VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2019 Bc. Tomáš Hipča

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

GENERÁTOR NEURONOVÝCH SÍTÍ PRO POTŘEBY MĚŘENÍ PODOBNOSTI OBRAZU
NEURAL NETWORK GENERATOR FOR IMAGE SIMILARITY MEASUREMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Tomáš Hipča

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. Radim Burget, Ph.D.

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Tomáš Hipča ***ID:*** 164731
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Generátor neuronových sítí pro potřeby měření podobnosti obrazu

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s problematikou hlubokých konvolučních neuronových sítí a proveďte rešerši posledních trendů v této oblasti. Navrhněte automatický softwarový generátor dopředných neuronových sítí, s pomocí kterého bude možné automaticky vytvářet různé varianty hlubokých neuronových sítí. Tyto neuronové sítě následně ověřte na databázi https://landmarkscvprw18.github.io/. Výsledky testování vhodně popište tabulkou hodnot a dosažené výsledky vhodně diskutujte.

DOPORUČENÁ LITERATURA:

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[2] Karpathy, Andrej, et al. "Large-scale video classification with convolutional neural networks." Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 2014.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: doc. Ing. Radim Burget, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Tato práce se zabývá návrhem a implementací automatického softwarového generátoru dopředných neuronových sítí pro klasifikaci obrazu. Teoretická část práce objasňuje pojmy jako neuronová síť nebo formální neuron. Dále práce prezentuje rozdělení neuronových sítí na základě typu jejich architektury sítě a stylu učení. Práce se zaměřuje na konkrétní typ neuronových sítí, a sice sítě konvoluční. Jsou prezentované vybrané výzkumy z této oblasti. Následují informace o implementaci použité v praktické části práce, tedy jaký byl zvolen programovací jazyk a který aplikační rámec byl použit. Stejně tak je obsahem stručný popis implementace, přehled implementovaných vrstev neuronové sítě, zvolená databáze fotografií a postup testování sítí. Výsledky toho testování jsou prezentovány a příslušně okomentovány.

KLÍČOVÁ SLOVA

Generátor neuronových sítí, Google-Landmarks, hluboká neuronová síť, Inception, InceptionV3, konvoluční neuronová sít, neuronová síť, strojové učení.

ABSTRACT

This thesis deals with designing an automatic generator of deep neural networks for image classification. Theoretical part clarifies what a neural network and formal neuron are. Furthermore, the types of neural network architectures are presented. The focus of this thesis is convolutional neural networks, several pieces of research from this field are mentioned. The practical part of this thesis describes information with regards to the implementation of neural network generator, possible frameworks and programming languages for such implementation. Brief description of the implementation itself is presented as well as implemented layers. Generated neural networks are tested on Google-Landmarks dataset and results are commented upon.

KEYWORDS

Convolutional neural network, deep neural network, Google-Landmarks, Inception, InceptionV3, machine learning, neural network generator, neural network.

HIPČA, Tomáš. *Generátor neuronových sítí pro potřeby měření podobnosti obrazu*. Brno, 2019, 70 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Radim Burget, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Generátor neuronových sítí pro potřeby měření podobnosti obrazu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Radimovi Burgetovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Neuronové sítě jsou velmi užitečným nástrojem zejména v případě, že přesně specifikovat algoritmus není možné a je příhodnější systému zadat jeho vstupy a požadované výstupy. Neuronová sít se poté "naučí"vhodné algoritmy pro správnou funkčnost tohoto systému. Pro takto fungující neuronovou síť jsou kritické dva aspekty: dostatečně velká a různorodá databáze vstupních a výstupních hodnot ale také vhodně zvolená struktura samotné neuronové sítě.

Tato práce se zaměřuje na generování neuronových sítí, které jsou později testovány pro klasifikaci obrazové množiny. V dnešní době je velkým problémem zejména zvolení vhodné struktury neuronové sítě, tento problém je možné redukovat použitím náhodně generovaných vrstev pro konstrukci neuronové sítě, tímto přístupem se otevírají možnosti struktur, které by lidský přistup mohl opomenout nebo zcela zavrhnout.

Nejdříve je vhodné vysvětlit, co neuronová síť je a jaká je její biologická předloha. Touto problematikou se zaobírá následující kapitola. Poté následuje ukázka několika možných topologií sítí a příklady využití těchto sítí v praxi. V práci je také zmíněno několik možných implementacích této technologie, jak s použitím výpočetního výkonu jednotky procesorové, tak grafické. Následně se práce zaměřuje na konvoluční neuronové sítě, na jejich parametry, ukázky, možné výhody a nevýhody oproti jiným architekturám. Tato architektura je nejvíce používaná při práci s obrazovými daty a využívanná například při klasifikaci objektů v obraze. Za zmínku také stojí předchůdce konvolučních neuronových sítí s názvem "neocognitron".

Následuje kapitola věnovaná samotnému generátoru neuronových sítí, popisu parametrů použitých pro generování vrstev, výpisu generovaných druhů vrstev, ale také možným knihovnám pro implementaci tohoto problému, jakož i volbě jazyka zvoleného pro implementaci v této práci. Výstupem tohoto generátoru jsou sítě pro využití v uživatelem vybrané aplikaci. Vygenerované sítě jsou závěrem testovány na zvolené databázi fotografií lokací. Výsledky úspěšnosti jednotlivých vygenerovaných sítí jsou porovnány a zdůvodněny.

1 Neuronové sítě

1.1 Co je neuronová síť

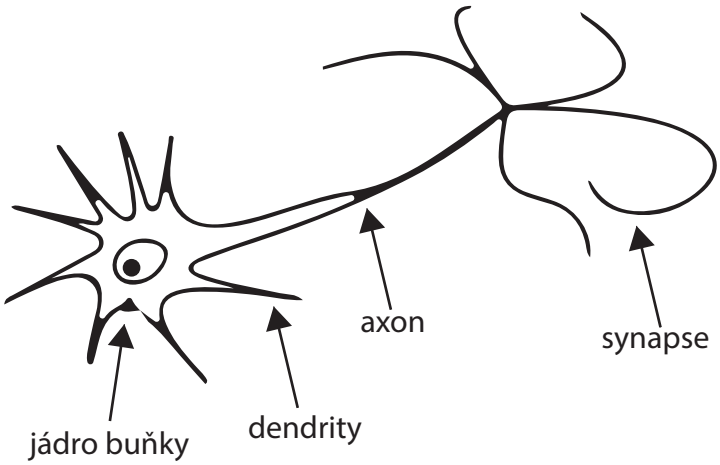
Jako neuronovou síť lze označit výpočetní model, který je schopný se zlepšovat (učit se) na základě předchozích zkušeností. Podle Haykina [5] je neuronová síť bohatě rozvětveným paralelním procesorem, schopným ukládat a využívat předešlé zkušenosti, složeným z jednoduchých výpočetních jednotek. Schopnost učit se umožnuje neuronové síti zobecnit problém, a tak poskytnout vhodný výsledek i pro nová data, se kterými se síť ještě nesetkala. Díky těmto vlastnostem je možné za pomoci několika neuronových sítí odhadovat výsledky komplexních, těžko řešitelných problémů.

Vstupem pro neuronovou síť je takzvaný dataset, jedná se o množinu dat rozdělených na data trénovací, sloužící pro trénování (učení) sítě, a data testovací, určená k ověření výkonnosti sítě. Podmínkou je, že podmnožina testovacích dat se nesmí vyskytovat v datech určených pro trénováni sítě, tímto způsobem je zajištěno, že síť není limitována pouze na již známá vstupní data. Jeden ze způsobů, jak rozdělit dataset, je například poměr 70:30, kdy je 70% dat použito pro trénování a zbylých 30% pro testování.

Schopnost sítí učit se je ve většině případů realizována použitím adaptačního algoritmu zpětného šíření chyby. Proces funkce tohoto algoritmu lze rozdělit na tři části, první z těchto částí je šíření vstupního signálu získaného zpracováváním síti poskytnutých dat. Následuje zpětné šíření chyby, zajišťující distribuci části chyby, kterou je třeba zpětně propagovat, tento faktor je vypočítán pomocí odchylek reálných výstupních hodnot od požadovaných výstupních hodnot neuronů umístěných na konci neuronové sítě. Pomocí těchto odchylek jsou v další etapě algoritmu aktualizovány váhy jednotlivých spojení mezi neurony [6].

1.2 Model neuronu

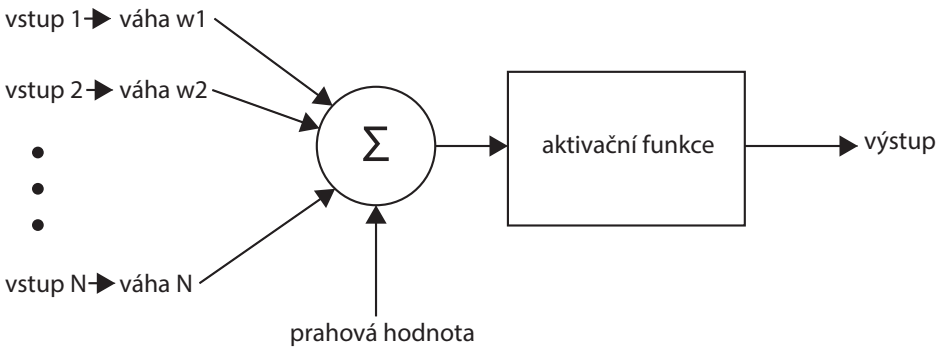
Elementárním prvkem obou druhů sítí, jak biologické, tak uměle vytvořené, je neuron, buňka schopná přenosu, uložení a následovnému zpracování informace [7]. Struktura biologického neuronu je schematicky znázorněna na následujícím obrázku.



Obr. 1.1: Schéma biologického neuronu

Z jádra neuronu vychází dendrity a axon, tyto výběžky slouží k přenosu signálů. Dendrity jsou použity pro příjem vstupního signálu, zatím co axon je využíván k přenosu signálu výstupního. Samotný přenost informací mezi dvěma neurony je realizován skrze synapse, unikátní mezineuronové rozhraní [6].

Pro reprezentaci biologického modulu byl vytvořen formální neuron, tato matematická reprezentace má n obecně reálných vstupů x, simulujících dendrity, a výstup y simulující axon. Ke každému ze vstupů je přiřazena jeho synaptická váha w, hodnota ovlivňující propustnost jednotlivého vstupu. Váha může nabývat jak hodnoty kladné, tak záporné, simuluje se tak chování biologického neuronu [8]. Pokud vážená suma vstupních hodnot překoná prahovou hodnotu neuronu, neuron se stane nabuzeným a na jeho výstupu lze nalézt signál odpovídající aktivační funkci neuronu [6].



Obr. 1.2: Schéma formálního neuronu

Matematicky lze formální neuron vyjádřit následovně. Vstup neuronu lze zapsat jako:

$$V = \sum_{i=1}^{N} w_i v_i \tag{1.1}$$

kde V označuje sumu vážených vstupů, v jsou vstupní signály a w jsou váhy pro jednotlivé vstupní signály.

Výstup neuronu poté bude:

$$Y = \varphi(V + p) \tag{1.2}$$

kde Y je výstup neuronu, φ je aktivační funkce neuronu, V je suma vážených vstupů a p je prahová hodnota neuronu.

Volba aktivační funkce ovlivňuje způsob, jakým bude učení neuronové sítě konvergovat. V praxi jsou využívány zejména následující aktivační funkce: funkce jednotkového skoku, funkce signum, funkce lineární a funkce sigmoidální [9].

1.3 Model sítě

Neuronová sít vzniká spojením několika neuronů, kdy výstup jednoho neuronu je vstupem neuronu dalšího (nebo i více). Vzhledem k počtu neuronů a stylu jejich propojení lze poté rozlišovat tyto sítě podle jejich topologie, tomuto rozdělení je věnována následující část této práce. Obecně, vzhledem k jejich využití, v neuronové síti nalezneme tři typy neuronů. Jedná se o neurony vstupní, skryté a výstupní. Jak již z názvu vypovídá, neurony vstupní nalezneme u vstupu do sítě, následují neurony skryté a poté neurony výstupní, umístěny na konci sítě [10].

Výpočty neuronové sítě lze matematicky vyjádřit za pomoci součinu matice a vektoru.

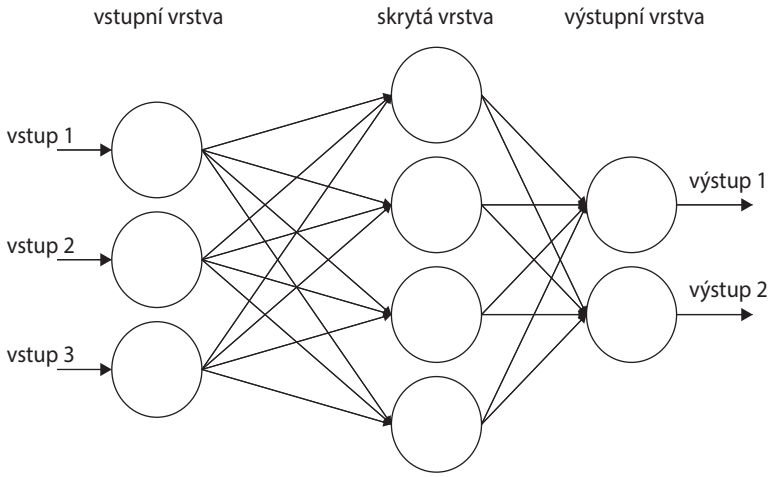
$$Y = A(W \times V) \tag{1.3}$$

Kde Y je výstupem neuronové sítě, W je maticí vah, V je vektorem vstupních hodnot a A je aktivační funkcí (za předpokladu stejné aktivační funkce pro všechny neurony).

1.4 Rozdělení sítí

1.4.1 Rozdělení podle topologie

Neuronové sítě je také možné dělit na dvě základní topologie, první z nich je cyklická síť, druhou je pak síť acyklická neboli dopředná. Jak již z názvů vyplývá, cyklické sítě obsahuji cykly, u nichž je možné například zpětnovazební propojení, naopak u sítích dopředných platí pravidlo, že výstupy neuronů vedou pouze z vrstev nižších do vrstev vyšších. Tímto propojením je možné dosáhnout vícevrstvé neuronové sítě [6].



Obr. 1.3: Schéma vícevrstvé neuronové sítě

1.4.2 Rozdělení podle typu učení

K procesu učení je využíván učící algoritmus modifikující synaptické váhy mezi neurony. Další možností učení sítě je dynamická změna architektury sítě pomocí modifikace jednotlivých neuronů a jejich vzájemného spojení. Pro modifikování vah mezi neurony je ve většině případů použit algoritmus zpětného šíření chyby, tento algoritmus se řadí do kategorie učení s učitelem. V případě použití tohoto algoritmu je třeba kontrolovat, zdali neuronová síť neuvázla v lokálním minimu, v tomto případě by nebyla schopna najít nejoptimálnější řešení [11].

Neuronové sítě je možné rozdělit do dvou skupin dle zvoleného postupu při trénování (učení) sítě. První, dříve již zmíněnou možností, je učení s učitelem, tento přístup vyžaduje dvojici dat, data vstupní a k těmto datům příslušná data výstupní. Síť si poté vypočítá odchylky svých odhadů od korektních výsledků a následně upraví hodnoty svých vah.

Druhý postup je učení bez učitele. Na rozdíl od předešlého typu tento postup vyžaduje pouze jedna data, a sice data vstupní. Tyto sítě jsou vhodné zejména v případech, kdy se data velmi rychle a neočekávaně mění a není možné v požadované době vytvořit sadu vstupních a patřičných výstupních dat [12].

1.5 Hardwarová implementace

Mnoho aplikačních rámců pro práci s neuronovými sítěmi, jako například v práci zvolený „TensorFlow“, umožňují výpočty realizovat jak za pomoci centrální procesorové jednotky, tak s použitím grafického procesoru. Vzhledem k možnosti celý proces výpočtů paralelizovat, a učení tak zkrátit mnohdy až o více jak polovinu, je upřednostňováno využití procesoru grafického.

Dalším možným hardwarem pro realizaci neuronových sítí jsou programovatelná hradlová pole (FPGA). Jedná se logické integrované obvody, které po porovnání s grafickými procesory nabízejí vynikající energetickou účinnost, avšak za cenu výpočetního výkonu. Tento problém však řeší nové generace FPGA společnosti Intel [13].

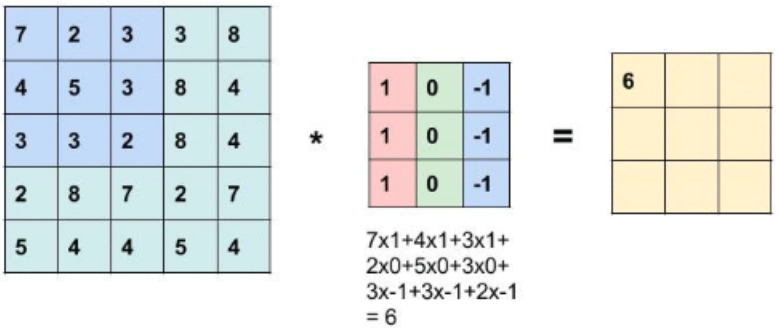
Posledním příkladem hardwaru umožňujícího realizaci neuronové sítě je zákaznický integrovaný obvod neboli ASIC. Tyto integrované obvody jsou navrženy pro specifickou činnost a na rozdíl od FPGA není možné ASIC přeprogramovat. Příkladem využití ASIC v oblasti neuronových sítí může být "Edge TPU"[1], integrovaný obvod vyvinutý společností Google pro práci s aplikačním rámcem TensorFlow.

1.6 Konvoluční neuronové sítě

Jedná se o sítě využívající ve svých vrstvách matematický nástroj konvoluce. Tato metoda staví na myšlence, že celý obraz má jednotné vlastnosti (je stacionární), je tedy možné obraz rozdělit na několik segmentů, a tyto segmenty zpracovávat individuálně. Výhodou těchto sítí, na rozdíl od jiných sítí, využívaných při výpočtu podobnosti obrazových vstupů, je možnost použití obrazu přímo, nikoli pouze předzpracovaných příznaků, jako vstupu [14].

Proces konvoluce probíhá následovně. Obraz je rozdělen na několik segmentů, na jednotlivé segmenty obrazu je postupně aplikována matice vah, nazývající se konvoluční jádro, výsledkem je konvoluce segmentu původního obrazu a konvolučního jádra.

[1] Webová stránka produktu: https://cloud.google.com/edge-tpu/



Obr. 1.4: Ilustrace procesu konvoluce

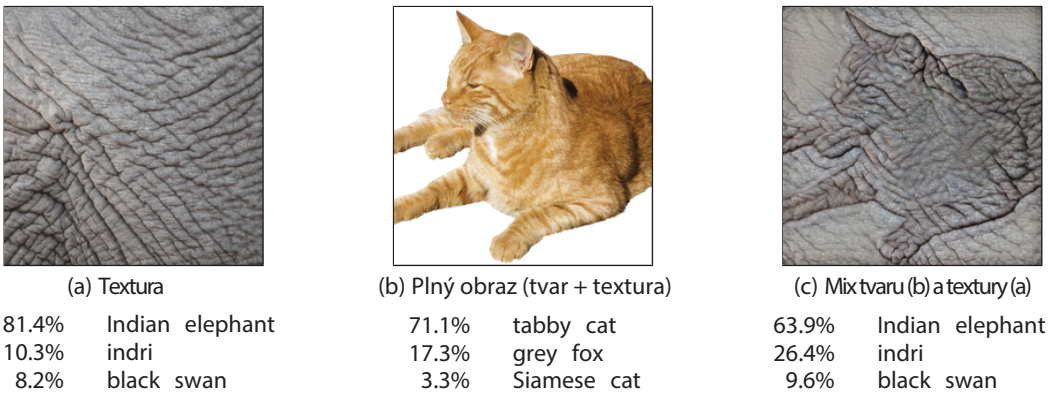
Na rozdíl od většiny dopředných sítí se konvoluční neuronové sítě rychleji učí, zejména díky menšímu množství propojení a parametrů při stejném počtu neuronů ve vrstvě, i přes jednodušší architekturu si sítě zachovávají velmi podobnou maximální teoretickou výkonost [15].

Předchůdcem tohoto typu neuronových sítí je "neocognitron". Jedná se o síť schopnou naučit se a rozpoznat sadu vzorů. Výsledek je jen velmi málo ovlivněn deformacemi obrazu, jako je například změna velikosti či relativní posun v pozici. Této robustnosti sítě je dosaženo postupnou tolerancí odchylek v pozici vzoru na každé z vrstev [16], která se ukázala jako úspěšnější oproti snaze tento proces provádět najednou, v jediné vrstvě. Neurony umístěné blíže vstupu sítě se zaměřují pouze na lokální znaky, se zvyšující vzdáleností neuronů od vstupu se zvyšuje i komplexnost vzorů, které tyto neurony vyhodnocují. Jedná se tedy o hierarchický přístup. Neurony nejvyšší vrstvy poté reagují pouze na, jim přiřazený, konkrétní vzor. Původně byla tato neuronová síť využita k rozpoznávání ručně psaných znaků.

1.6.1 Postup konvolučních sítí při klasifikaci objektu

Společně s rozšířením konvolučních neuronových sítí je stále více řešena otázka samotného fungování těchto sítí. Zejména díky jejich rozsáhlému využití pro výpočetní modely lidského zrakového vjemu tvarů a rozpoznávání objektů. Je známo, že lidský zrak se zaměřuje především na tvar a kontury objektu. Tento způsob vnímání člověku umožňuje klasifikovat objekt i v případě záměny textury tohoto objektu. Vzhledem k tomuto faktu je rozšířena teorie, že konvoluční neuronové sítě, stejně jako člověk při kategorizaci, využívají především tvarů a kontur objektů. Geirhos a spol. ve své práci [1] tento přístup označují jako hypotézu tvaru. Tuto hypotézu potvrzuje několik vědeckých prací [17], [18].

Geirhos a spol. [1] ovšem prezentují i poněkud odlišnou hypotézu, autory pojmenovanou jako hypotézu textury. Tato myšlenka je založena na experimentech Gatys a spol. [19] a Brendela a Bathge [20], které poukazují na schopnost konvolučních neuronových sítí klasifikovat objekt pouze pomocí textury. V provedených experimentech byl tvar objektů na vstupu zcela deformován. Lze tedy předpokládat, že textura objektu je pro klasifikaci objektu pomocí konvoluční neuronové sítě důležitější, než samotný tvar klasifikovaného objektu.



Obr. 1.5: Výsledky experimentu hypotézy textury [1]

Sklon k preferování textury nad tvarem objektu mají podle autorů zejména konvoluční sítě trénované pomocí datasetu Image-Net. Součástí práce [1] je také experiment testující výkonnost konvoluční neuronové sítě preferující tvar nad texturou. Této vlastnosti bylo dosaženo trénováním sítě na uzpůsobené trénovací množině, zohledňující předešle zmíněný fakt. Takto trénovaná síť vykazuje větší robustnost a dosahuje přesnějších výsledků.

1.7 Siamské neuronové sítě

Siamské neuronové sítě jsou velmi slibným typem neuronových sítí za předpokladu, že k učení sítě máme pouze jeden příklad od každé možné třídy. Tento typ neuronové sítě se skládá ze dvou totožných sítí (jak architekturou, tak vahami) na jejichž vstup přivedeme obrazy, ve vyšších vrstvách jsou pak tyto sítě propojeny pomocí funkce počítající rozdílnost znaků vyšších vrstev. Koch a spol. [21] také uvádí, že nejlepší výsledky přináší architektury využívající několika konvolučních vrstev, hustě propojených vrstev a následnou funkci pro výpočet rozdílnosti.

2 Technologie a výzkum

Neuronové sítě, zejména pak sítě konvoluční, jsou velmi užitečným nástrojem při práci s obrazy, jako je například klasifikace, zaostření, komprese, či výpočet podobnosti obrazu. Tyto schopnosti je pak možné využívat například pro rozpoznávání objektů zachycených kamerou autonomního automobilu [22] nebo k automatické diagnóze pacienta ze snímků pořízených nemocničním vybavením. Traore a spol. například prezentují využití konvolučních neuronových sítí ke klasifikaci mikroskopických vzorků cholery a malárie [23]. Tímto ovšem využití neuronových sítí zdaleka nekončí, sítě lze také využít jako nástroj pro predikci hodnot. Příkladem tohoto využití je například předpověď ceny nemovitostí na základě historie jejich cen, případně jiných parametrů.

2.1 Konvoluční neuronová síť Krizhevsky a spol.

Prvním příkladem architektury neuronové sítě je konvoluční neuronová síť prezentována Alexem Krizhevskym a spol. [15]. Jedná se o neuronovou síť určenou pro klasifikaci 1,2 milionu obrazových souborů do 1000 kategorií. Tato síť dosáhla v soutěži "ImageNet LSVRC-2010"výsledku 17% chybovosti [1], se kterým jednoznačně předčila tehdejší konkurenci. Tento model se skládal ze 650 000 neuronů rozložených do osmi vrstev, přesněji, do pěti konvolučních vrstev a tří plně propojených vrstev. Úspěšnost této sítě lze přisuzovat zejména vhodné implementaci dvourozměrné konvoluce a využití regularizační metody dropout, fungující na principu nulování výstupní hodnoty náhodných neuronů skrytých vrstev. Výstup každého neuronu má tedy 50% šanci, že bude vynulován. Tato metoda byla využita v prvních dvou plně propojených vrstvách, a výrazně tak vylepšila výkonost neuronové sítě na testovací sadě.

Obdobný model se také umístil na prvním místě v soutěži "ILSVRC-2012"s chybovostí 15,3%, přičemž ve výsledném pořadí druhý model dosáhnul chybovosti 26,2% [2] .

2.2 ResNet

Práce prezentovaná Hem a spol. [24] se zabývá otázkou, zdali je možné dosáhnout lepších výsledků neuronových sítí pouhým přidáváním dalších vrstev jedné za druhou. Experimenty provedené v této publikaci tento přístup jasně vyvracejí. Přesnost

[1] Této chybovosti bylo dosáhnuto v kategorii "top-5 error rate"

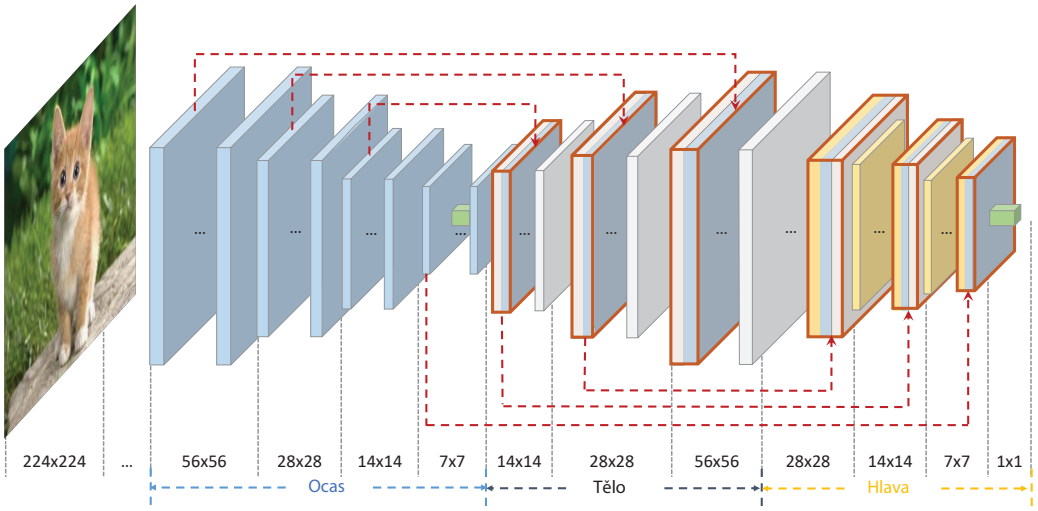
[2] Kategorie "top-5 error rate"

sítí se s příliš velkým počtem vrstev naopak snižuje. Tento výzkum tak podporuje hypotézu vznesenou v této diplomové práci. Pro snížení nepřesnosti neuronových sítí, vzniklé velkým počtem vrstev v architektuře, je využito „zkratek“, umožňujících propojení přímo nenavazujících vrstev. Bloky vrstev vytvořené s použitím tohoto přístupu nesou název reziduální bloky a jsou využity například v architektuře Inception, kterou je možné generovat pomocí v práci implementovaného generátoru. Použitím těchto propojení není nijak zvýšen počet parametrů architektury ani její výpočetní náročnost. Minimalizují však negativní dopady při velkém počtu neuronových vrstev.

2.3 FishNet

Jedná se o typ architektur určený specificky pro detekci a segmentaci obrazových vstupů. Velkou výhodou těchto architektur je schopnost získat sémantické informace i z příznaků ve vysokém rozlišení. FishNet architektury spojují výhody architektur určených pro funkčnost jak na úrovni pixelů, regionů, ale i celého snímku. Informace získané z jednotlivých úrovní jsou využity pro zvýšení přesnosti na ostatních úrovních.

Architektura FishNet je rozdělena na tři části. Směrem od vstupu do sítě se jako první nachází „ocas“, který se zaměřuje na příznaky malého rozlišení. Následuje „tělo“, specializující se na příznaky vysokého rozlišení a jejich sémantický význam. Poslední část, „hlava“, využívá informace z předchozích částí a dále je umocňuje [2].



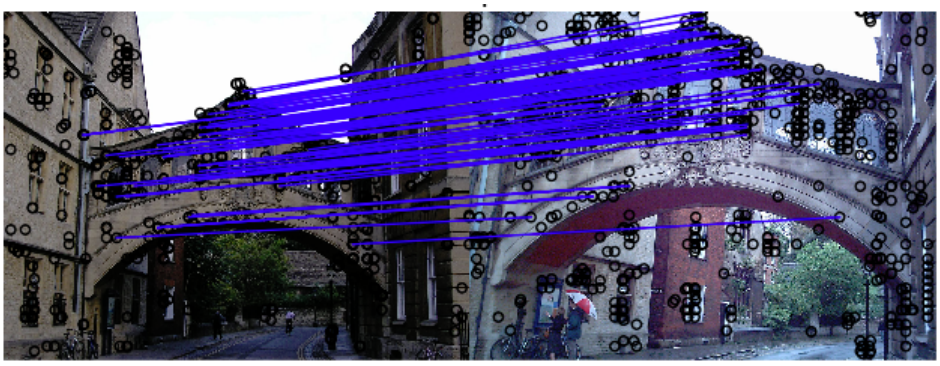
Obr. 2.1: Architektura FishNet [2]

2.4 DELF

Pro vyhledávání podobných obrazů ve velkém datasetu, jako tomu je například v případě soutěže "Landmark Retrieval"(blíže popsána v kapitole dataset), je možné zvolit přístup zvaný DELF [3] , který se stal jednoznačným vítězem v této kategorii. Využitím tohoto algoritmu je možné extrahovat sémantické informace z určité oblasti obrazu a následně nalézt geometricky podobnou oblast v datasetu. Tento přístup je zejména efektivní v případech, kdy databáze neobsahuje žádný podobný objekt nebo je na snímku mnoho objektů v pozadí [3].

Proces algoritmu DELF lze rozdělit na čtyři dílčí části. První z těchto částí je extrakce hustě lokalizovaných znaků (dense localized feature extraction), následuje selekce významných bodů (keypoint selection), redukce dimenzionality (dimensionality reduction) a závěrečné indexování a vyhledávání (indexing and retrieval). Extrakce znaků je provedena pomocí plně propojené konvoluční sítě, přesněji, je použita část modelu neuronové sítě ResNet50 [24], předtrénována na množině dat ImageNet [25] a následně dotrénována pro rozeznávání lokálních deskriptorů. Vzhledem k množství nepotřebných deskriptorů je jejich počet redukován pomocí metody selekce významných bodů. Tímto postupem je docíleno vyšší přesnosti a výpočetní efektivity celého systému. Dalšího zlepšení systému dosáhneme redukcí dimenzionality. Tento proces je používán pro aplikace tohoto typu poměrně běžně [26].

Následující obrázek ilustruje nalezení shodných oblastí na dvou snímcích zachycujících stejný objekt z jiných úhlů. Kruhy značí pro algoritmus zajímavé části obrazu, pokud je obdobná část nalezena na druhém snímku, oblasti jsou propojeny modrou úsečkou.



Obr. 2.2: Ilustrace procesu DELF [3]

[3]DELF je zkratkou pro Deep Local Feature

2.5 Redukce objemu databáze

S rostoucí přesností neuronových sítí roste čas učení neuronové sítě i objem potřebných datasetů. Možné řešení problému s dlouhou dobou učení a velkým objemem trénovacích dat prezentují Gonzalez-Diaz a Paluzo-Hidalgo [27]. Autoři navrhují postup pro výběr reprezentační podmnožiny datasetu, jehož základní myšlenkou je redundance informací potřebných k učení sítě u prvků množiny datasetu nacházející se na stejném nebo podobném místě topologického prostoru. Jedná se tedy o využití algoritmu založeného na proximálním grafu [28].

Odstraněním některých z těchto redundantních prvků lze dle autorů docílit snížení doby učení, aniž by síť významně utrpěla na přesnosti. Ověření tohoto tvrzení bylo provedeno na hustě propojené dopředné neuronové síti se sto neurony. Tato sít je testována na třech datasetech (originálním a dvěmi redukovanými s Hausdorffovou vzdáleností 0.32 a 0.60). Výsledné odhady sítě s redukovanými datasety se poté pohybovaly v rozmezí 95 až 96 % původního odhadu s kompletním datasetem. Bohužel, vzhledem k chybějícím údajům o době učení, je těžké určit, zdali je použití této metody výhodné.

2.6 AdaBound

AdaBound je poměrně novým projektem zaměřujícím se na kombinaci přístupů k realizaci optimalizačních metod neuronových sítí. AdaBound kombinuje adaptivní změnu a stochastický gradientní pokles k sestrojení optimalizační metody dosahující lepších výsledků oproti samostatnému využití těchto metod. Zásadním problémem adaptivního nastavení parametru míry učení (learning rate), je možnost nastavit tento parametr extrémně nízký, nebo naopak vysoký. Ve výsledku se špatné nastavení tohoto parametru projevuje slabou schopností neuronových sítí zobecňovat problémy a obecně horší výkoností. AdaBound proto limituje množinu možných hodnot pro míru učení. Nastavení těchto limitů probíhá dynamicky a zaručuje tak souvislý přechod z adaptivních metod k stochastickému gradientnímu poklesu.

Liangchen a spol. [29], ve své práci prezentují dvě varianty optimalizačních algoritmů, jsou jimi AdaBound a AMSBound. Při jejich použití může na začátku procesu trénování dosahovat míra učení hodnot od nuly do nekonečna. S postupujícím se učením se tyto limity postupně přibližují až je docíleno konstantních hodnot změn. V práci je také zmíněno, že tento přístup není jediným možným. Nabízí se například postup s využitím postupného úpadku míry učení, tuto možnost je však třeba experimentálně ověřit.

2.7 AdaNet

AdaNet, projekt zveřejněný týmem Google AI v říjnu letošního roku, je zaměřen na kombinování v praxi rozšířených, mnohdy předtrénovaných, architektur a dynamické úpravě vah vrstev k vytvoření neurální sítě dosahující co nejlepších výsledků pro danou problematiku. Jedná se tedy o problematiku predikce modelu strojového učení. AdaNet je aplikačním rámcem založeným na TensorFlow, čerpajícím z předchozích výzkumů Googlu, jako například automatického strojového učení založeného na evolučních algoritmech. Schopnost generovat celé neuronové sítě, stejně tak jako modifikovat sítě stávající, dělá z AdaNetu výbornou pomůcku pro uživatele bez předchozích zkušeností s návrhem architektury neuronové sítě.

Při generování neuronové sítě AdaNet vytvoří jednoduchý lineární model a postupně navrhuje několik možných architektur, následně porovnává jejich odchylku v přesnosti a schopnost zobecnit daný problém. V případě, že použití dané architektury vede ke zvýšení přesnosti neuronové sítě bez většího dopadu na její schopnost zobecňovat, je tato architektura zakomponována do výsledného modelu. Tímto způsobem je možné vytvořit architekturu sítě, kdy neurony vrstvy mohou být propojeny s neurony jakékoli vrstvy, nikoli pouze s neurony vrstvy následující. Cortes a spol. [30] popisují AdaNet jako schopný generování jak klasických architektur neuronových sítí, tak i poněkud exotičtějších variant, jako například modely prezentované He a spol. [24] či Huang a spol. [31].

Přestože autoři prezentují řešení pouze pro binární klasifikátor, zaručují funkčnost tohoto přístupu pro klasifikátor více tříd. Je také nezbytné zmínit, že se jedná o učení s učitelem. Závěrem jsou prezentovány výsledky AdaNetu na datasetu "CIFAR-10"[32] poukazující na schopnost AdaNetu konkurovat tradičním přístupům pro výběr architektury neuronové sítě k tomuto problému.

2.8 Auto-Keras

Auto-Keras je snahou o snížení výpočetní náročnosti při generování architektur neuronových sítí. Jedná se o aplikační rámec inspirovaný technologií NAS, který však ke hledání optimálních architektur využívá „network morphism“. Jedná se o variaci dědění vlastností neuronových sítí, kdy potomek zdědí vědění svého předchůdce. Učení potomka je rychlejší a potomek dosáhne lepších výsledků. Potomka je také možné modifikovat vložením vrstvy, nebo přeskočením původního spojení. Jak již bylo zmíněno, takto vytvořenou neuronovou síť je pro nárůst výkonnosti nutno trénovat pouze krátkou dobu.

Pro hledání architektur je využit algoritmus Bayeské optimalizace, který se skládá se tří rekurentních kroků. Prvním krokem je načtení stávající architektury

a jejích výsledků. Ve druhém kroku se vygeneruje architektura potomka a dojde k mírným modifikacím architektury. Třetí krok spočívá v učení vygenerované architektury a získání jejího reálného výkonu. Délka tohoto učení není konstantní, nýbrž dynamicky se měnící. Proces učení se ukončí, pokud nedojde k poklesu odchylky během několika epoch. Vhodné je také zmínit, že Auto-Keras využívá paralelizaci CPU a GPU. Další optimalizací je také adaptace strategie hledaní v závislosti na velikosti grafické paměti, pokud je odhadovaná velikost architektury příliš paměťově náročná, navrhovaná architektura je zahozena.

Autoři [33] upozorňují zejména na problém návaznosti některých vrstev neuronové sítě, který může vzniknout při využití této metody. K překonání problémů spjatých se změnou architektury bylo vytvořeno jádro neuronové sítě, které měří počet operací nutných k modifikaci architektury. Dále byl zoptimalizován algoritmus vyhledáváni ve stromové struktuře modifikací architektur. Pro modifikaci architektury potomka je možné využít čtyř operací:

- Vložení vrstvy
- Rozšíření výstupního tensoru vrstvy, nebo přidání více filtrů
- Vložení aditivního spojení mezi vrstvami
- Vložení konvolučního spojení mezi vrstvami

Ukázky vygenerovaných architektur pomocí Auto-Keras jsou k nalezení v příloze práce (Obr. A.9 a A.10).

2.9 NASNet

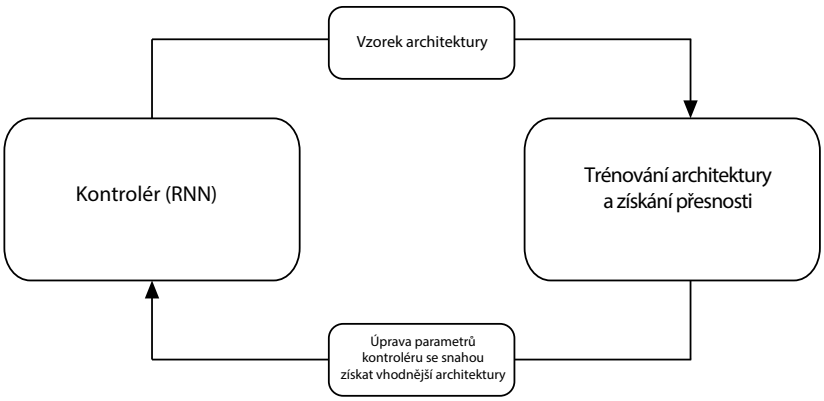
Návrh architektur neuronových sítí pro zpracování velkých datasetů je vysoce náročnou operací. Navrhovaný způsob generování architektur [34] se snaží náročnost těchto operací snížit. Přístup spočívá ve využití malého datasetu k návrhu lehce škálovatelné architektury, kterou je po rozšíření možné využít i na mnohonásobně větších datasetech. Pro generovaní bloků je využita technologie NAS. Tato metoda je založena na teorii, že většina architektur konvolučních neuronových sítí má opakující se charakter. Jedná se o kombinace konvolučních filtračních bank, nelinearit a malého množství vhodně zvolených propojení.

Autoři ve své publikaci [34] využívají pro generaci bloků neuronové sítě dataset CIFAR-10. Nejlepší nalezený blok je poté ve finální architektuře pro dataset ImageNet několikrát využit, vždy s rozdílným nastavením parametrů. Ke zvýšení úspěšnosti takto generovaných architektur přispívá i využití nové regularizační metody „ScheduledDropPath“, která v tomto případě do značné míry eliminuje problém přeučení.

Tento způsob generování architektur neuronových sítí je možné uplatnit nezávisle na hloubce neuronové sítě nebo na velikosti vstupních obrazů. Generátor totiž nevy-

tváří přímo architekturu ale jakousi množinu možných spojení jednotlivých bloků. Je zde také umožněno přizpůsobit generování podle vyžadované hardwarové náročnosti výsledné architektury. Změny se týkají počtu konvolučních bloků a filtračních vrstev.

Základním stavebním blokem NAS je „RNN“, rekurentní neuronová síť, zastávající funkce kontroléru. RNN porovnává výsledky různých architektur využívajících vygenerovaných bloků. Pomocí výsledků těchto testů je upraveno nastavení kontroléru za účelem generování výkonnějších architektur.



Obr. 2.3: Cyklus generování architektury

Vygenerované architektury jsou složeny ze dvou typů konvolučních bloků. První, „standardní“ konvoluční blok vrací mapu znaků bez jakékoli změny rozměrů. Druhý blok, „redukční“, navrací, jak již název napovídá, mapu znaků zmenšenou, v tomto případě o polovinu. Stejně, jako v případě architektur Inception či ResNet, jsou počty opakování bloků měněným parametrem, závislým na náročnosti konkrétního klasifikačním problému. Generované bloky mají konstantní velikost pěti vrstev.

Pro generování bloků je možné využít jak přístup deterministický, tak stochastický. Z publikovaných experimentů [34] je zřejmé, že stochastický přístup je efektivnější v případě menšího počtu generovaných testovacích architektur. Pakliže je generováno více než 10 000 architektur, je jednoznačně vhodnější zvolit přístup deterministický.

2.10 Generování náhodných neuronových sítí s využitím klasických modelů náhodných grafů

Tým vědců z Facebook AI Research velmi nedávno zveřejnil výsledky svého výzkumu z oblasti náhodné generace architektur neuronových sítí [35]. Výsledné architektury

nesou název RandWire. Na rozdíl od generátorů jako je NAS se tento výzkum snaží maximálně limitovat vliv člověka na samotné generování. Zatímco NAS, má člověkem jasně danou množinu možných kombinací architektur, které je možné generovat, vědci z Facebook AI Research staví na trojici skupin klasických modelů náhodných grafů. Jedná se o modely Erdős-Rényi [36], Barabási–Albert [37] a Watts–Strogatz [38]. S pomocí těchto modelů jsou generovány neorientované grafy. Generátor náhodných grafů založený na modelu Watts–Strogatz je součástí praktického výstupu této diplomové práce.

Proces generování náhodného grafu pomocí modelu Watts–Strogatz probíhá následovně. Nejdříve je vytvořen daný počet vrcholů, tyto vrcholy jsou následně umístěny do kruhu. Každý vrchol je poté propojen k danému počtu sousedících vrcholů z obou stran. Takto je vytvořen základní graf, nad kterým jsou nyní prováděny operace pro vytvoření náhodných spojení. Postupným procházením vrcholů ve směru hodinových ručiček může na základě náhodné hodnoty dojít k přesměrování stávající hrany na náhodný vrchol, rozdílný od vrcholu, nad kterým je tato operace prováděna. Při přesměrování je potřeba zamezit sestrojení duplicitních hran. Průchod všemi vrcholy je opakován podle zadaného počtu sousedů.

Aby mohl být takto vygenerovaný neorientovaný graf převeden na architekturu neuronové sítě, je třeba jej upravit na acyklický orientovaný graf. Dále je vhodné vygenerovat vstupní a výstupní vrchol grafu. Síť má tedy pouze jeden, jasně určený, vstup a výstup. Takto upravený graf je možné převést na architekturu neuronové sítě. Orientace hran nyní představuje směr toku dat v neuronové síti.

Každý vrchol grafu, vyjma vrcholu vstupního a výstupního, reprezentuje tři operace. První z operací zajišťuje kombinaci vstupních dat. Z dat vstupujících do vrcholu je vytvořena jejich suma, přičemž každý vstup má svou váhu, obdobně jako je tomu u modelu neuronu. Váhy se mohou dynamicky měnit a nabírají pouze kladných hodnot. Kombinace tímto stylem, na rozdíl od zřetězení, zamezuje nárustu výpočetní obtížnosti následujících konvolucí. Druhou operací je konvoluce nad daty. Je důležité, aby tato transformace nepozměnila počet kanálů. Zachování stejného počtu vstupních kanálů jako kanálů výstupních, je kritické pro možnost tato data dále kombinovat. Poslední operací je vyslání dat do všech výstupních hran.

V oblasti klasifikace obrazu není ve většině aplikací žádoucí, aby síť pracovala s plným rozlišením vstupu po celou dobu průchodu sítí. Architektury sítí jsou proto rozděleny na několik částí, které pracují s rozdílným rozlišením. K vytvoření takto strukturované architektury je proto využito spojení několika neuronových sítí, vygenerovaných shora prezentovaným způsobem. Návaznost těchto částí je řešena pomocí vstupního a výstupního vrcholu každé sítě. Tyto vrcholy jsou propojeny a při přechodu do další části je zajištěno zdvojení počtu kanálů.

Generované neuronové sítě jsou autory [35] testovány pro klasifikaci na datasetu

ImageNet, obsahujícího 1000 tříd [25]. Z těchto tesů vyplývá, že model Watts–Strogatz dosahuje ze všech klasických modelů náhodných grafů nejlepších výsledků. Dalším závěrem je, že průměrná přesnost všech generovaných architektur je konkurenceschopná se stávajícími, člověkem designovanými a optimalizovanými, generátory architektur neuronových sítí.

3 Implementace generátoru neuronových sítí a testování sítí

Praktickým výstupem této práce jsou generátory neuronových sítí implementované v programovacím jazyce Java a Python. První z generátorů přijímá jako vstupní parametry počet generovaných sítí, přibližný počet vrstev v sítích [1], tvar vstupu, dimenze vstupu a požadovaný tvar výstupu. Výstupem tohoto generátoru jsou soubory obsahující kód pro vytvoření těchto sítí za pomocí Keras a TensorFlow. Generátor vygeneruje síť s počtem vrstev určeným uživatelem, kterým následovně, v závislosti na tvaru vstupu či výstupu předchozí vrstvy, přidělí náhodně určený typ vrstvy. Pro správnou funkčnost generátoru je tedy nezbytné, aby každá vrstva měla přístup k několika parametrům svého předchůdce. Mezi tyto parametry patří například údaj o jaký typ vrstvy se jedná a jaký tvar má její výstup. Díky této struktuře je možné implementovat opatření, díky kterým lze počet některých kombinací vrstev eliminovat, nebo je naopak preferovat více. Také je ošetřena funkčnost sítě v případě rozdílných dimenzí vstupů u vrstev, které po sobě následují. Generátor je schopen generovat typy vrstev, jako jsou například vrstvy konvoluční, vrstvy typu dropout, maxpooling, flatten a batch normalization (detailněji popsaných v následující části práce). Další možností generátoru je využití concatenate vrstvy, umožňující zřetězit některé z předchozích vrstev. Kromě generování jednotných vrstev je generátor schopen do generované sítě začlenit bloky vrstev. Takovýto blok je tvořen seskupením několika na sebe navazujících vrstev. Tímto postupem je možné generovat modifikace některých z široce používaných modelů neuronových sítí. Generované sítě je poté možné využít v jakékoli implementaci využívající Keras. Pokud uživatel specifikuje, že výstupní architektura má být siamská neuronová síť, generátor architekturu doplní o blok zajišťující tuto funkcionalitu.

Druhý generátor je do značné míry inspirován výzkumem [35]. Vstupem pro tento generátor je typ klasického modelu náhodného grafu, požadovaný počet vrcholů grafu, tvar vstupních dat a tvar výstupu sítě. Výstupem generátoru je náhodná architektura neuronové sítě, splňující uživatelem zadané parametry, která byla sestrojena podle náhodného grafu.

3.1 Vrstvy neuronové sítě

V této podkapitole jsou zmíněny vybrané vrstvy využívané v neuronových sítích. Všechny tyto vrstvy jsou také implementovány v praktické části práce.

[1] vzhledem k možnosti generováni bloků vrstev

Konvoluční vrstvy Conv1D a Conv2D jsou základem pro jakoukoli konvoluční neuronovou síť. V závislosti na číslici předcházející písmenu D je možné tyto vrstvy využít ke konvoluci vektorů či matic. Využitím těchto vrstev získává síť na robustnosti. V případě využití sítě pro klasifikaci obrazu jde především o redukuci vlivu pozice objektu v obraze a případném dopadu rozmazání hran na výsledném stavu sítě.

Vrstva Dense, jedná se o hustě (plně) propojenou vrstvu sítě, právě tyto vrstvy stojí za většinou "učení se"v neuronové síti.

Dropout vrstva je využívána pro regulaci overfitting (přeučení). V tomto stavu síť výborně reaguje na trénovací množinu a již předešle zpracovaná data, avšak při testování nových dat je odhalena neschopnost sítě generalizovat problém a správně zpracovat neznáma data. Vrstva vynuluje výstup u uživatelem zadané části neuronů vrstvy a tím zajistí, že síti není umožněno výsledek zakládat na výstupu jednoho neuronu.

Vrstvy typu MaxPooling (MaxPooling1D, MaxPooling2D), stejně jako konvoluční vrstvy typu Conv, je možné využít jak u vektorů, tak matic. Vrstvy tohoto typu napomáhají k redukci přetrénování (overfitting) a snižují výpočetní náročnost díky redukci rozměrů vstupu. Například z obrazu o rozměrech 224x224 pixelů se po využití MaxPooling2D vrstvy stává obraz o rozměrech 112x112 pixelů.

Vrstvy typu BatchNormalization, které redukují rozptyl vstupů při změně parametrů předchozí vrstvy v rámci učení a umožnují tak urychlení učícího procesu. Vzhledem k vlastnostem těchto vrstev, lze v některých případech jejich použitím eliminovat potřebu využití dropout vrstev.

Vrstvy typu Flatten, umožňující zploštění vstupních dat, jsou využívány především v posledních vrstvách neuronové sítě, například jako předposlední vrstva redukující výstup na uživatelem požadovaný tvar, například jako počet tříd klasifikační neuronové sítě.

Vrstvy Concatenate umožňují zřetězení předchozích vrstev. Generování vrstev tohoto typu je záměrně limitováno pouze na druhou polovinu generované sítě, čímž se zvyší šance, že generátor v předchozích vrstvách nalezne dvě, ke zřetězení vhodné, vrstvy. Výběr možných vrstev probíhá následujícím způsobem. Při generování první vrstvy druhé poloviny sítě je sestaven list tvarů výstupů již vytvořených vrstev. Algoritmus tento list prohledává, a v případě, že najde dva shodné výstupy, uloží je do listu kandidátů pro zřetězení. Následně je vygenerováno náhodné číslo, určující, která z dvojic vrstev bude použita jako vstupní parametr pro vrstvu Concatenate. V případě, že generátor nenajde vhodné vrstvy ke zřetězení, je vygenerována vrstva jiného typu.

Vzhledem ke zvolené implementaci je možné nastavit pravděpodobnost výskytu jednotlivých typů vrstev, aniž by došlo k porušení funkčnosti generované sítě v dů-

sledku nevhodně zvoleného typu či tvaru výstupu vrstev. Generátor je možné v budoucnosti rozšířit o další typy vrstev.

3.2 Aplikační rámce

Je třeba také krátce zmínit aplikační rámce, s jejichž použitím je možné podobný generátor sestrojit. V době realizace této práce je jasným favoritem v oblasti strojového učení a neuronových sítí TensorFlow[2], jedná se o volně přístupný aplikační rámec pro číslicové výpočty původně vyvinutý vědci a inženýry z týmu Google brain. Tento aplikační rámec je implementován v jazyce Python, více o tomto programovacím jazyce bude řečeno později.

Další variantou může být například aplikační rámec Deep Learning for Java[3], který, jak už je z názvu patrné, je implementován v jazyce Java. Tento aplikační rámec, stejně jako TensorFlow, je volně šiřitelný. Samotné výpočty jsou implementovány v programovacím jazyce C a C++.

Možnou alternativou je také aplikační rámec Caffe[4], disertační práce Yangqing Jia, studenta Kalifornské univerzity v Berkeley. Caffe je implementováno v jazyce Java s případným uživatelským rozhraním v jazyce Python.

Zajímavým konkurentem je také aplikační rámec PyTorch. Jedná se o aplikační rámec implementovaný v jazyce Python. V předchozích letech byl PyTorch využíván zejména pro vědecké a výzkumné účely, v poslední době se však prosazuje i v oblasti komerčního využití. Při použití PyTorch je výhodné použít knihovnu fastai, která staví na PyTorch a nabízí ještě jednodušší a rychlejší práci s neuronovými sítěmi.

3.3 Použitý jazyk

Vzhledem k použití dříve zmiňovaného aplikačního rámce TensorFlow, byl k testování vygenerovaných architektur zvolen vysokoúrovňový skriptovací programovací jazyk Python[5]. Výhodou tohoto jazyka je především jednoduchost jeho syntaxe. Mezi další výhody patří například dostupnost knihoven díky databázi PyPi.

K implementaci samotných generátorů byl použit programovací jazyk Java a Python. Jazyk Java byl využit v první implementaci generátoru, a to zejména z důvodů snadnější lokalizace a odstranění případných chyb v implementaci. Později se tato volba ukázala jako ne příliš šťastná, v době implementace nenabízel TensorFlow

[2]Dostupný na: https://www.tensorflow.org/
[3]Dostupný na: https://deeplearning4j.org/
[4]Dostupný na: http://caffe.berkeleyvision.org/
[5]Dostupný na: https://www.python.org/about/

vhodné programovací rozhraní (API). Z tohoto důvodu bylo zajištění správných dimenzí vstupních tensorů a ověření návaznosti vrstev obecně, velmi obtížnou úlohou.

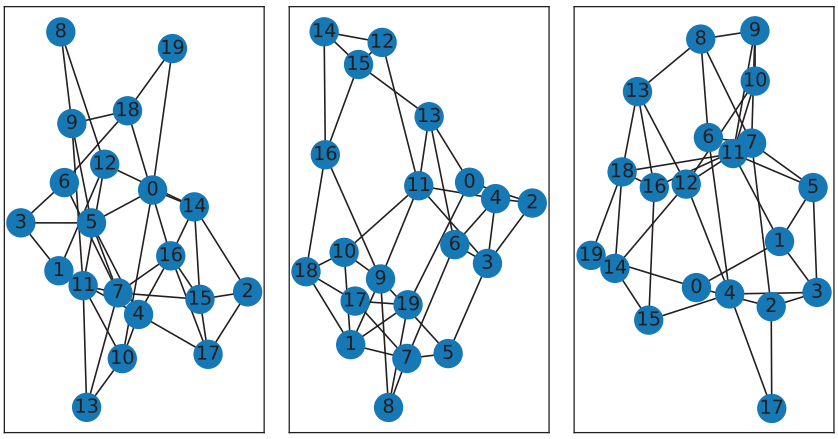
Pro implementaci druhého generátoru byl z předešlých důvodů zvolen jazyk Python. Díky možnosti využití programovacího rozhraní TensorFlow, je možné řadu těchto problému identifikovat již při implementování generátoru, nikoli až při testování výsledné architektury. Dalším důvodem pro zvolení programovacího jazyka Python je možnost využít knihovnu networkx [39] , umožňující generování a práci s grafy.

3.4 Implementace

Implementace prvního generátoru je provedena za pomoci vygenerování několika neuronových vrstev, ze kterých je posléze několik náhodně vybráno a začleněno do vytvořené neuronové sítě. Pro správnou funkčnost je třeba, aby každá vrstva měla přístup k několika parametrům svého předchůdce. Mezi tyto parametry patří například o jaký typ vrstvy se jedná, či jaké jsou dimenze jejího vstupu. Díky této struktuře je možné implementovat opatření, kterými lze počet některých kombinací vrstev redukovat, nebo je naopak preferovat více. Ke kompletaci neuronové sítě je použito sekvenční metody, která je přímočařejší k implementaci, ale nepřináší takovou kontrolu nad procesem vytvoření, jako její alternativy.

V případě implementace druhého generátoru, je nejdříve vygenerována základní verze grafu podle uživatelem zvoleného typu. Tento graf je následně náhodně modifikován. Tímto přístupem je zajištěno generování skutečně náhodné architektury, která není ovlivněna podmínkami pevně danými autorem generátoru. Na základě tohoto grafu je vytvořena architektura neuronové sítě. Každý vrchol grafu je ve finální architektuře reprezentován skupinou vrstev neuronové sítě. Obrázek 3.1 demonstruje možné varianty vygenerovaného grafu. Směr toku dat je vždy od menšího čísla vrcholu ke většímu.

Proces vytvoření neuronové sítě je implementován s využitím dědičnosti tříd aplikačního rámce TensorFlow. Tento přístup přináší hned několik výhod za cenu nutnosti přesně definovat průchod dat modelem. Hlavní výhodou je možnost využít větvících podmínek při sestrojení architektury. Každý vrchol je ve výsledné architektuře realizován blokem vrstev. První vrstvou tohoto bloku je rektifikovaná lineární jednotka. Následující vrstva je hloubková dvou dimenzionální konvoluce. Po této vrstvě je využita konvoluce klasická. Celý blok je následně ukončen vrstvou typu BatchNormalization. Pakliže má vrchol více vstupních hran, jsou tato data sjednocena sumarizací nad jednotkovou maticí. Tímto postupem je zachován tvar vstupních dat. Protože konvoluční neuronové sítě mají v drtivé většině využití pouze jeden vstup a výstup, jdou tyto vrstvy uměle vygenerovány jako poslední.



Obr. 3.1: Ukázka generovaných grafů

3.5 Databáze

Sítě budou učeny a testovány za pomoci nyní volně přístupné databáze společnosti Google s názvem Google-Landmarks[6], skládající se z více než dvou milionů fotografií, zobrazujících třicet tisíc geografických lokací. Zveřejnění tohoto datesetu je spjato s vyhlášením dvou soutěžních kategorií serveru Kaggle. První, s názvem "Landmark Recognition", si klade za cíl sestrojit neuronovou síť se schopností správně rozeznat lokaci zobrazenou na fotografii. Druhá kategorie, "Landmark Retrieval", je zaměřena na nalezení fotografií zobrazujících stejnou lokaci jako vstup sítě.

Obr. 3.2: Ukázka datasetu Google-Landmarks [4]

[6]Blog popisující dataset: https://goo.gl/bN66nL

Společně s datasetem byl veřejnosti také zpřístupněn zdrojový kód konvoluční neuronové sítě s názvem Large-Scale Image Retrieval with Attentive Deep Local Features, vhodný k předešle zmíněným aplikacím [4].

3.6 Google Colaboratory

Pro samotné testování architektur je využita cloud služba Google Colaboratory. Tento projekt byl vyvinut ke zlepšení přístupnosti strojového učení. Jedná se o verzi Jupyter notebooku, kterou je možné používat bez jakéhokoli předešlého nastavování. Colaboratory podporuje jazyky Python 2.7 a Python 3.6, který tato diplomová práce využívá. Tato služba byla vybrána zejména díky možnosti využít službou volně poskytované GPU pro trénování a testování vygenerovaných architektur. Paměť tohoto GPU je necelých 15 GB. Vzhledem k možnosti zneužití služby pro těžbu kryptoměn, je maximální délka výpočtů limitována na 12 hodin. Tato limitace je kritická zejména při velkých datasetech. Alternativou za tuto službu může být například služba Kernels poskytovaná portálem Kaggle, nebo sada produktů Microsoft Azure Machine Learning.

3.7 Testování vygenerovaných sítí

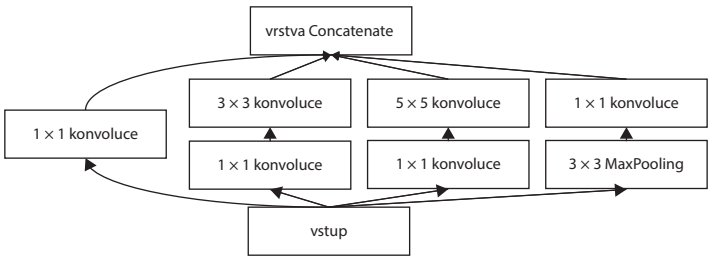
Níže je popsán postup testování sítí. Prvním cílem bylo generování sítí schopných správné funkčnosti na datasetu ručně psaných číslic MNIST [40], mnohdy označovaném jako "Hello world!"neuronových sítí.

Tímto testováním je zaručena základní funkčnost všech generovaných sítí v případě jejich využití k řešení klasifikačního problému. Vzhledem k triviálnosti problému klasifikace datasetu ručně psaných číslic MNIST, byly sítě následovně testovány na datasetu MNIST fashion, který obsahuje 10 tříd oblečení, zaručuje větší robustnost testování sítě při zachování parametrů datasetu, jako je počet tříd, či tvar vstupních vektorů.

Dalším testem vygenerovaných sítí je podmnožina datasetu Google-Landmarks, obsahující 4 675 obrázků rozdělených do 100 tříd. Ke zpracování výsledků je použit nástroj TensorBoard, umožňující vizualizaci architektury sítě, výsledků nebo například možnost analyzovat vstupní dataset.

Původní generování pomocí náhodného výběru vrstev se ukázalo pro zadaný dataset jako nevhodné, přesto, že sítě na MNIST datasetu vykazovaly uspokojující výsledky. Při použití na datasetu Google-Landmarks dosahovaly tyto sítě ve většině případů pouze úspěšnosti kolem 30% při klasifikací 10 tříd. Příklady architektur těchto generovaných sítí, stejně jako jejich výsledky, jsou součástí přílohy A.1, A.2,

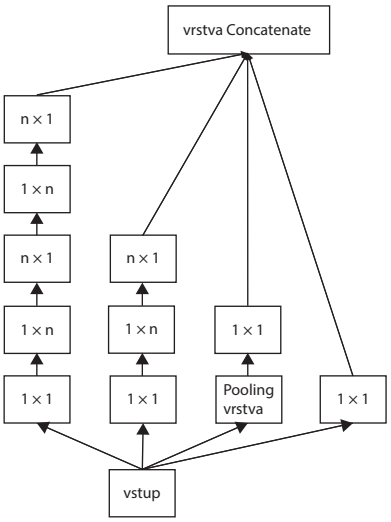
A.3. Z těchto důvodů bylo nutné generátor upravit změnou šance k vygenerování určitých typů vrstev a implementováním možnosti vygenerovat blok vrstev, jehož základní myšlenkou je modifikace známých architektur Inception [41] a InceptionV3 [42]. Obě tyto architektury jsou založeny na principu spojení několika rozdílných konvolučních vrstev. Tato myšlenka je jasně patrná ze schématu architektury sítě Inception (obrázek 3.3).



Obr. 3.3: Schéma bloku Inception

Tato architektura byla pro implementaci v praktické části této práce modifikována odstraněním větvě využívající vrstvy typu MaxPooling a Conv2D. Výsledky vygenerovaných sítí, využívajících dvojici předešle popsaných Inception bloků, jasně předčily náhodně generované architektury, a to až o více jak dvojnásobek (viz tabulka 4.1 na straně 40).

Jak již bylo zmíněno, byla implementována i modifikace architektury InceptionV3 [42]. Tato architektura čerpá z úspěchů svého předchůdce Inception [41] a snaží se ho nadále vylepšit. V rámci práce implementovaná architektura konkrétně modifikuje následující verzi této architektury.



Obr. 3.4: Schéma bloku InceptionV3

Modifikace spočívají opět v odstranění větve využívající vrstvy typu MaxPooling a Conv2D. Při generování bloku této architektury je zvoleno náhodné číslo v rozsahu 3 až 10 a dosazeno za proměnnou n. Je tedy možné generovat různé kombinace rozdílných verzí této architektury v jedné síti. Příklady generovaných sítí s využitím bloků obou architektur, jak Inception, tak InceptionV3, jsou prezentovány v příloze této práce na straně 51.

Z výsledků, zejména na 100 třídách (strana 66), je patrné, že sítě při testování nejsou schopny dostatečně zobecňovat. Tento problém se sice projevuje velmi dobrou výkoností sítí na trénovací množině, avšak přesnost sítě na množině validační je nedostatečná. K redukci tohoto problému byl generátor modifikován, přesněji, byl zvýšen rozsah parametru vrstvy dropout, ovlivňující schopnost sítě zobecňovat. Dalším možným řešením tohoto problému je změna parametrů generátoru vstupních dat, umožňující například aplikování rotace či posunu.

Při testování vygenerovaných sítí na rozšířeném datasetu se začaly projevovat nedostatky a limity implementace tímto způsobem. Generování s využitím pouhé modifikace stávajících architektur nebyla schopna dosáhnout konkurence schopných výsledků. Pro testování obsáhlejšího datasetu bylo nutné generovat architektury o několika násobném počtu vrstev než v předchozích případech, což s sebou přinášelo problém velké paměťové náročnosti. Díky implementaci v jazyce Java nebylo také možné tyto problémy jednoduše odbourat. Generování tímto stylem se z předešle zmíněných důvodů ukázalo jako nevhodné.

Jako velmi perspektivní se ovšem ukázalo generovaní s využitím klasickým modelů náhodných grafů. Kombinací těchto postupů a využitím plného potenciálu apli-

kačního rámce TensorFlow, díky změně jazyka implementace, bylo možné docílit generování opravdu náhodných architektur s možností detailního nastavení parametrů. Na rozdíl od předchozího, dovoluje tento způsob implementace generovat komplexní, plně náhodné architektury, aniž by při nárůstu velikosti sítě došlo k nekontrolovatelnému vzrůstu paměťové náročnosti.

4 Výsledky testování neuronových sítí

V této kapitole jsou prezentovány ukázky možných generovaných neuronových sítí (vyobrazeny v příloze, strana 51) a jejich výsledky na podmnožině databáze Google-Landmarks. Sítě byly testovány na problematice klasifikování s 10, 100, 1000 a 3472 tříd. V případě 10 tříd je síť učena pomocí 413 fotografií a poté testována na 200 fotografiích, při klasifikaci 100 tříd, jsou tyto hodnoty 3147 a 1528. Test o 1000 třídách obsahuje 45373 trénovacích fotografií a 22172 fotografií pro validaci funkčnosti sítě. Finální test 3472 tříd obsahuje 165346 trénovacích a 80917 validačních fotografií. Fotografie byly z důvodu snížení hardwarové náročnosti zmenšeny na rozlišení 128x128 pixelů.

Při testování sítí je viditelná jasná korelace mezi komplexností problému a nutností využít komplexnější architekturu neuronové sítě. Sítě využívající pouze základních architektur s limitovaným počtem vrstev, jako například síť 1.1 (schéma A.1 v příloze), jsou vhodné zejména pro méně komplexní úlohy, jakou je například klasifikace 10 tříd, ve které tato síť dosáhla validační přesnosti 78% (jak je patrné z grafu A.13). Naopak, architekturou komplexnější neuronové sítě, jako například síť 2.5, schéma A.8, využívající bloky architektur Inception, byly schopné dosáhnout validační přesnosti až 64% při klasifikování 1000 tříd (graf A.25). Vzhledem ke tvaru a hodnotám křivek přesnosti a odchylky, lze předpokládat, že tato přesnost není maximální dosažitelnou a při delším učení sítě by mělo dojít k jejímu vzrůstu. Křivka odchylky také naznačuje velmi dobrou schopnost sítě zobecňovat daný problém díky vhodně zvolené pozici a nastavení vrstvy typu dropout.

To, že pouhá komplexnost architektury sítě neznamená vyšší přesnost, je viditelné na síti 2.2 (schéma A.5) a jejím výsledku při klasifikaci 1000 tříd (grafy A.24). Z nich je patrné, že síť není schopna se naučit vhodné postupy pro klasifikaci fotografií. Tuto chybu lze nejspíše přisuzovat nevhodnému nastavení vrstvy typu dropout, díky kterému má většina výstupů předchozích vrstev jen velmi limitovaný vliv na hodnotu finálního výstupu sítě. Je tedy nutné volit vhodné nastavení jednotlivých vrstev neuronové sítě.

Následující tabulka prezentuje výsledky testovaných sítí na podmnožině databáze Google-Landmarks při klasifikaci 10 možných tříd. Z tabulky je patrné, že v této situaci neexistuje korelace mezi přesností sítí a počtem jejich vrstev. Při generování sítí je nutné se zaměřit na schopnost sítě dostatečně zobecňovat a použité vrstvy vhodně propojit.

Tab. 4.1: Výsledky generovaných neuronových sítí při klasifikaci 10 tříd

Neuronová síť	Počet vrstev	Trénovací přesnost [%]	Validační přesnost [%]
1.1	7	100	78
1.2	33	31	34
1.3	25	69	44
2.1	**22**	**100**	**89**
2.2	20	99	83
2.3	26	99	83

Tabulka 4.2 prezentuje výsledky sítí při klasifikaci 100 tříd. Z těchto výsledků je patrné, že sítě jsou schopny se velmi dobře naučit klasifikaci trénovací množiny, nejsou však schopné tyto zkušenosti aplikovat na nová data, jedná se tedy o problém přeučení neuronové sítě.

Tab. 4.2: Výsledky generovaných neuronových sítí při klasifikaci 100 tříd

Neuronová síť	Trénovací přesnost [%]	Validační přesnost [%]
2.1	100	62
2.2	**100**	**63**
2.3	99	56
2.4	98	58
2.5	99	61

Nadcházející tabulka zachycuje výsledky dvou neuronových sítí použitých pro klasifikaci 1000 tříd. Tyto výsledky naznačují, že vrstvy sítě 2.2 nebyly vhodně vygenerovány a síť tak není schopna správné funkčnosti. Tento jev je více patrný z grafu A.24 na straně 67. Jako velmi perspektivní se však jeví neuronová síť 2.5 (schéma A.8), neboť tato síť ukazuje jen relativně malé známky neschopnosti zobecňovat a dosahuje velmi vysoké přesnosti.

Tab. 4.3: Výsledky generovaných neuronových sítí při klasifikaci 1000 tříd

Neuronová síť	Trénovací přesnost [%]	Validační přesnost [%]
2.2	21	19
2.5	**73**	**64**

Tabulka, která následuje, porovnává výkon nejúspěšnější z generovaných neuronových sítí pro klasifikaci 1000 tříd (síť 2.5) a předučené sítě InceptionV3 [42], dostupné v aplikačním rámci TensorFlow (v tomto případě byla označena jako síť referenční). Při stejném nastavení generátoru vstupních dat vykazuje referenční síť větší známky přeučení. Tuto chybu sítě by bylo možné redukovat použitím vrstev typu dropout. Z výsledků (grafy A.26) je také patrné, že obě sítě dosáhly při testování velmi podobných hodnot validační přesnosti, přesto, že referenční síť obsahuje několikanásobně větší počet vrstev a její architektura je komplexnější. Z těchto výsledků lze tedy usoudit, že v tomto případě vyšší počet vrstev, stejně tak jako komplexnější architektura sítě, přináší pouze velmi malé zvýšení její přesnosti.

Tab. 4.4: Výsledky generované a referenční sítě při klasifikaci 1000 tříd

Neuronová síť	Trénovací přesnost [%]	Validační přesnost [%]
2.5	73	64
Referenční síť	**85**	**68**

Výsledky finálního testování 3472 tříd jsou prezentovány níže. Z těchto výsledků je patrné, že náhodné generování architektur založené na modifikaci nevedlo ke zvýšení přesnosti. Průběhy testování jsou k nalezení v příloze práce (str. 68, 69).

Tab. 4.5: Výsledky generované a referenční sítě při klasifikaci 3472 tříd

Neuronová síť	Trénovací přesnost [%]	Validační přesnost [%]
3.1	3	3
3.2	17	18
Referenční síť	**65**	**63**
Referenční síť (nepředučena)	36	33

Následující tabulka poukazuje na značný rozptyl počtu parametrů i přes totožné nastavení generátoru náhodných architektur. Nutné je zmínit, že počet parametrů úzce souvisí s paměťovými nároky při použití sítě.

Tab. 4.6: Počet parametrů generovaných sítí prvního generátoru

Požadovaný počet vrstev	Počet parametrů
10	382 307 987
10	1 443 639 825
10	757 202 606
20	881 900 103
20	306 938 064
20	2 038 916 683

Pro porovnání obou přístupů generování je prezentována také tabulka počtu parametrů vygenerovaných architektur pro druhý implementovaný generátor.

Tab. 4.7: Počet parametrů generovaných sítí druhého generátoru

Počet grafů v architektuře	Požadovaný počet vrcholů v grafu	Počet parametrů
4	10	498 548
4	10	498 524
4	10	434 790
4	20	943 156
4	20	994 166
4	20	994 163

Z provedených experimentů lze konstatovat, že generátor implementovaný v jazyce Java, generující architektury pomocí modifikace stávajících Inception architektur je do značné míry nevhodný, zejména pro neschopnost plně kontrolovat paměťové nároky generovaných architektur. Vzhledem ke stochastickému přístupu je efektivně použitelné jen velmi malé procento vygenerovaných sítí. Časová neefektivnost tohoto přístupu spočívá zejména v nutnosti trénovat velké množství sítí, kde jen velmi malá podmnožina generovaných sítí vykazuje kladné výsledky. Pro zvolený dataset se ukázalo jako vhodnější využit klasické architektury InceptionV3 předtrénované na datasetu ImageNet.

Na rozdíl od prvního generátoru, generátor implementovaný v jazyce Python, využívající klasické modely náhodných grafů, dovoluje do značné míry kontrolovat paměťovou náročnost generované architektury při zachování náhodného procesu generování. Implementace v jazyce Python zároveň umožňuje použití nejnovějších technologií v Tensorflow, verze 2.0. Jedním z příkladů těchto nově dostupných technologií je „eager execution“, umožňující vykonávání kódu obdobně, jako je tomu

u typického Python kódu. Je tak možné využít řídících struktur programu přímo v průběhu generování architektur neuronových sítí.

5 Závěr

Neuronové sítě je možné využit v nespočetném množství aplikací, od nápovědy vyhledávače, přes predikci hodnot nemovitostí, po klasifikaci objektů na obraze. Neuronové sítě jsou v dnešní době velmi populárním nástrojem, a to zaslouženě. Díky schopnosti řešit vysoce paralelizované a komplexní úlohy si neuronové sítě vysloužily zájem předních organizací a firem v oblasti výpočetní techniky. Giganti jako Google a Microsoft vyvíjí veřejnosti přístupné aplikační rámce a knihovny pro experimentální účely. Příkladem takovéhoto aplikačního rámce je například Tensorflow, který je v této práci také prakticky využit. Dalším příkladem může být AdaNet [30], nadstavba Tensorflow, umožňující řízené generování neuronových sítí pro specifický problém. Vzhledem ke komplexnosti těchto problémů je výzkum v oblasti neuronových sítí velmi slibný. Dokazují to i portály pořádající, mnohdy veřejnosti volně přístupné, soutěže s finančním ohodnocením. Jedním z těchto portálů je například kaggle.com. Zapojení veřejnosti je zejména vhodné při sběru dat pro pozdější zpracování. Neuronové sítě mnohdy vyžadují pro své učení databáze o velikostech stovek tisíců hodnot.

Práce z těchto důvodů objasňuje problematiku neuronových sítí, jejich elementární jednotku, neuron, jejich možné dělení, jak podle přístupu k učení sítě, tak podle síťové architektury. Blíže prezentuje základní princip konvolučních neuronových sítí, které jsou využity v praktické části této práce pro klasifikaci fotografií lokací z databáze Google-Landmarks [3]. Kromě předešle zmíněných témat, práce čtenáři prezentuje několik soudobých výzkumů v oblasti neuronových sítí.

Praktická část této práce je zaměřena na vývoj a implementaci dvou automatických softwarových generátorů dopředných neuronových sítí, schopných vytvářet náhodné varianty konvolučních neuronových sítí. Vygenerované sítě jsou ověřeny na databázi Google-Landmarks[3] a výsledky jsou následně patřičně prezentovány. Jako hlavní přínos této práce lze tedy považovat vývoj a implementaci generátorů v jazyce Java a Python, generujících náhodné neuronové sítě použitelné v aplikačním rámci TensorFlow. Testování vygenerovaných sítí a prezentace jejich výsledků je rovněž součástí praktického výstupu. Z těchto výsledků jsou zvolené postupy generování okomentovány a porovnány.

Výsledek provedených experimentů je následující. V porovnání s architekturou InceptionV3 nepřinesla stochastická modifikace architektur Inception na zadaném datasetu zvýšení přesnosti. Hlavní nevýhodou tohoto přístupu je limitovaná kontrola nad paměťovými nároky generovaných architektur a nízké procento přínosných modifikací, zejména u architektur s větším počtem skrytých vrstev.

Literatura

[1] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019. URL: `https://openreview.net/forum?id=Bygh9j09KX`.

[2] Shuyang Sun, Jiangmiao Pang, Jianping Shi, Shuai Yi, and Wanli Ouyang. Fishnet: A versatile backbone for image, region, and pixel level prediction. *CoRR*, abs/1901.03495, 2019. URL: `http://arxiv.org/abs/1901.03495`, `arXiv:1901.03495`.

[3] Hyeonwoo Noh, Andre Araujo, Jack Sim, and Bohyung Han. Image retrieval with deep local features and attention-based keypoints. *CoRR*, abs/1612.06321, 2016. URL: `http://arxiv.org/abs/1612.06321`, `arXiv:1612.06321`.

[4] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Largescale image retrieval with attentive deep local features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3456–3465, 2017.

[5] Simon S. Haykin. *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition, 2009.

[6] Eva Volná. Evoluční algoritmy a neuronové sítě. *Ostrava: Ostravská univerzita v Ostravě*, 2013.

[7] Jiří Kupka. Konvoluční neuronové sítě v analỳze obrazu. 2016.

[8] Ben Kröse, Ben Krose, Patrick van der Smagt, and Patrick Smagt. An introduction to neural networks. 1993.

[9] Simon Haykin and Neural Network. A comprehensive foundation. *Neural networks*, 2(2004):41, 2004.

[10] Eva Volná. Neuronové sítě 1. *Ostrava: Ostravská univerzita v Ostravě. Vydání: druhé*, 2008.

[11] David Smolík et al. Identifikace obličej pomocí metod počítačového vidění. 2017.

[12] Kidong Lee, David Booth, and Pervaiz Alam. A comparison of supervised and unsupervised neural networks in predicting bankruptcy of korean

firms. *Expert Systems with Applications*, 29(1):1 – 16, 2005. URL: http://www.sciencedirect.com/science/article/pii/S0957417405000023, doi:https://doi.org/10.1016/j.eswa.2005.01.004.

[13] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Ong Gee Hock, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, and Guy Boudoukh. Can fpgas beat gpus in accelerating next-generation deep neural networks? In *Proceedings of the 2017 ACM/-SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pages 5–14, New York, NY, USA, 2017. ACM. URL: http://doi.acm.org/10.1145/3020078.3021740, doi:10.1145/3020078.3021740.

[14] Michal Hradiš and Aleš Láník. Klasifikace scén a tagování obrázků. Technical report, 2014. URL: http://www.fit.vutbr.cz/research/view_pub.php.en?id=11031.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130, 1988.

[17] Jonas Kubilius, Stefania Bracci, and Hans P. Op de Beeck. Deep neural networks as a computational model for human shape sensitivity. *PLOS Computational Biology*, 12(4):1–26, 04 2016. URL: https://doi.org/10.1371/journal.pcbi.1004896, doi:10.1371/journal.pcbi.1004896.

[18] Nikolaus Kriegeskorte. Deep neural networks: A new framework for modeling biological vision and brain information processing. *Annual Review of Vision Science*, 1(1):417–446, 2015. URL: https://doi.org/10.1146/annurev-vision-082114-035447, arXiv:https://doi.org/10.1146/annurev-vision-082114-035447, doi:10.1146/annurev-vision-082114-035447.

[19] Thomas S. A. Wallis, Christina M. Funke, Alexander S. Ecker, Leon A. Gatys, Felix A. Wichmann, and Matthias Bethge. A parametric texture model based on deep convolutional features closely matches texture appearance for humansWallis et al. *Journal of Vision*, 17(12):5–5, 10 2017. URL: https://doi.org/10.1167/17.12.5, arXiv:https://jov.arvojournals.org/arvo/content_public/journal/jov/936521/i1534-7362-17-12-5.pdf, doi:10.1167/17.12.5.

[20] Wieland Brendel and Matthias Bethge. Approximating CNNs with bag-of-local-features models works surprisingly well on imagenet. In *International Conference on Learning Representations*, 2019. URL: https://openreview.net/forum?id=SkfMWhAqYQ.

[21] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

[22] Jong Hyun Kim, Ganbayar Batchuluun, and Kang Ryoung Park. Pedestrian detection based on faster r-cnn in nighttime by fusing deep convolutional features of successive images. *Expert Systems with Applications*, 114:15 – 33, 2018. URL: http://www.sciencedirect.com/science/article/pii/S0957417418304354, doi:https://doi.org/10.1016/j.eswa.2018.07.020.

[23] Boukaye Boubacar Traore, Bernard Kamsu-Foguem, and Fana Tangara. Deep convolution neural network for image recognition. *Ecological Informatics*, 48:257 – 268, 2018. URL: http://www.sciencedirect.com/science/article/pii/S1574954118302140, doi:https://doi.org/10.1016/j.ecoinf.2018.10.002.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL: http://arxiv.org/abs/1512.03385, arXiv:1512.03385.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. URL: http://arxiv.org/abs/1409.0575, arXiv:1409.0575.

[26] Hervé Jégou and Ondrej Chum. Negative evidences and co-occurrences in image retrieval: the benefit of PCA and whitening. In *ECCV - European Conference on Computer Vision*, Firenze, Italy, October 2012. URL: https://hal.inria.fr/hal-00722622.

[27] Rocio Gonzalez-Diaz, Eduardo Paluzo-Hidalgo, and Miguel A. Gutiérrez-Naranjo. Representative datasets for neural networks. *Electronic Notes in Discrete Mathematics*, 68:89 – 94, 2018. Discrete Mathematics Days 2018. URL: http://www.sciencedirect.com/science/article/pii/S1571065318301070, doi:https://doi.org/10.1016/j.endm.2018.06.016.

[28] Joseph SB Mitchell and Wolfgang Mulzer. 32 proximity algorithms.

[29] Liangchen Luo, Yuanhao Xiong, and Yan Liu. Adaptive gradient methods with dynamic bound of learning rate. In *International Conference on Learning Representations*, 2019. URL: https://openreview.net/forum?id=Bkg3g2R9FX.

[30] Corinna Cortes, Xavi Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. *CoRR*, abs/1607.01097, 2016. URL: http://arxiv.org/abs/1607.01097, arXiv:1607.01097.

[31] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL: http://arxiv.org/abs/1608.06993, arXiv:1608.06993.

[32] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, 2009. URL: http://www.cs.toronto.edu/~{}kriz/learning-features-2009-TR.pdf.

[33] Haifeng Jin, Qingquan Song, and Xia Hu. Efficient neural architecture search with network morphism. *CoRR*, abs/1806.10282, 2018. URL: http://arxiv.org/abs/1806.10282, arXiv:1806.10282.

[34] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. URL: http://arxiv.org/abs/1707.07012, arXiv:1707.07012.

[35] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring Randomly Wired Neural Networks for Image Recognition. *arXiv e-prints*, page arXiv:1904.01569, Apr 2019. arXiv:1904.01569.

[36] On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.

[37] Romualdo Pastor-Satorras, Miguel Rubi, and Albert Diaz-Guilera. *Statistical mechanics of complex networks*, volume 625. Springer Science & Business Media, 2003.

[38] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440, 1998.

[39] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[40] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, Nov 2012. doi:10.1109/MSP.2012.2211477.

[41] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL: http://arxiv.org/abs/1409.4842, arXiv:1409.4842.

[42] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL: http://arxiv.org/abs/1512.00567, arXiv:1512.00567.

[43] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1):43–52, Dec 2010. URL: https://doi.org/10.1007/s13042-010-0001-0, doi:10.1007/s13042-010-0001-0.

[44] Michal Hradiš, Jan Kotera, Pavel Zemčík, and Filip Šroubek. Convolutional neural networks for direct text deblurring. In *Proceedings of BMVC 2015*. The British Machine Vision Association and Society for Pattern Recognition, 2015. URL: http://www.fit.vutbr.cz/research/view_pub.php?id=10922.

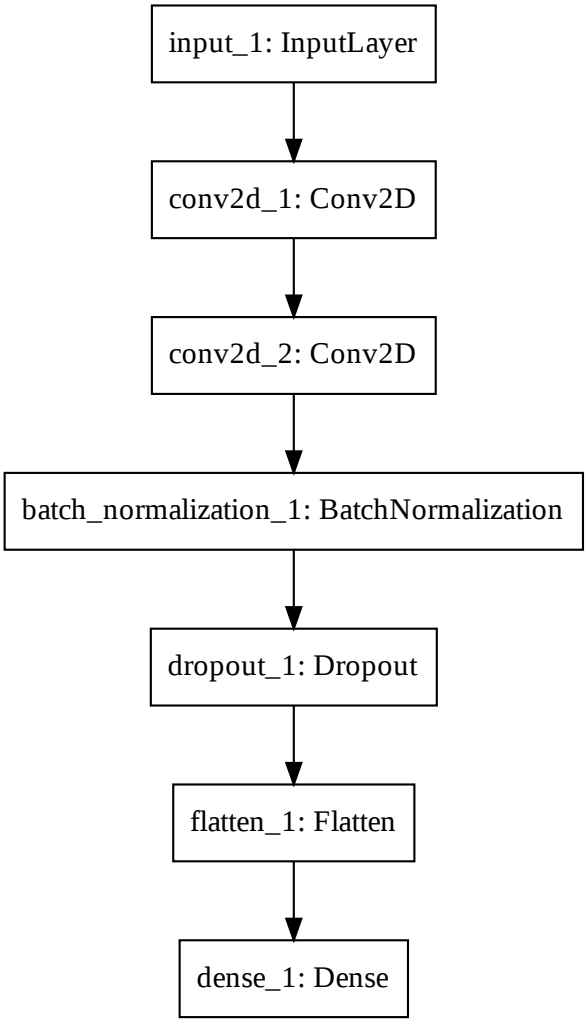
[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[46] Matthew Mackay, Paul Vicol, Jonathan Lorraine, David Duvenaud, and Roger Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *International Conference on Learning Representations*, 2019. URL: https://openreview.net/forum?id=r1eEG20qKQ.

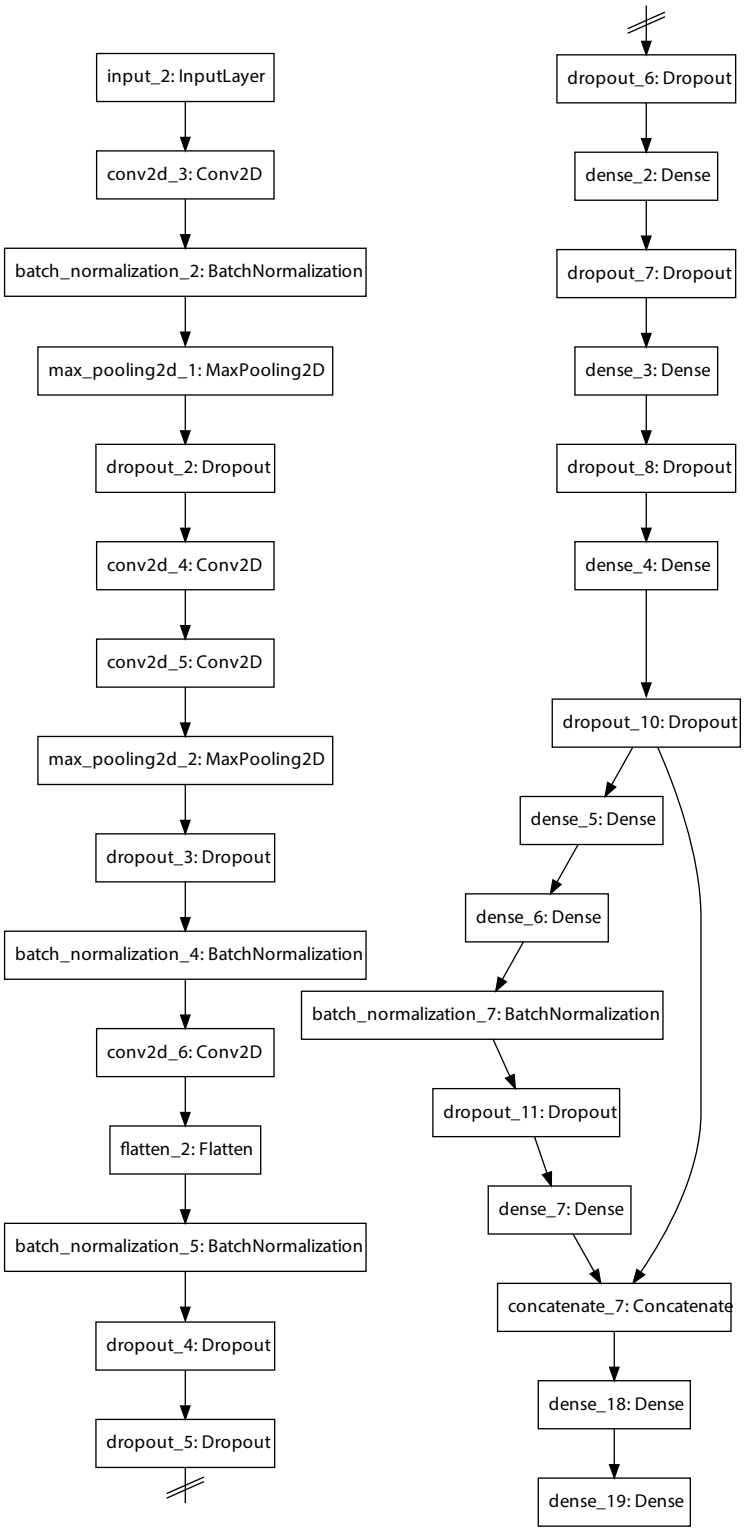
Seznam příloh

A Ukázky generovaných neuronových sítí a jejich výsledky

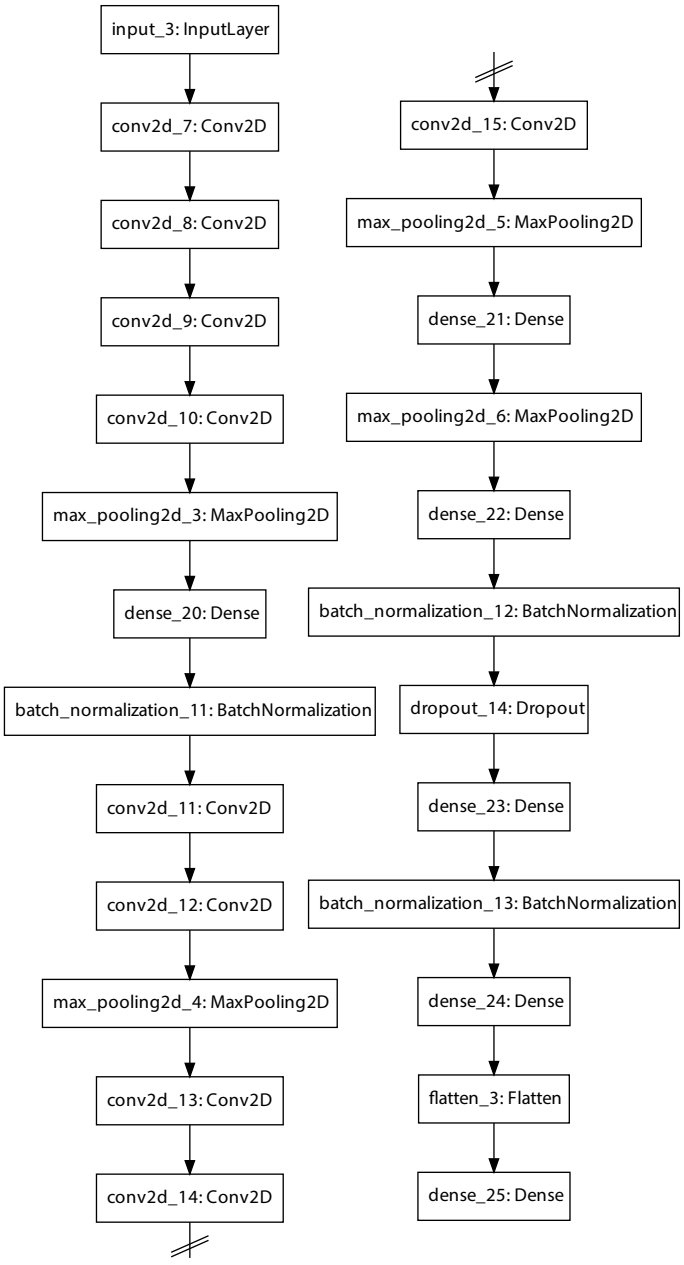
Následující strany obashují schémata architektur vygenerovaných neuronových sítí. Příloha dále obsahuje výsledky veškerého testování.



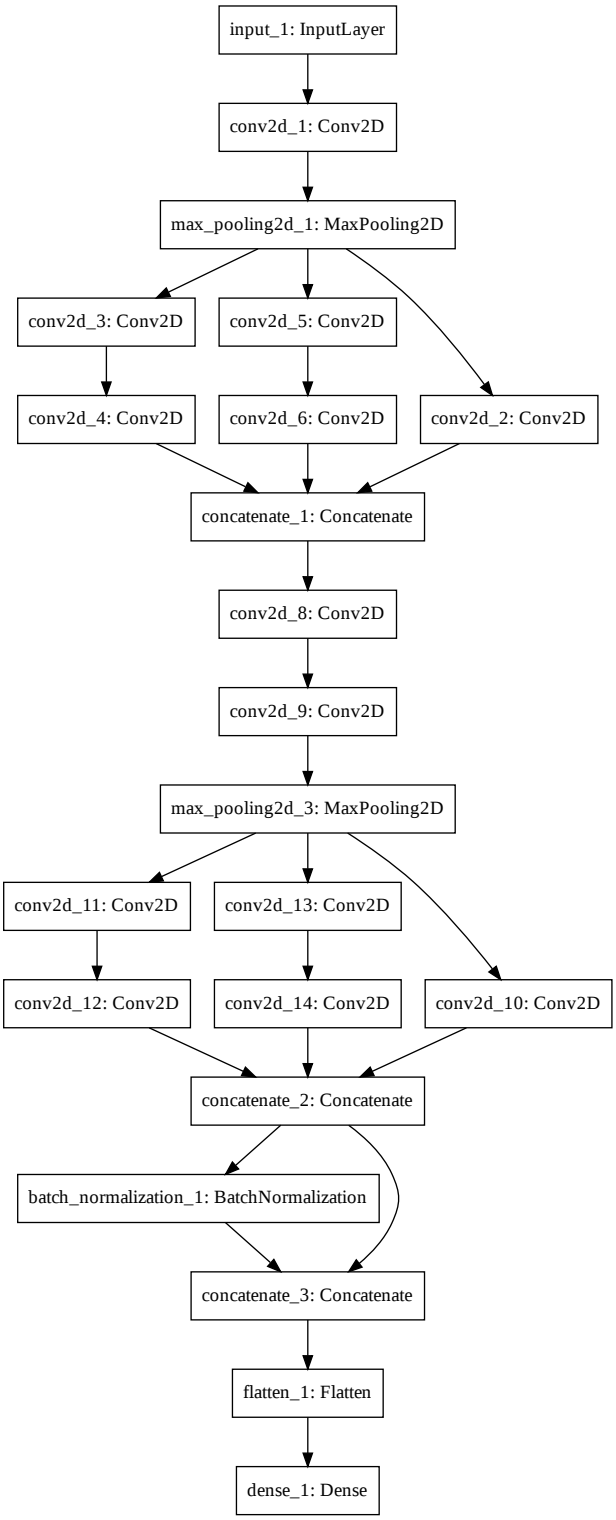
Obr. A.1: Neuronová síť 1.1



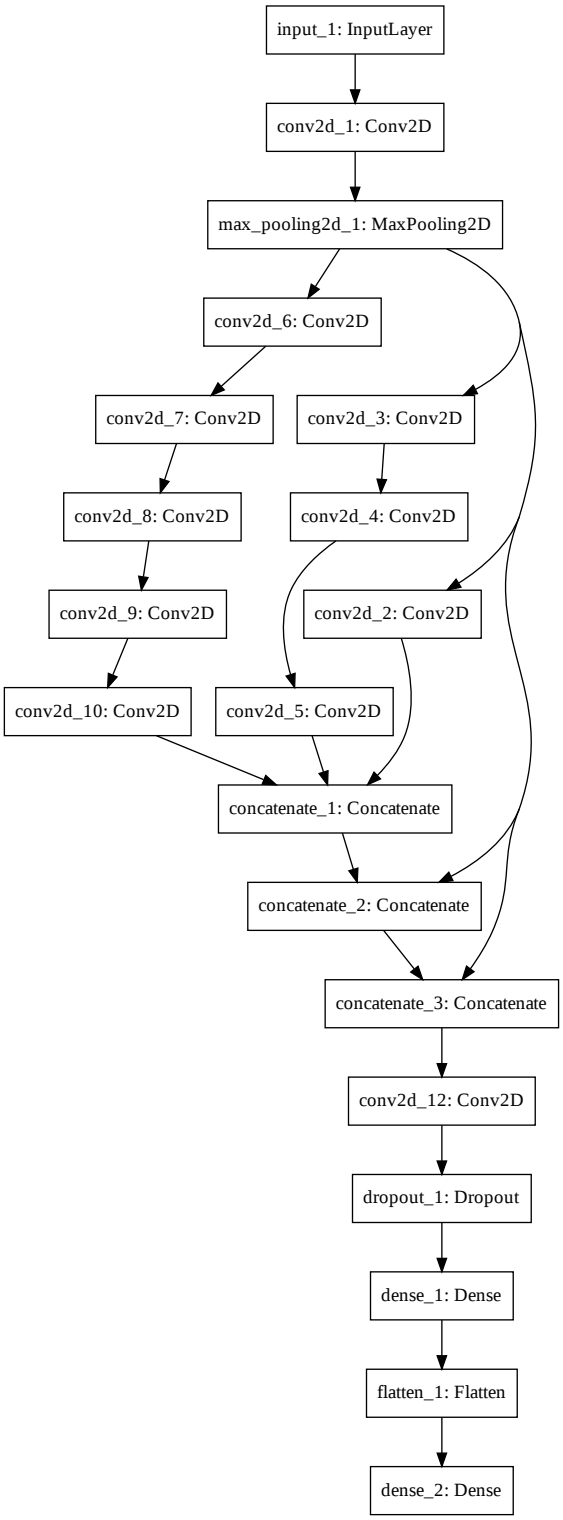
Obr. A.2: Neuronová síť 1.2



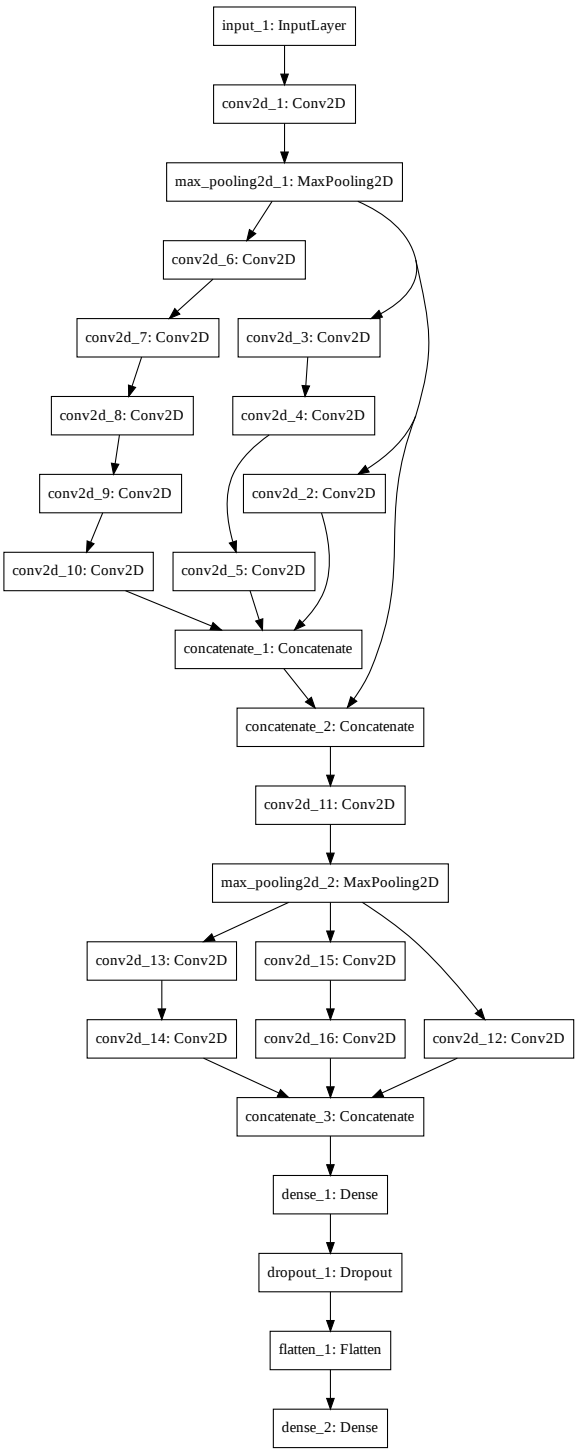
Obr. A.3: Neuronová síť 1.3



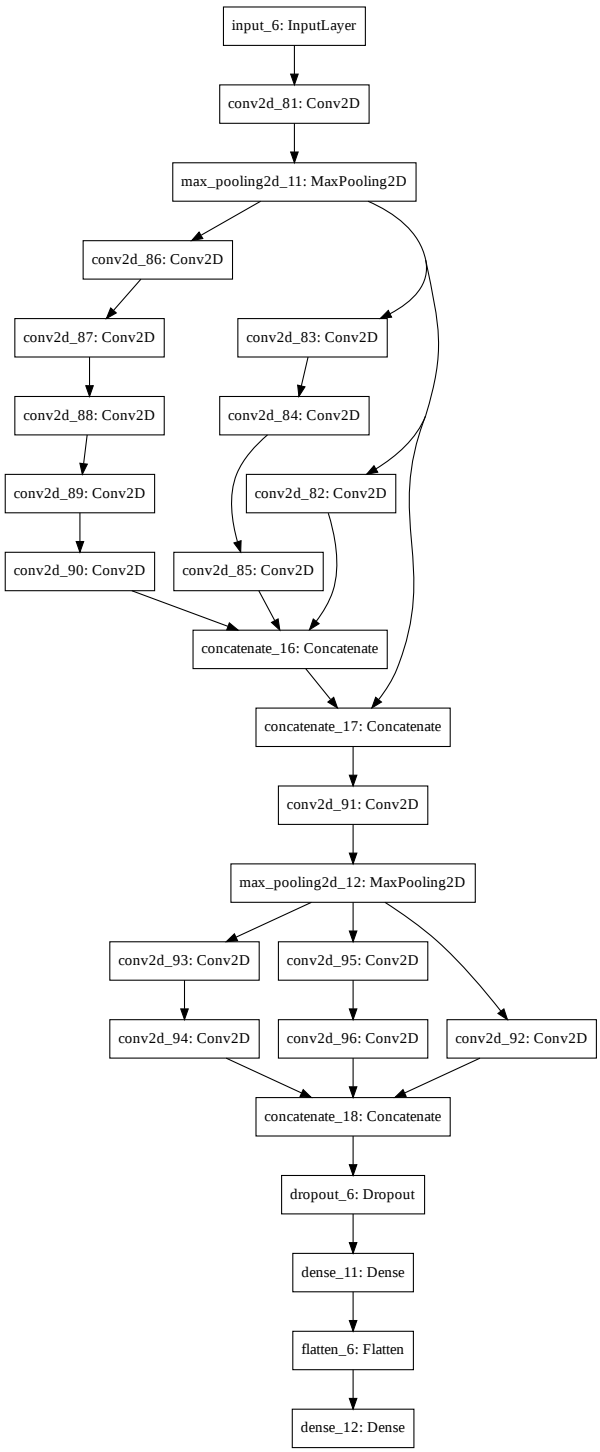
Obr. A.4: Neuronová síť 2.1



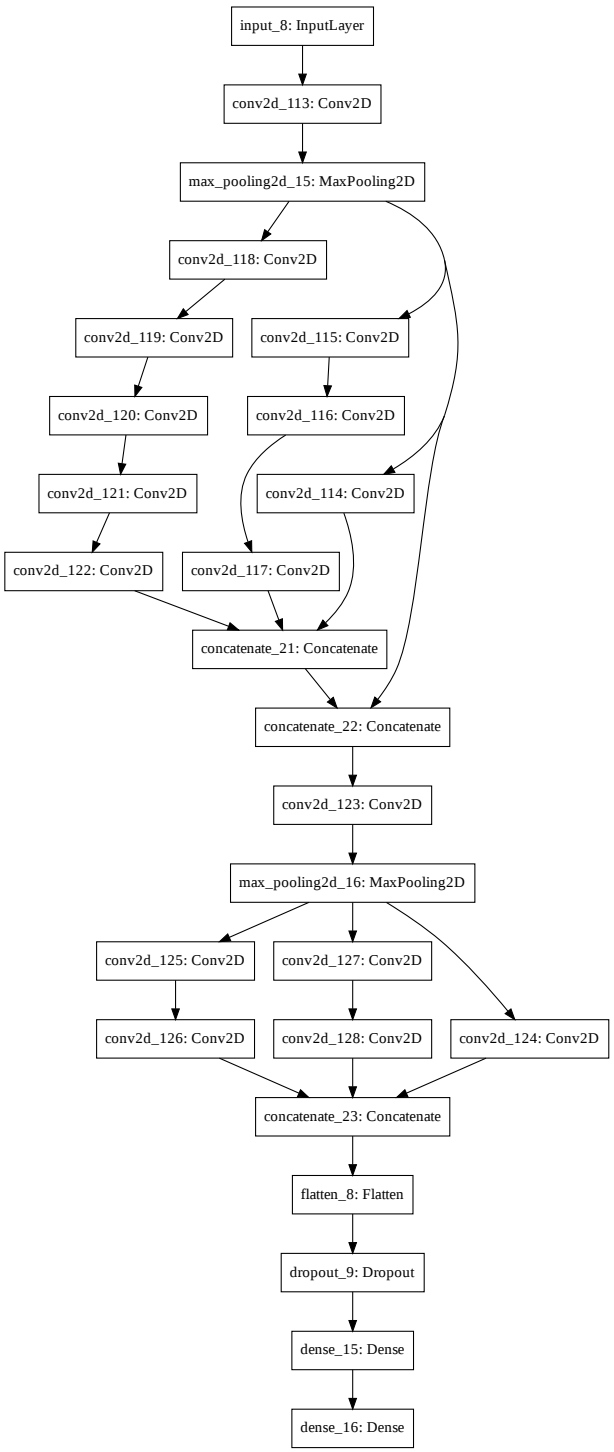
Obr. A.5: Neuronová síť 2.2



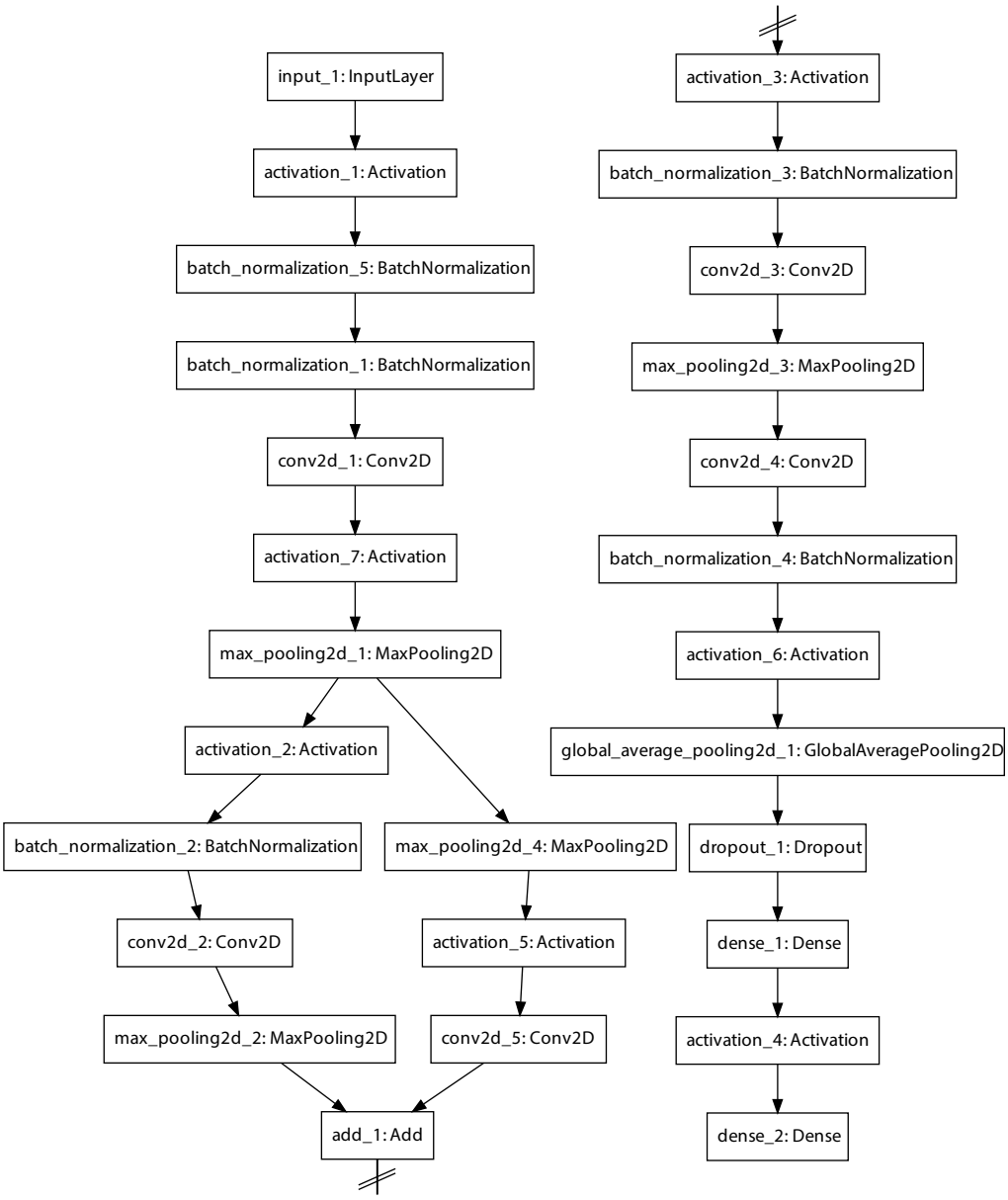
Obr. A.6: Neuronová síť 2.3



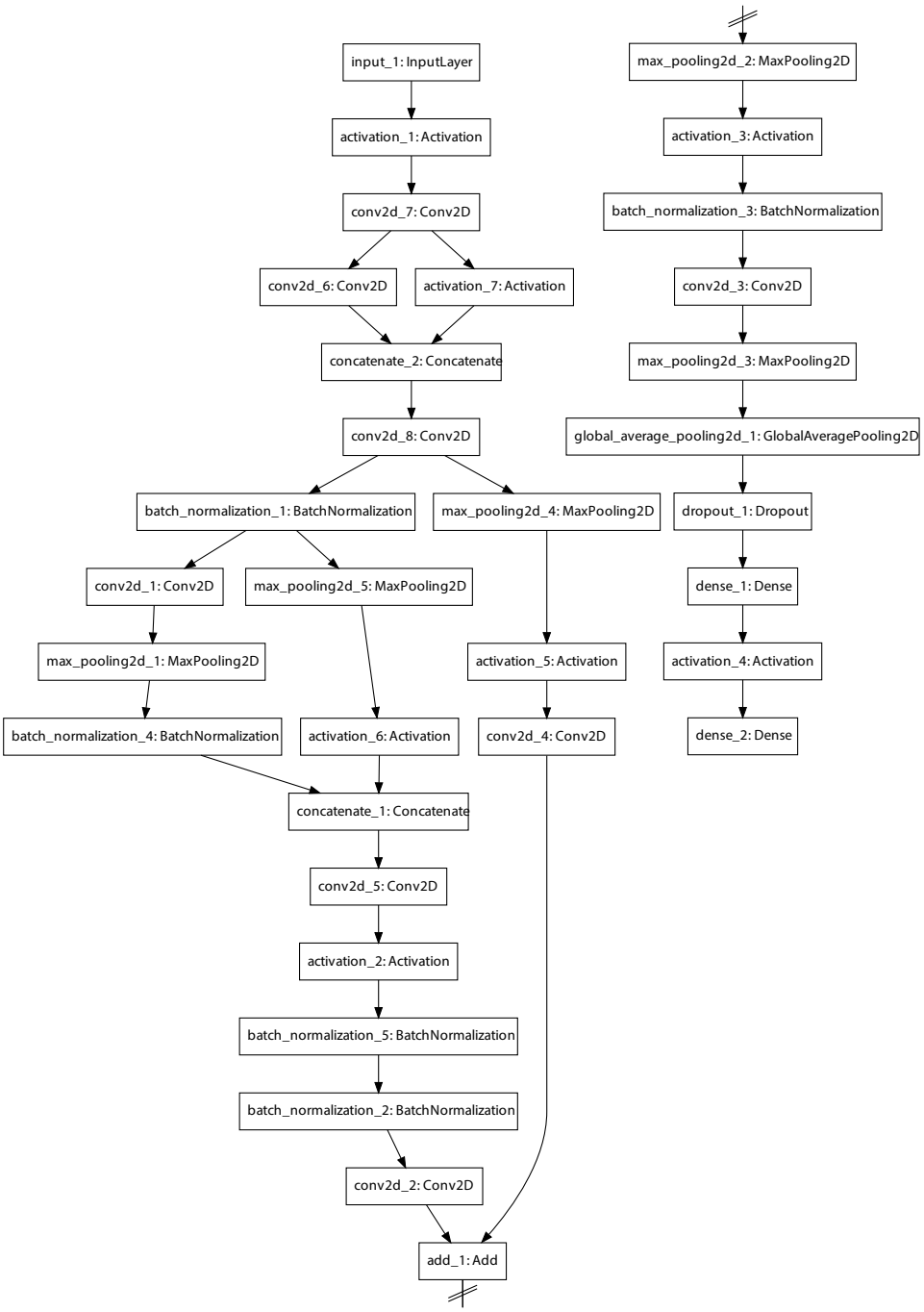
Obr. A.7: Neuronová síť 2.4



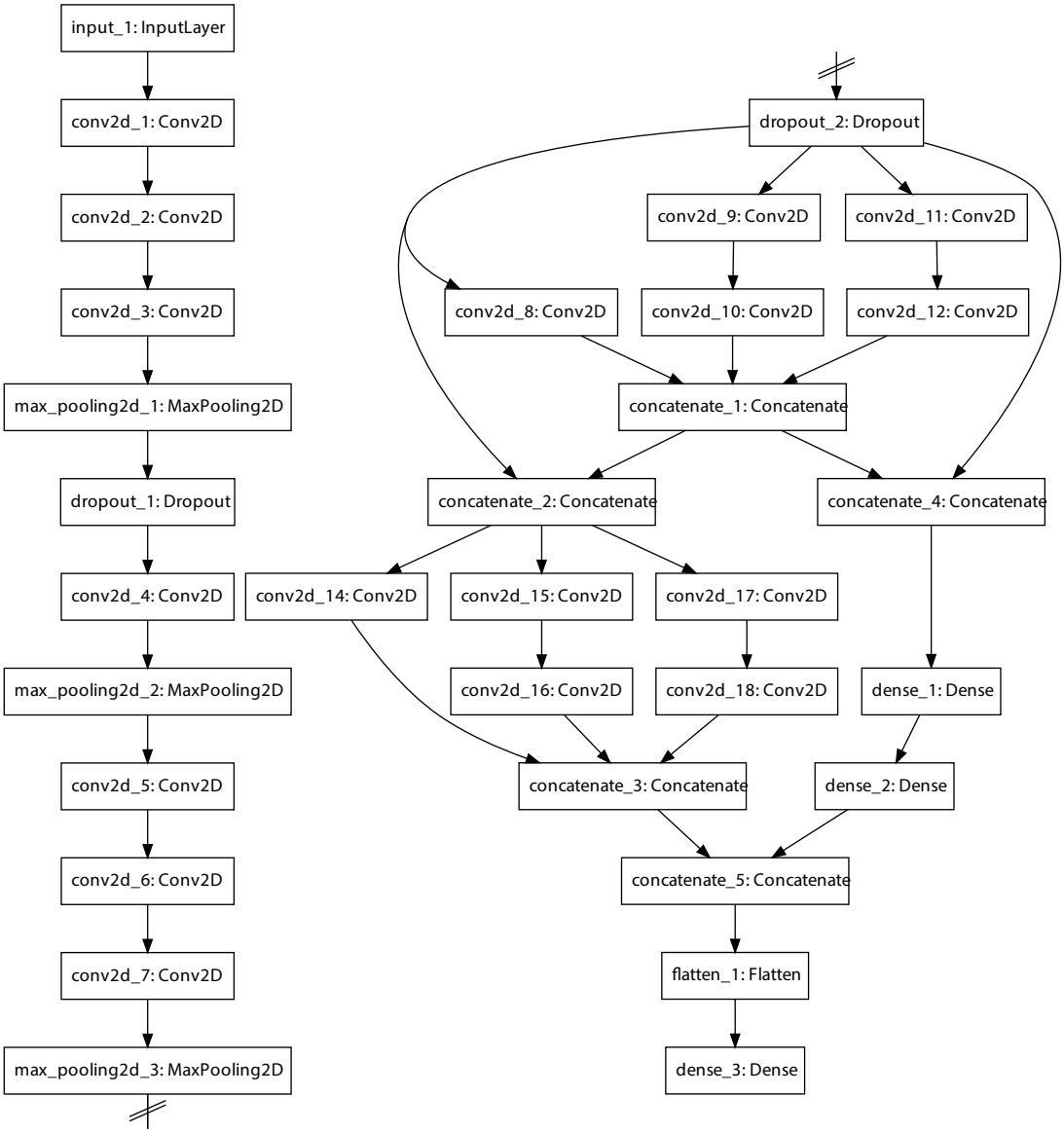
Obr. A.8: Neuronová síť 2.5



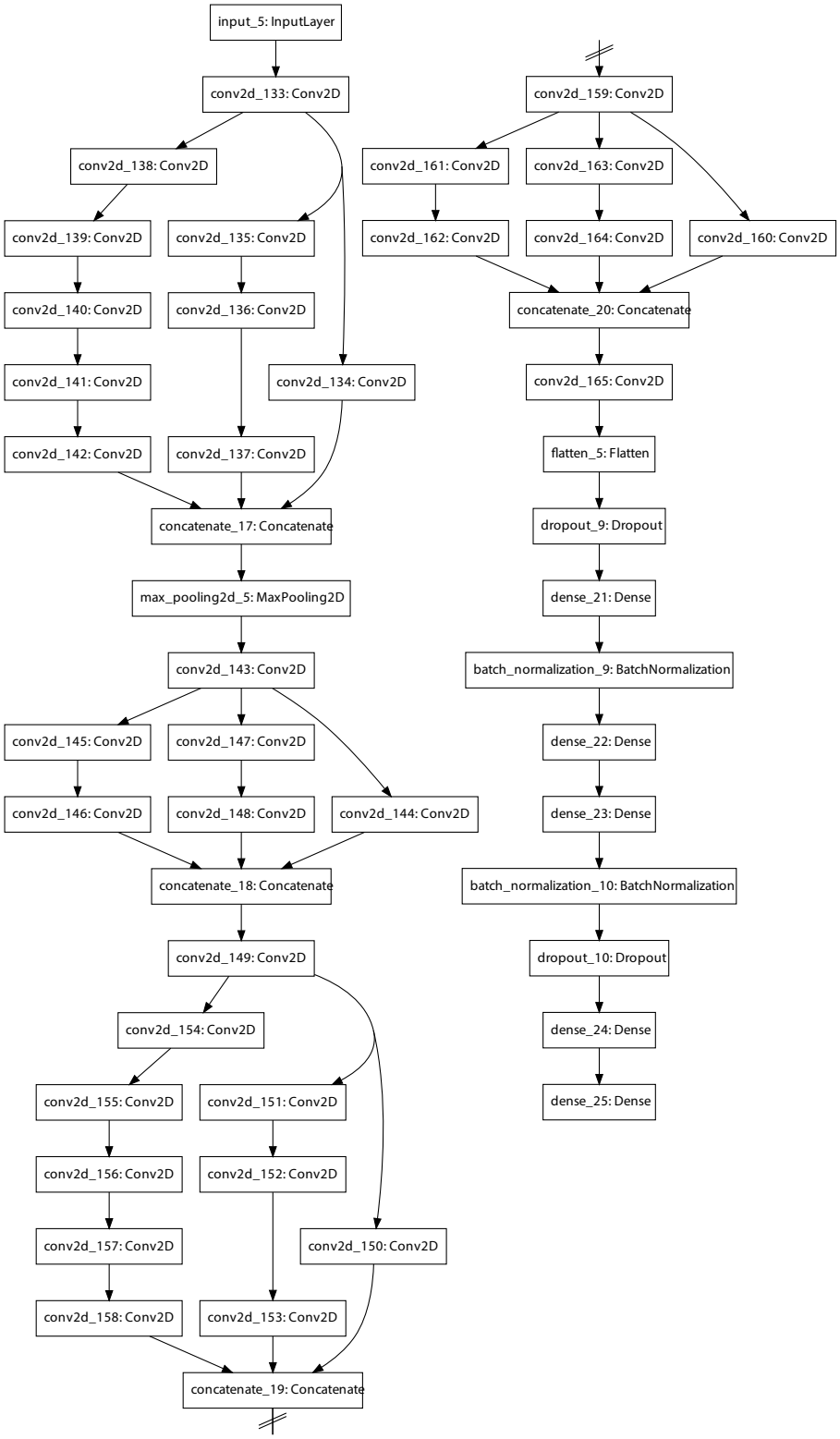
Obr. A.9: Neuronová síť Auto-Keras 1



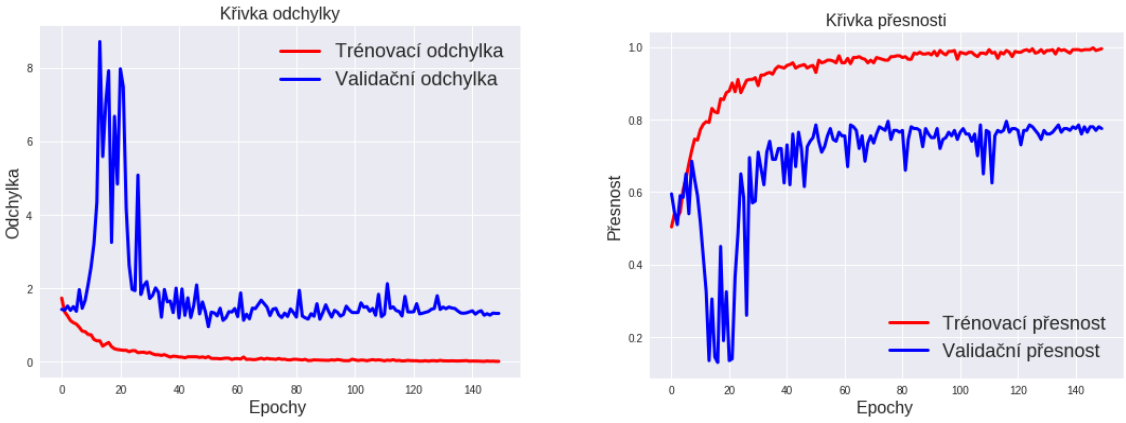
Obr. A.10: Neuronová síť Auto-Keras 2



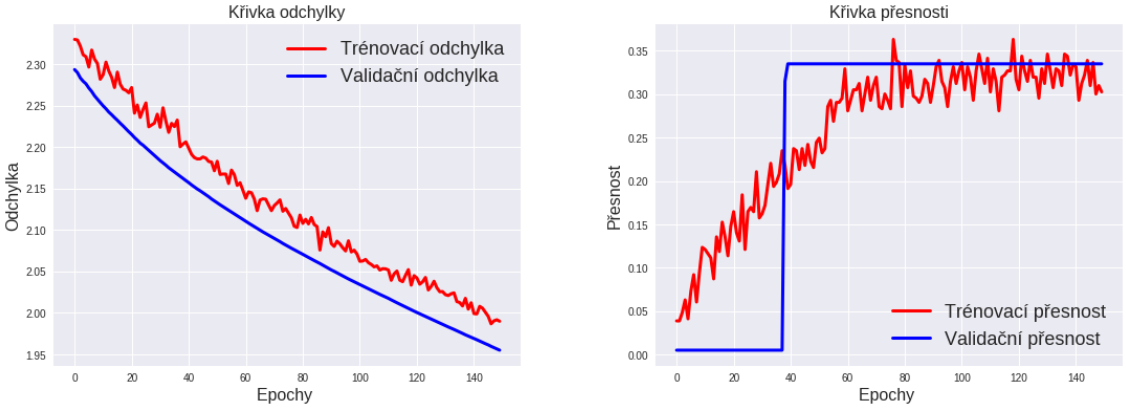
Obr. A.11: Neuronová síť 3.1



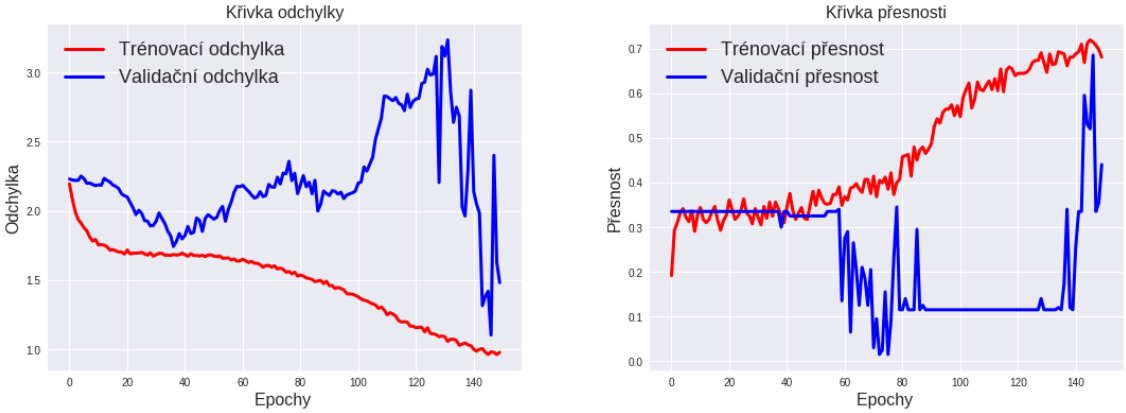
Obr. A.12: Neuronová síť 3.2



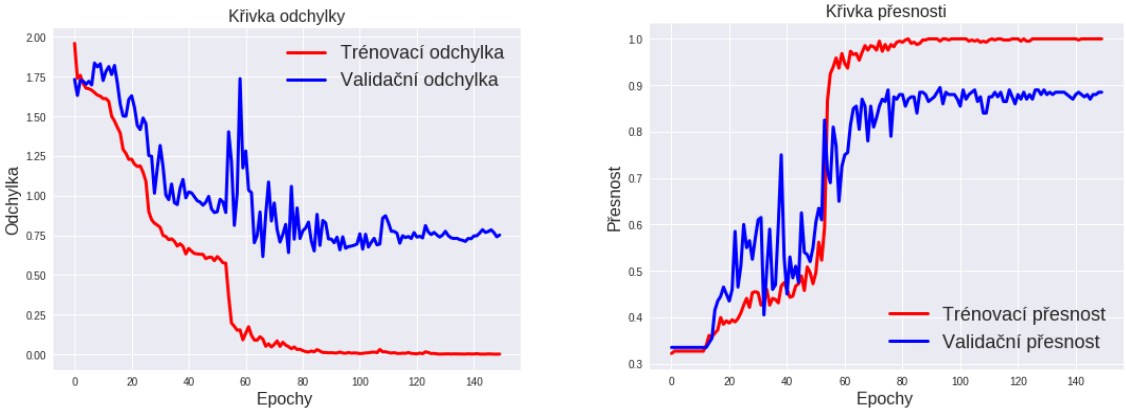
Obr. A.13: Výsledky sítě 1.1 - 10 tříd



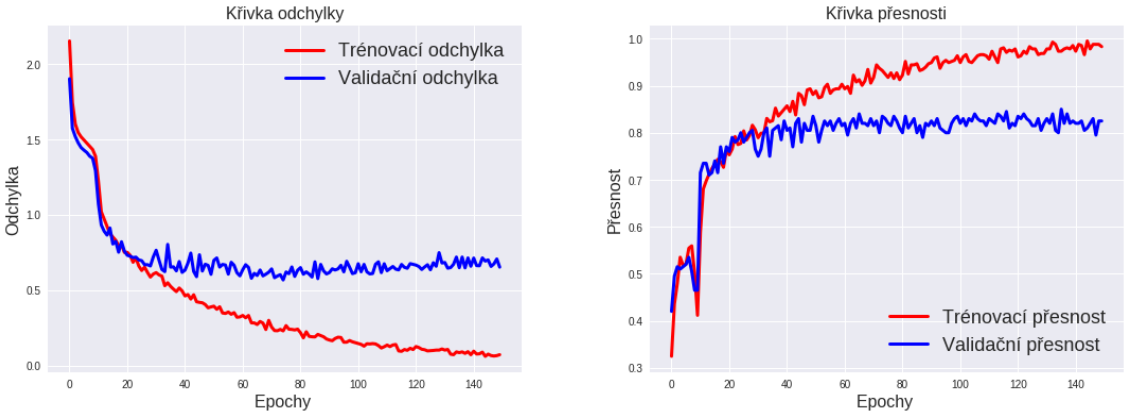
Obr. A.14: Výsledky sítě 1.2 - 10 tříd



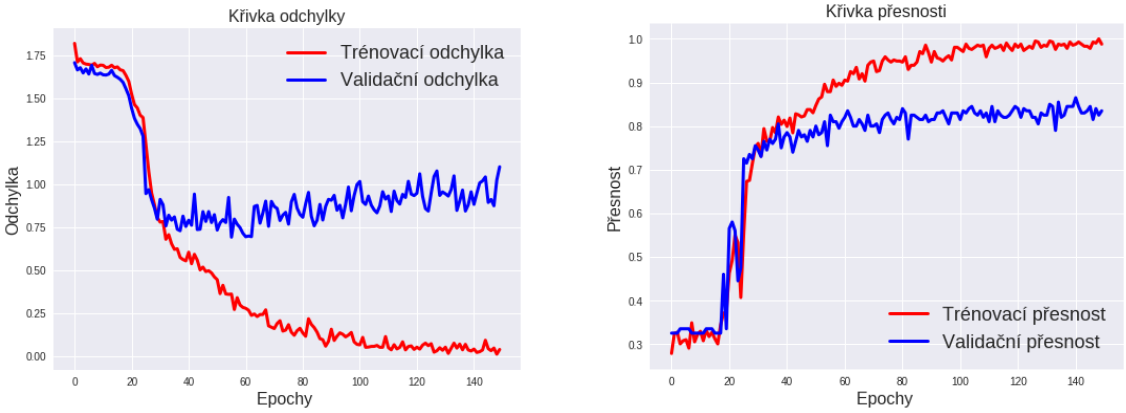
Obr. A.15: Výsledky sítě 1.3 - 10 tříd



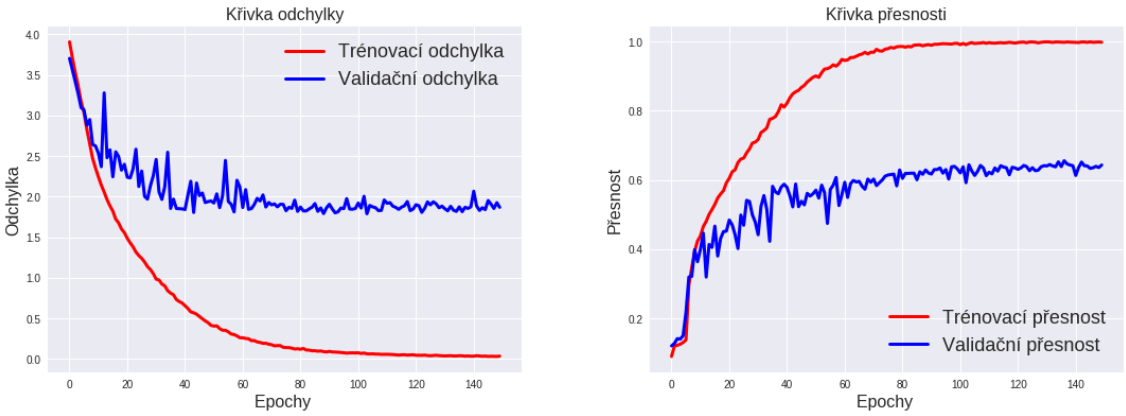
Obr. A.16: Výsledky sítě 2.1 - 10 tříd



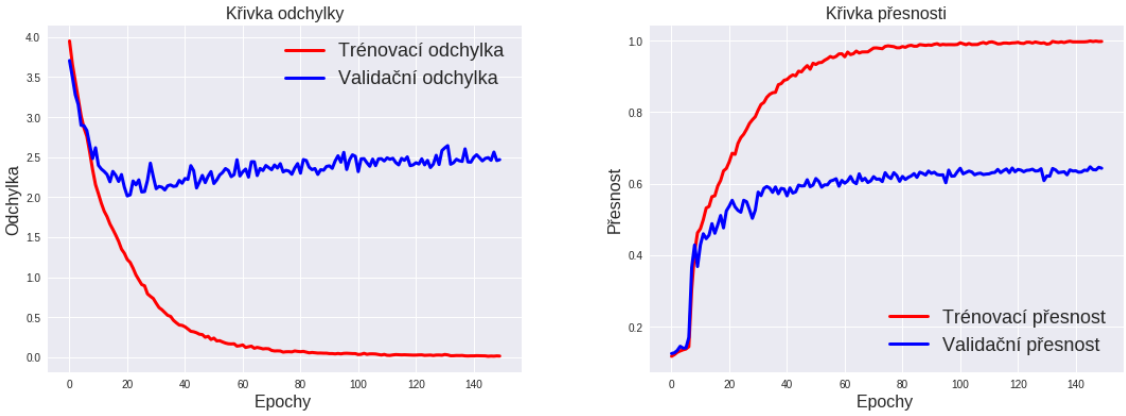
Obr. A.17: Výsledky sítě 2.2 - 10 tříd



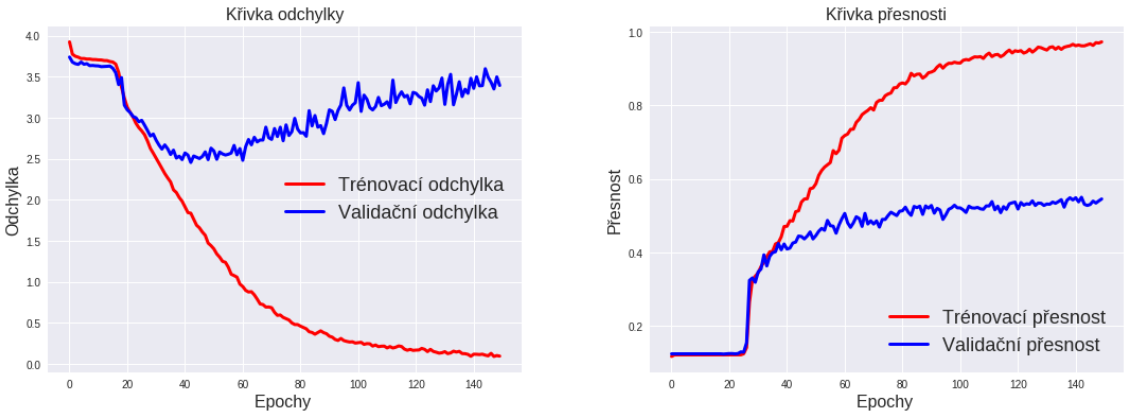
Obr. A.18: Výsledky sítě 2.3 - 10 tříd



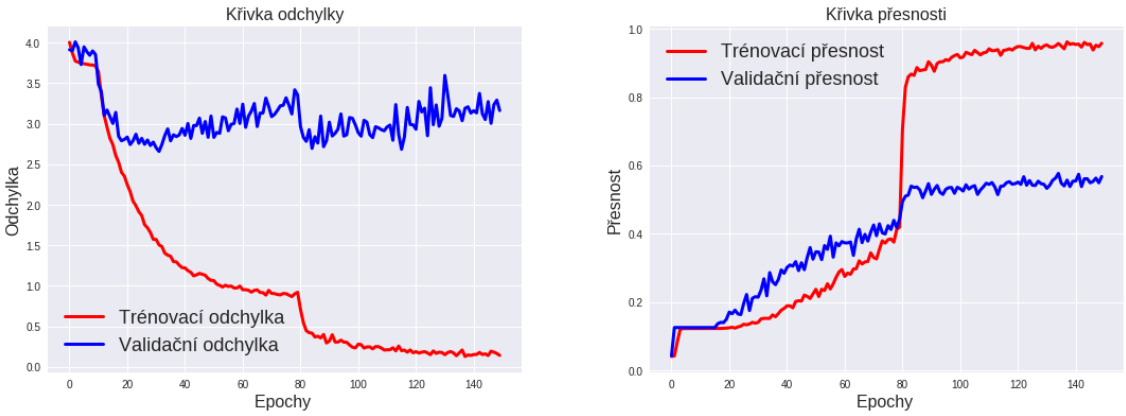
Obr. A.19: Výsledky sítě 2.1 - 100 tříd



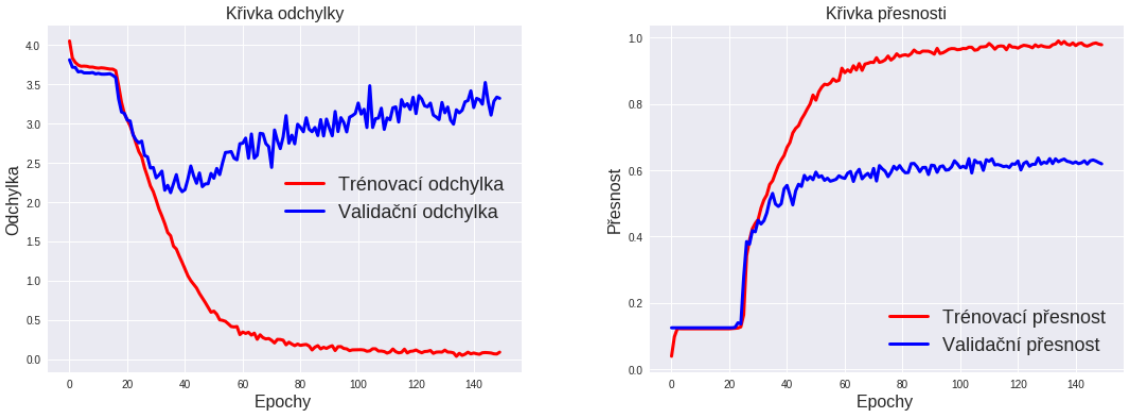
Obr. A.20: Výsledky sítě 2.2 - 100 tříd



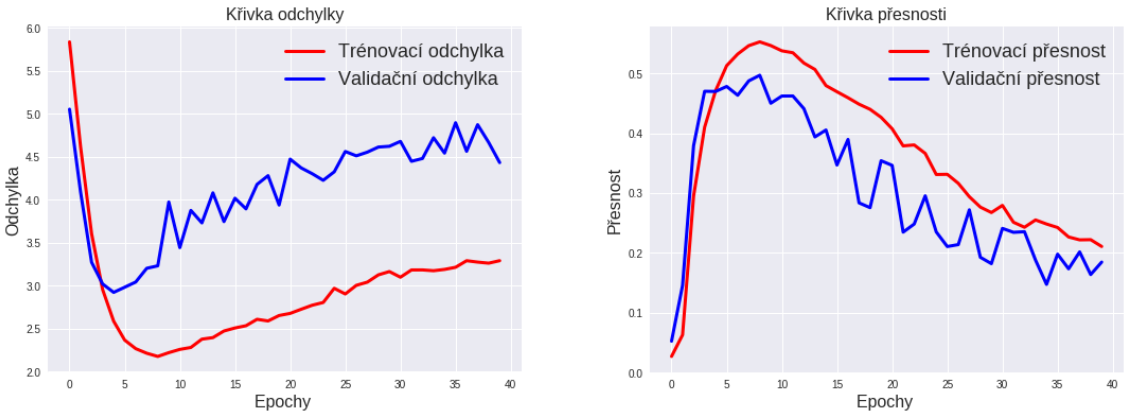
Obr. A.21: Výsledky sítě 2.3 - 100 tříd



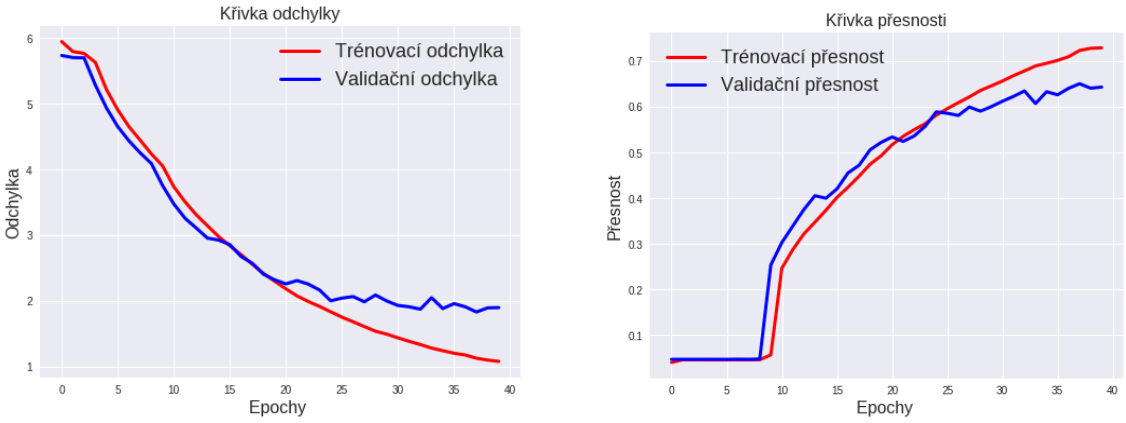
Obr. A.22: Výsledky sítě 2.4 - 100 tříd



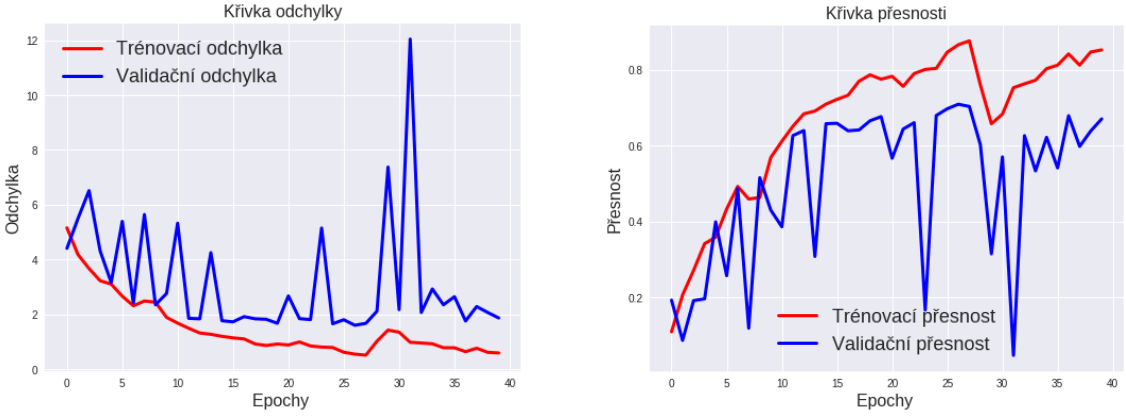
Obr. A.23: Výsledky sítě 2.5 - 100 tříd



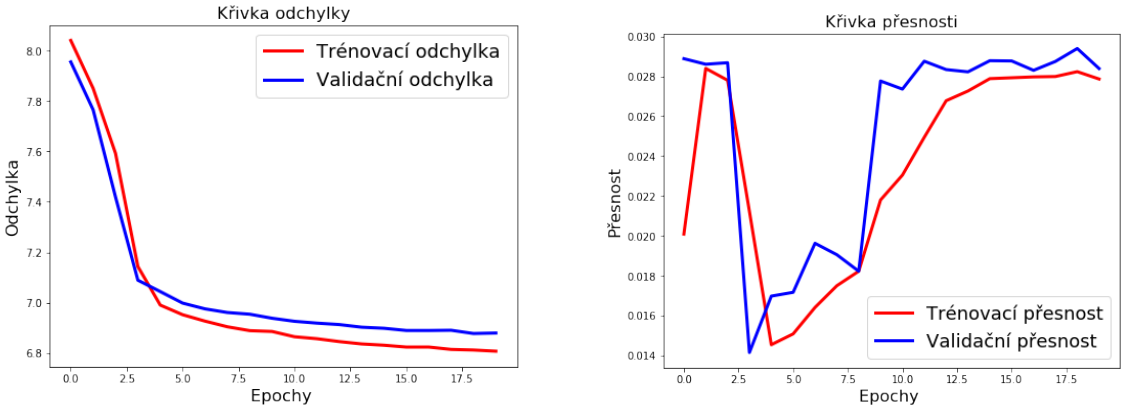
Obr. A.24: Výsledky sítě 2.2 - 1000 tříd



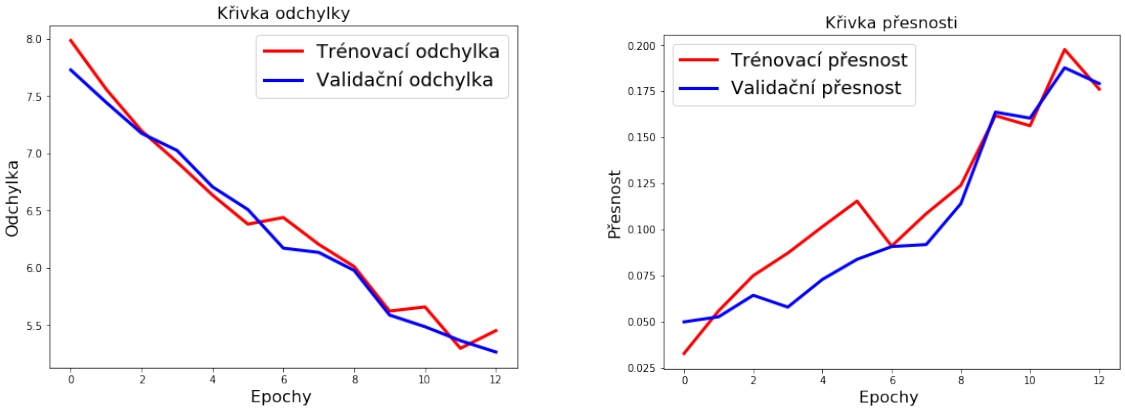
Obr. A.25: Výsledky sítě 2.5 - 1000 tříd



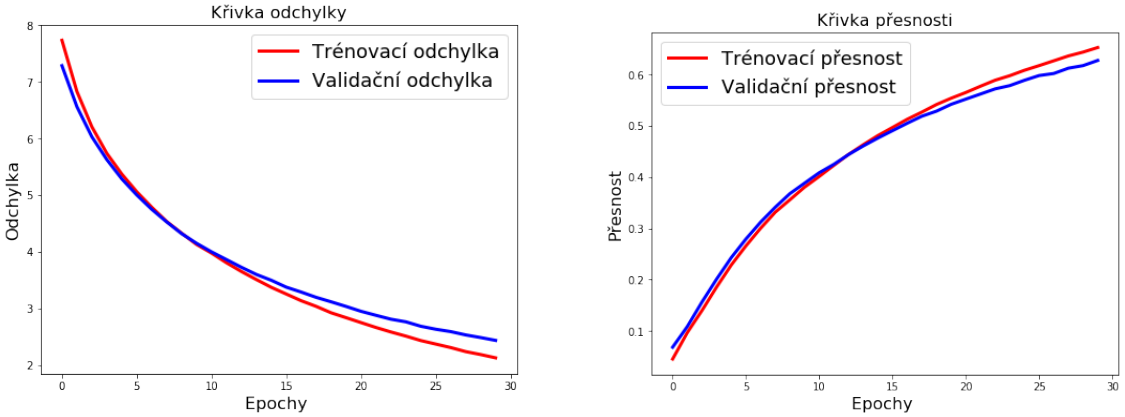
Obr. A.26: Výsledky sítě InceptionV3 dostupné z kerasu - 1000 tříd



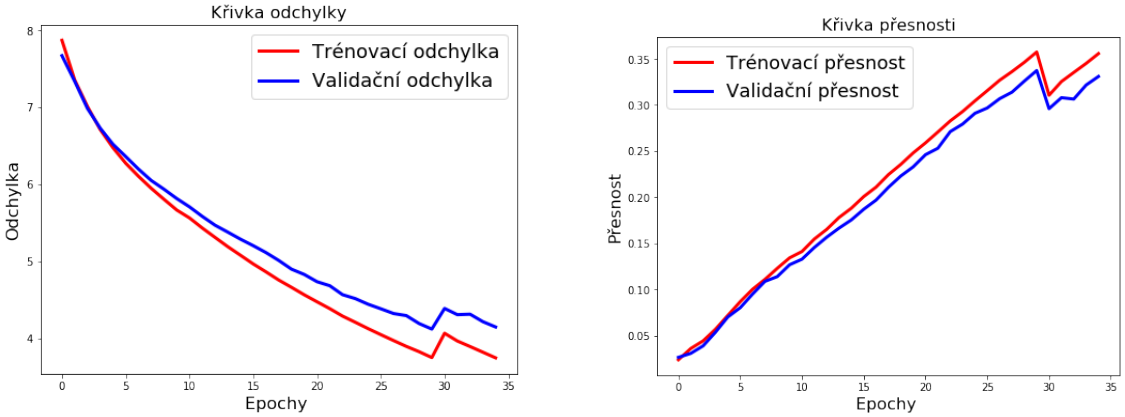
Obr. A.27: Výsledky sítě 3.1 - 3472 tříd



Obr. A.28: Výsledky sítě 3.2 - 3472 tříd



Obr. A.29: Výsledky sítě InceptionV3 dostupné z kerasu - 3472 tříd



Obr. A.30: Výsledky sítě InceptionV3 dostupné z kerasu (nepředučena) - 3472 tříd

B Obsah přiloženého CD

Následující část přílohy popisuje strukturu dat na přiloženém médiu. Software byl vyvinut a testován s využitím JDK 12 a Python 3.6.4

- `/` ..... kořenový adresář přiloženého CD
  - `generator1_java` ..... soubory prvního generátoru
    - `zdrojovy_kod`
      - `DP_2019_hipca.zip`
    - `DP_2019_hipca.jar` ..... spustitelný soubor generátoru
    - `NetworkTesting.py` ..... skript pro výpis parametrů architektury
  - `generator2_python` ..... soubory druhého generátoru
    - `graph.py` ..... pomocný skript
    - `graph_to_network.py` ..... skript pro generování architektur
    - `requirements.txt` ..... list potřebných balíčků
  - `google_colab` ..... soubory využity v Google Colaboratory
    - `utils` ..... skripty pomocných nástrojů
    - `test_generator1.ipynb` ..... skript pro testování sítí generátoru 1
    - `test_generator2.ipynb` ..... skript pro testování sítí generátoru 2
  - `pdf_prace` ..... PDF diplomové práce
    - `xhipca00.pdf`