN-GA [15]	95.22	2.9M	35

Tabulka 1. Srovnání parametrů CNN, které byly získány pomocí různých přístupů (jeden GPU den znamená, že algoritmus strávil jeden den na jednom GPU).

4. Navržené řešení

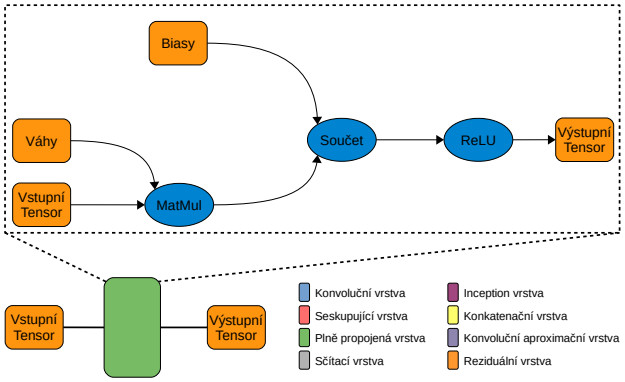
Pro automatizovaný návrh CNN architektur byla zvolena metoda CGP, kde jednotlivé uzly CGP mřížky představují určité vrstvy CNN modelu. CGP tak hledá architekturu sítě CNN, přičemž učení vah je ponecháno na algoritmu stochastického gradientního sestupu s využitím zpětného šíření chyby.

Pro práci s neuronovými sítěmi byla zvolena knihovna TensorFlow. Hlavními dvěma motivátory pro volbu této knihovny byla podpora akcelerace procesu učení na GPU a fakt, že výzkumná skupina *Evolvable Hardware* pracuje na rozšíření *tf-approximate*[2] knihovny TensorFlow o využití aproximačních jednotek.

Implementovaný program vytváří z výpočetních grafů knihovny TensorFlow jednotlivé vrstvy konvo-

[2] https://github.com/ehw-fit/tf-approximate

lučních neuronových sítí. Na obrázku 4 je uveden příklad implementace jedné vrstvy CNN, konkrétně plně propojené vrstvy, pomocí výpočetního grafu. Jak obrázek naznačuje, tak obdobným způsobem lze realizovat libovolnou vrstvu, jako je například konvoluční, seskupující, sčítací či konkatenační. Z těchto jednoduchých vrstev lze vytvořit i složitější výpočetní celky, jako je reziduální blok (popsaný v sekci 3) nebo tzv. *Inception* modul [16].



Obrázek 4. Příklad implementace plně propojené vrstvy pomocí TensorFlow výpočetního grafu.

Funkční uzly CGP mřížky obsahují implementaci individuálních vrstev CNN architektury, jak je uvedeno na obrázku 5. Každý DAG tak představuje nějakou validní architekturu CNN. Nastavením počtu parametrů CGP mřížky může uživatel ovlivnit, jaké architektury bude navržený program vytvářet. Počet sloupců mřížky c udává maximální hloubku navrhovaných CNN. Pomocí počtu řádků r lze zase dát programu na výběr z několika alternativ funkčních uzlů v každém sloupci. Parametr $l-$back pak udává omezení navrhovaných CNN z pohledu možných propojení jednotlivých vrstev. Tento parametr rovněž umožňuje vytváření užitečných *skip* propojení.

Dalším užitečným vstupem, který může uživatel programu poskytnout, je definice samotné výpočetní mřížky CGP. Vhodně rozmístěné funkční uzly ve výpočetní mřížce mohou totiž ulehčit hledání efektivních CNN architektur.

Primárním vstupem CGP je obrázek, vyjádřený jako 3D tensor, s dimenzemi výška, šířka a počet kanálů vstupního obrázku. CGP pracuje s tensory s hodnotami typu float. Primárním výstupem CGP je 1D tensor (vektor), vyjadřující příslušnost vstupního obrázku do jedné z výstupních tříd. Výstupní tensor pro tyto účely používá kódování 1 z N.

4.1 Mutace

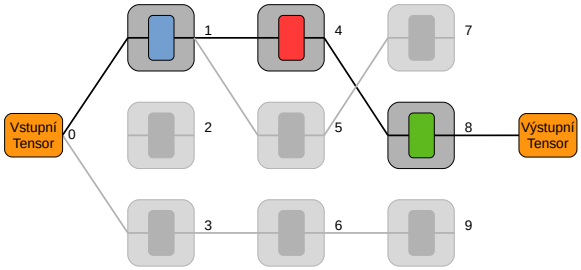
Implementovaný program pro zakódování jedinců nevyužívá klasické kódování chromozomu jako sekvence celých čísel, jak je tomu u většiny genetických algoritmů. V tomto případě je jedinec reprezentován přímo jako DAG, tedy dvojice $G = (U, E)$, kde U značí množinu uzlů a E množinu hran. Mutace je prováděna na grafu G, přičemž při mutaci dochází pouze ke změně propojení uzlů. V terminologii genetického programování to znamená, že mutace mění pouze propojovací a výstupní geny chromozomu. Funkční geny zůstávají nezměněny.

Mutace probíhá tak, že je náhodně vybrán uzel k mutaci $n \in U$. Pro uzel n jsou odstraněny všechny vstupní hrany, tedy $E = E \setminus \{(x,n) \mid x \in U\}$. Následně je spočítána množina

$$L(n) = \{u \mid u \in U \text{ a } u \text{ je uzel respektující parametr l-back pro vybraný uzel } n\}$$

Z množiny $L(n)$ je následně náhodně vybrán jeden uzel $m \in L(n)$. Do množiny hran E je poté přidána nová hrana (m,n), tedy $E = E \cup (m,n)$. Pokud uzel n reprezentuje funkci s vyšší aritou než je 1, tak je proces přidání nové hrany opakován.

Mutace jedince je prováděna do té doby, dokud není vytvořen aktivní podgraf spojující vstupní uzel s výstupním.



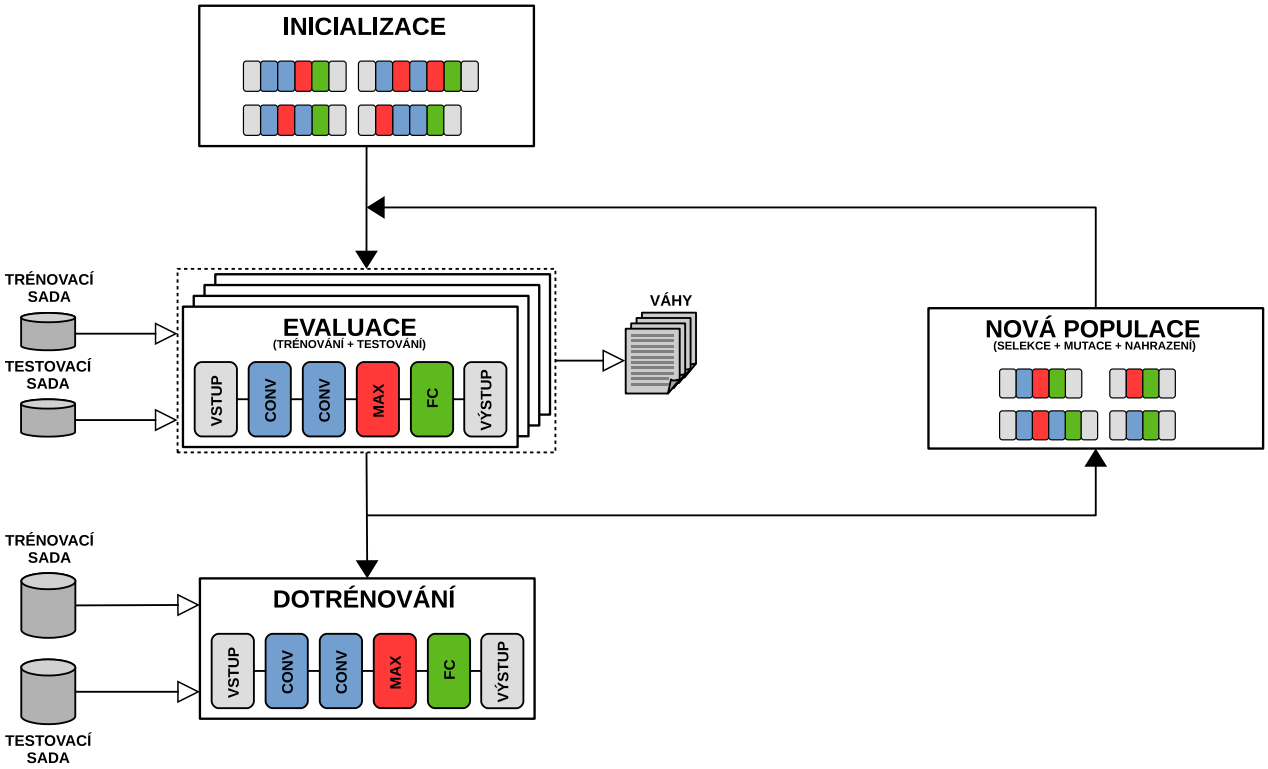
Obrázek 5. Ukázka DAG v CGP mřížce představující nalezené řešení, skládající se z konvoluční vrstvy (modrá), seskupující vrstvy (červená) a plně propojené vrstvy (zelená).

4.2 Prohledávací algoritmus

Implementovaný program využívá prohledávací algoritmus založený na evoluční strategii (1 + 4). Diagram prohledávacího algoritmu je uveden na obrázku 6.

Na úvod jsou vygenerovány 4 kandidátní řešení. Algoritmus dále vstupuje do smyčky, která provádí evaluaci každého jedince. Ta se skládá z trénování jedinců pomocí náhodně vybrané podmnožiny trénovací datové sady. Následné testování jedinců je opět prováděno na náhodně vybrané podmnožině testovací datové sady. Všichni jedinci v určité generaci jsou trénováni a testování na stejných, náhodně vybraných podmnožinách originálních datových sad. Tento postup byl převzat z [13].

Během evaluace je jedincům přiřazena fitness hodnota podle jejich přesnosti a počtu parametrů. Výpočet



Obrázek 6. Diagram prohledávacího algoritmu. Jak je zde znázorněno, tak všichni jedinci v každé generaci jsou trénováni a testováni s využitím náhodně vybrané podmnožiny originální datové sady. Naučené váhy každého jedince jsou uloženy do souboru. Po skončení algoritmu je nejlepší jedinec (do)trénován na originální datové sadě.

fitness je proveden pomocí vzorce

$$f = a \times \left(\frac{k}{\log(p)+1} + 1 \right), \qquad (1)$$

kde a značí přesnost, p počet parametrů a koeficient k udává vliv počtu parametrů na výslednou fitness hodnotu. Tato fitness funkce byla převzata z [17].

Z ohodnocených jedinců je poté vybrán nejlepší jedinec – *rodič*. Pokud je v současné populaci více jedinců s nejlepší fitness hodnotou, tak je vybrán jedinec, který ještě nebyl rodičem. V případě, že je takovýchto jedinců více, je zvolen náhodný z nich.

S využitím operátoru mutace, popsaném v podsekci 4.1, jsou z rodiče vytvořeni potomci, kteří tvoří populaci následující generace.

Hlavní smyčka prohledávacího algoritmu končí, až je dosaženo maximálního počtu generací. Nejlepší nalezené řešení je následně (do)trénováno s využitím originální datové sady.

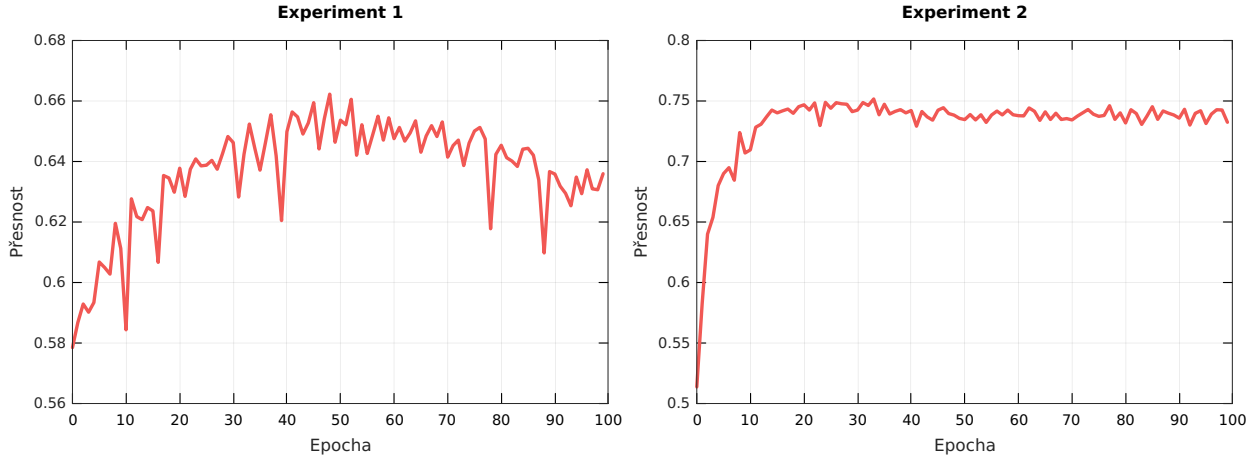
4.3 Uložení nejlepších vah

Z popisu prohledávacího algoritmu plyne, že v každé generaci jsou všichni jedinci podrobeni trénování před tím, než je určena jejich přesnost. Do další generace se však dostanou jen někteří jedinci, z čehož plyne, že trénování některých jedinců bylo zbytečné a jejich naučené váhy se zahodí. Další negativní vlastností tohoto přístupu je to, že noví jedinci budou muset začínat od začátku, s náhodně vygenerovanými váhami.

Implementovaný program tento problém řeší tak, že každý funkční uzel v CGP výpočetní mřížce si pamatuje naučené váhy nejlepšího jedince, který tento uzel využíval. Noví jedinci si pak při trénování načtou tyto váhy a nebudou tak muset začínat od znovu. Jelikož jsou ale rozměry váhového tensoru každého funkčního uzlu určeny tím, ke kterému funkčnímu uzlu je připojen, tak se mohou rozměry váhového tensoru pro ten samý uzel lišit. V tomto případě je váhový tensor buď oříznut nebo doplněn náhodnými hodnotami tak, aby odpovídal potřebným rozměrům.

5. Experimenty

Experimenty byly prováděny na datové sadě CIFAR-10, která se skládá z 60 000 barevných obrázků o velikosti 32×32, rozdělených do 10 tříd. Datová sada je dále rozdělena na trénovací sadu obsahující 50 000 vzorků a testovací sadu o velikosti 10 000 vzorků. Experimenty byly spuštěny na stroji se čtyřmi grafickými kartami NVIDIA GTX 1080 (Pascal), 8GB RAM, které byly použity pro akceleraci procesu učení CNN. Všechny ostatní výpočty byly prováděny na CPU.



(a) Přesnost během finálního dotrénování nejlepšího řešení, nalezeného v experimentu 1.

(b) Přesnost během finálního dotrénování nejlepšího řešení, nalezeného v experimentu 2.

Obrázek 7. Grafy experimentů.

5.1 Experiment 1

První experiment měl za cíl navrhnout architekturu CNN s použitím jednoduchých výpočetních bloků, jako jsou konvoluční, seskupující, konkatenační, sčítací a plně propojené vrstvy.

5.1.1 Nastavení programu

CGP mřížka obsahovala 6 řádků a 31 sloupců s parametrem l-back nastaveným na 8. Mřížka byla vytvořena tak, že sloupce blíže primárnímu vstupu obsahovaly konvoluční vrstvy s větší velikostí filtru a menším počtem výstupních kanálů. Čím blíže se pak sloupce blížily výstupu CGP mřížky, tím se velikosti filtrů zmenšovaly a počty výstupních kanálů zvyšovaly. Tento postup byl zvolen z toho důvodu, že hlubší vrstvy si mohou dovolit mít více výstupních kanálů, jelikož pracují s menšími vstupy. Konvoluční sloupce CGP mřížky byly příležitostně proloženy sloupci se seskupujícími, konkatenačními a sčítacími vrstvami. Poslední sloupec se skládal z plně propojených vrstev s počtem neuronů 64, 128, 256, 512, 1024 a 2048.

Počet generací prohledávacího algoritmu byl nastaven na 50, přičemž algoritmus v každé generaci pracoval s populací velikosti 4. V každé generaci bylo z trénovací datové sady náhodně vybráno 10 000 vzorků a z testovací datové sady 5 000 náhodných vzorků. Pomocí této mini-trénovací sady bylo provedeno trénování každého jedince po dobu 10 epoch. Přesnost každého kandidátního řešení pak bylo určeno evaluací na mini-testovací sadě. Parametr k fitness funkce 1 byl nastaven na 0.5.

Po skončení prohledávacího algoritmu bylo nejlepší nalezené řešení (do)trénováno na plné trénovací sadě po dobu 100 epoch. Výsledná přesnost pak byla spočítána pro originální testovací sadu.

5.2 Experiment 2

Cílem druhého experimentu bylo navrhnout architekturu CNN s využitím ResBloku, popsaném na obrázku 3.

5.2.1 Nastavení programu

CGP mřížka obsahovala 6 řádků a 15 sloupců s parametrem l-back nastaveným na 4. Jednotlivé sloupce CGP mřížky obsahovaly ResBloky, konvoluční, seskupující, sčítací a konkatenační vrstvy s různými parametry. Poslední sloupec se skládal z plně propojených vrstev s počtem neuronů 64, 128, 256, 512, 1024 a 2048.

Nastavení prohledávacího algoritmu tohoto experimentu bylo shodné s nastavením v experimentu 1.

5.3 Výsledky

Přesnosti nejlepších nalezených řešení obou experimentů jsou ukázány na obrázku 7.

Výsledkem prvního experimentu bylo řešení s přesností 64.5 % a počtem parametrů 146178. Graf 7a zobrazuje průběh učení nejlepšího řešení při finálním dotrénování, které trvalo 100 epoch. Jak je vidět, tak přesnost dosáhla až k 66 % a přibližně od 50. epochy se snižovala. To ukazuje, že při procesu učení začalo docházet k přeučení (overfitting), kdy model začal ztrácet schopnost generalizace.

Výsledkem druhého experimentu bylo řešení s přesností 74.5 % a počtem parametrů 475722. Graf 7b zobrazuje průběh učení nejlepšího řešení při finálním dotrénování, které trvalo 100 epoch. Jak je vidět, tak přesnost dosáhla až k 75 % a přibližně od 30. epochy se opět začala pozvolna snižovat. To opět ukazuje na přeučení.

6. Závěr

Cílem této práce bylo navrhnout a implementovat neuroevoluční algoritmus pro automatizovaný návrh architektur konvolučních neuronových sítí. Toho bylo dosaženo použitím speciální techniky CGP. Pro práci s CNN byla zvolena knihovna TensorFlow, umožňující akceleraci výpočtů na GPU. Pro ověření funkčnosti implementovaného programu byly provedeny dva experimenty na datové sadě CIFAR-10, které dosáhly přesnosti 64.5 % s 146178 parametry a 74.5 % s počtem parametrů 475722.

Výsledkem této práce je program, implementující evoluční návrh architektur konvolučních neuronových sítí. Implementovaný program rovněž umožňuje uživateli specifikovat si různé požadavky a omezení na navrhované CNN. Uživatel tak může programu říci, jaké výpočetní jednotky může používat nebo jaká může být minimální či maximální hloubka sítě. Z praktického pohledu je tato možnost velmi užitečná, uvážíme-li, že bychom například hledali architekturu CNN pro nějaké zařízení s omezenými výpočetními prostředky.

6.1 Pokračování práce

Styl, jakým je program navržen a implementován podporuje možnost rozšíření CGP mřížky o nové, výkonnější funkční bloky. Implementovaný program může být rovněž použit pro návrh takovýchto bloků. Další alternativou je inicializace CGP nejlepšími známými CNN a hledání kompromisu mezi přesností a velikostí sítě.

Jiným podstatným rozšířením tohoto programu je využití aproximačních jednotek, které na úkor přesnosti snižují požadavky na příkon nebo plochu na čipu. Rozšíření *tf-approximate* implementuje tyto aproximační jednotky, ale nepodporuje proces učení. Tyto jednotky by ale mohly být lehce integrovány do implementovaného programu tak, že v procesu evaluace jedince by trénování probíhalo na přesných jednotkách, zatímco testování na těch aproximovaných. Výsledkem programu by pak byla naučená architektura CNN, která by mohla být na čipu implementována aproximačními jednotkami a tak ušetřit cenné zdroje vestavěného zařízení.

Poděkování

Rád bych poděkoval svému vedoucímu práce panu profesoru Ing. Lukáši Sekaninovi, Ph.D. za odborné vedení, konzultace, čas a cenné rady při tvorbě této práce. Dále bych rád poděkoval doktoru Ing. Vojtěchu Mrázkovi a docentu Ing. Jiřímu Jarošovi, Ph.D. za zprostředkování přístupu na školní výpočetní server.

Literatura

[1] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. *Neural Architecture Search: A Survey*. Journal of Machine Learning Research, 20(55):1–21, 2019.

[2] Md Ashiqur Rahman. *Neural Architecture Search (NAS) - The Future of Deep Learning*, June 2019. URL: https://towardsdatascience.com/neural-architecture-search-nas-the-future-of-deep-learning-c99356351136.

[3] Kenneth O. Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. *Designing neural networks through neuroevolution*. Nature Machine Intelligence, 1(1):24–35, 2019.

[4] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *International Conference on Learning Representations*, 2017.

[5] Lewei Yao, Hang Xu, Wei Zhang, Xiaodan Liang, and Zhenguo Li. *SM-NAS: Structural-to-Modular Neural Architecture Search for Object Detection*, 2019. URL: https://arxiv.org/abs/1911.09929.

[6] Ning Zhu. *Neural Architecture Search for Deep Face Recognition*, 2019. URL: http://arxiv.org/abs/1904.09523.

[7] Patrick Siarry, Alain Petrowski, and Sana Ben Hamida. *Metaheuristics*, chapter Evolutionary Algorithms. Springer International Publishing, 2016.

[8] J. F. Miller. *Cartesian Genetic Programming*, pages 17–34. Natural Computing Series. Springer Berlin Heidelberg, 2011.

[9] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[10] Xin Yao. *Evolving artificial neural networks*. Proceedings of the IEEE, 87(9):1423–1447, September 1999.

[11] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[12] Maryam Mahsal Khan and Gul Muhammad Khan. A novel neuroevolutionary algorithm:

Cartesian genetic programming evolved artificial neural network (cgpann). In *Proceedings of the 8th International Conference on Frontiers of Information Technology*, New York, NY, USA, 2010. Association for Computing Machinery.

[13] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 497–504, New York, NY, USA, 2017. Association for Computing Machinery.

[14] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, July 2017.

[15] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G. Yen. Automatically designing CNN architectures using genetic algorithm for image classification. 2018.

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[17] Filip Badáň and Lukáš Sekanina. Optimizing convolutional neural networks for embedded systems by means of neuroevolution. In *Theory and Practice of Natural Computing*, pages 109–121. Springer International Publishing, Listopad 2019.