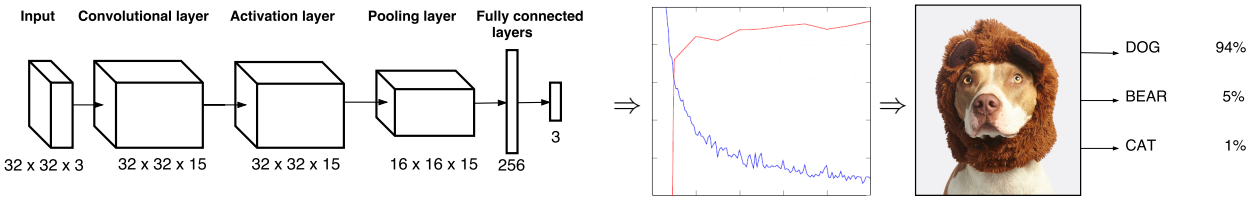
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2018
http://excel.fit.vutbr.cz

TypeCNN – knihovna pro práci s konvolučními neuronovými sítěmi podporující různé datové typy

Petr Rek*



Abstrakt

Cílem výzkumné práce, která je popsána v tomto příspěvku, je vytvořit knihovnu pro práci s konvolučními neuronovými sítěmi – od návrhu, přes trénování až po inferenci. Knihovna sama o sobě je psána velmi jednoduše a obsahuje také množství funkcionality. Specifickým rozšířením je pak nezávislost na datovém typu, od čehož je také odvozen název knihovny – *TypeCNN*. Každá vrstva sítě může mít různé typy pro váhy, inferenci a učení. Mezi vrstvami jsou tyto typy nezávislé. Za účelem vyhodnocení je do knihovny přidán i vlastní datový typ simulující aritmetiku v pevné řádové čárce a knihovna tedy dokáže simulovat provádění sítě v této reprezentaci. A to s různými nastaveními počtu bitů před a za řádovou čárkou. Jaký vliv má tato změna na přesnost klasifikace je podrobeno experimentům.

Klíčová slova: Konvoluční neuronová síť — Neuronová síť — Fixed point — Aproximace

Přiložené materiály: Knihovna na GitHub

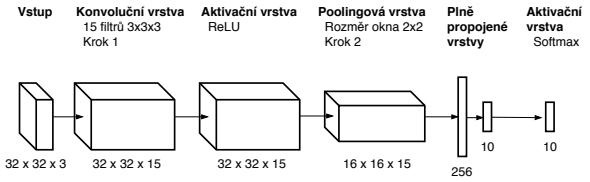
*xrekpe00@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

1. Úvod

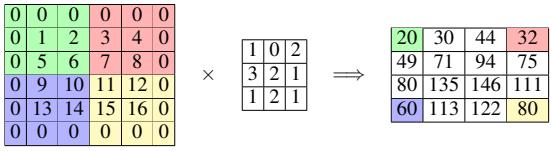
Umělá inteligence je v současnosti odvětvím, které se velmi rychle vyvíjí a počet aplikací umělé inteligence rok od roku roste. Podstatnou roli hrají také konvoluční neuronové sítě, které jsou využívány především pro zpracování strukturovaných informací jako je obraz nebo zvuk. Můžeme je najít například v autonomních vozidlech nebo algoritmech využívaných společnostmi *Google* či *Facebook*.

Cílem výzkumné práce, která je představena v tomto příspěvku, je vytvořit knihovnu pro práci s konvolučními neuronovými sítěmi. Cílem knihovny samozřejmě není konkurovat existujícím knihovnám. Čím se tato knihovna odliší od jiných bude její jednoduchá implementace a také nezávislost na datovém typu. V rámci každé vrstvy mohou existovat až tři datové typy – pro váhy, pro inferenci a pro učení. To samo o sobě není zajímavé v případě, že budeme používat pouze datové typy s plovoucí řádovou čárkou, avšak různou šířkou. Proto je knihovna za účelem vyhodnocení doplněna i implementací datového typu s pevnou řádovou čárkou a operací nad ním. Pro tento typ je možno specifikovat počet bitů před a za řádovou čárkou. V závislosti na okolnostech pak vhodně zvolená kombinace typů může vést k vyšší přesnosti, vyšší rychlosti a nižší spotřebě energie, což je podstatné zejména v případě využití v mobilních či vestavěných zařízeních.

Primárním cílem je vytvořit samotnou knihovnu, která bude poskytovat dostatečné množství funkcí a bude rozumně rychlá. Bylo tedy nutné nastudovat problematiku konvolučních neuronových sítí, provést návrh knihovny včetně podpůrných funkcí (*parsery, perzistence*), implementovat ji, zefektivnit a řádně otestovat. Aby se však knihovna odlišovala od jiných



Obrázek 1. Příklad architektury CNN



Obrázek 2. Vizualizace konvoluční operace

dostupných, byl zvolen již zmíněný sekundární cíl – nezávislost na datovém typu, především se zaměřením na datový typ s pevnou řádovou čárkou.

Knihoven pro konvoluční neuronové sítě vzniklo v průběhu let větší množství – od menších projektů až po ty, které jsou dnes běžně využívané v praxi. Mezi ty nejznámější patří například *TensorFlow* [1] nebo *Caffe* [2]. Vzhledem k masivní paralelizaci a implementaci pro grafické procesory jsou nejen velmi rychlé, ale poskytují i velké množství funkcionality. V rámci této práce takovýmto knihovnám nelze konkurovat, což ani není cílem. Takovéto knihovny jsou vzhledem k velkému množství funkcionality a urychlením typicky velmi těžko čitelné, což je nevýhoda, kterou menší projekt nemusí trpět.

Z pohledu využití aproximované reprezentace je nutno zmínit takzvanou *kvantizaci* [3], její 8 bitová varianta je součástí *TensorFlow* (který ale mezivýpočty provádí na větší bitové šířce) nebo jako rozšíření dostupná i pro *Caffe* (framework *Ristretto* [4], který však vnitřně stále počítá v plovoucí řádové čárce). Kvantizace se však zásadně liší od reprezentace s pevnou řádovou čárkou, jak bude dále ukázáno.

Příspěvek je strukturován následovně. V kapitole 2 je ve zkratce popsána problematika konvolučních neuronových sítí a reprezentace s pevnou řádovou čárkou. V kapitole 3 je blíže představena vytvořená knihovna. V kapitole 4 je provedena evaluace knihovny a také experimenty s reprezentacemi parametrů sítě. Dosavadní výsledky práce jsou pak shrnuty v kapitole 5.

2. Konvoluční neuronové sítě

Konvoluční neuronové sítě (dále *CNN*) jsou rozšířením dopředných neuronových sítí (dále *NN*) a velmi často je také obsahují. Používají se pro zpracování informací s pevně danou strukturou. Například pixely mají pevně dány souřadnice, které je nutno zachovat, aby obrázek dával smysl. Stejně tak zvuk má pevně dán čas a s ním spojené hodnoty. Výhodou CNN oproti NN je významná redukce počtu učitelných parametrů a lepší škálovatelnost. Složitější modely NN mohou obsahovat i desítky milionů vah a učení takových sítí je velmi náročné. Oproti tomu existují konvoluční sítě s daleko menším počtem parametrů, ale srovnatelnou či lepší přesností.

Konvoluční neuronová síť se skládá z vrstev, které jsou typicky řazeny lineárně za sebe (existují ale i implementace podporující grafové uspořádání), viz Obrázek 1. Takovýchto vrstev existuje velké množství, zde zmiňme čtyři základní [5], které by neměly chybět v žádné knihovně:

- konvoluční vrstva – má učitelné filtry a provádí operaci konvoluce nad vstupem,
- plně propojená vrstva – ekvivalent dopředné neuronové sítě bez skrytých vrstev, typicky využita na konci sítě,
- poolingová vrstva – provádí redukci rozměrů (šířky a výšky) trojrozměrných matic, urychluje výpočet,
- aktivační vrstva – zavádí nelinearitu do sítě, častá je například operace *ReLU*.

Nejpodstatnější je konvoluční vrstva, která dala jméno celým CNN. Konvoluční vrstva se skládá z předem definovaného počtu trojrozměrných filtrů, jejichž šířky a výšky jsou taktéž předem definovány a jejichž hloubka je dána hloubkou vstupu. Každý takový filtr je pak dále doplněn biasem (podobně jako v NN). Tyto filtry jsou s předem definovaným krokem posouvány po vstupním obrázku. Každé takové posunutí provádí sčítání násobků odpovídajících si hodnot a produkuje jeden akumulovaný výsledek. Každý filtr tak produkuje dvourozměrný výstup a hloubka výstupu konvoluční vrstvy je pak dána počtem filtrů. Operace konvoluce je znázorněna na Obrázku 2. Hodnoty filtrů (nazývané také váhy) a hodnota biasu jsou předmětem učení.

2.1 Aproximovaná reprezentace

Jak již bylo zmíněno, tak v rámci experimentů bude pro účely vyhodnocení knihovny z hlediska nezávislosti na datovém typu využita reprezentace s pevnou řádovou čárkou. Počet bitů před a za řádovou čárkou přímo udává minimální a maximální hodnotu, stejně jako rozdíl mezi dvěma po sobě jdoucími čísly.

Pro úplnost je také vhodné popsat již zmíněnou techniku kvantizace. V tomto případě je součástí definice pouze celková bitová šířka a také minimální a maximální vyjádřitelná hodnota. Vzdálenost mezi dvěma po sobě jdoucími čísly je pak zvolena tak, aby

	Kvantizace	Pevná řádová čárka
Definice	Šířka: 8 bitů Minimum: -1.573 Maximum: 4.954	Šířka: 8 bitů Celá část: 4 bity Desetinná část: 4 bity
Příklad	$0 = -1.573$ $128 = 1.703298...$ $255 = 4.954$	$0 = -8$ $128 = 0$ $255 = 7.9375$

Tabulka 1. Příklad kvantizace a pevné řádové čárky při využití 8 bitů (lze vyjádřit 256 hodnot)

bylo možné vyjádřit celý interval (čím větší interval, tím větší krok).

Názorněji je rozdíl mezi oběma přístupy znázorněn v Tabulce 1. Výhodou kvantizace je možnost vyjádřit prakticky libovolný interval (a s tím související vyšší přesnost) a nemůže tedy dojít k přetečení, což je problémem v případě reprezentace s pevnou řádovou čárkou. Avšak kvantizace (především tak, jak je používána například v knihovně *Tensorflow*) má vyšší režii než reprezentace s pevnou řádovou čárkou.

3. Knihovna TypeCNN

Knihovna *TypeCNN* je vytvořena v jazyce *C++* a dostupná pro operační systémy *Linux* a *Windows*. Knihovna umožňuje navrhnout, natrénovat a poté využívat konvoluční neuronovou síť. Z pohledu funkcionality nabízí následující:

- Konvoluční neuronová síť:
  - Výpočetní vrstvy – Konvoluční, Poolingová (average i max), Plně propojená, Drop-out, Aktivační,
  - Aktivační funkce - ReLU, Leaky ReLU, Sigmoid, Tanh, Softmax,
  - Ztrátové funkce – Mean squared error, Cross-entropy,
  - Optimalizátory – SGD, Momentum, Nestorov momentum, Adagrad, Adam,
- Podpůrné prostředky:
  - XML schéma pro specifikaci CNN,
  - Modul perzistance (načítání a ukládání),
  - Konzolové i programové rozhraní pro CNN i NN,
  - Parsery pro formáty IDX, Binary a PNG,
  - Podpora až tří datových typů v rámci jedné vrstvy, nezávislé mezi vrstvami,
  - Datový typ `FixedPoint` – reprezentace v pevné řádové čárce s definovatelným počtem bitů před a za řádovou čárkou.

Součástí implementace jsou také jednotkové testy (*unit testy*) a ukázkové programy popisující práci s knihovnou. Při implementaci byl kladen důraz na nezávislost na datovém typu a také snadnou pochopitelnost základních principů CNN. Snahou taktéž bylo vytvořit efektivní kód, neboť trénování konvolučních neuronových sítí je časově velmi náročné. Příklady zrychlení, které stojí za zmínku, jsou předpočítání hran pro výpočet výstupních hodnot v konvoluční a poolingové vrstvě a také jednorozměrné uložení trojrozměrných matic (a práce s jednorozměrným formátem v maximální možné míře).

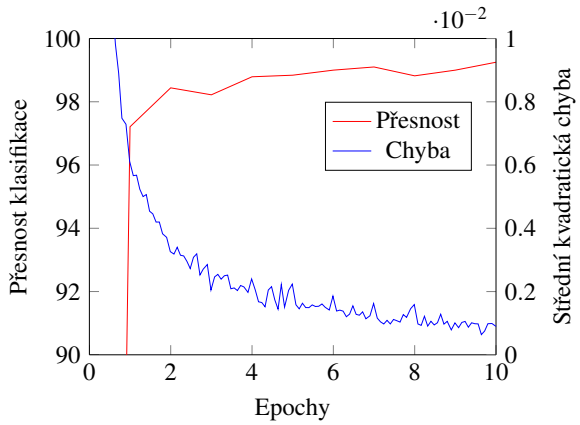
Významným rozšířením, které mělo zásadní vliv na implementaci, byla nezávislost na datovém typu. Té bylo dosaženo rozsáhlým využitím šablon napříč veškerými moduly knihovny. Datové typy jsou nezávislé mezi vrstvami (což ale vyžaduje speciální rozhraní, viz příklady, klasické uživatelské rozhraní to neumožňuje) a každá vrstva může mít tři na sobě nezávislé datové typy:

- *WeightType* – typ pro načítání/ukládání vah a práci s nimi během inference,
- *ForwardType* – typ výpočtů během inference,
- *BackwardType* – typ výpočtů a vah během trénování.

To není příliš zajímavé, pokud jsou používány pouze datové typy `float` nebo `double` (neboť rozdíly v trénování nejsou velké, `double` vede k mírnému zlepšení). Proto byl přidán typ `FixedPoint<F,D>`, kde F specifikuje počet bitů před řádovou čárkou a D počet bitů za řádovou čárkou. Musí platit $F > 1 \wedge 1 < F + D < 29$.

V případě vhodné datové šířky se převedení reprezentace s plovoucí řádovou čárkou na reprezentaci s pevnou řádovou čárkou projeví přidáním šumu, se kterým se CNN dokáží vyrovnat.

V případě, že využíváme datový typ `float` se šířkou 32 bitů a váhy převedeme na 8 bitů s pevnou řádovou čárkou, tak dosáhneme čtyřnásobné redukce místa, které váhy zabírají na disku (což u velkých sítí představuje i desítky MB). V případě, že jsme pak schopni celou CNN provádět v této reprezentaci, tak se nabízí využití ve vestavěných zařízeních, která nemohou pracovat s reprezentací s plovoucí řádovou čárkou nebo pro které hraje spotřeba významnou roli. V tomto případě je však třeba dávat pozor na možnost přetečení, která hrozí u menších bitových šířek. I proto jsou operace nad datovým typem `FixedPoint` saturované – v případě, že při provádění operace dojde k přetečení, respektive podtečení, je výsledek zarovnán na maximální, respektive minimální, hodnotu. Samotné trénování CNN je pak natolik přesnou disciplínou, že vyžaduje daleko větší přesnost (prakticky se neobejde bez reprezentace s plovoucí řádovou čárkou) a gradienty je nutné uchovávat v této přesnější reprezentaci (stejně tak je nutno si uchovávat přesnější hodnoty vah během trénování).



Obrázek 3. Průběh učení na datové sadě MNIST

4. Vyhodnocení a experimenty

V první části této kapitoly bude vyhodnocena implementace knihovny a poté budou provedeny experimenty s vlivem datového typu na schopnost sítě učit se.

4.1 Evaluace knihovny

Evaluace knihovny se skládá ze dvou částí. První z nich je vyhodnocení knihovny na klasických datových sadách určených pro hluboké učení – *MNIST* [6] a *CIFAR-10* [7].

Ve druhé části je pak knihovna porovnána s dalšími čtyřmi knihovnami z hlediska rychlosti a kvality učení.

4.1.1 Evaluace na datových sadách

MNIST je datová sada obsahující ručně psané černobílé číslice $0-9$ o rozměru 28×28 a *CIFAR-10* [7] je datová sada obsahující barevné obrázky o deseti kategoriích a rozměru 32×32. Zatímco MNIST je velmi jednoduchou úlohou (jednoduchá neuronová síť je schopná dosáhnout úspěšnosti okolo 97%), tak CIFAR-10 je o poznání náročnější (stejná síť dosáhne úspěšnosti pouze okolo 40%).

Pro datovou sadu MNIST byla zvolena architektura založená na *LeNet-5* [8], ve které byly pozměněny aktivační vrstvy na *Leaky ReLU*. Po 10 epochách trénování bylo dosaženo úspěšnosti až 99.25%, průběh trénování lze vidět na Obrázku 3. Ty nejlepší architektury s dalšími vylepšeními dosahují úspěšnosti okolo 99.7% [9]. Výsledek vyšší než 99.2% lze tedy pokládat za demonstraci schopnosti knihovny učit se.

Při datové sadě CIFAR-10 jsou pro dobré výsledky zapotřebí rozsáhlé architektury s mnoha konvolučními vrstvami (a desítkami až stovkami filtrů v každé z nich). Trénování takové sítě ve vytvořené knihovně by však trvalo velmi dlouhou dobu. Byla proto zvolena menší architektura založená na *LeNet-5*, která však obsahovala více konvolučních vrstev a filtrů. Na této architektuře bylo po 20 epochách dosaženo výsledku

Název knihovny	Doba trénování	Úspěšnost
SimpleCNN	749s	97.18%
TinyDNN	152s	98.15%
Keras (TensorFlow)	743s	98.18%
TypeCNN	530s	98.22%

Tabulka 2. Porovnání čtyř knihoven při trénování CNN po dobu 10 epoch

62.20%, což je hodnota, která se u menší sítě dá považovat za úspěch. Ty nejlepší architektury ale dosahují úspěšnosti až 96.5% [9].

4.1.2 Porovnání s jinými knihovnami

Druhou částí pak bylo porovnání schopnosti knihovny se učit – jak z pohledu kvality, tak z pohledu rychlosti. Porovnání bylo provedeno vůči jiným knihovnám pro konvoluční neuronové sítě. Pro tento účel byla náhodně zvolena architektura, která je součásti repozitáře knihovny *SimpleCNN*, a s totožnými parametry trénována na zvolených knihovnách na datové sadě MNIST po dobu 10 epoch. Výsledné hodnoty jsou průměrem z 5 experimentů. Těmito knihovnami byly:

- SimpleCNN [10] – jednoduchá knihovna v `C++`, malá podpora funkcionality a bez urychlení,
- TinyDNN [11] – rozsáhlá knihovna v `C++`, široká funkcionalita, velmi efektivní,
- Keras [12] (TensorFlow [1]) – prostřednictvím knihovny *Keras* lze využívat více různých knihoven pro hluboké učení (má jednoduché a jednotné rozhraní), v experimentu byla použita knihovna *Tensorflow*, což je rozsáhlá knihovna s rozhraním v `Pythonu` (implementace v `C++`), široká funkcionalita, včetně implementace pro grafické procesory.

Výsledky v Tabulce 2 samozřejmě nemají za cíl prokázat, že některá knihovna je horší než jiná. Každá knihovna má svá specifika. Všechny knihovny byly přeloženy pro co největší rychlost. K výsledku kombinace *Keras* + *Tensorflow* je potřeba dodat, že i když je toto spojení knihoven často využíváno, tak také lze dohledat případy, kdy samotné *Tensorflow* by bylo daleko rychlejší – což byl nejspíš i tento případ. Je potřeba také dodat, že testována byla pouze implementace pro *CPU* – varianta pro *GPU* by byla daleko rychlejší.

4.2 Experimenty s datovými typy

Cílem experimentů v této kapitole je jednak ověřit úspěšnost implementace nezávislosti na datovém typu, ale také zjistit vliv aproximované reprezentace s pevnou řádovou čárkou na schopnost sítě učit se. Předmětem experimentů je pouze zjistit dopady na přesnost natrénované sítě, nikoliv na dobu trénování – cílem totiž není urychlit trénování na klasických počítačích.

Datový typ	Úspěšnost na validační sadě	
	Před dotrénováním	Po dotrénování
`float`	97.61%	97.84%
`FixedPoint<14,14>`	97.61%	97.85%
`FixedPoint<8,8>`	97.70%	97.88%
`FixedPoint<12,4>`	74.80%	97.71%
`FixedPoint<6,2>`	20.84%	95.09%
`FixedPoint<4,4>`	10.28%	94.66%
`FixedPoint<3,5>`	11.65%	84.95%

Tabulka 3. Vliv reprezentace na klasifikační přesnost NN, trénování provedeno na `float` po dobu 10 epoch, dotrénování poté na daném typu po dobu 5 epoch

4.2.1 Neuronová síť

Několik plně propojených vrstev v zásadě odpovídá neuronové síti. První z experimentů byl proveden na třívrstvé neuronové síti s počtem vstupních neuronů 744, počtem neuronů ve skrytých vrstvách 256, 64 a 10 neurony ve výstupní vrstvě. Skryté vrstvy používají aktivační funkci *sigmoid* a výstupní vrstva *softmax*. Experiment byl proveden na datové sadě *MNIST*.

Experiment se skládal ze dvou částí. V první části byla neuronová síť během 10 epoch natrénována na datovém typu `float` s konečnou úspěšností 97.61% (byla ponechána rezerva pro další trénování). Další trénování poté bylo prováděno s různými datovými typy po dobu dalších 5 epoch. Trénování pro nižší bitovou šířku je obtížné a velmi závisí na učícím koeficientu η. Čím menší šířka, tím menší tento koeficient musí být. V Tabulce 3 je shrnuta úspěšnost ihned po převodu a nejlepší dosažená úspěšnost během pěti epoch dotrénování. Uvedený datový typ byl totožný pro `ForwardType` a `WeightType`.

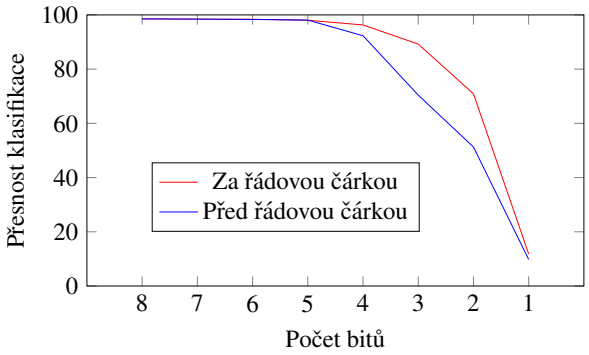
Lze si povšimnout, že v případě dostatečné šířky (16 a více bitů) síť netrpí žádnou ztrátou přesnosti (často i naopak). V případě využití 8 bitů je síť stále schopná se učit, avšak vliv na úspěšnost už je znatelný. Dobře patrný je také vliv přetékání – zatímco s málo bity za řádovou čárkou se síť vyrovná dobře, s méně bity před řádovou čárkou již ne.

Ve druhém experimentu je pak stejná neuronová síť od počátku trénována na daném typu po dobu 10 epoch. Oproti prvnímu experimentu je pro srovnání podstatné i počáteční nastavení vah, každý experiment byl tedy proveden pětkrát a výsledné hodnoty jsou průměrem. V závislosti na bitové šířce je upravován učící koeficient a v Tabulce 4 je vždy nejlepší dosažený výsledek během zmíněných 10 epoch.

Neboť jsou počáteční váhy nastavovány na velmi nízké hodnoty tak v případě nízké bitové šířky jsou vždy 0, což znamená, že síť se není schopna učit. Proto byl přidán mechanismus, který náhodně vygenerované

Datový typ	Úspěšnost na validační sadě
`float`	97.61%
`FixedPoint<14,14>`	97.53%
`FixedPoint<8,8>`	97.76%
`FixedPoint<12,4>`	96.79%
`FixedPoint<5,3>`	92.45%
`FixedPoint<4,4>`	77.30%

Tabulka 4. Výsledky trénování NN po dobu 10 epoch od počátku v daném typu



Obrázek 4. Závislost přesnosti na počtu bitů před a za řádovou čárkou, počet bitů druhé části byl vždy 8 (trénováno na datovém typu `float`, poté převedeno a dotrénováno na daném typu po dobu 5 epoch)

váhy násobí vhodným koeficientem tak, aby tomuto problému bylo zabráněno.

Jak je patrné z Tabulky 4, opět platí, že v případě dostatečné bitové šířky (16 a více bitů) je schopnost sítě učit se prakticky neovlivněna. V případě 8 bitového typu je již vliv patrný, avšak síť je stále schopná učení a pro jednodušší úlohy by pak stále byla použitelná.

4.2.2 Konvoluční neuronová síť

V případě konvolučních neuronových sítích se jedná o daleko větší problém a to zejména pro datové typy s nízkou bitovou šířkou, to lze vidět na Obrázku 4 (pro jednoduchou síť použitou pro srovnání knihoven).

V prvním experimentu byla využita konvoluční neuronová síť založená na architektuře *LeNet-5* (tři konvoluční vrstvy, dvě plně propojené vrstvy). Trénována byla po dobu 10 epoch a to na přesnost 98.23% (byla ponechána rezerva pro další trénování). Poté byly datové typy inference a vah převedeny do reprezentace s pevnou řádovou čárkou a po dobu 5 epoch proběhlo dotrénování.

V Tabulce 5 si lze všimnout, že konvoluční neuronová síť je schopna pracovat bez problému v 16 bitech, avšak 8 bitů již znamená velký pokles v kvalitě trénování.

Ve druhém experimentu byl typ `ForwardType` tedy vždy `FixedPoint<8,8>` a měnil se pouze datový typ `WeightType`. V Tabulce 6 si lze všimnout,

Datový typ	Úspěšnost na validační sadě	
	Před dotrénováním	Po dotrénování
`float`	98.23%	98.44%
`FixedPoint<14,14>`	94.84%	98.45%
`FixedPoint<8,8>`	94.03%	98.50%
`FixedPoint<12,4>`	14.56%	97.80%
`FixedPoint<4,4>`	10.12%	26.36%

Tabulka 5. Vliv reprezentace na klasifikační přesnost CNN, trénování provedeno na typu `float` po dobu 10 epoch, dotrénování poté na daném typu po dobu 5 epoch

Datový typ vah	Úspěšnost na validační sadě	
	Před dotrénováním	Po dotrénování
`FixedPoint<8,8>`	94.03%	98.50%
`FixedPoint<2,6>`	94.28%	98.47%
`FixedPoint<3,5>`	90.77%	98.40%
`FixedPoint<4,4>`	79.24%	98.26%
`FixedPoint<2,2>`	9.80%	88.39%

Tabulka 6. Vliv reprezentace na klasifikační přesnost CNN, trénování provedeno na typu `float` po dobu 10 epoch, dotrénování poté s `FixedPoint<8,8>` pro inferenci a daným datovým typem pro váhy

že pokud je síť oproštěna od problému přetékání, stačí pouhých 8 bitů pro reprezentaci vah a schopnost sítě učit se je taktéž prakticky nezměněna. Pro jednoduché úlohy by pak bylo možno použít například i 4 bity.

5. Závěr

Hlavním cílem této práce bylo vytvořit funkční knihovnu pro práci s konvolučními neuronovými sítěmi, která bude do jisté míry nezávislá na datovém typu.

Z předchozích kapitol je patrné, že knihovna je v současné době plně funkční. Avšak existuje mnoho vylepšení, které je možno provést – od nové funkcionality až po zrychlení. Knihovna také stále prochází úpravami s ohledem na efektivnost a i na kvalitu kódu.

Podařilo se implementovat i rozšíření spočívající v nezávislosti knihovny na datovém typu. To bylo ukázáno na experimentech s datovým typem odpovídajícím reprezentaci s pevnou řádovou čárkou. Bylo také ukázáno, že provádění inference v této reprezentaci nutně neznamená, že síť ztratí svou přesnost. Velmi však závisí na počtu bitů před a za řádovou čárkou. Trénování v této reprezentaci, zejména pro nižší šířky, je také problematičtější vzhledem k vyšší citlivosti.

Díky tomuto rozšíření by nad touto práci bylo možno založit další, které dále prozkoumají vliv aproximace (nejen reprezentace v pevné řádové čárce, ale i dalších reprezentací a případně také aproximovaných operací) na konvoluční neuronové sítě.

Poděkování

Na tomto místě bych rád poděkoval vedoucímu práce Prof. Ing. Lukáši Sekaninovi, PhD., za cenné rady při jejím vypracování.

Literatura

[1] Martín Abadi, Ashish Agarwal, and Paul Barham et al. TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/, 2015. Software available from tensorflow.org.

[2] Yangqing Jia and Evan Shelhamer et al. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[3] Building a quantization paradigm from first principles. https://github.com/google/gemmlowp/blob/master/doc/quantization.md, 2015.

[4] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[5] Andrej Karpathy. Lecture notes for COMPSCI 682 (Neural Networks: A modern introduction). http://cs231n.github.io/, Leden 2015.

[6] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[7] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Diplomová práce, Department of Computer Science, University of Toronto*, 2009.

[8] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[9] Rodrigo Benenson. What is the class of this image? 2016.

[10] *can1357*. SimpleCNN: Simple convolutional neural network library. https://github.com/can1357/simple_cnn/, 2016.

[11] Taiga Nomi. TinyDNN: header only, dependency-free deep learning framework in c++14. https://github.com/tiny-dnn/, 2013.

[12] François Chollet et al. Keras. https://keras.io, 2015.