České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství
Obor: Aplikace informatiky v přírodních vědách

Rozpoznávání typů automobilů pomocí konvolučních neuronových sítí

Car Type Recognition Using Convolutional Neural Networks

DIPLOMOVÁ PRÁCE

Vypracoval: Bc. Kryštof Filip
Vedoucí práce: Mgr. Dana Majerová, Ph.D.
Rok: 2023

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství Akademický rok 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	**Bc. Kryštof Filip**
Studijní program:	**Aplikace informatiky v přírodních vědách**
Specializace:	–
Název práce česky:	**Rozpoznávání typů automobilů pomocí konvolučních neuronových sítí**
Název práce anglicky:	**Car Type Recognition Using Convolutional Neural Networks**
Jazyk práce:	**čeština**

Pokyny pro vypracování:

1. Nastudujte teorii konvolučních neuronových sítí (CNN).
2. Připravte vhodnou množinu dat (fotografie různých typů automobilů).
3. Seznamte se s nástrojem Deep Network Designer ve výpočetním prostředí MATLAB.
4. Popište a porovnejte některé předtrénované modely CNN dostupné v nástroji Deep Network Designer.
5. Využijte transfer learning pro rozpoznávání obrazů z dané datové množiny.
6. Vytvořte vlastní model CNN a aplikujte ho na danou datovou množinu.
7. Popište a zhodnoťte dosažené výsledky.

Doporučená literatura:

[1] BACH, Renee. Deep Learning Onramp. *Self-Paced Online Courses - MATLAB & Simulink* [online]. Natick (Massachusetts, USA): The MathWorks, 2022 [cit. 2022-10-07]. Dostupné z: `https://matlabacademy.mathworks.com/details/deep-learning-onramp/deeplearning`

[2] FERLITSCH, A. *Deep Learning Patterns and Practices.* USA (Shelter Island, NY): Manning, 2021. ISBN 978-1-61729-826-4.

[3] GOODFELLOW, I., COURVILLE, A., and BENGIO, Y. *Deep Learning.* Cambridge: MIT Press, 2016. ISBN 978-0-262-03561-3. Dostupné také z: `http://www.deeplearningbook.org`.

[4] KUBAT, M. *An Introduction to Machine Learning.* 3rd edition. Switzerland: Springer, 2021. ISBN 978-3-030-81934-7.

[5] NIELSEN, M. A. *Neural Networks and Deep Learning* [online]. Determination Press, 2015 [cit. 2022-10-07]. Dostupné z: `http://neuralnetworksanddeeplearning.com/`.

Jméno a pracoviště vedoucího práce:

Mgr. Dana Majerová, Ph.D.
Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, ČVUT v Praze

Jméno a pracoviště konzultanta:

–

Majerová
.....
vedoucí práce

Datum zadání diplomové práce: 14. 10. 2022

Termín odevzdání diplomové práce: 3. 5. 2023

Doba platnosti zadání je dva roky od data zadání.

.....
garant oboru

.....
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská
-3-

.....
děkan

V Praze dne 14. 10. 2022

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne ....

Bc. Kryštof Filip

Poděkování

Chtěl bych poděkovat Mgr. Daně Majerové, Ph.D. za vedení mé diplomové práce a podnětné připomínky při jejím zpracovávání.

Bc. Kryštof Filip

Název práce:
Rozpoznávání typů automobilů pomocí konvolučních neuronových sítí

Autor: Bc. Kryštof Filip

Studijní program: Aplikace přírodních věd
Obor: Aplikace informatiky v přírodních vědách
Druh práce: Diplomová práce

Vedoucí práce: Mgr. Dana Majerová, Ph.D.
Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze

Abstrakt: V posledních letech se konvoluční neuronové sítě stávají stále populárnějšími skrz všemi obory. Předtrénované sítě dosahují slibných výsledků při klasifikaci obrázků. Tato práce porovnává výkon klasifikace různých typů automobilů pěti předtrénovaných (Googlenet, Alexnet, Squeezenet, Resnet-50 a DarkNet19) a několika dalších konvolučních neuronových sítí. Pomocí různých webscrapainogvých technik, zahrnující scraping Google obrázků, Bing obrázků, sítě Pinterest a webových inzertních stránek bylo získáno tisíce snímků pro každou kategorii. Empirické výsledky ukazují, že určité sítě mají v této specifické klasifikační úloze lepší výsledky než jiné. Tato práce poskytne kompletní výsledky a podrobnosti o použité metodologii.
Klíčová slova: Neuronové sítě, konvoluční neuronové sítě, automobil, transfer learning

Title:
Car Type Recognition Using Convolutional Neural Networks

Author: Bc. Kryštof Filip

Abstract: In recent years, convolutional neural networks have become increasingly popular in various fields, and pre-trained models have shown promising results in image classification tasks. This contribution compares the performance of five pre-trained (Googlenet, Alexnet, Squeezenet, Resnet-50 and DarkNet19) and several other convolutional neural networks in distinguishing between different car species. Thousands of images per category were obtained from various sources using different web scraping techniques, including Google images, Bing images, Pinterest, second-hand and advertising sites. Our empirical results show that certain models outperform others in this specific classification task. This work will provide complete results and further details on the methodology.
Key words: Neural networks, convolutional neural networks, car, transfer learning

Obsah

Úvod

Tato práce se zabývá teorií, tréninkem, použitím a srovnáním konvolučních neuronových sítí. Motivací pro tvorbu byla primárně rostoucí popularita a výkonnost neuronových sítí, která vybízí k prozkoumání a porozumnění této oblasti. Cílem práce je vytvořit teoretický aparát a jeho následnou aplikací prozkoumat výkonnost různých technik a metod z oblasti konvolučních neuronových sítí. Praktická část bude navíc obsahovat mnohé testování, které by mělo ilustrovat použití zkoumaných sítí v reálném prostředí. Dále budou konstruovány vlastní konvoluční neuronové sítě, které budou srovnány s mnohými předtrénovanými sítěmi.

První kapitola se zaměřuje na motivaci vzniku umělých nerunových sítí a komentuje rozdíly mezi biologickým a umělých neurone. V druhé části první kapitoly jsou zavedeny 4 druhy umělých neurony v chronologickém pořadí.

Druhá kapitola se zaměřuje na způsob učení v neuronových sítích. Bude zaveden robustní matematický aparát, na který budou navazovat další teoretické kapitoly. První část se věnuje metodě gradientního sestupu a jeho použití v oblasti neuronových sítí, druhá část potom algoritmu zpětného šíření chyby včetně jeho důkazu.

Ve třetí kapitole budou zavedeny a komentovány nejrůznější techniky užívané pro zlepšování výkonu neuronových sítí, a to převážně těch, které budou užity v praktické části práce. Zavedeny budou techniky regularizace, techniky redukující jev nestabilních gradientů a alternativy metody gradientního sestupu.

Čtvrtá kapitola pojednává a zavádí konvoluční neuronové sítě. Kapitola vychází ze zavedené teorie z minulých kapitol a rozšiřuje ji o speciální metody používaných v problematice konvolučních neuronových sítí.

V páté kapitole je popsán postup tvorby datasetu. Komentovány budou podmínky kladené na dataset, metody, jakými budou data získávána a nakonec bude řešena i automatizace eliminace nechtěných dat. V neposlední řadě bude uvedena kompletní struktura datasetu.

V předposlední, šesté, kapitole budou popsány a ilustrovány zkoumané konvoluční neuronové sítě, a to jak předtrénované, tak vlastní. Celkem bude uvedeno 5 předtrénovaných konvolučních neuronových sítí. Vlastní sítě jsou rozděleny na základní a komplexní. Základní budou předmětem optimalizace hyperparametrů, komplexní bude naopak předmětem studování složitějších architektur.

Poslední kapitola shrnuje dosažené výsledky všech zkoumaných sítí. Krom toho budou komentovány zajímavé pozorování vyplývající z dosažených výsledků.

Kapitola 1

Neurony a neuronové sítě

1.1 Motivace

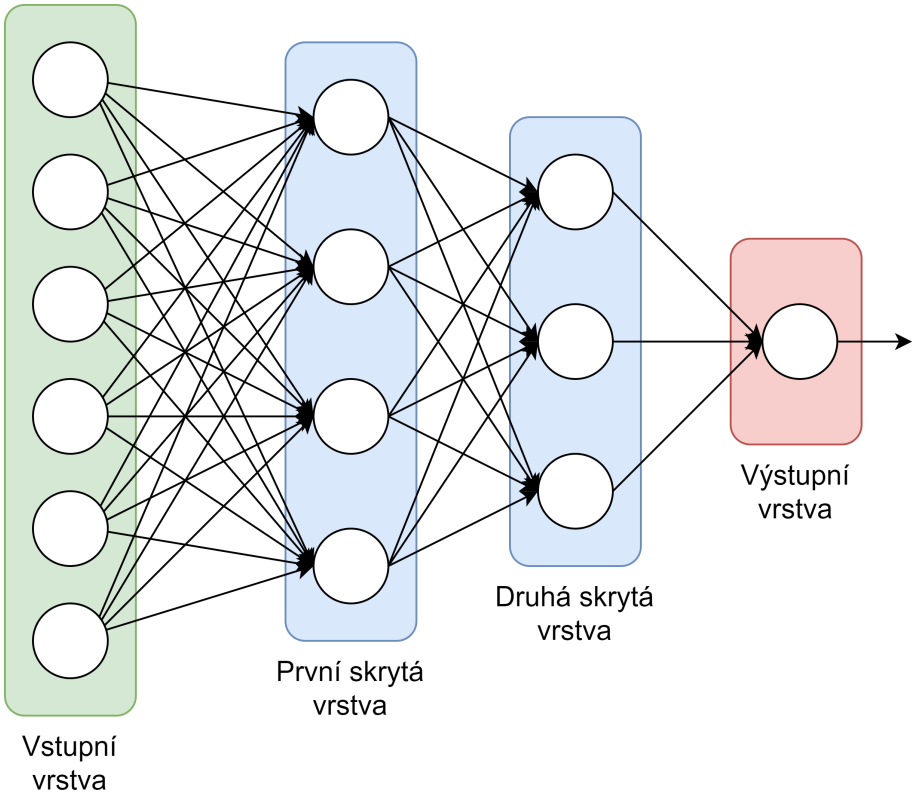
Jak již z názvu vyplývá, počítačové, někdy též umělé neuronové sítě jsou inspirovány funkcí biologických neuronů, ač je mezi umělým a biologickým neuronem několik zásadních rozdílů.

1.1.1 Obecná terminologie

Na síti z obrázku 1.1 lze dobře interpretovat názvosloví, které bude provázet celou tuto práci. Síť neuronů vizualizujeme jako ohodnocený graf, kde uzly značí jednotlivé neurony a hrany spoje mezi nimi. Jak již bylo poznamenáno, v síti existují vrstvy neuronů. Vrstvy jsou shluky neuronů, kde jednotlivé neurony v rámci jedné vrstvy nesdílí žádnou hranu a naopak jsou spojené se všemi neurony z vrstev sousedících – ve smyslu každý s každým. Neurony z jedné vrstvy zakreslujeme na vertikálu, jednotlivé vrstvy pak na horizontálu. První vrstvu neuronů nazveme vrstvou vstupní a neurony v ní vstupními neurony, přičemž počet vstupů může být libovolný. Poslední vrstvu nazveme vrstvou výstupní a analogicky neurony v ní výstupními neurony, i zde platí, že výstupních neuronů může být libovolně mnoho. Vrstvy, které se nacházejí mezi vstupní a výstupní vrstvou, nazveme vrstvami skrytými. Často budu neuronové sítě obecně charakterizovat jako "síť s N skrytými vrstvami" nebo ekvivalentně jako "síť s hloubkou N". Sít hloubky 1 budeme nazývat mělkou, síť s hloubkou ≥ 2 pak hlubokou. Ilustrace přiložena na obrázku 1.1. Sítě, kterými informace prostupují jedním směrem bez bez smyček (tak jak tomu je na obrázku 1.1), se nazývají jednosměrné nebo dopředné. Nebude-li upřesněno, pojmem neuronová síť bude rozuměna jednosměrná/dopředná neuronová síť.

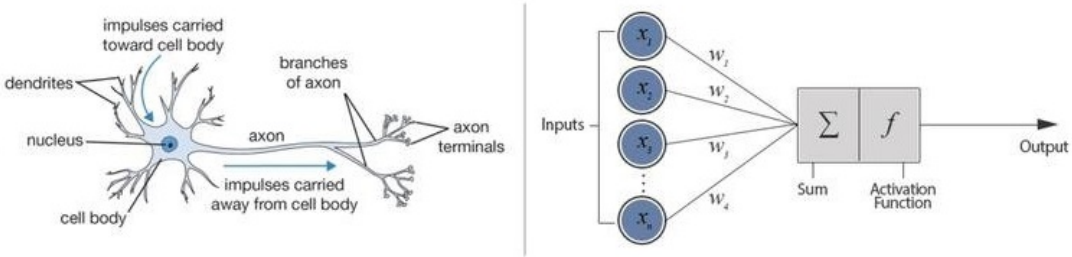
1.1.2 Rozdíl mezi biologickými a umělými neurony

Název neuronová síť společně s odvážnými novinovými titulky by mohla evokovat, že umělý neuron je implementací toho biologického v počítačovém prostředí, a tedy,



Obrázek 1.1: Popis neuronové sítě.

že neuronové sítě jsou umělou formou lidského mozku. Dokonce lze na internetu jednoduše vyhledat společnosti, které tvrdí, že dokáží nakopírovat neurony z vašeho mozku do počítače a zajistit tak nesmrtelné vědomí. Mezi stavbou a funkcí umělého a biologického neuronu je ovšem spousta rozdílů a pojem neuronová síť vychází pouze z inspirace funkce biologického neuronu. V této krátké sekci se pokusím jednoduše popsat nejzásadnější rozdíly mezi dvěma zmiňovanými neurony. K hlubšímu pochopení rozdílnosti slouží i zbytek kapitoly o teorii NN.



Obrázek 1.2: Model biologického a umělého neuronu. [25]

V 50. letech 20. století přišel A. F. Huxley s novými objevy o fungování mozku, tehdy byl poprvé popsán biologický neuron. To se stalo pro mnohé informatiky velkou inspirací, neboť myšlenka lidského neuronu byla poměrně snadno implementovatelná do počítačového světa. V roce 1957 se tak objevila první zmínka umělého neuronu od F. Rosenblatta – sekce 1.2.1. Náčrt a podobnost je možné sledovat na přiloženém

obrázku 1.2. Funkce umělého neuronu je v obecném pojetí stejná jako u neuronu biologického: neuron přijímá signály a v závislosti na jejich síle pošle signál dál, kde signál zpracovávají další neurony. [29][13][14]

Připomeňme, že perceptron má na vstupu binární hodnoty, které neurony zpracovávají a vyhodnocují pomocí nespojité aktivační funkce. Toto omezení bránilo v konstrukci hlubokých sítí, neboť v takovém případě není možné jednoduše zjistit, jak jednotlivé neurony ovlivňují výstup ze sítě, což je stěžejní parametr při trénování sítě (tréninkem nebo učením sítě se rozumí nastavování parametrů, tedy vah a biasů, tak, aby síť správně vyhodnocovala výstupy). Z tohoto důvodu vznikl nový druh umělých neuronů, který má na svých vstupech a výstupech reálná čísla a neurony obsahují spojitou aktivační funkci. Každý neuron je tím pádem diferencovatelná funkce více proměnných, ze které můžeme pro jednotlivé váhy a bias spočítat parciální derivaci s ohledem na výstup neuronu a následně upravit jednotlivé parametry tak, aby neuron nebo celá neuronová síť vracela zamýšlené výstupy (tento pohled je velice zjednodušený a celý problém je zevrubně popsán v kapitole 2.1). Výše popsaný typ umělých neuronů se používá v sítích dodnes a bude od tohoto okamžiku tím, co budu označovat jako umělý neuron. Definice konkrétní aktivační funkce byla záměrně vynechána, neboť bude předmětem dalších kapitol.

Níže jsou v bodech, na základě kritérií, popsané rozdíly mezi neuronem umělým a biologický, přičemž rozdíly jsou výhradně vztaženy k plně propojeným neuronovým sítím.

Funkce

- **Umělý neuron:** Imituje sílu vstupních signálu váhami, každý vstup je vynásoben přiřazenou váhou a vyhodnocen aktivační funkcí (používaných aktivačních funkcí existuje několik, totožná architektura sítě s odlišnými aktivačními funkcemi bude dávat jiné výsledky). Výstup neuronu závisí na použité aktivační funkci, pro některé platí, že za určitých okolností vrátí nulovou hodnotu a imituje se tak stav, kdy biologický neuron signál neodesílá dál, jiné nulovou hodnotu nevrátí pro žádný vstup z reálných čísel, a tak vždy posílají signál určité magnitudy, ač třeba velmi malý, dále.
- **Biologický neuron:** Do neuronu jde vícero nervových spojů, každý z nich posílá signály jiné síly. Překročí-li síla signálů na vstupu určitou hranici, pošle neuron signál dále. Některé signály jsou důležitější než ostatní a mohou vyvolat odeslání výstupového signálu jednodušeji.

Spoje mezi neurony

- **Umělý neuron:** Architektura neuronové sítě je daná a nemění se v průběhu učení. Neurony v jednotlivých vrstvách jsou propojeny "každý s každým", nemusí tedy vznikat nové. Zánik spojů se dá imitovat nastavením váhy dané hrany na 0.

- **Biologický neuron:** Nervové spoje se mohou zesilovat nebo zeslabovat. Mohou vznikat nové spoje a zanikat stávající.

Výstup neuronu

- **Umělý neuron:** Umělé neurony mají na svém výstupu spojité hodnoty, signál tedy vždy posílají dál, jen s jinou magnitudou (případně posílají signál nulový).
- **Biologický neuron:** Neurony mají binární výstupovou hodnotu, signál buď pošlou dále, nebo nikoliv.

Počet neuronů

- **Umělý neuron:** Většina neuronových sítí má v závislosti na jejich komplexitě od stovek po statisíce neuronů. Mohutné umělé neuronové sítě nepřináší kýžené zlepšení výkonu, a to kvůli způsobu, jakým jsou trénovány (více o tomto tématu v kapitole 3.11), obecně tak neplatí ekvivalence mezi velikostí sítě a její přesností. Srovnání v počtu neuronů s lidským mozkem je však neúplné. Umělé neurony totiž mezi sebou nejsou propojeny v rámci jedné skryté vrstvy stejně a v rámci vrstev nesousedících (neplatí u rekurentních nebo LSTM sítích). Mimoto jsou umělé neurony různými techniky schopné extrahovat sofistikovanější vlastnosti ze vstupních dat.
- **Biologický neuron:** Lidský mozek obsahuje asi 86 miliard neuronů a mezi 100 až 1000 biliony synapsí (spojů mezi neurony). [12]

Průstup signálu sítí

- **Umělý neuron:** Všechny umělé sítě prochází a počítají výstupy neuronů v síti postupně od vstupních neuronů, přes skryté vrstvy, až po neurony výstupní.
- **Biologický neuron:** Biologické neuronové sítě mohou vysílat signály prostupující sítí asynchronně a paralelně.

Rychlost

- **Umělý neuron:** Rychlost, s jakou signál putuje sítí a s jakou dokáže neuron signál odeslat, závisí na hardwaru, který neuronová síť využívá, a tedy tyto veličiny nenesou žádnou informaci.
- **Biologický neuron:** Některé neurony dokáží vyhodnotit vstupy a odeslat signál až 200-krát na vteřinu a signály putují v závislosti na jejich síle mezi 0,61 po 119 metrů za vteřinu. Frekvence a rychlost signálu je součást informace, kterou signál přenáší. [24] [18]

Energetická náročnost

- **Umělý neuron:** Neuronové sítě se často trénují na výkonných grafických kartách, které samy o sobě dokáží brát i více jak 300 Wattů (NVIDIA RTX A6000) a obvykle pracují při teplotách vyšších desítek stupňů.
- **Biologický neuron:** Lidský mozek spotřebuje zhruba 20 % celkové tělesné energie, i přesto se jedná o pouhých 20 Wattů při tělesné teplotě pod 37 °C. [33]

Učení

- **Umělý neuron:** Umělé sítě zdokonalují své predikce pouze v rámci jejich tréninku, při následném užití neuronových sítí ke generování predikcí už k žádnému zlepšování výkonu nedochází.
- **Biologický neuron:** Mozek se učí při každém opakování, není rozdíl mezi tréninkem (pokud nějaký absolvujeme) a použitím. Jinými slovy, náš mozek se učí stále za pochodu.

Biologická a umělá neuronová síť se tedy liší v mnoha zásadních vlastnostech týkajících se stavby, funkce nebo učení. Umělé neuronové sítě dokáží ve vybraných problémech a precizním tréninku lidský mozek překonat, na druhou stranu zatím zcela jistě nedokáží obsáhnout šířku lidského vědění, dorovnat se úrovni generalizace, kterou lidský mozek disponuje, tvořivosti nebo kreativitě. Můžeme je mezi sebou srovnávat a argumentovat o jejich výhodách, nevýhodách a schopnostech, není ale vhodné a pravdivé je vnímat jako ekvivalentní.

1.2 Typy umělých neuronů

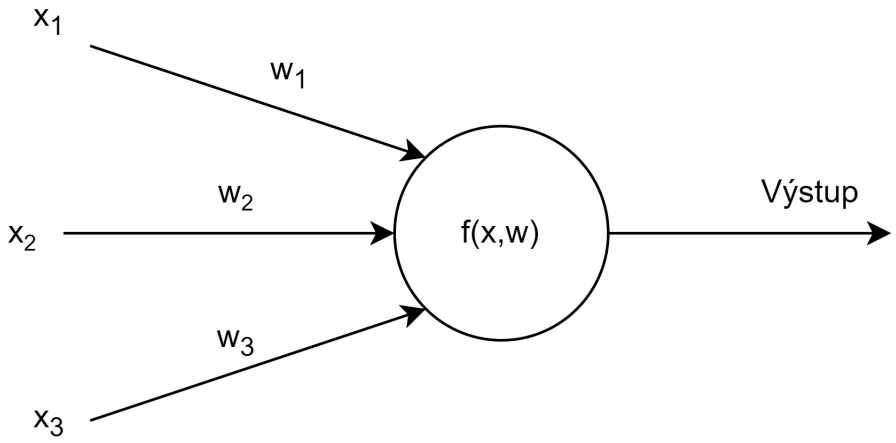
Tato sekce byla zpracována na základě knih *Neural Networks and Deep Learning*, NIELSEN M.A. [26] a *Deep Learning*, GOODFELLOW I. et al. [10].

Cílem této kapitoly je popsat některé typy umělých neuronů, aby bylo následně možné zavést neuronovou síť (dále také NN – z anglického *neural network*). jsou zaměřeny na zavedení neuronových sítí. Okomentována bude motivace pro jejich vznik, fungování, učení a specializované tvary neuronových sítí. Dále budou uvedena a vysvětlena omezení neuronových sítí a formy řešení těchto omezení.

Teorie neuronových sítí se rozvíjí od 50. let minulého století, její rozsáhlost tedy daleko překračuje možnosti této práce. Proto bude definován jen určitý průřez celým odvětvím neuronových sítí, který byl specificky vybrán tak, aby obsáhl formy neuronových sítí využívaných v praktické části práce.

1.2.1 Perceptron

S myšlenkou zkonstruování umělého neuronu se začalo již v 50. letech 20. století, jeho vývoj a popis pak přišel o dekádu později v knize *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms* [29]. Její autor – Frank Rosenblatt – v knize zkonstruoval nejjednodušší formu umělého neuronu – perceptron.



Obrázek 1.3: Perceptron.

Perceptron je funkce $f(w_j)$ jedné či libovolně mnoha binárních proměnných x_j, z nichž každá je na svém vstupu vynásobena váhou w_j. Váhy na vstupech perceptronu udávají jakousi význačnost daného vstupu na výstup. V případě perceptronu má funkce, jež se nazývá aktivační, tvar součtu násobků proměnných s váhami. Výstup aktivační funkce je též binární, zjednodušeně řečeno výstup udává, zda byl perceptron aktivován a posílá signál dál, či nikoliv.

Přesněji, perceptron je typ umělého neuronu z obrázku 1.3, kde

$$f(\vec{x},\vec{w}) = \begin{cases} 0 & \text{pokud} \quad \sum_j w_j x_j \leq T \\ 1 & \text{pokud} \quad \sum_j w_j x_j > T \end{cases} \tag{1.1}$$

přičemž $x_j \in \{0,1\}$, $w_j \in \mathbb{R}$, $T \in \mathbb{R}$ a T je takzvaný práh citlivosti (anglicky *threshold value*).

Pro přímočařejší práci s tímto vztahem budu však nadále používat ekvivalentní vztah (1.2).

$$f(\vec{x},\vec{w}) = \begin{cases} 0 & \text{pokud} \quad \vec{w}\cdot\vec{x}+b \leq 0 \\ 1 & \text{pokud} \quad \vec{w}\cdot\vec{x}+b > 0 \end{cases} \tag{1.2}$$

kde b nazýváme bias a platí $b = -T$ a vztahem $\vec{w}\cdot\vec{x}$ se rozumí skalární součin dvojice vektorů, tedy platí $\vec{w}\cdot\vec{x} = \sum_j w_j x_j$.

Perceptron je tedy jakási rozhodovací struktura založená na váženém vstupu. Příkladem může být rozhodovací proces o koupi nového automobilu, uvažujme například:

- Cena automobilu je důležitým faktorem, tedy výrok, zda je automobil levný, bude mít váhu 3.

- Provozní náklady jsou dalším ze zásadních rozhodovacích faktorů, a tak fakt, že je automobil ekonomický, dostane váhu 2.
- Naopak výkon automobilu je spíše jen výhoda, nikoliv zásadní prerekvizita – přidělíme váhu 0,5.
- Barva automobilu hraje jen mizivou roli v našem rozhodování, a tedy přidělíme váhu 0,2.
- Zároveň si opravdu nepřejeme, aby byl automobil poruchový, bude-li poruchový, za žádnou cenu si ho nechceme pořídit, přidělíme tedy tomuto výroku zápornou váhu -5.
- Zbývá určit práh citlivosti, řekněme, že si automobil pořídíme, pouze pokud bude levný, s nízkou spotřebou a nějakou přidanou hodnotou k tomu. Práh citlivosti bude tedy 5,1.

V tabulce 1.1 jsou zaneseny vybrané hodnoty rozhodovacích parametrů a výstup perceptronu.

Nízká cena	Nízké provozní náklady	Vysoký výkon	Líbivá barva	Poruchovost	Vážený vstup	Koupě
1	0	1	1	0	3,7	0
1	1	1	1	1	0,7	0
1	1	1	0	0	5,5	1
1	1	0	1	0	5,2	1

Tabulka 1.1: Vybrané hodnoty rozhodovacích parametrů a příslušný výstup perceptronu.

Je zřejmé, že změnou jednotlivých vah a prahu citlivosti měníme rozhodovací proces perceptronu. Tímto způsobem lze sítí perceptronů modelovat jakýkoliv rozhodovací proces nebo lépe – libovolnou logickou funkci. S odkazem na výše uvedený primitivní příklad s jedním perceptronem je zajímavé zauvažovat, co dokáže rozhodnout síť perceptronů na obrázku 1.1, kde perceptrony v druhé vrstvě se rozhodují na základě výstupu perceptronů z první vrstvy, zjednodušeně tedy rozhodují na vyšší, abstraktnější a komplexnější úrovni. I taková síť je ovšem co do velikosti neporovnatelná s reálně používanými neuronovými sítěmi. Na obrázku 1.1 máme celkem 8 neuronů ve třech vrstvách, perceptrony ze sousedních vrstev jsou spojeny každý s každým váhou ohodnocenými hranami a každý perceptron obsahuje práh citlivosti. Váhy a prahy citlivosti jsou parametry, jejichž změnami měníme chování celé sítě. Vhodně nastavenými parametry sítě (opomeňme zatím způsob, jak parametry správně nastavit) bude síť na základě vstupů správně určovat výstup. Počet parametrů v síti je tedy hlavním měřítkem její komplexity. Více parametrů znamená, že je síť schopná řešit komplexnější problémy. Na obrázku 1.1 máme celkem 39 vah a 8 prahů citlivosti, celkem tedy 47 parametrů. Pro srovnání, vyspělé neuronové sítě obsahují miliony, někdy i miliardy parametrů (neuronová síť ChatGPT společnosti OpenAI obsahuje 175 miliard parametrů [7]).

1.2.2 Sigmoid neuron

Již v předchozích kapitolách byl komentován problém s perceptrony a nastíněn modifikovaný druh neuronu, který pracuje s reálnými čísly a spojitou aktivační funkcí. Právě tato podkapitola uceleně zavede typ umělého neuronu, který jako první umožnil konstrukci hlubokých, výkonných a obecně použitelných neuronových sítí [35].

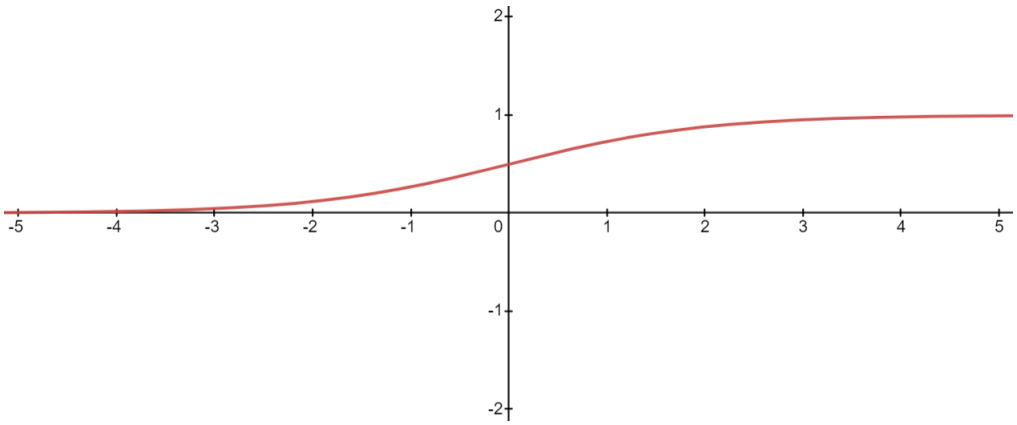
Abychom mohli neuronovou síť trénovat – nastavovat její parametry tak, aby správně vyhodnocovala vstupy – je zapotřebí zjistit vliv každého z parametrů na výstup celé sítě. Připomeňme, že neuronová síť je sestavena z mnoha složených funkcí. Vliv jejich parametrů na výstup sítě tak lze obstarat užitím parciálních derivací. V tomto ohledu však není perceptron ideální volbou, neboť obsahuje skokovou aktivační funkci, která není spojitě diferencovatelná.

Mějme neuron z obrázku 1.3 s aktivační funkcí:

$$f(\vec{x}, \vec{w}) = \sigma(\vec{w} \cdot \vec{x} + b)\,, \quad \text{kde} \tag{1.3}$$

$$\sigma(z) = \frac{1}{1 + \mathrm{e}^{-z}} \tag{1.4}$$

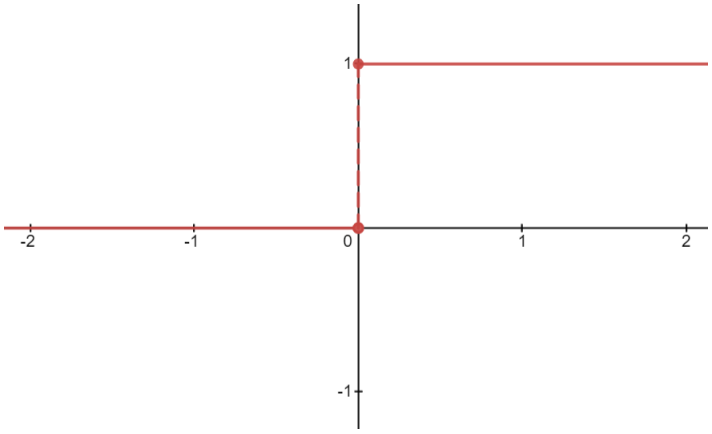
kde $\vec{x} \in [0,1]^k$, $\vec{w} \in \mathbb{R}^k$, $z \in \mathbb{R}$. Výše zmíněná funkce se nazývá sigmoida, proto bude tento typ neuronu dále označován jako sigmoid neuron. Graf sigmoidy je uveden na obrázku 1.4, graf skokové funkce na obrázku 1.5. Povšimněme si podobností se



Obrázek 1.4: Graf sigmoidy.

skokovou funkcí používanou v perceptronu, je-li $z = \vec{w} \cdot \vec{x} + b \gg 0$, potom $\mathrm{e}^{-z} \approx 0$ a tedy $\sigma(z) \approx 1$. Naopak je-li $z \ll 0$, potom $\mathrm{e}^{-z} \to \infty$ a tedy $\sigma(z) \approx 0$. Na rozdíl od skokové funkce je ale sigmoida spojitě diferencovatelná, a tak lze spočítat vliv změny parametrů sigmoid neuronu na jeho výstup. Předpokládejme tedy sigmoid neuron s n vstupy. Parametry takového neuronu bude tvořit množina n vah $\{w_1, w_2, \ldots, w_n\}$ a bias b. provedeme-li změnu i-té váhy tak, že $w_i^* = w_i + \Delta w_i$, a biasu tak, že $b^* = b + \Delta b$, lze aproximovat změnu na výstupu neuronu $\Delta\Theta$ jako

$$\Delta\Theta \approx \frac{\partial\,\Theta}{\partial\,w_i}\,\Delta w_i + \frac{\partial\,\Theta}{\partial\,b}\,\Delta b. \tag{1.5}$$



Obrázek 1.5: Graf skokové funkce.

$\Delta\Theta$ je tedy lineární funkce, a to i v případě, kdy budeme měnit všechny váhy na vstupu. V takovém případě máme množinu změn vah $\{\Delta w_1, \Delta w_2, \ldots, \Delta w_n\}$ a jednoduchou úpravou výrazu (1.5) dostáváme

$$\Delta\Theta \approx \sum_{j=1}^{n} \frac{\partial\,\Theta}{\partial\,w_j}\,\Delta w_j + \frac{\partial\,\Theta}{\partial\,b}\,\Delta b. \tag{1.6}$$

Kde $\Delta w_j \in \{\Delta w_1, \Delta w_2, \ldots, \Delta w_n\}$ a $w_j \in \{w_1, w_2, \ldots, w_n\}$.

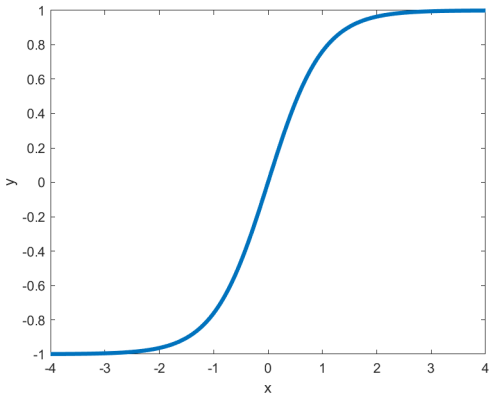
1.2.3 Tanh neuron

Doposud byl představen jen jeden typ neuronu, který bude využíván v praktické části práce, a to neuron se sigmoidní aktivační funkcí. Pochopitelně se nejedná o jediný typ široce využívaného neuronu, proto budou v následujících podkapitolách představeny dvě alternativy, které budou předmětem testování v praktické části – tanh (čteno *tanč*) neuron, neboli neuron s aktivační funkcí hyperbolického tangens a ReLU neuron.

Hyperbolický tangens a sigmoida jsou si velice podobné, přesněji pro ně platí následující závislost

$$\sigma(z) = \frac{1 + \tanh(\frac{z}{2})}{2}\,. \tag{1.7}$$

Jejich podobnost lze spatřit i z grafu hyperbolického tangens na obrázku 1.6. Jediným zásadním rozdílem je jejich obor hodnot. Zatímco pro sigmoidu platí $H_\sigma \in (0, 1)$, tak hyperbolický tangens nabízí širší pole působnosti $H_{\tanh} \in (-1, 1)$. Zřejmě tedy tanh neuron oproti sigmoid neuronu dovoluje v síti záporné aktivace. Použití záporných aktivací má v neuronových sítích jednu důležitou konsekvenci. Předpokládejme užití sigmoid neuronu, potom nám algoritmus zpětného šíření chyby říká, že gradient asociovaný s váhou w_{jk}^{l+1} má tvar $a_k^l \delta_j^{l+1}$. Protože jsou všechny aktivace $a_k^l > 0$ bude mít gradient stejné znaménko jako chyba δ_j^{l+1}. Toto pozorování vyústí v fakt, že pokud je chyba $\delta_j^{l+1} > 0$, potom vlivem gradientního sestupu budou všechny váhy v tomto kroku klesat, naopak pokud $\delta_j^{l+1} < 0$, budou všechny váhy růst. Tedy



Obrázek 1.6: Graf hyperbolického tangens.

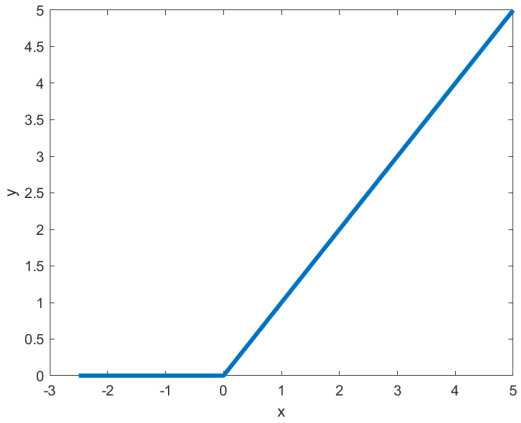
všechny váhy spojené s inkriminovaným neuronem budou buď dohromady klesat, nebo růst. Tento jev nemusí být ideálním, neboť některé váhy mohou profitovat ze snižování, zatímco jiné z růstu. Aby bylo takové chování umožněno, je potřeba, aby člen a_k^l vracel i záporné hodnoty, což právě hyperbolický tangens splňuje. Navíc platí $\tanh(-z) = -\tanh(z)$, tedy hyperbolický tangens je symetrický okolo 0, takže lze předpokládat, že přibližně polovina vah ve výše popsaném problému bude klesat, zatímco přibližně druhá polovina vah růst. [8]

1.2.4 ReLU neuron

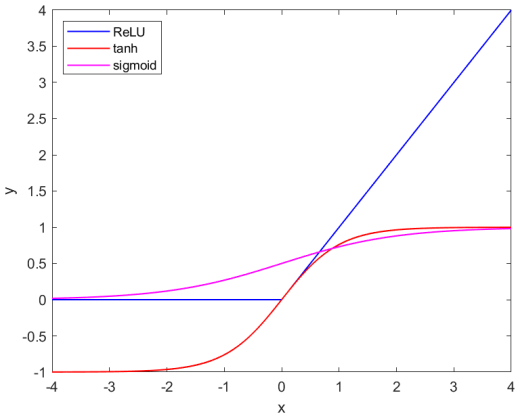
Posledním typem neuronu, který bude použit v praktické části, je ReLU neuron (zkratka z anglického *rectified linear unit*). Aktivační funkce ReLU neuronu má následující tvar

$$\mathrm{ReLU}(z) = \max(0, z) = \max(0, \vec{w} \cdot \vec{x} + b)\,. \tag{1.8}$$

Graf funkce ReLU je přiložen na obrázku 1.7, srovnání všech tří zmíněných aktivačních funkcí potom na obrázku 1.8. Již na první pohled je zřejmé, že ReLU funkce je poměrně odlišná od sigmoidy i hyperbolického tangens. Mnohé články však ukazují, že použití ReLU neuronu výrazně zvyšuje přesnost NN v problému klasifikace obrázků. Odpověď na otázku, proč tomu tak je, však nebyla nikdy prokázána, jedná se pouze o empirické zkušenosti. Jednoduchým pozorováním lze alespoň předpokládat, proč ReLU často překonává jiné aktivační funkce. Víme, že sigmoid neurony se, podobně jako tanh neurony, se přestávají učit (případně je jejich učení velmi pomalé), jakmile jsou saturovány, tedy jakmile se jejich výstup přibližuje 0 nebo 1 (fakt, který je způsobený derivací sigmoidy a tanh v těchto bodech). V porovnání s tím aktivační funkce ReLU nikdy nebude vlivem zvyšování vstupních hodnot saturována, neboť má pro $\forall z > 0$ konstantní první derivaci, a tedy nedochází v takovém případě k zpomalování učení. Navíc platí $\mathrm{ReLU}(z) = 0, \forall z \leq 0$, tedy i příslušný gradient je v takovém případě nulový a učení se zcela zastaví. Tato dvě pozorování nám dávají představu o tom, jak ReLU neuron funguje, a konsekventně i možné důvody, proč často ReLU neuron dosahuje lepších výsledků. Tyto teorie však zatím nejsou potvrzeny, stejně jako otázka, zda neexistuje jiný typ neuronu, který by dosahoval ještě výrazně lepších výsledků než ReLU. Je vhodné také podotknout, že nelze



Obrázek 1.7: Graf ReLU funkce.



Obrázek 1.8: Graf funkce ReLU, sigmoidy a hyperbolického tangensu.

obecně uvažovat o ReLU neuronu jako o jediném a nejlepším kandidátovi. Prostor parametrů, jež ovlivňují výkonnost NN, je obrovský, tudíž jediný možný způsob, jak se přesvědčit o vhodnosti daného typu neuronu pro daný problém a danou NN, je experiment a empirická data. [20] [17] [9]

Kapitola 2

Učení v neuronových sítích

Tato sekce byla zpracována na základě knih *Neural Networks and Deep Learning*, NIELSEN M.A. [26] a *Deep Learning*, GOODFELLOW I. et al. [10].

2.1 Gradientní sestup

V tuto chvíli již víme, jak sestavit síť sigmoid neuronů. Pokud takové síti náhodně přidělíme váhy a biasy, bude zpracovávat vstupy, které prostoupí celou sítí a vrátí reálnou hodnotu v intervalu [0, 1]. Otázkou zůstává, jak nastavit zmíněné parametry sítě tak, aby vracela správné predikce. K vyřešení tohoto problému bude požit algoritmus gradientního sestupu.

Jedná se o takzvané učení s učitelem, takže nutnou prerekvizitou k trénování NN je soubor dat (dále dataset). Dataset je soubor mnoha důsledně do kategorií roztříděných a popsaných dat, které se využívají k trénování a následnému testování výkonu NN. Dataset se zpravidla bez výjimky náhodně rozděluje na dvě (až tři, pokud je použit validační dataset) oddělené podmnožiny – data pro učení a data pro testování (v rozdílných poměrech, velice často například v poměru 80:20). Síť se trénuje pouze na datech určených k učení. Po dokončeném tréninku sítě se použijí testovací data, aniž by se s nimi síť předtím setkala, k určení výkonnosti. Tímto způsobem lze ověřit, zda NN dokáže správně detekovat unikátní vlastnosti každé kategorie a ve výsledku tak kvalitně generalizovat. Síť schopná dobré generalizace dokáže správně predikovat výsledky i pro data, která nikdy neviděla – jinými slovy, dosáhne vysoké přesnosti na testovacích datech. Zmíněnými daty mohou typicky být čísla, obrázky, písmena, 3D mapy, rengenové či MR snímky a mnoho dalších. Velmi důležitým aspektem je velikost datasetu, pro zajištění kvalitních výsledků se zpravidla pracuje s tisícovkami dat pro každou kategorii. Dobrým příkladem je známý dataset MNIST [5], který obsahuje 70 000 černobílých fotografií ručně psaných číslic 1 až 9. Každá kategorie tak disponuje 7 000 obrázky.

Označme jeden libovolný vstup jako x a pro tento vstup zamýšlený výstup jako $y = y(x)$ (příkladem budiž NN pro rozpoznávání čísel výše zmíněného datasetu MNIST, pro vstup zobrazující číslo 4 bude platit $y(x) = (0, 0, 0, 0, 1, 0, 0, 0, 0, 0)^{\mathrm{T}}$).

Abychom mohli kvantifikovat správnost výsledků, definujeme funkci C[1] (anglicky *cost function* či *loss function*) jako

$$C(\vec{w}, b) = \frac{1}{2n} \sum_{x} \|y(\vec{x}) - \vec{a}\|^2 . \tag{2.1}$$

Kde $\vec{w}$ značí vektor všech vah v síti, b množinu všech biasů, n celkový počet vstupů, $\vec{a}$ vektor výstupů při vstupu $\vec{x}$ a sumace je aplikována na celou množinu vstupů. Funkce dána vztahem 2.1 se nazývá střední kvadratická chyba nebo také účelová funkce (z anglického *MSE – mean squared error*). Nicméně střední kvadratická chyba není jediná účelová funkce, kterou lze použít. Alternativní účelové funkce budou okomentovány v příštích kapitolách.

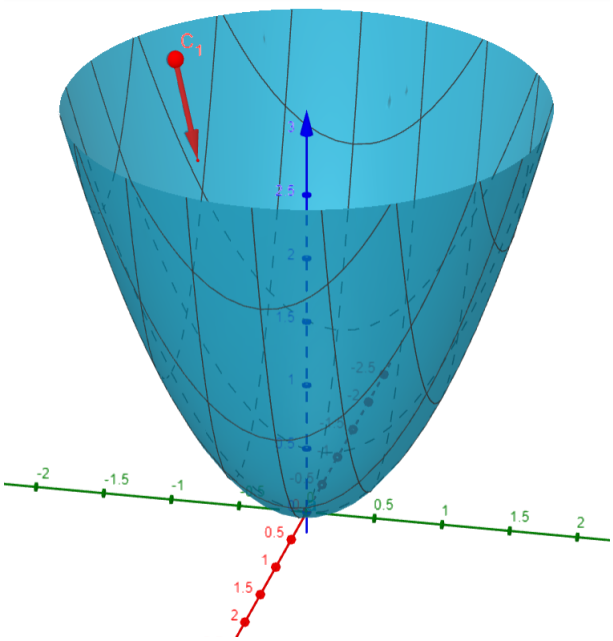
Je snadné nahlédnout, jak se účelová funkce MSE chová. Pro každý vstup vrátí druhou mocninu vzdálenosti mezi výstupem a správným řešením. Po sumaci přes všechny vstupy výsledek zprůměrujeme a dostáváme jakousi průměrnou chybu sítě s ohledem na celou množinu vstupů. Cílem tréninku sítě tak bude funkci C ze vztahu (2.1) jako funkci vah a biasů minimalizovat. Povšimněme, že při tréninku se provádí minimalizace chyby sítě a nikoliv přímo maximalizace přesnosti. Problém v přímé maximalizaci přesnosti je fakt, že počet správných predikcí není spojitou funkcí vah a biasů. Většina malých změn v parametrech neovlivní predikce na výstupu, ač mohou tyto změny pozitivně či negativně ovlivnit celkovou spolehlivost a schopnost generalizace sítě. Opačně tomu je v případě účelové funkce, ta vrátí lehce odlišnou hodnotu pro jakoukoliv malou změnu v parametrech sítě. Z tohoto důvodu budeme vždy minimalizovat účelovou funkci namísto maximalizace počtu správných predikcí.

Funkce C je zákonitě funkcí velkého množství proměnných, pro přehlednou ilustraci gradientního sestupu předpokládejme, že je funkcí pouze dvou proměnných – v_1 a v_2 – a její tvar bude dán vztahem $C(v_1, v_2) = v_1^2 + v_2^2$. Výše zmíněná funkce již zřejmě nemá návaznost na NN, je definována tak, aby usnadnila ilustraci algoritmu gradientního sestupu.

Problém minimalizace funkce více proměnných lze řešit analyticky spočtením prvních dvou derivací a následnou analýzou lokálních minim. Tento způsob řešení je samozřejmě validním a přesným, ovšem prakticky použitelným pouze v případě jednotek proměnných. Máme-li funkci velkého množství proměnných, což účelové funkce v neuronových sítích zcela jistě jsou, stává se analytický přístup natolik komplikovaným, že je praktický nepoužitelný.

Mnohem vhodnější způsob je právě algoritmus gradientního sestupu. Gradient je diferenciální operátor, jenž vyjadřuje směr a velikost největší změny. Uvažujme graf funkce $C(v_1, v_2)$, náhodným určením parametrů v_1 a v_2 dostaneme náhodný bod C_1. Gradient v bodě C_1 bude udávat směr a velikost jakési nejrychlejší cesty k maximu. V případě hledání minima budeme využívat opačný směr gradientu, který zřejmě povede k minimu. Názorná ukázka viz obrázek 2.1. Gradientní sestup potom není nic jednoduššího, než následování, po malých krocích, opačného směru gradientu.

[1] Proměnná C bude v textu označovat jak účelovou funkci, tak hodnotu chyby, tedy výstup účelové funkce. Vždy bude však z kontextu jasné, zda se jedná o funkci, či její výstup.



Obrázek 2.1: Ilustrace gradientního sestupu.

Přesněji, ze vztahu (1.5) víme, že pro funkci $C(v_1, v_2)$ platí

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \,. \tag{2.2}$$

Definováním vektoru změn $\Delta\vec{v} = (\Delta v_1,\, \Delta v_2)^{\mathrm{T}}$ a vektoru gradientů $\nabla C = \left(\frac{\partial C}{\partial v_1},\, \frac{\partial C}{\partial v_2}\right)^{\mathrm{T}}$ pak vztah (2.2) zjednodušíme na

$$\Delta C \approx \nabla C \cdot \Delta\vec{v} \,. \tag{2.3}$$

Aplikováním myšlenky postupného gradientního sestupu budeme v každém kroku hledat takové $\Delta\vec{v}$, aby platilo $\Delta C < 0$. Protože je $\Delta\vec{v}$ volný parametr, který můžeme určit libovolně, je snadné najít takové Δv, abychom zaručili zápornost ΔC. Přesněji se bude jednat o tvar

$$\Delta\vec{v} = -\eta \nabla C \,. \tag{2.4}$$

Kde parametr $\eta \in \mathbb{R}^+$ nazýváme rychlost učení (z anglického *learning rate*) a nejčastěji platí $\eta \in [0,\, 1]$. Parametr η, který bude často zmiňován v nadcházejících kapitolách, určuje, jak velké změny v parametrech budeme v každém kroku gradientního sestupu používat, jinými slovy, jak velké kroky budeme podněcovat v opačném směru gradientu.

Dosazením vztahu (2.4) do rovnice (2.3) dostáváme

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2 \,. \tag{2.5}$$

Protože $\eta > 0$ a z podstaty platí $\|\nabla C\|^2 \geq 0$, máme zaručeno (za předpokladu, že je aproximace ze vztahu (2.5) je dostatečně přesná), že $\Delta C \leq 0$.

Parametry (v tomto případě dvojice parametrů v_1, v_2) budou v každém kroku aktualizovány dle následujícího vztahu:

$$\vec{v} \to \vec{v'} = \vec{v} - \eta \nabla C \,. \tag{2.6}$$

Výše uvedený gradientní sestup byl popsán užitím funkce dvou proměnných, je však jednoduché stejný postup aplikovat na funkci libovolně mnoha proměnných. Uvažujme tedy funkci n proměnných $C(v_1, v_2, \ldots, v_n)$, potom pro ΔC v závislosti na Δv platí vztah (2.3), kde vektor gradientů je rozšířen na celou množinu parametrů, tedy $\nabla C = \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_n}\right)^{\mathrm{T}}$ a analogicky uvažujeme vektor změn $\Delta v = (\Delta v_1, \Delta v_2, \ldots, \Delta v_n)^{\mathrm{T}}$. Obdobně jako v případu dvou proměnných můžeme určit změny parametrů dle vztahu (2.4), a zajistit tím nekladnost ΔC. Následně provedeme aktualizaci parametrů dle vztahu (2.6).

2.1.1 Kvalita metody gradientního sestupu

Gradientní sestup byl prezentován jako metoda pro hledání minima, aniž by byla okomentována kvalita této metody. V této sekci bude jednoduše ukázáno, že gradientní sestup je dokonce optimální strategií pro hledání minima.

Předpokládejme, že chceme najít takové $\Delta\vec{v}$, pro které v daném bodě snížíme hodnotu funkce C maximálním možným způsobem (jinými slovy minimalizujeme ΔC), přičemž je zadána velikost $\Delta\vec{v}$ jako $\|\Delta\vec{v}\| = \varepsilon$, pro nějaké malé fixní $\varepsilon > 0$.

Připomeňme si nejdříve tvar funkce ΔC, který je zadán jako $\Delta C \approx \nabla C \cdot \Delta\vec{v}$, a dále Cauchyho-Schwarzovu nerovnost. Mějme vektory $\vec{u}$ a $\vec{v}$ v unitárním prostoru $\mathbb{V}$, potom platí:

$$\vec{u} \cdot \vec{v} \leq \|\vec{u}\|\|\vec{v}\|, \quad \forall\, \vec{u}, \vec{v} \in \mathbb{V} \tag{2.7}$$

Prohodíme-li strany rovnice, pak platí:

$$-\|\vec{u}\|\|\vec{v}\| \leq \vec{u} \cdot \vec{v}, \quad \forall\, \vec{u}, \vec{v} \in \mathbb{V} \tag{2.8}$$

Navíc rovnost nastává právě tehdy, když jsou vektory $\vec{u}$ a $\vec{v}$ lineárně závislé, tedy $\vec{u} = \alpha\vec{v}$, $\alpha > 0$.

Z Cauchyho-Schwarzovy podmínky tak máme:

$$-\|\Delta\vec{v}\|\|\nabla C\| \leq \Delta\vec{v} \cdot \nabla C \implies -\varepsilon\|\nabla C\| \leq \Delta\vec{v} \cdot \nabla C \tag{2.9}$$

Je zřejmé, že pro minimalizaci ΔC musí platit ve vztahu (2.9) rovnost, tedy musí platit $\Delta\vec{v} = -\alpha\nabla C$. Předpokládejme α takové, že $\alpha = \frac{\varepsilon}{\|\nabla C\|}$. Dostáváme tedy

$$\Delta\vec{v} = -\frac{\varepsilon}{\|\nabla C\|}\,\nabla C. \tag{2.10}$$

Po zpětném dosazení do rovnice (2.9) dojdeme několika algebraickými úpravami k rovnosti.

$$-\varepsilon\|\nabla C\| \leq \Delta\vec{v} \cdot \nabla C \leq -\frac{\varepsilon}{\|\nabla C\|}\,\nabla C \cdot \nabla C \tag{2.11}$$

$$-\varepsilon\|\nabla C\| \leq \Delta\vec{v} \cdot \nabla C \leq -\frac{\varepsilon}{\|\nabla C\|}\,\|\nabla C\|^2 \tag{2.12}$$

$$-\varepsilon\|\nabla C\| \leq -\varepsilon\|\nabla C\| \tag{2.13}$$

Což dokazuje, že $\Delta\vec{v}$ ve tvaru $\Delta\vec{v} = -\eta\nabla C$ ze vztahu (2.4) pro $\eta = \frac{\varepsilon}{\|\nabla C\|}$ minimalizuje ΔC (při fixní velikosti $\Delta\vec{v}$ jako $\|\Delta\vec{v}\| = \varepsilon$), a tedy gradientní sestup je optimální strategie pro hledání minima.

2.1.2 Užití gradientního sestupu v neuronových sítích

Bylo ukázáno, jak aplikovat gradientní sestup na funkci více proměnných, v této sekci uzavřeme kapitolu o gradientním sestupu jeho užitím v NN.

Jediným rozdílem bude obměna parametrů a užití účelové funkce. Parametry neuronové sítě jsou váhy w_k a biasy b_l, gradient účelové funkce ∇C tedy bude mít jim odpovídající komponenty, přesněji pro n vah a m biasů bude tvar ∇C následující:

$$\nabla C = \left(\frac{\partial C}{\partial w_1}, \ldots, \frac{\partial C}{\partial w_n}, \frac{\partial C}{\partial b_1}, \ldots, \frac{\partial C}{\partial b_m}\right) \tag{2.14}$$

V neposlední řadě aktualizujeme váhy a biasy.

$$w_k \to w_k' = w_k - \eta\frac{\partial C}{\partial w_k} \tag{2.15}$$

$$b_l \to b_l' = b_l - \eta\frac{\partial C}{\partial b_l} \tag{2.16}$$

2.1.3 Stochastický gradientní sestup

V praktickém užití se v neuronových sítí primárně používá takzvaný stochastický gradientní sestup, který oproti tomu standardnímu za užití aproximace výrazně snižuje časovou náročnost výpočtů. Povšimněme tvar účelové funkce z (2.1) $C = \frac{1}{n}\sum_x C_x$, kde $C_x = \frac{\|y(x)-a\|^2}{2}$, jinými slovy se jedná o průměr výstupů účelové funkce přes všechny vstupy. Pro výpočet jednoho kroku gradientního sestupu je pak potřeba určit gradient ∇C_x pro každý vstup a následně určit $\nabla C = \frac{1}{n}\sum_x \nabla C_x$. V neuronových sítích se však často setkáváme s tisícovkami vstupních dat, pro úsporu výpočetního času se proto používá stochastický gradientní sestup, který aproximuje ∇C pouze na malém, náhodně vybraném vzorku ze vstupních dat.

Přesněji, mějme malé $r \in \mathbb{N}$ a množinu náhodně vybraných vstupů $x_1, x_2, \ldots, x_r$, této množině budeme říkat anglickým názvem mini-batch. V závislosti na velikosti mini-batche r, dostáváme více či méně přesnou aproximaci ∇C_{x_j} pro ∇C_x.

$$\nabla C = \frac{\sum_x \nabla C_x}{n} \approx \frac{\sum_{j=1}^{r} \nabla C_{x_j}}{r} \tag{2.17}$$

Následná aktualizace vah a biasů je pak analogicky pozměněna na:

$$w_k \to w_k' = w_k - \frac{\eta}{r}\sum_j \frac{\partial C_{X_j}}{\partial w_k}, \tag{2.18}$$

$$b_l \to b_l' = b_l - \frac{\eta}{r}\sum_j \frac{\partial C_{X_j}}{\partial b_l}, \tag{2.19}$$

kde sumace probíhá přes všechny vstupy x_j daného mini-batche.

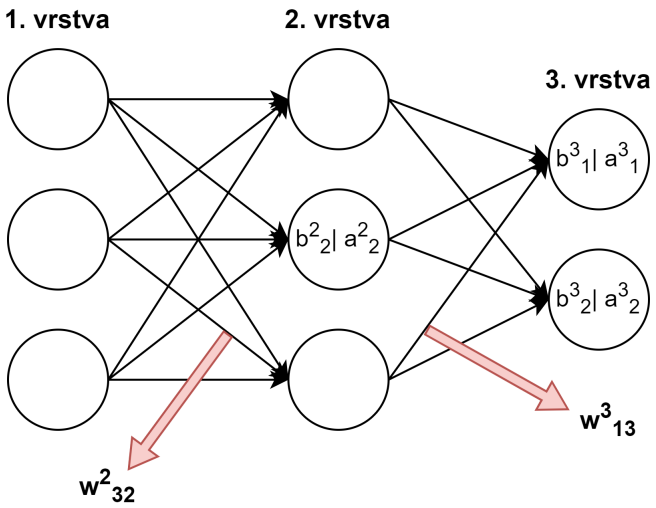
Výše uvedený algoritmus nazýváme stochastickým gradientním sestupem. Při trénování neuronových sítí používáme tento algoritmus iterativně, přičemž mini-batch je náhodně inicializován v každé iteraci. Vyčerpáme-li všechna vstupní data, nazýváme tento stav epochou. Často bude v práci hovořeno o tréninku NN na určitý počet epoch, čímž je myšleno, že každý vstup byl v rámci tréninku použit "epoch-krát".

2.2 Algoritmus zpětného šíření chyby

V minulé sekci byl zaveden algoritmus stochastického gradientního sestupu pro hledání minima a jeho užití v neuronových sítích. Bez komentáře ale zůstal problém spočtení parciálních derivací, jež se v algoritmu nacházejí. Jak se ukáže, vypočítat dané parciální derivace není v případě neuronových sítí zcela přímočaré a bude z tohoto důvodu zaveden algoritmus zpětné šíření chyby (anglicky *backpropagation*) pro jejich spočtení. Tato kapitola se bude opírat o článek *Learning representations by back-propagating errors* ([30]), který užívá algoritmus zpětného šíření chyby pro neuronové sítě.

2.2.1 Terminologie

Ze všeho nejdřív bude nutné zavést značení, které bude užito k nadcházející definici. Váhou w^l_{jk} budeme rozumět váhu na hraně jdoucí z k-tého neuronu ve vrstvě $(l-1)$ do j-tého neuronu v l-té vrstvě. Analogicky budeme používat značení b^l_j pro bias j-tého neuronu v l-té vrstvě a a^l_j pro aktivaci (výstup z aktivační funkce) j-tého neuronu v l-té vrstvě. Pro názornost je přiložena ilustrace na obrázku 2.2.



Obrázek 2.2: Anotace vah, biasů a aktivací.

Užitím výše zmíněného zápisu můžeme zapsat aktivaci j-tého neuronu v l-té vrstvě

jako

$$a_j^l = \sigma\left(\sum_k w_{jk}^l \, a_k^{l-1} + b_j^l\right), \tag{2.20}$$

kde σ značí aktivační funkci a sumace probíhá přes všechny neurony v $(l-1)$-ní vrstvě.

Vztah (2.20) lze zjednodušit maticovým zápisem. Předpokládejme s neuronů v $(l-1)$-ní vrstvě a t neuronů v l-té vrstvě, potom matice vah $W^l \in \mathbb{R}^{t\times s}$ má tvar

$$W^l = \begin{pmatrix} w_{11}^l & w_{12}^l & \cdots & w_{1s}^l \\ w_{21}^l & w_{22}^l & \cdots & w_{2s}^l \\ \vdots & \vdots & \ddots & \vdots \\ w_{t1}^l & w_{t2}^l & \cdots & w_{ts}^l \end{pmatrix}. \tag{2.21}$$

Obdobně definujeme vektor biasů b^l a aktivací a^l jako

$$\vec{b}^l = \left(b_1^l, b_2^l, \ldots, b_t^l\right), \tag{2.22}$$

$$\vec{a}^l = \left(a_1^l, a_2^l, \ldots, a_t^l\right). \tag{2.23}$$

Užitím maticového zápisu lze vztah (2.20) jednoduše přepsat pro aktivace všech neuronů v l-té vrstvě v následujícím kompaktním tvaru

$$\vec{a}^l = \sigma\left(W^l \, \vec{a}^{l-1} + \vec{b}^l\right), \tag{2.24}$$

kde aktivační funkci aplikujeme po složkách, platí tedy $\sigma\left(v\right)_j = \sigma\left(v_j\right)$.

Poznamenejme, že účelová funkce MSE ze vztahu (2.1) má užitím této notace následující tvar

$$C = \frac{1}{2n}\sum_x \left\|y(x) - a^L(x)\right\|^2, \tag{2.25}$$

kde $\vec{y} = y(\vec{x})$ je stále vektorem zamýšlených výstupů, n značí celkový počet vstupních dat, L výstupní (poslední) vrstvu sítě, sumace probíhá přes všechna vstupní data $\vec{x}$ a $a^L = a^L(\vec{x})$ je vektor aktivací poslední vrstvy (tedy výstupů sítě) při vstupu $\vec{x}$.

V nadcházejících kapitolách bude z důvodu přehlednosti vztah (2.24) psán jako $\vec{a}^l = \sigma\left(\vec{z}^l\right)$, kde

$$\vec{z}^l = W^l \, \vec{a}^{l-1} + \vec{b}^l. \tag{2.26}$$

2.2.2 Předpoklady kladené na účelovou funkci

V kapitole 2.1 byla prezentována účelová funkce MSE – formule (2.1). Ač bude MSE nadále často použita jako příklad účelové funkce, jedná se pouze o jednu z mnoha funkcí, jež je možné použít k měření chyby. Je tedy důležité postavit teoretický základ dostatečně obecný, aby byl aplikovatelný na různé účelové funkce. Krom toho budou v této sekci prezentovány dva předpoklady kladené na účelovou funkci, které jsou nezbytné pro definici algoritmu zpětného šíření chyby.

První podmínkou je platnost

$$C = \frac{1}{n} \sum_x C_x \,, \tag{2.27}$$

kde C_x opět značí míru chyby pro jeden vstup x. Jednoduše řečeno podmínka předpokládá, že celkovou chybu lze přepsat jako průměr chyb pro jednotlivé vstupy, kde chyba je udávána účelovou funkcí. V případě funkce MSE je tato podmínka splněna, neboť chyba jednoho vstupu x je $C_x = \frac{1}{2}\|y - a^L\|^2$. Důvod pro tuto podmínku je jednoduchý, algoritmus zpětného šíření chyby bude udávat hodnoty parciálních derivací vah a biasů pro jednotlivé vstupy, parciální derivace $\frac{\partial C}{\partial w^l_{jk}}$ a $\frac{\partial C}{\partial b^l_j}$ poté stanovíme jako průměr parciálních derivací všech vstupů.

Druhou podmínkou buď

$$C_x = C(a^L) \,. \tag{2.28}$$

Ta říká, že chyba sítě musí být funkcí výstupů sítě. Funkce MSE je funkcí výstupů, a tak i tuto podmínku splňuje.

2.2.3 Definice algoritmu zpětného šíření chyby

Již máme zavedeno vše tak, abychom mohli přejít k samotnému algoritmu zpětného šíření chyby[2] (anglicky *backpropagation*).

Nejdříve definujme parametr δ^l_j jako

$$\delta^l_j = \frac{\partial C}{\partial z^l_j} \,, \tag{2.29}$$

kde $z^l_j = (z^l)_j$ z definice (2.26). Parametr δ^l_j nazveme chybou j-tého neuronu v l-té vrstvě. V podobném znění jako v předchozích kapitolách definujeme vektor chyb neuronů v l-té vrstvě jako $\vec{\delta}^l$. Myšlenka algoritmu zpětného šíření chyby spočívá v určení δ^l pro každou vrstvu a, užitím tohoto vektoru, následném vyjádření $\frac{\partial C}{\partial w^l_{jk}}$ a $\frac{\partial C}{\partial b^l_j}$. Přesněji půjde o 4 definované vztahy, které budou dohromady popisovat, jak spočíst chybu a z ní vyjádřit ony parciální derivace.

1. Pro parametr δ^L_j, kde L značí poslední/výstupní vrstvu platí

$$\delta^L_j = \frac{\partial C}{\partial a^L_j} \sigma'(z^L_j). \tag{2.30}$$

   Popíšeme-li tuto formuli slovy, tak parciální derivace vyjadřuje velikost změny celkové chyby sítě jako funkci aktivace j-tého neuronu v L-té, poslední vrstvě, protože ale je navíc aktivace závislá na aktivační funkci (jedná se o funkci složenou) tak je potřeba vyjádřit i velikost změny aktivace v závislosti na jejím vstupu. Tato formule platí jen a pouze pro poslední vrstvu neuronové

[2] Není chyba jako chyba: slovo chyba se v pozdější fázi práce bude vztahovat k procentuálnímu vyjádření správně klasifikovaných dat neuronovou sítí, naopak u algoritmu zpětného šíření chyby se slovo chyba vztahuje k účelové funkci.

sítě, pro vrstvy zbylé je výpočet chyby popsán v příštím bodě. Zároveň, pro ucelenost, níže uvádím vztah (2.30) v jeho maticové formě, která bude nadále v textu využívána primárně.

$$\delta^L = \nabla_{\vec{a}} C \odot \sigma'(z^l)\,, \tag{2.31}$$

kde $\odot$ značí násobení po složkách – pro dva vektory $\vec{a}$, $\vec{b}$ stejné délky platí $(\vec{a} \odot \vec{b})_j = a_j b_j$, a $\nabla_{\vec{a}} C$ je vektor parciálních derivací $\frac{\partial C}{\partial a_j^L}$. Poznamenejme, že jak aktivační, tak účelová funkce jsou předem známy a není příliš složité určit jejich parciální, či obyčejné derivace.

2. Parametr δ_j^l, kde l značí jakoukoliv vrstvu mimo tu poslední, je počítán ve vztahu k δ_j^{l+1}. A to přesně jako

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'\left(z_j^l\right)\,, \tag{2.32}$$

opět bude primárně používán maticový tvar

$$\vec{\delta}^l = \left(\left(W^{l+1}\right)^T \vec{\delta}^{l+1}\right) \odot \sigma'(\vec{z}^l)\,. \tag{2.33}$$

Kombinací formulí (2.31) a (2.33) již lze spočíst chybu pro všechny vrstvy sítě, přičemž se zřejmě postupuje iterativně od poslední vrstvy (za použití (2.31)) zpětně (za použití (2.33)) k té první.

3. Za užití spočítaných chyb lze vyjádřit parciální derivaci celkové chyby podle biasu j-tého neuronu v l-té vrstvě $\frac{\partial C}{\partial b_j^l}$ jednoduše jako

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l\,. \tag{2.34}$$

4. A obdobně pak parciální derivaci celkové chyby podle váhy na hraně z k-tého neuronu v $(l-1)$ vrstvě do j-tého neuronu v l-té vrstvě $\frac{\partial C}{\partial w_{jk}^l}$ jako

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l\,. \tag{2.35}$$

Výše uvedená čtveřice vztahů velice jednoduše zprostředkovává parciální výsledné chyby dané účelovou funkcí podle všech vah a biasů v síti. Následnou aplikací stochastického gradientního sestupu tyto váhy a biasy aktualizujeme za účelem minimalizace účelové funkce. Opakováním kombinace těchto dvou metod lze postupně minimalizovat celkovou chybu, čímž se jako vedlejší produkt zlepšuje přesnost klasifikace neuronové sítě. Tím jest popsán postup trénování neuronové sítě.

2.2.4 Vlastnosti algoritmu zpětného šíření chyby

Algoritmus zpětného šíření chyby je v tuto chvíli definovaný a popsaný, čímž se naskýtá prostor na okomentování některých vlastností a omezení, jež z definic vyplývají.

- Z výrazu (2.35) je vidno, že parciální derivace podle vah je závislá na aktivaci neuronu z předchozí vrstvy. Aktivace je přitom reálné číslo z intervalu [0, 1]. Stane-li se, že $a_k^{l-1} \to 0$, pak bude i příslušná parciální derivace směřovat k nule, což vyústí v malou změnu příslušné váhy při aplikaci stochastického gradientního sestupu. Navíc se stejná hodnota aktivace a_k^{l-1} šíří do všech neuronů v l-té vrstvě, tedy všechny váhy putující z inkriminovaného neuronu budou sdílet podobný scénář. V takovémto případě budeme říkat, že se dané váhy učí pomalu.

- Výraz (2.30), potažmo (2.31) a výraz (2.33) obsahují stejnou proměnnou, a to derivaci aktivační funkce σ. Připomeňme, že aktivační funkce mohou mít různorodé tvary, pro příklad však uvažujme sigmoidu ze sekce 1.2.2. Ta se pro parametry $z_j^l = \mathbb{R} \setminus [0,\, 1]$ stává téměř konstantní, což implikuje $\sigma'\left(z_j^l\right) \approx 0$. Z toho vyplývá, že váhy a biasy se budou učit pomalu, pokud je aktivace daného neuronu buď ≈ 0, nebo ≈ 1 a váhy nebo biasy nejsou dostatečně veliké na to, aby tento rozdíl vykompenzovaly. Takovému stavu říkáme, že je neuron saturovaný a jeho učení se zastavilo nebo je pomalé.

- Z definic je mimo jiné vidět jedna z velkých výhod algoritmu zpětného šíření chyby, a sice že umožňuje spočítat pro jeden vstup všechny parciální derivace vah a biasů z jednoho dopředného průchodu sítě a jednoho zpětného průchodu sítě. Uvažme použití konvenční cesty počítání derivací z definice, pokud bychom měli v sítí milion vah a biasů dohromady, pro jeden vstup by to znamenalo, z definice derivace, spočtení milionu účelových funkcí, jinými slovy provedení milionu dopředných průchodů sítě.

2.2.5 Důkaz algoritmu zpětného šíření chyby

1. Chceme dokázat, že platí $\delta_j^L = \frac{\partial C}{\partial a_j^L}\, \sigma'(z_j^L)$. Z definice (2.29) víme, že

$$\delta_j^L = \frac{\partial C}{\partial z_j^L}\,, \tag{2.36}$$

   kde z_j^L je argumentem funkce a_k^L a ta je argumentem funkce C. Z pravidel o derivaci složené funkce dostáváme

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L}\,, \tag{2.37}$$

   kde sumace probíhá přes všechny neurony k ve výstupní vrstvě. Aktivace a_k^L je ale závislá pouze na takovém z_j^L, kde $k = j$, pro všechny ostatní případy

budou členy sumy rovny nule. Dostáváme tak

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \,. \tag{2.38}$$

A protože $a_j^L = \sigma\left(z_j^L\right)$, přepíšeme vztah (2.38) ekvivalentně na

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'\left(\partial z_j^L\right) \,. \tag{2.39}$$

Tím je důkaz dokončen.

2. Chceme dokázat, že platí $\delta^l = \left(\left(W^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'(z^l)$. Opět rozvineme definici (2.29).

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} \tag{2.40}$$

Z definice (2.26) víme, že

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma\left(z_j^l\right) + b_k^{l+1} \,. \tag{2.41}$$

Výraz napravo lze derivovat podle z_j^l jako

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'\left(z_j^l\right) \,. \tag{2.42}$$

Zpětným dosazením do (2.40) dostáváme

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'\left(z_j^l\right) \,, \tag{2.43}$$

což není nic jiného než dokazovaný vztah rozepsaný po jednotlivých komponentech, viz (2.32). Důkaz je tímto hotový.

3. Chceme dokázat, že platí $\frac{\partial C}{\partial b_j^l} = \delta_j^l$. Připomeňme, že b_j^l je argumentem funkce z_j^l, navíc platí

$$\frac{\partial z_j^l}{\partial b_j^l} = 1. \tag{2.44}$$

Rozepíšeme si b_j^l podle pravidla derivace složené funkce, máme

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial C}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} = \frac{\partial C}{\partial z_j^l} \,. \tag{2.45}$$

S užitím definice (2.29) tak platí

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} = \delta_j^l \,. \tag{2.46}$$

Tímto je důkaz dokončen.

4. Chceme ukázat, že platí $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$. Postup bude velice podobný předchozímu důkazu. Parametr w^l_{jk} je argumentem pouze funkce z^l_j, která je argumentem funkce a^l_j. Opět tedy $\frac{\partial C}{\partial w^l_{jk}}$ rozepíšeme jako

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial a^l_j} \frac{\partial a^l_j}{\partial z^l_j} \frac{\partial z^l_j}{\partial w^l_{jk}} . \tag{2.47}$$

Přičemž platí

$$\frac{\partial z^l_j}{\partial w^l_{jk}} = a^{l-1}_k. \tag{2.48}$$

Zpětným dosazením tak dostáváme

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial a^l_j} \frac{\partial a^l_j}{\partial z^l_j} a^{l-1}_k = \frac{\partial C}{\partial z^l_j} a^{l-1}_k . \tag{2.49}$$

Opět užitím definice (2.29) dostáváme zamýšlenou rovnici a dokončujeme důkaz

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j} a^{l-1}_k = a^{l-1}_k \, \delta^l_j . \tag{2.50}$$

Tímto jsou všechny 4 formule zpětného šíření chyby dokázané.

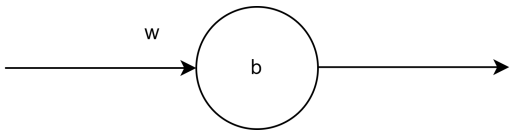
Kapitola 3

Techniky zlepšování výkonu neuronových sítí

3.1 Účelová funkce křížové entropie

Doposud byla jako příklad uváděna účelová funkce MSE, v praktické části však z důvodů lepších výsledků budou konstruovány neuronové sítě využívající účelovou funkci křížové entropie. V této kapitole okomentuji omezení účelové funkce MSE a zavedu účelovou funkci křížové entropie, která tato omezení redukuje.

Pro lepší názornost budeme používat příklad jednoho neuronu s jedinou vstupní hranou, všechna pozorování budou však aplikovatelná na jakkoliv velikou neuronovou síť. Ilustrace takového neuronu je na obrázku 3.1. Předpokládejme trénování



Obrázek 3.1: Ilustrační neuron s jedinou vstupní hranou.

tohoto jediného neuronu, dle učení zavedeného v minulých kapitolách. Víme, že takový neuron bude v průběhu učení měnit váhu w a bias b v závislosti na parciálních derivacích $\frac{\partial C}{\partial w}$, $\frac{\partial C}{\partial b}$. Jinými slovy rychlost učení neuronu je závislá na parciálních derivacích účelové funkce podle váhy a biasu. Připomeňme tvar účelové funkce MSE, který je v tomto případě jediného neuronu následující

$$C = \frac{(y-a)^2}{2}. \tag{3.1}$$

Protože $a = \sigma(z)$, kde $z = wx + b$, tak výše zmíněné parciální derivace spočteme

jako

$$\frac{\partial C}{\partial w} = (y - a)\,\sigma'(z)\,x\,, \tag{3.2}$$

$$\frac{\partial C}{\partial b} = (y - a)\,\sigma'(z)\,. \tag{3.3}$$

Předpokládejme případ, kdy inicializujeme učení tohoto neuronu náhodným určením vah a biasu, přičemž výstup neuronu a bude zcela opačný než zamýšlená hodnota y – pro ilustraci použijme příklad, kdy $a = 0$ a $y = 1$. Ony parciální derivace budou rovny

$$\frac{\partial C}{\partial w} = \sigma'(z)\,x\,, \tag{3.4}$$

$$\frac{\partial C}{\partial b} = \sigma'(z)\,. \tag{3.5}$$

Parciální derivace v tomto případě závisí pouze na derivaci aktivační funkce, která je v našem případe sigmoidou. Připomeňme, že na krajích sigmoidy její derivace klesá k nule. Určíme-li při inicializaci tréninku váhu a bias tak nešťastně, že pro tyto parametry bude derivace sigmoidy blízká nule, bude se neuron učit velmi pomalu – bude saturovaný. Toto chování je však zcela opačné než bychom předpokládali. Příklad jsme definovali záměrně tak, aby na výstupu neuronu byla výrazně chybná hodnota, což evokuje dojem, že i míra změny váhy a biasu by v takovém případě měla být velká, jinými slovy, aby v takovém případě probíhalo učení rychle. Náš příklad však ukazuje, že tomu může být zcela opačně a neuron se zpočátku nebude učit téměř vůbec. Tato anomálie vyústí v dlouhý a neefektivní trénink. Navíc, jak bude později uvedeno, tento jev se může projevit v jakkoliv velké neuronové sítí. Existuje vícero řešení tohoto problému. První, které bude uvedeno, je změna účelové funkce právě na funkci křížové entropie.

Účelová funkce křížové entropie je pro jeden výstupní neuron dána vztahem

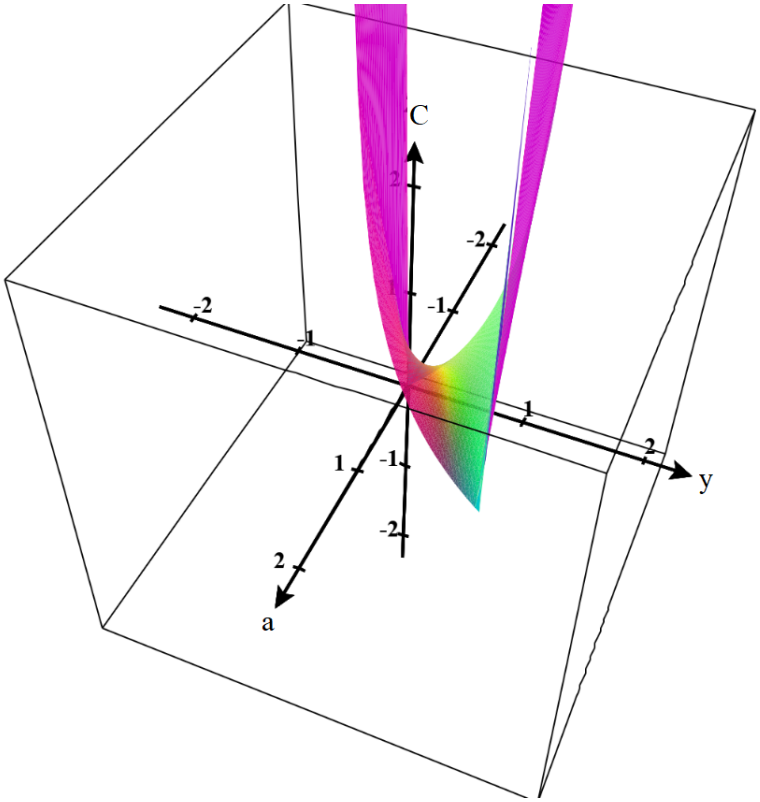
$$C = -\frac{1}{n}\sum_x \left[y \ln a + (1 - y)\ln(1 - a)\right]\,, \tag{3.6}$$

kde n je celkový počet vstupních dat a sumace probíhá přes všechna vstupní data x. Pro více-neuronovou výstupní vrstvu budeme navíc sčítat přes všechny výstupní neurony j

$$C = -\frac{1}{n}\sum_x \sum_j \left[y_j \ln a_j^L + (1 - y_j)\ln\left(1 - a_j^L\right)\right]\,. \tag{3.7}$$

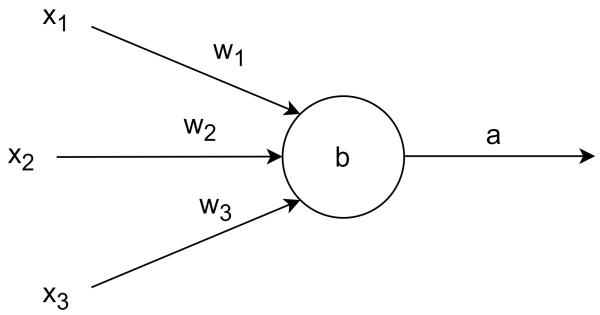
Všimněme si podobností s funkcí MSE:

- Funkce je nezáporná. Argumenty obou logaritmů jsou jistě z intervalu [0, 1], tedy logaritmy vracejí zápornou hodnotu a zároveň je celý výraz násoben -1.
- Funkce klesá, pokud se a přibližuje k y, tedy platí $y = r \wedge a \approx r \implies C \approx 0$, $\forall r \in [0, 1]$. Není přímočaré tuto vlastnost z předpisu funkce vyčíst, proto pro názornost přikládám její graf na obrázku 3.2.



Obrázek 3.2: Graf funkce křížové entropie.

Krom základních společných aspektů ale tato funkce oproti funkci MSE netrpí na výše zmíněný jev saturace při velké chybě. Abychom ukázali, proč tomu tak je, předpokládejme jiný typ neuronu, a to sice neuron se třemi vstupními hranami a jedním výstupem (obrázek 3.3). Spočtěme nyní parciální derivaci funkce křížové



Obrázek 3.3: Ilustrační neuron s třemi vstupními hranami.

entropie podle vah.

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n} \sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)} \right) \frac{\partial \sigma}{\partial w_j} = \tag{3.8}$$

$$= -\frac{1}{n} \sum_x \left(\frac{y}{\sigma(z)} - \frac{(1-y)}{1-\sigma(z)} \right) \sigma'(z)\, x_j = \tag{3.9}$$

$$= \frac{1}{n} \sum_x \frac{\sigma'(z)\, x_j}{\sigma(z)(1-\sigma(z))} \left(\sigma(z) - y\right). \tag{3.10}$$

Pro příklad použijeme opět sigmoidní aktivační funkci z definice (1.4), jejíž derivace má tvar $\sigma'(z) = \sigma(z)(1-\sigma(z))$. Dosazením do rovnice (3.10) dostaneme

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j \left(\sigma(z) - y\right), \tag{3.11}$$

analogicky potom dostáváme i parciální derivaci účelové funkce podle biasu.

$$\frac{\partial C}{\partial b} = \frac{1}{n} \sum_x \left(\sigma(z) - y\right). \tag{3.12}$$

Z poměrně nic neříkajícího zápisu jsme se dostali k jednoduché formuli, ze které je jasně vidět, že parciální derivace podle vah a biasu je závislá na rozdílu zamýšleného výstupu od aktivace. Tedy i rychlost učení neuronu je na tomto rozdílu závislá, přičemž zřejmě platí, že čím více jsou tyto dva parametry odlišné, tím větší je rozdíl, a tím pádem rychlejší učení. Na praktickém příkladu jsme tak ukázali, že účelová funkce křížové entropie, na rozdíl od funkce MSE, garantuje vhodné chování v situaci, kdy máme výrazný rozdíl mezi hodnotou na výstupu a hodnotou zamýšlenou – čím větší chyba na výstupu, tím větší rychlost učení.

3.2 Softmax

Užití účelové funkce křížové entropie není jediný způsob, jak zmírnit zpomalování učení neuronů. Jedna z dalších metod je funkce softmax, která bude využita ve všech neuronových sítích v praktické části. Překvapivě však nepramení její využití v druhé části práce z její schopnosti zmírnit zpomalování učení jako spíše z výhod, kterými disponuje při klasifikačních problémech.

Funkce Softmax je speciální aktivační funkce, která se výhradně používá v poslední, výstupní vrstvě. Nejedná se tedy o aktivační funkci ve smyslu sigmoidy, protože softmax není přímou alternativou k ostatním aktivačním funkcím, neboť se v síti používá pouze v poslední vrstvě, zatímco ve všech ostatních neuronech bude jedna z klasických aktivačních funkcí (například zmíněná sigmoida). Softmax má následující tvar

$$a_j^L = \sigma(z_j^L) = \frac{\mathrm{e}^{z_j^L}}{\sum_k \mathrm{e}^{z_k^L}}, \tag{3.13}$$

kde sumace probíhá přes všechny výstupní neurony k. Všimněme si jedné zásadní vlastnosti, a sice

$$\sum_j a_j^L = \frac{\sum_j \mathrm{e}^{z_j^L}}{\sum_k \mathrm{e}^{z_k^L}} = 1\,. \tag{3.14}$$

Jinými slovy, součet výstupů všech výstupních neuronů je nutně roven 1. Navíc je z definice patrné, že softmax, stejně jako sigmoida, vrací hodnoty v intervalu $[0, 1]$. Kombinací těchto dvou pozorování je zřejmé, že chování funkce softmax připomíná pravděpodobnostní rozdělení, což je v případě klasifikačních problémů užitečná vlastnost. Představme si, že klasifikujeme obrázky pomocí neuronových sítí do 10 kategorií. Použijeme-li softmax jako aktivační funkci výstupní vrstvy, dostaneme na výstupu hodnoty pro jednotlivé kategorie, které lze interpretovat jako pravděpodobnost, že vstup do dané kategorie patří. Jako příklad vezměme již několikrát zmíněný datový soubor ručně psaných jednotkových čísel MNIST. Výstupem sítě pro jeden vstupní obrázek by mohl být třeba vektor $a^L = \left[\frac{5}{100}, \frac{6}{100}, \frac{15}{100}, \frac{2}{100}, \frac{8}{100}, \frac{2}{100}, \frac{6}{100}, \frac{5}{100}, \frac{5}{100}, \frac{1}{100}\right]^T$. V takovém případě by si síť nebyla zcela jistá, o jaké číslo se jedná. Nejpravděpodobněji by šlo o číslo 9, a to s pravděpodobností 50 %, že opravdu jde o obrázek s číslem 9.

3.3 Přeučení

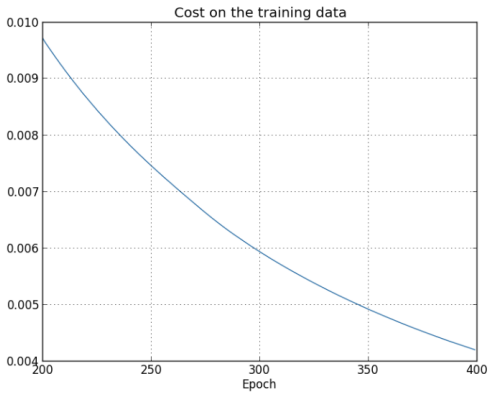
Takzvané přeučení (anglicky *overfitting*) je jeden z nejzásadnějších problémů tréninku neuronových sítí. Cílem naprosté většiny dobře natrénovaných NN je vysoká schopnost generalizace – síť trénujeme na známých datech a požadujeme, aby byla schopna správné klasifikace dat neznámých. Tomu, jak dobře umí síť detekovat podstatné jevy v datech, a tím pádem správně klasifikovat, se často říká právě generalizace. Z toho pramení i důvod, proč se při tréninku rozděluje dataset na tréninkový a testovací, jednoduše nás zajímá mnohem méně úspěšnost, s jakou síť klasifikuje data, na kterých je učená, než úspěšnost na datech, která "nikdy neviděla". Overfitting je právě ten nechtěný stav, kdy rapidně klesá nebo stagnuje úspěšnost klasifikace (obrázek 3.5) a skokově roste chybovost (obrázek 3.4), neboli hodnota, kterou vrací účelová funkce, na testovacích datech, přičemž není neobvyklé, že úspěšnost na tréninkových datech atakuje 100 % (obrázek 3.7) a účelová funkce klesá k nule (obrázek 3.6). Při overfittingu tedy síť až příliš detailně detekuje jevy na trénovacích datech, které nejsou sdíleny obecně skrz danou kategorii, a tím ztrácí úspěšnost na datech testovacích. Připomeňme, že velké neuronové sítě mají často miliony či miliardy parametrů. Takové množství volných parametrů je schopno aproximovat ohromně komplexní funkci, jinými slovy zachytit velké množství velmi komplexních jevů v datech.

Příkladem s tendencí k přeučení budiž klasifikace vozidel na automobily, nákladní vozy a autobusy, kde bychom síť s miliony parametry učili pouze na jednom konkrétním typu auta, nákladního vozu a autobusu, přičemž v testovacích datech by byly vozy různých značek. Za předpokladu, že budeme síť trénovat dostatečně dlouho, dostaví se jev overfittingu, neboť by se síť až příliš soustředila na aspekty dané značky (například logo nebo tvary, kterými se daná značka identifikuje). Tento příklad je

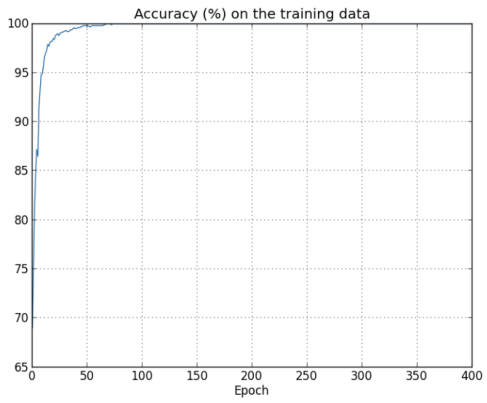
záměrně přehnaný, slouží pouze jako demonstrace, obecně ale k overfittingu dochází, za předpokladu dostatečně dlouhého trénování, i na mnohem pestřejších datasetech.

Z minulého paragrafu ihned vyplývá nejdůležitější faktor pro omezení overfittingu, a sice velikost a vlastnosti datasetu. V ideálním případě chceme síť trénovat na co největším a nejrůznorodějším datasetu. Ideální však praktické prostředí nikdy není a mnohdy je získávání dat drahé, časově náročné nebo přímo nemožné. Druhým faktorem pramenícím z pozorování je velikost sítě, přesněji počet parametrů – čím více parametrů, tím spíše dochází k overfittingu. Zároveň ale víme, že větší a hlubší sítě mají větší potenciál pro rozhodování komplexnějších problémů. Snižování parametrů sítě je tak v rozporu s cílem vytváření výkonné NN. Z těchto důvodu bylo zavedeno několik technik, které si kladou za cíl minimalizovat jev přeučení. V následujících sekcích budou popsané vybrané techniky, které budou zároveň používané v praktické části.

Níže jsou přiloženy grafy dobře zřetelného jevu přeučení (grafy[1] byly převzaty z [26]). Poznamenejme, že chybovostí v popiskách grafu je myšlena hodnota účelové funkce, přesnost je potom procentuální vyjádření správně klasifikovaných vstupů.



Obrázek 3.4: Ukázka jevu přeučení: graf chybovosti na trénovacích datech. [26]

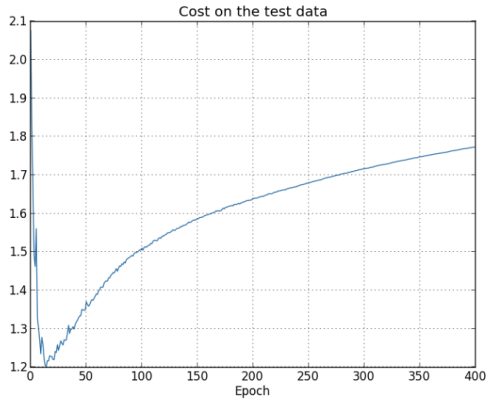


Obrázek 3.5: Ukázka jevu přeučení: graf přesnosti na trénovacích datech. [26]

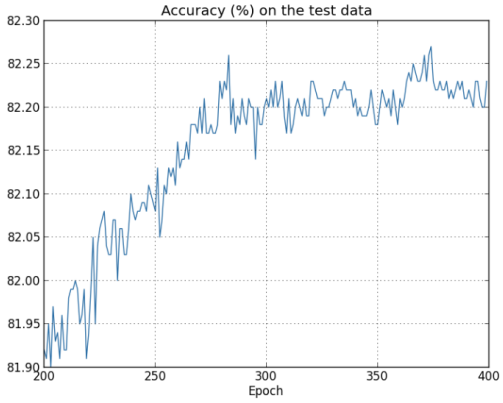
3.3.1 Poznámka ke schopnosti generalizace

Schopnost generalizace je tedy jeden z hlavních atributů tréninku NN a existuje spousta technik, které se snaží síť přimět dobře generalizovat. Všechny tyto techniky mají ale své hranice a rozhodně se dobře generalizující síť nedá přirovnat ke schopnostem generalizace lidského mozku. Představme si, že nám někdo ukáže pouze několik obrázků zvířete, které jsme nikdy předtím neviděli, s velkou pravděpodobností budeme následně schopni takové zvíře rozeznat na stovkách dalších obrázků,

[1] Jedná se o grafy z tréninku NN s 23860 parametry, jednou skrytou vrstvou a 30 neurony v této vrstvě. Trénovacím datasetem bylo 1000 náhodně vybraných obrázků z datasetu MNIST. Trénink probíhal s konstantní mírou učení (learning rate) 0,15 a velikostí mini-batche 10. Zastavovacím kritériem bylo 400 epoch.



Obrázek 3.6: Ukázka jevu přeučení: graf chybovosti na testovacích datech. [26]



Obrázek 3.7: Ukázka jevu přeučení: graf přesnosti na testovacích datech. [26]

a to i za předpokladu, že bude zobrazeno z jiného úhlu nebo v jiném prostředí. Toto je schopnost, kterou neuronové sítě z několika obrázků jednoduše nedokáží. Nastává otázka, proč nezkusíme do umělých neuronových sítí zakomponovat imitaci techniky generalizace lidského mozku. Pravda je taková, že v současné době zatím nevíme přesně, jakým způsobem je takto dobré generalizace schopen. Dá se předpokládat, že s postupem času věda odpovědi na tyto otázky najde a bude velmi zajímavé sledovat, jaký to bude mít vliv na umělé neuronové sítě.[16]

Na druhou stranu, vezmeme-li v potaz, že sítě s miliony parametry se dokáží naučit generalizovat velmi dobře na 50 000 obrázcích (takové příklady budou uvedeny v praktické části v kapitole 7), tak se to zdá být nepochopitelné. Analogicky je toto situací, kdy se snažíme vytvořit polynomiální model miliontého stupně na 50 000 dat. Dle teorie, kterou jsme zavedli, by takový model měl zcela jistě overfittovat, a přeci tomu tak není. Proč tomu tak není, je otázkou, na kterou neznáme přesnou odpověď, neboť to jednoduše nevíme. Je usuzováno, že "dynamika učení pomocí gradientního sestupu má ve vícevrstvých sítích schopnost sebe-regularizačního efektu[22]"(citace přeložena z angličtiny).

3.4 L2 regularizace

Technikám, které mají za cíl snižovat jev přeučení, se často říká regularizační. V této sekci bude popsána takzvaná L2 regularizace, která bude v praktické části užita ve všech sítích.

L2 regularizace tkví v přidání takzvaného regularizačního členu do účelové funkce (takovou účelovou funkci nazveme regularizovaná účelová funkce). Regularizační člen je v případě L2 regularizace suma čtverců všech vah v síti vynásobená takzvaným

regularizačním parametrem a podělená počtem dat v datasetu.

$$L_2 = \frac{\lambda}{2n} \sum_w w^2 \,, \tag{3.15}$$

kde $\lambda \in \mathbb{R}^+$ je regularizační parametr. V případě účelové funkce křížové entropie dostáváme po přidání regularizačního členu

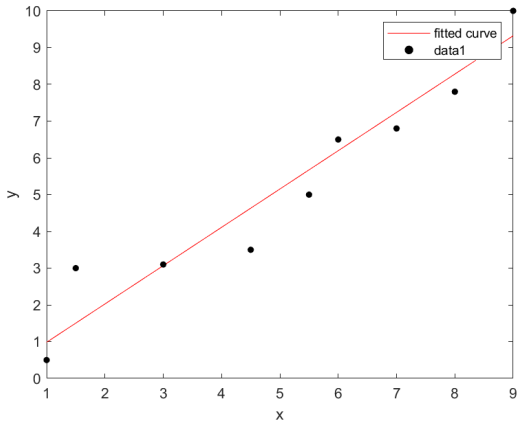
$$C = -\frac{1}{n} \sum_x \sum_j \left[y_j \ln a_j^L + (1 - y_j) \ln \left(1 - a_j^L\right) \right] + \frac{\lambda}{2n} \sum_w w^2 \,. \tag{3.16}$$

Protože se při tréninku sítě minimalizuje účelová funkce, do které se po přidání regularizačního členu propisují i velikosti vah, je zřejmé, že bude algoritmus učení do jisté míry udržovat váhy malé. Navíc si všimněme, že se v sumě nachází druhá mocnina vah, to má za účel exponenciálně penalizovat váhy velké a naopak favorizovat váhy malé. Míra, jakou v průběhu tréninku povolujeme velikost vah vzhledem k účelové funkci, je dána regularizačním parametrem λ. Obvykle nastavujeme regularizační parametr na logaritmické škále v intervalu $[0; 0{,}1]$, přičemž 0 zřejmě značí neregularizovanou účelovou funkci.

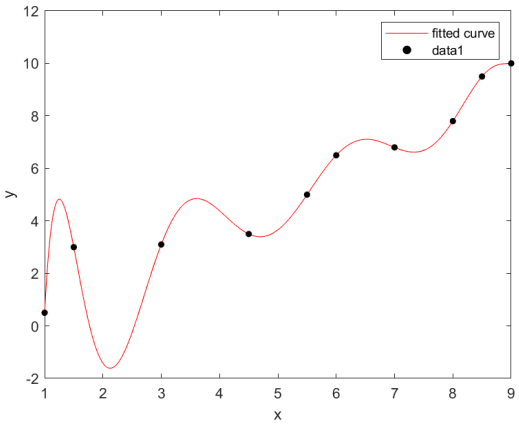
Krom omezování přeučení regularizace omezuje i uvíznutí v chybném lokálním minimu. Představme si trénink sítě bez regularizace. Váhy v síti budou mít tedy větší tendenci se v průběhu učení zvětšovat a velikost vektoru vah bude růst. A protože gradientní sestup provádí jen velmi malé změny v závislosti na míře učení (*learning rate*), tak se vektor vah příliš nezmění a bude stále ukazovat velice podobným směrem. Tento fakt zapříčiní, že se v průběhu učení řádně "neprozkoumá" prostor vah a hrozí, že algoritmus hledání minima "přehlédne" lepší lokální minima, neboť se k nim jednoduše nedostane.

Tímto však není vysvětleno, proč regularizace omezuje přeučení. Za tímto účelem nejdříve předpokládejme jednoduchý dataset pozorování, na kterém budeme konstruovat model. Graf bodů je přiložen na obrázcích 3.8 a 3.9. Hledáme tedy závislost parametru y na x. V grafu je 10 bodů, budeme-li tedy chtít konstruovat naprosto přesný model, lze použít polynomiální funkci devátého stupně (obrázek 3.9). Alternativní možností je zkonstruovat pouze model lineární (obrázek 3.8).

Otázkou však je, který z těchto modelů lépe pozorování generalizuje a dokáže předpovědět hodnoty pro parametry x, které již nebudou součástí pozorovaného datasetu. Na tuto otázku nikdy neexistuje přímá odpověď, může se stát, že závislost opravdu odpovídá polynomiální funkci, na druhou stranu však mohou být body lehce ovlivněné šumem (například přesností pozorovacího zařízení) a závislost y na x bude ve výsledku odpovídat přesně jednoduché lineární funkci. Ač nelze určit, která z funkcí bude predikovat lepší hodnoty, stále je možné na tomto příkladu pozorovat určité zákonitosti. Předpokládejme tedy, že necháme model predikovat hodnotu y pro x, které je daleko od naměřených hodnot z datasetu. Mezi přístupem užití polynomiální funkce devátého stupně a lineární funkce bude v takovém případě obrovský rozdíl, neboť první z modelů bude unášen devátou mocninou, zatímco lineární model bude růst konstantně. Toto pozorování lze jednoduše převést i na neuronové sítě, kde bude mít mnohem větší konsekvence než u těchto dvou modelů. Mějme NN s



Obrázek 3.8: Lineární model.



Obrázek 3.9: Polynomiální model devátého stupně.

regularizovanou účelovou funkcí, taková síť bude mít tendenci se trénovat s malými vahami. Tyto malé váhy znamenají, že se její chování nebude příliš měnit, pokud mírně změníme několik vstupních dat. Jinými slovy, malé odlišnosti ve vstupních datech příliš neovlivní výstup sítě. To ale také znamená, že bude pro tuto síť málo pravděpodobné, že by začala v datech rozpoznávat (učit se) šumy, které obsahují. Naopak, síť se bude převážne učit z jevů, které budou sdíleny ve většině datasetu.

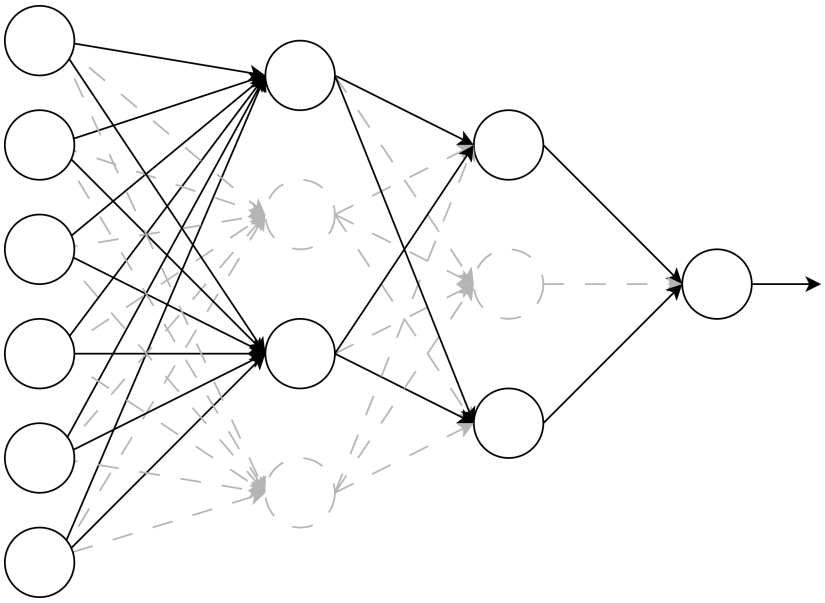
Představíme-li si nyní jinou síť obsahující velmi velké váhy, pak je mnohem pravděpodobnější, že i malý rozdíl ve vstupních datech vyústí ve velkou změnu na výstupu. Tedy taková síť bude mít větší tendenci se učit i velice komplexní a ojedinělé jevy, které ve výsledku nemusí být podstatné (což mimo jiné odpovídá definici přeučení). Vezmeme-li v potaz, jaké problémy jsou nejčastěji řešeny pomocí neuronových sítí, pak je zřejmé, že budeme očekávat schopnost rozeznávat pouze všeobecné jevy, které determinují danou kategorii. Uvedeno na příkladu – neuronová síť rozeznávající jeden typ vozidla (myšleno ve smyslu Škoda Octavia): takové auto může mít různorodé barvy, disky kol, provedení exteriéru, může mít "rakev" na střeše nebo tažné zařízení, pořád ale půjde o jeden typ vozidla, které budeme chtít klasifikovat do stejné kategorie. Na druhou stranu můžeme mít typ problému, kde naopak bude od sítě vyžadováno, aby i menší změny na vstupu vyústily ve větší změnu na výstupu. Příkladem takového problému by mohlo být rozpoznávání jednoho typu mince, která by byla zachycena vždy v identickém prostředí. Ač jsou takovéto typy problémů v oblasti NN výjimečné a jsou spíše řešeny prostřednictvím jiných metod než NN, lze pomocí regularizačního parametru míru regularizace sítě nastavovat. V praxi se tak v závislosti na typu problému a architektury sítě zkouší trénink pro různé regularizační parametry.

3.5 Dropout

Dropuot je další z regularizačních technik, s níž bude experimentováno v druhé části práce. Myšlenka dropoutu je však značně odlišná od ostatních. Chování dropoutu

bude opět ilustrováno na příkladu.

Uvažujme síť z obrázku 1.1, kterou budeme trénovat s užitím techniky dropout. Ta spočívá v dočasném a náhodném vymazání určitého podílu skrytých neuronů. Procento skrytých neuronů, jež budou dropoutem dočasně vymazány, udáváme dropout parametrem z intervalu $[0, 1]$. Na obrázku 3.10 je jedna z možných architektur sítě po aplikaci dropoutu na síť z obrázku 1.1. Trénink potom pokračuje na modifikované



Obrázek 3.10: Ilustrace dropoutu.

síti s menším počtem skrytých neuronů po dobu jednoho mini-batche. Následně se aktualizují váhy přetrvávajících neuronů, smazané se do sítě vrátí a smaže se nový náhodný výběr daného podílu skrytých neuronů. Tento postup se opakuje po celou dobu tréninku. Všimněme si, že při tréninku s dropout technikou se uvažuje pouze daný podíl neuronů, tedy i aktualizace vah a biasů se bude v každém mini-batchi týkat pouze této množiny neuronů. Označme tento podíl jako $1-\rho$, kde ρ tedy značí podíl smazaných neuronů. Po skončení tréninku tak budou váhy a biasy zvětšeny, a tedy po skončení tréninku je třeba všechny váhy a biasy normalizovat násobkem $\frac{1}{\rho}$.

Hlavní myšlenka metody dropout je snadná. Vychází z empirického pozorování, kdy se často k zvýšení přesnosti používá množství různých architektur trénovaných na stejném datasetu a následně se mezi nimi vybírá nebo nějakým způsobem průměruje síť s nejlepšími výsledky. Každá změna v síti znamená, že se nějakým způsobem změní jejich chování, dosáhnou, ač třeba lehce, jiných výsledků a především je možné, že se začnou přeučovat z jiného důvodu, a tedy průměrování mezi různými sítěmi má velkou šanci jev overfittingu omezit. Dropout se snaží tuto techniku používání vícero sítí imitovat v mnohem uživatelsky příjemnější a výpočetně méně náročné podobě. Důvod, proč dropout napomáhá omezovat overfitting, je shrnut citací jednoho z původních článků, které dropout zmiňují: “Tato technika omezuje komplexní koadaptace neuronů, neboť neuron nemůže spoléhat na přítomnost jiných konkrétních neuronů. Je tedy nucen k učení robustnějších jevů, které jsou užitečné ve spojení s

mnoha různými a náhodnými podmnožinami dalších neuronů"[20] (citace přeložena z angličtiny).

3.6 Umělé rozšiřování datasetu

Posledním komentovanou technikou zmírňující jev overfittingu je umělé rozšíření datasetu. Již v kapitole 3.3 bylo řečeno, že rozšiřování datasetu je nejspolehlivějším způsobem na prevenci přeučení, přičemž tato úloha je často časově náročná až nemožná. Nabízí se tak řešení rozšířit dataset uměle. Tato technika je převážně používá na hlubokém učení 4 na obrazových vstupech (možno použít i pro zvukové vstupy), a to různými distorzemi obrazu. Může jít o otočení, radiální, tangenciální, či elastické distorze. Vyspělé knihovny pro hluboké učení mají tuto funkci často implementovanou a je velice jednoduché dataset uměle rozšířit. Je však vždy dobré mít na paměti, že umělé rozšíření nikdy plně nekompenzuje nedostatek dat. Použití této techniky má své místo, pokud disponujeme dostatečně velkým datasetem, poté lze sledovat, zda se přesnost sítě umělým rozšířením dat zlepšila.

3.7 Gradientní sestup se zakomponovanou hybností

Stochastický gradientní sestup je velice účinnou metodou pro minimalizaci účelové funkce mnoha proměnných, nejde ovšem o jediný přístup, který se v neuronových sítích pro minimalizaci účelové funkce používá. Jednoduchou modifikací gradientního sestupu získáme alternativní metodu využívající jakousi hybnost vah (tento modifikovaný přístup bude dále uváděn jako MBGD – zkratka z anglického *momentum-based gradient descent*).

K definování přístupu MBGD je nejdříve potřeba zavést rychlosti vah $v_j \in \mathbb{R}$ pro každou jednotlivou váhu w_j. Potom aktualizační pravidlo vah přístupu MBGD je dáno vztahem

$$v \to v' = \mu v - \eta \nabla C, \tag{3.17}$$

$$w \to w' = w + v', \tag{3.18}$$

kde $\mu \in [0, 1]$ je parametr zvaný koeficient hybnosti, který uvádí, jakou mírou bude hybnost v průběhu tréninku narůstat. Analogicky je pak toto pravidlo aplikováno i na biasy v síti.

Aby bylo pochopitelné, jak zavedení hybnosti funguje, představme si příklad, kdy $\mu = 1$, a kdy se algoritmem MBGD pohybujeme v prostoru směrem dolů. V takovém případě budou každým krokem algoritmu MBGD rychlosti růst, a tedy posun směrem dolů bude v každém kroku větší. Z tohoto důvodu se k lokálnímu minimu dostaneme mnohem rychleji, zároveň ale také platí, že jakmile lokálního minima dosáhneme, tak vlivem nastřádaných rychlostí v parametru v lokální minimum v dalších krocích mineme a budeme pokračovat v daném směru do té doby, dokud se naopak nenastřádají rychlosti v opačném směru. Zjednodušeně si lze algoritmus

MBGD představit jako vhozený míček do mísy, míček bude od okraje mísy nabírat rychlost směrem ke dnu. Jakmile se ale na dno dokutálí, vlivem své hybnosti dno mine a bude určitou chvíli oscilovat, než se nakonec na dně zastaví. Přesně toto chování se algoritmus MBGD snaží imitovat, přičemž parametr μ značí, jak rychle chceme, aby se hybnost kumulovala. Parametr μ si lze na příkladu s míčkem představit jako tření, kdy platí, že čím větší máme tření, tím menší rychlost míček směrem dolů získá, a o to méně mine dno. Primární výhodou algoritmu MBGD oproti gradientním sestupu, za předpokladu vhodně nastaveného koeficientu hybnosti, je možnost urychlit trénink.

3.8 RMSprop

[32]
V neuronových sítích je velice obtížné správně určit správný learning rate, přičemž právě learning rate je hyperparametr, který významně ovlivňuje celý trénink. Účelová funkce je často více citlivá v určitých směrech parametrického prostoru a v jiných citlivá méně. V takovém případě je složitým úkolem pro oba parametry určit společný learning rate. Tím se dostáváme k myšlence, zda není možné learning rate určovat automaticky pro každý parametr zvlášť. Právě tuto myšlenku aplikuje algoritmus s názvem RMSprop (z anglického *root mean square propagation*).

Hlavní myšlenkou této metody je zachovávat klouzavý průměr čtverců gradientů pro každou váhu a následně druhou odmocninou tohoto klouzavého průměru podělit gradient z aktualizačního pravidla vah. Přesněji potom

$$E[g^2] \to E[g^2]' = \beta E[g^2] + (1-\beta)\left(\frac{\partial C}{\partial w}\right)^2 \tag{3.19}$$

$$w \to w' = w - \frac{\eta}{\sqrt{E[g^2]' + \varepsilon}}\frac{\partial C}{\partial w}\,, \tag{3.20}$$

kde $E[g^2]$ je klouzavý průměr čtverců gradientů dané váhy a β je parametr klouzavého průměru (jakási "délka klouzavosti") a ε je šum zabraňující dělení nulou, standardní hodnota parametru β je 0,9.

3.9 ADAM

[32]
ADAM (z anglického *Adaptive Moment Estimation*) je optimalizační metody kombinující MBGD a RMSprop metody dohromady. Využívá čtverců gradientů pro určování rychlosti učení jednotlivých vah a hybnosti, přičemž ta je určována z klouzavého průměru gradientů namísto gradientu jako takového (případ SGDM).

Máme tedy klouzavý průměr čtverců gradientů $E[g^2]$ a moment m, pro něž jsou

definovány aktualizační pravidla

$$E[g^2] \rightarrow E[g^2]' = \beta_2 E[g^2] + (1-\beta_2)\left(\frac{\partial C}{\partial w}\right)^2 \tag{3.21}$$

$$m \rightarrow m' = \beta_1 m + (1-\beta_1)\left(\frac{\partial C}{\partial w}\right), \tag{3.22}$$

kde β_1, β_2 jsou parametry klouzavého průměru respektive koeficient momentu. Aktualizační pravidlo vah je potom zadáno jako

$$w \rightarrow w' = w - \frac{\eta m'}{\sqrt{E[g^2]' + \varepsilon}}, \tag{3.23}$$

kde ε je opět šum zabraňující dělení nulou. Standardními hodnoty jsou 0,9 respektive 0,999. Ze způsobu, jakým je aktualizační pravidlo zavedeno vyplývá, že hodnota, o kterou jsou váhy modifikovány je invariantní na velikost gradientu, což je vítanou vlastností v místech s malými hodnotami gradientu – sedlové oblasti či krátery. Z mých zkušeností konstatuji, že ADAM optimalizace konvergovala ze zmíněných metod nejrychleji.

3.10 Batch normalizace

[37]
Normalizační technika normalizující aktivace v skrytých vrstvách – průběhu sítě. Normalizace probíhá v rámci jednoho mini-batche, odtud název batch normalizace. Efektem zahlazení prostoru účelové funkce a konsekventně i rovnoměrněji soustřednými vrstevnicemi v tomto prostoru je docíleno prediktivního a stabilního chování gradientů, což následně ústí v rychlejší trénink.

Normalizace aktivace jednoho neuronu je řízená průměrem a rozptylem hodnot aktivací tohoto neuronu v rámci jednoho mini-batche. Mějme vážený vstup v l-té vrstvě z_i^l pro všechny vstupy $i \in \mathcal{B} = \{x_{1\ldots m}\}$, kde $\mathcal{B}$ množina mini batchů, potom batch normalizovaný vážený vstup $\hat{z}_i^l$ je dán vztahem

$$\hat{z}_i^l = \gamma \frac{z_i^l - \mu_{\mathcal{B}}}{\sqrt{\mathrm{S}_{\mathcal{B}}^2 + \varepsilon}} + \beta, \quad \text{kde} \tag{3.24}$$

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} z_i^l \qquad \text{a kde} \tag{3.25}$$

$$\mathrm{S}_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}\left(z_i^l - \mu_{\mathcal{B}}\right). \tag{3.26}$$

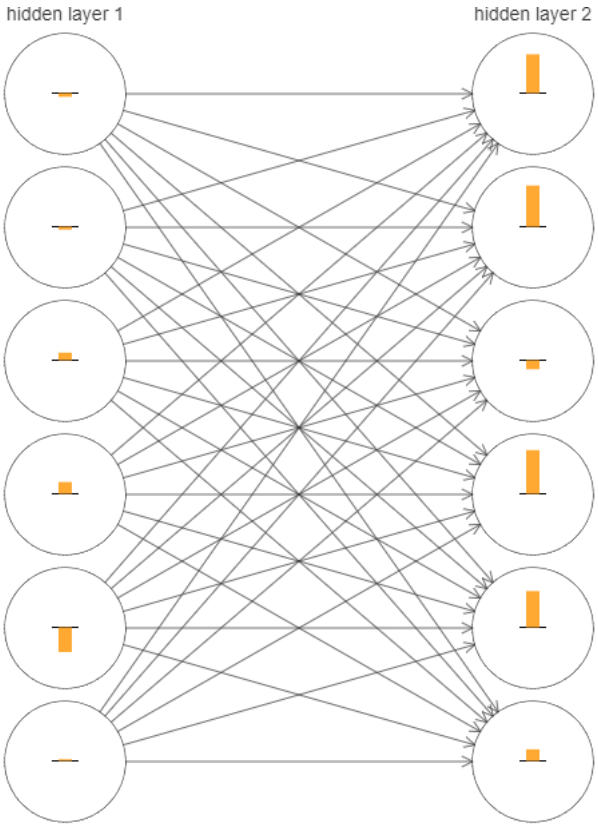
Kde γ a β jsou parametry batch normalizace a ε je šum zabraňující dělení nulou. Parametr β posunuje střední hodnotu a γ směrodatnou odchylku, tedy batch normalizace produkuje hodnoty z rozdělení se střední hodnotu β a směrodatnou odchylkou γ. Navíc jsou oba parametry, analogicky k váhám a biasům, optimalizovány v rámci gradientního sestupu.

batch normalizace znatelně zrychluje průběh učení, snižuje významnost inicializace vah a do jisté míry regularizuje neuronovou síť.

3.11 Problém nestabilního gradientu

Problém nestabilního gradientu je přetrvávajícím důvodem, proč stále nejsou používané gigantické NN s tisíci a miliony vrstev. Nejde ani tolik o fakt, že by trénink trval dlouhou dobu, jako právě o problém nestabilního gradientu, který se vyskytuje v každé hlubší síti.

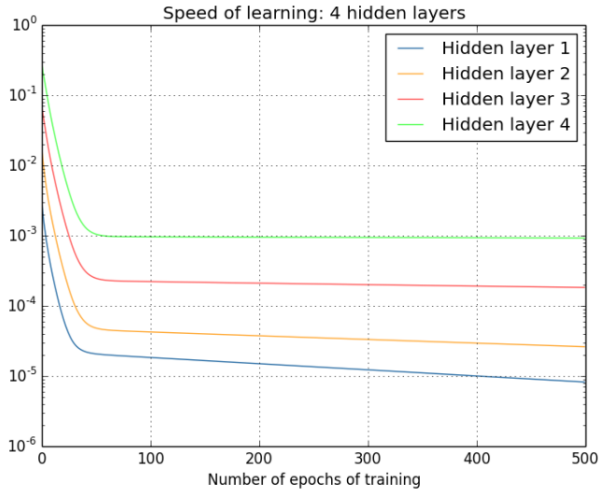
Pozorování nestabilního gradientu ukážu na níže přiloženém příkladu, jenž je vypůjčen ze zdroje [26]. Níže je přiložený graf (obrázek 3.11) neuronové sítě s vyobrazenými velikostmi aktualizací biasů v první a druhé skryté vrstvě po tréninku na prvním mini-batchi. Oranžový sloupec v každém neuronu zobrazuje parametr δ_j^l, přičemž již z algoritmu zpětného šíření chyby víme, že platí $\delta_j^l = \frac{\partial C}{\partial b_j^l}$.



Obrázek 3.11: Ilustrace učení neuronů ve dvou skrytých vrstvách. [26]

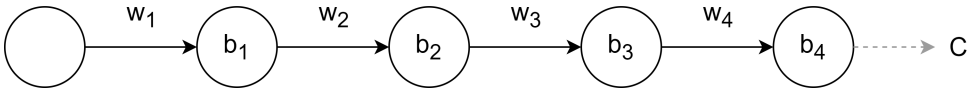
Na první pohled je zřejmé, že velikosti sloupců v neuronech, tedy velikosti gradientů $\frac{\partial C}{\partial b_j^l}$ jsou v první skryté vrstvě podstatně menší než ve druhé, tedy biasy v první vrstvě se učí podstatně pomaleji, než biasy ve vrstvě druhé. Zde poznamenejme, že gradienty vah budou vykazovat velmi podobný jev, neboť jsou závislé na stejném parametru δ_j^l. Přesněji zjišťujeme, že $\|\delta^1\| \approx 0{,}07$ a $\|\delta^2\| \approx 0{,}31$, kde $\|\delta^l\|$ značí velikost vektoru δ^l, který lze přibližně interpretovat jako rychlost učení l-té vrstvy. Neurony z první vrstvy se tedy v tomto příkladu učí zhruba 4krát pomaleji než neurony z vrstvy druhé. Přidáme-li do sítě další dvě skryté vrstvy, tak dostáváme

$\|\delta^1\| \approx 0{,}003\,, \|\delta^2\| \approx 0{,}017\,, \|\delta^3\| \approx 0{,}070\,, \|\delta^4\| \approx 0{,}285$ (graf rychlostí učení čtyř skrytých vrstev v závislosti na délce učení je přiložen na obrázku 3.12). Navíc empiricky víme, že přidáním dalších vrstev klesají gradienty podobným způsobem dále k nule. U velmi hlubokých vrstev je tak gradient velmi blízký 0 a inkriminované váhy a biasy téměř nevykazují žádné učení. Tomuto jevu se říká mizející gradient (anglicky *vanishing gradient*). [21]



Obrázek 3.12: Graf rychlosti učení tří skrytých vrstev.
[26]

Abychom zjistili, proč jev mizejícího gradientu vzniká, vytvořme jednoduchou hlubokou neuronovou síť se třemi vrstvami po jednom neuronu v každé vrstvě. Graf takové sítě je přiložen na obrázku 3.13.

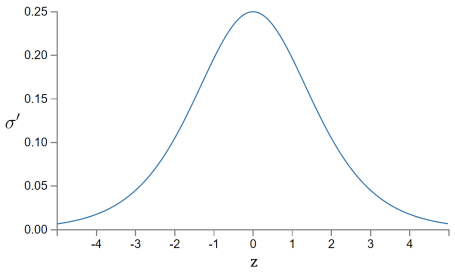


Obrázek 3.13: Graf sítě se třemi skrytými vrstvami po jednom neuronu.

V této síti budeme hledat gradient $\frac{\partial C}{\partial b^1}$. Ten je dle algoritmu zpětného šíření chyby roven

$$\frac{\partial C}{\partial b^1} = \sigma'(z_1) \cdot w_2 \cdot \sigma'(z_2) \cdot w_3 \cdot \sigma'(z_3) \cdot w_4 \cdot \sigma'(z_4) \cdot \frac{\partial C}{\partial a_4}\,, \tag{3.27}$$

kde předpokládáme sigmoidu jako aktivační funkci. Z předpisu je dobře vidět, že krom členu posledního je gradient $\frac{\partial C}{\partial b^1}$ produktem členů $w_j\sigma'(z_j)$. Z minulých kapitol připomeňme na obrázku 3.14 tvar první derivace sigmoidy.



Obrázek 3.14: První derivace sigmoidy.

Maxima dosahuje v bodě 0 a platí $\sigma'(0) = 0{,}25$. Inicializace vah probíhá nejčastěji náhodným určením z normálního rozdělení, po inicializaci bude pro všechny váhy platit $\|w_j\| < 1$, tedy pro většinu vah bude platit $\|w_j\sigma'(z_j)\| < 0{,}25$. Z tohoto pozorování vyplývá, že vztah (3.27) je součinem, v naprosté většině případů, členů v absolutní hodnotě menších než 0,25, a bude tak s přibývajícími členy exponenciálně klesat. Aplikováno na příklad z obrázku 3.13 to znamená, že gradient $\frac{\partial C}{\partial b^1}$ bude asi 64krát menší než gradient $\frac{\partial C}{\partial b^4}$.

V minulém paragrafu jsme okomentovali pouze příklad, kdy se mizející gradient projevoval po inicializaci vah. Parametry sítě se však v průběhu tréninku mohou měnit a je tedy možné, že pokud porostou, nebude se jev mizejícího gradientu projevovat. Na druhou stranu, všimněme si, že pokud váhy porostou v průběhu tréninku tak, aby platilo $\|w_j\sigma'(z_j)\| > 1$, dostaneme ve vzorovém vztahu (3.27) produkt čísel v absolutní hodnotě větších než 1, a tedy gradient $\frac{\partial C}{\partial b^1}$ bude exponenciálně růst. Problému popsaném v poslední větě říkáme explodující gradient.

Ač byla kapitola započata problémem mizejícího gradientu, zakončena byla problémem gradientu explodujícího. Na těchto příkladech je patrné, proč nese kapitola název problém nestabilního gradientu. Problémem tréninku neuronových sítí pomocí gradientního sestupu není přímo ani jeden z problémů, primárně jde o celkově nestabilní gradienty, které se s přibývajícím počtem skrytých vrstev projevují primárně v počátečních vrstvách neuronové sítě, přesněji jejich intenzita sílí směrem počátku sítě.

3.11.1 Poznámky k nestabilním gradientům

Mizející gradient je častějším problémem

Zvláště u sigmoid neuronů se častěji setkáváme s problémem mizejícího gradientu. K zastavení jevu mizejícího gradientu musí platit $\|w_j\sigma'(z_j)\| > 1$. Jediným způsobem, jak tuto nerovnost splnit, je zvětšením váhy w_j. Váhy jsou však i argumentem aktivační funkce, neboť $\sigma(z) = \sigma(\vec{w}\vec{a} + b)$, a tedy i argumentem její derivace. S odkazem na tvar první derivace sigmoidy je zřejmé, že s dostatečně rostoucí váhou (za předpokladu fixního $\vec{a}$, b) bude současně první derivace sigmoidy klesat.

Mizející gradient se projevuje i v komplexních sítích

Předpokládejme síť s mnoha skrytými vrstvami a mnoha neurony v jednotlivých vrstvách. Algoritmus zpětného šíření chyby říká, že gradient v l-té vrstvě je řízen proměnnou δ^l a je dán vztahem

$$\vec{\delta^l} = \Sigma'(z^l)(W^{l+1})^T \Sigma'(z^{l+1})(W^{l+2})^T \dots \Sigma'(z^L)\nabla_{\vec{a}} C\,, \tag{3.28}$$

kde $\Sigma'(z^l)$ je diagonální matice, jejíž prvky jsou hodnoty $\sigma'(z^l)$ a $\nabla_a C$ je vektor parciálních derivací $\frac{\partial C}{\partial a_j^L}$. Situace je tedy podobná jako v případě vrstev s jedním neuronem, parametr $\vec{\delta^l}$ je součinem členů tvaru $(W^j)^T\Sigma'(z^j)$. Navíc matice $\Sigma'(z^j)$ má na diagonále prvky menší než $\frac{1}{4}$. Za předpokladu, že matice vah $(W^j)^T$ nebudou příliš veliké, tak členy $(W^j)^T\Sigma'(z^j)$ povedou k jevu mizejícího gradientu. Nestabilita gradientu je tak analogická s příkladem vrstev s jedním neuronem.

ReLU omezuje problém mizejícího gradientu

ReLU neurony se ukazují jako dobré řešení mizejícího gradientu. Připomeňme tvar funkce ReLU z obrázku 1.7. Zřejmě platí, že první derivace funkce ReLU je konstantní, přesněji rovna 1, pro hodnoty větší než 0. To znamená, že pro kladné argumenty je ReLU rezistentní vůči mizejícímu gradientu. Na druhou stranu je funkce ReLU odpovědná za takzvaný umírající gradient (anglicky *dying gradient*) pro hodnoty menší 0. Pro takové případy se vlivem ReLU neurony zcela "vypnou" a přestanou se učit. Alternativou, která předchází tomuto problému je modifikace zvaná *Leaky ReLU*, přesněji

$$\text{LeakyReLU} = \max(\frac{z}{100}, z)\,. \tag{3.29}$$

ReLU navíc není rezistentní vůči jevu explodujícího gradientu. Omezení tohoto jevu při použití ReLU neuronů je možné změnou inicializace vah a biasů (inicializují se na hodnoty kladné a blízké nule – příkladem je *He inicializace*), použitím regularizačních technik nebo batch normalizace.

Kapitola 4

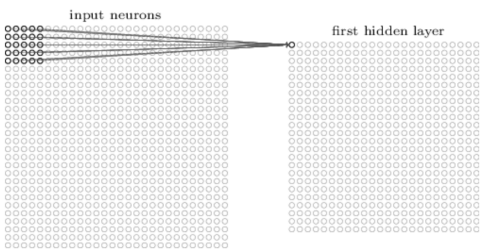
Konvoluční neuronové sítě

V minulých kapitolách byly zmíněny pouze sítě s plně propojenými vrstvami – každý neuron z $(l-1)$-ní vrstvy je propojen z každým neuronem z l-té vrstvy (viz obrázek 1.1). Předpokládejme obrazové vstupy, potom je počet vstupních neuronů v plně propojených vrstvách roven počtu pixelů v obraze. Vezmeme-li v potaz, jaké jevy jsou v obrázcích rozpoznávány (příkladem křivky, hrany), je k zamyšlení, zda je přístup plně propojených vrstev vhodný. Alternativou by bylo použití architektury, která dokáže čerpat informace i z prostorově strukturních jevů, a právě tuto schopnost mají konvoluční neuronové sítě (dále také jako CNN z anglického *convolutional neural network*).

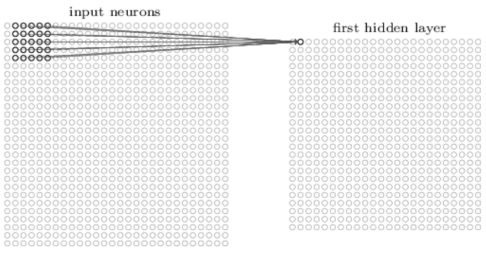
Konvoluční neuronové sítě zpracovávají maticové vstupy a aplikují konvoluce na celé části maticových vstupů. Z tohoto důvodu jsou převážně používány u obrazových vstupů. Fakt, že síť pracuje s maticemi dat, umožňuje rychlejší učení, konstrukci hlubších a přesnějších sítí. Konvoluční neuronové sítě, přesněji pak konvoluční vrstvy, stojí na 3 hlavních myšlenkách – lokální receptivní pole (z anglického *local receptive fields*), sdílení vah a biasů a pooling.

4.1 Lokální receptivní pole

Již bylo řečeno, CNN zpracovávají maticové vstupy, nebylo však upřesněno, jakým způsobem matice prostupují sítí. Právě tento problém je řešený lokálními receptivními poli, kdy se nepropojují jednotlivé pixely s jednotlivými neurony, ale rovnou celé oblasti pixelů. Představme si obrazový vstup o velikostech 25×25 pixelů, potom příkladem lokálního perceptivního pole může být oblast 5×5 pixelů, jejíž jednotlivé pixely budou propojeny s jedním neuronem v další vrstvě. Jestliže máme ve vstupní vrstvě matici o rozměrech 25×25 pixelů, pak bude možné v této matici najít 441 (21 na vertikále, 21 na horizontále) různých receptivních polí o velikosti 5×5. Každý jednotlivý pixel jednotlivého pole bude spojen s jedním neuronem v další vrstvě, jinými slovy v další vrstvě bude matice o velikosti 21×21 neuronů a do každého neuronu povede 25 váhou ohodnocených hran. I zde platí, že každý neuron má svůj bias. Ilustrace tohoto procesu je přiložena na obrázcích 4.1 a 4.2. Výše popsané pole se pohybuje původní maticí pouze o jeden pixel, tomu tak ovšem nemusí



Obrázek 4.1: Ilustrace lokálního receptivního pole – 1. krok.
[26]



Obrázek 4.2: Ilustrace lokálního receptivního pole – 2. krok.
[26]

být. Velikost posunu receptivního pole po matici pixelů z předchozí vrstvy udává parametr, jemuž říkáme stride (v překladu délka kroku). Všimněme si, že zvětšováním délky kroku se bude zmenšovat matice neuronů v další vrstvě. V uvedeném příkladu byla použita délka kroku rovna 1, i přesto se velikost matice neuronů v druhé vrstvě zmenšila. Mohou ale nastat případy, kdy si nebudeme přát, aby matice neuronů v další vrstvě byla menších rozměrů. Tento problém lze jednoduše vyřešit umělým rozšířením vstupní matice do všech směrů nulovými (možno použít i jednotkové) hodnotami (zvětšení nebo také ohraničení se anglicky nazývá *padding*). Zároveň byl použit výraz matice pixelů, to zřejmě platí pouze ve vstupní vrstvě, v dalších vrstvách již jde o maticově uspořádané neurony. Poznamenejme také, že v příkladu je užita jedna vstupní matice, to odpovídá vstupnímu obrazu v odstínech šedi. V případě barevného RGB obrazu je každým vstupem trojice matic.

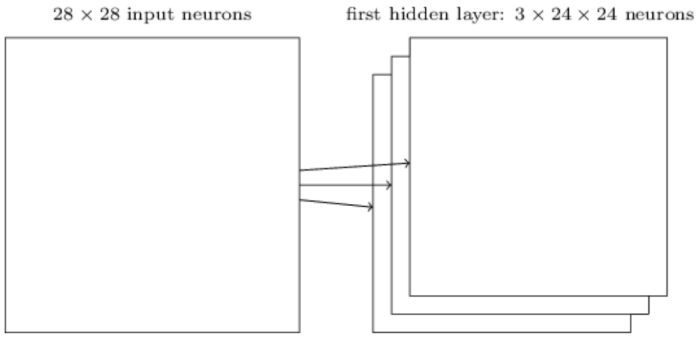
4.2 Sdílení vah a biasů

V minulé sekci bylo uvedeno, že jednotlivé neurony receptivního pole jsou spojeny vahou ohodnocenou hranou s jedním konkrétním neuronem v další vrstvě. Krom toho také navíc platí, že každé receptivní pole je s neuronem z další vrstvy spojeno stejnými vahami a biasem, tedy pro výstup (j,k)-tého neuronu v l-té vrstvě platí

$$\sigma\left(b^l + \sum_j \sum_k W^l_{p,q} a^{l-1}_{j+p,k+q}\right), \tag{4.1}$$

kde b^l je sdílený bias v l-té vrstvě, $W_{p,q}$ je matice sdílených vah v l-té vrstvě, $a^{l-1}_{j+p,k+q}$ je aktivace z $(l-1)$-ní vrstvy a σ je sigmoida, lze však použít jakoukoliv jinou aktivační funkci. Jinými slovy, všechny neurony dané vrstvy detekují stejné jevy, pouze v odlišných místech obrazu v první skryté vrstvě a na odlišných místech v matici aktivací v dalších skrytých vrstvách. Tato vlastnost, mimo jiné, zaručuje CNN invarianci vůči translaci – zkoumaný jev může být kdekoliv v obraze. Matice aktivací se často nazývá anglicky jako *feature map* a matice sdílených vah a sdílený bias jako *kernel* nebo *filter*. Ve většině CNN narazíme na použití vícera kernelů, kde každý detekuje jiné jevy. Lze spatřit konvoluční vrstvy používající desítky až tisíce kernelů.

Velikost kernelů je jedním z hyperparametrů sítě, obvykle se používají velikosti sahající od 1×1 po 15×15, vždy se však jedná čtvercové kernely a jejich délka v obou směrech je vždy lichým číslem (z důvodu aliasing problémů). Na obrázku 4.3 je ilustrace vrstvy používající 3 kernely o velikosti 5×5, velikostí kroku 1 a bez užití ohraničení. Použití sdílených vah a biasů má krom schopnosti invariance vůči



Obrázek 4.3: Ilustrace konvoluční vrstvy s třemi kernely.
[26]

translaci další zásadní konsekvence, a těmi jsou rychlost učení, částečná resistence vůči overfittingu a nestabilitě gradientu. Předpokládejme použití kernelu velikosti 5×5, potom pro jednu feature mapu v jedné vrstvě trénujeme pouhých 25 vah a 1 bias. Kdybychom uvažovali například 30 feature map, pak pro danou vrstvu máme celkem 750 vah a 30 biasů k učení. Oproti tomu v případě plně propojené vrstvy s feature mapou o velikosti 25×25 a 30 skrytými neurony ve vrstvě další bychom dostali dohromady $25 \times 25 \times 30 = 18750$ vah a dalších 30 biasů. Takové srovnání sice není validní, neboť jsou tyto architektury z důvodu principiálních rozdílu nesrovnatelné, avšak poskytuje dobrý vhled na efektivitu učení konvolučních sítí. Dále je tu částečná resistence konvolučních vrstev vůči overfittingu, která opět pramení z užití sdílených vah. Stejné váhy v jedné vrstvě totiž znamenají, že jsou kernely nucené se učit jevy napříč celou vstupní maticí, a tak je méně pravděpodobné, že by se naučily detekovat malé a ojedinělé jevy. V neposlední řadě jsou při učení konvoluční vrstvy více rezistentní vůči nestabilnímu gradientu, a to jednoduše proto, že ve srovnání s plně propojenými vrstvami obvykle pro stejnou míru komplexity sítě obsahují méně parametrů, kde počet parametrů je u nestabilního gradientu primárním faktorem.

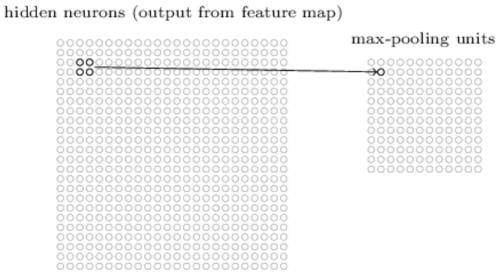
Nastává otázka, jak feature mapy v různých vrstvách sítě vypadají a co zobrazují. Bohužel ale jde o nic neříkající hodnoty v intervalu $[0, 1]$, jež lze interpretovat jako intenzitu. Po zobrazení půjde jen o množinu více či méně tmavých pixelů. Porozumět, co CNN v danou chvíli detekuje, je náročná disciplína, která je v současné době předmětem vědeckého zkoumání. [40]

4.3 Pooling

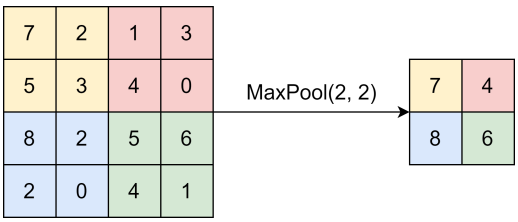
Posledním výrazným členem CNN jsou pooling vrstvy. Ty jsou často používány bezprostředně po konvolučních vrstvách a slouží k zjednodušení a "zhuštění" informací

získaných konvolucemi. Přesněji, pooling vrstva přebírá aktivaci z vrstvy předchozí a na oblastech s předem definovanou velikostí sumarizuje jednou ze sumarizačních funkcí hodnoty v oblasti do hodnoty jediné. Jedna z neznámějších sumarizačních strategií se nazývá maxpooling, která z dané oblasti vrátí maximální hodnotu. Podobně jako u konvolučních vrstev, i pooling prochází aktivačními maticemi (připomeňme, že kernelů bývá více) z předešlé vrstvy v krocích předdefinované délky, dokud neprojde všemi hodnotami. Případně lze použít na vstupní matici také ohraničení. Maxpooling o velikosti 5×5 s velikostí kroku 1 a bez užití ohraničení je ilustrován na obrázku 4.4, detailnější pohled pak na obrázku 4.5.

Alternativní poolingové strategie zahrnují například average pooling (průměrování), L2 pooling (odmocnina součtu čtverců) nebo globální max nebo average pooling. Je zřejmé, že matice na výstupu pooling vrstvy bude vždy menších rozměrů než na vstupu. V praktickém užití CNN se vstupní matice postupně průchodem sítí zmenšují v rozměrech. Na tento jev lze nahlížet jako na zhušťování a detekování stále obecnějších a komplexnejších informací, přičemž také dochází k redukci parametrů. Po průchodu konvolučními vrstvami tak není neobvyklé spatřit feature mapu o jednotkách či desítkách hodnot. Obvykle se pak na takovou feature mapu na konci neuronové sítě aplikuje před vyčíslením výsledků plně propojená vrstva, která propojí každou kategorii na výstupu s každým neuronem z poslední skryté vrstvy. Z tohoto faktu také vyplývá, že se u konvolučních vrstev většinou nepoužívá technika dropoutu.



Obrázek 4.4: Ilustrace MaxPoolingu. [26]



Obrázek 4.5: Ilustrace MaxPoolingu – detailní pohled.

Krom klasických poolingových metod zmíněných výše existují také globální poolingové metody, přesněji jde o *global average pooling* – GAP a o *global max pooling* – GMP. Global average pooling je často alternativou pro plně propojenou vrstvu v konvolučních neuronových sítích, neboť mění maticové feature mapy ve vektor. Myšlenka této metody spočívá v předpokladu, že počet feature map na výstupu poslední konvoluční vrstvy odpovídá počtu kategorií ve vrstvě klasifikační. V takové případě lze na feature mapy aplikovat GAP, který spočte průměr všech hodnot každé feature mapy a následně tento vektor hodnot přiřadí k vektoru klasifikací (opomenuta byla softmax vrstva, která je, za účelem jednoduché normalizace hodnot, téměř vždy vložena mezi GAP či GMP a klasifikaci). Krom GAP metody se používá i GMP, která místo průměru bere z feature mapy maximum. Zjevným rozdílem global pooling metod oproti plně propojené vrstvě je absence učících se parametrů a tedy zna-

telný pokles počtu parametrů v sítí. Mimo jiné může použití GAP či GMP vyústit ve větší robustnost vůči prostorovým translacím. V případě, kdy plně propojenou vrstvu vyměníme za globální poolingové metody, musí počet feature map odpovídat počtu kategorií na výstupu sítě. Globalní poolingové metody jsou často využívány při problémech lokalizace objektů v obraze.

Kapitola 5

Dataset

Tato kapitola je věnována datasetu. Ač to nemusí být na první pohled zřejmé, tvorba vhodného datasetu je velice zdlouhavá činnost, která si v mém případě vyžádala zhruba třetinu času věnovaného celé práci. I z tohoto důvodu jsem se rozhodl věnoval datasetu celou kapitolu. Popsána bude struktura datasetu, podmínky kladené na dataset a způsoby jeho získávání. Připomeňme, že obsahem datasetu mají být konkrétní typy automobilů, čímž bude rozuměno, že například Škoda Octavia je jedním konkrétním typem kategorie.

5.1 Podmínky

V první kapitole byl dataset uveden jako primární faktor ovlivňující jev přeučení. Pokud tedy mají být cílem práce konstrukce a trénink neuronových sítí dosahujících velmi vysokou přesnost, je nutnou podmínkou vytvořit kvalitní dataset. Otázkou tedy je, jaké vlastnosti datasetu jsou potřebné k tomu, aby byl prohlášen za kvalitní.

Počet obrázků v jednotlivých kategoriích je jistě jednou z hlavních podmínek. Jistě platí, že čím více dat, tím lépe, na druhou stranu, benefity s rostoucím počtem dat jsou v určitou chvíli zastřeny časovou náročností jejich získávání. Ze zkušeností mohu konstatovat, že pro můj případ datasetu roste od určitého počtu dat časová náročnost pro získání každého dalšího obrázku neúměrně rychle. Je tak rozumné najít jakési ekvilibrium mezi časovou náročností získání jednoho obrázku a benefitu, který pramení z potlačení přeučení. Empiricky stanovený cíl byl, po konzulaci s vedoucí práce, určen na 5 000 až 10 000 obrázků v každé kategorii. Stanoveno dále bylo minimálně 5 kategorií s tím, že konečný počet bude záviset na časových možnostech. Každá kategorie by ideálně měla obsahovat přibližně stejný počet dat a v neposlední řadě by měl být minimalizován počet duplicit.

Dalším důležitým faktorem je, řekněme, rozmanitost dat. Vztaženo na obrázky automobilů by se mělo jednat o následující vlastnosti:

- Klasifikace se bude týkat pouze exteriéru automobilů, mezi daty se tedy nesmí nacházet obrázky interiérů.

- Automobil by měl být zachycen z co možná nejvíce směrů, nechtěné jsou jen obrázky z ptačí perspektivy.
- Obrázky mohou zobrazovat celý nebo jen část automobilu, vždy by se ale měl nacházet alespoň na přibližně 30 % záběru.
- Problém generačních proměn a faceliftů je nutno řešit úsudkem, je-li o několik generací starší automobil výrazně odlišný, neměly by tyto generace být zakomponovávány do dat. Ze zkušeností se tento milník nejčastěji nachází někde okolo přelomu milénia. Většina automobilů v datasetu je tedy zhruba z posledních 20 let.
- Automobil může být různých barev, různých disků kol nebo mít exteriérové modifikace ve smyslu sportovních exteriérových paketů, střešní rakve, taxíkářské úpravy, tažného zařízení.
- Vyloučeny budou těžce bourané vozy.
- Snímek může zobrazovat více automobilů, a to i jiných typů.
- Důraz bude kladen na rozmanitost prostředí, ve kterém se automobil nachází. Nemělo by se tedy jednat pouze o novinářské a inzertní fotografie, ale i o snímky z reálného provozu v reálném a různorodém prostředí – město, dálnice, příroda a jiné.
- Kategorie by měly obsahovat jak automobily odlišné, tak i velmi podobné. Přesněji je cílem mít mezi kategoriemi automobilů zástupce malých, středních, sportovních nebo SUV vozů, přičemž alespoň některé z těchto skupin by měly mít mezi kategoriemi vícero zástupců. Důvodem je následné porovnávání úspěšnosti neuronových sítí v závislosti na podobnosti automobilů.

5.2 Metody získávání datasetu

Jestliže je cílem mít 5 000 až 10 000 snímků v každé kategorii, je zřejmé, že ruční stahování z inzertních stránek není zcela vhodnou metodou. Abych byl schopen takové množství obrázků na internetu najít a stáhnout, byla potřeba nějaká forma automatizace.

Nejjednodušším možným způsobem se zdá být využití již existujících datasetů. V případě automobilů přichází v úvahu jako jeden z mála pouze *Stanford car dataset* [19]. V oboru neuronových sítí se jedná o známý dataset s více jak 500 citacemi, který pro mnoho výzkumů slouží jako obecně uznávaný dataset pro srovnávání přesnosti sítí. Bohužel se však jedná o dataset s mnoha kategoriemi, kde v každé z nich je zhruba 60 snímků, což neodpovídá požadavkům na dataset konstruovaný pro tuto práci. Externích datasetů jsem tedy pro svoji práci nevyužil.

Použita naopak byla forma automatizace hledání a ukládání dat z internetu zvaná web-scraping. Pro tyto účely byl použit jazyk Python. Celkem šlo o 4 metody web-scrapingu, jejichž kombinací jsem dosáhl požadované představy o velikosti a rozmanitosti datasetu.

5.3 Scraping Google obrázků

Využití nejznámějšího "*search-enginu*" je zcela automatická volba. Problémem ovšem je, že vyhledávač Google je rezistentní vůči skriptům, které se snaží přistoupit k jeho vyhledávacím službám. Jediným způsobem, jak docílit scrapingu Google stránek, je zřízením účtu ve službě *Google Cloud Console* (dříve *Google Developers Console*) a aktivací produktu *Custom Search API* [3]. Tato služba je sice placená, ale nabízí trial verzi na prvních 90 dnů zdarma. Dále je možné využít knihovnu *Google-Images-Search* (v době psaní práce šlo o verzi 1.4.6), pomocí klíče vygenerovaného pro *Custom Search API* lze potom tuto knihovnu využít pro prohledávání a ukládání výsledků z vyhledávače obrázků.

Zde však nastává další problém, a sice že pro jedno vyhledávání (ve smyslu Škoda Octavia) vyhledávač v naprosté většině případů, ač často proklamuje miliardy nalezených výsledků, nenajde více jak 200 obrázků, s průměrem kolem 140 obrázků na vyhledávání. Tento jev lze pozorovat i přístupem přes webové prohlížeče, kde po chvilce listování velice jednoduše narazíme poslední výsledek. Jediným možným řešením je tak formulovat nejrůznější vyhledávací dotazy a obrázky následně pomocí skriptu přejmenovat a všechny přesunout do jediné složky. Z mé zkušenosti lze při použití 100 různých dotazů nalézt a uložit okolo 15 000 obrázků. Níže přikládám některé z používaných dotazů.

VW Golf 8 DSG	VW Golf 8 active	VW Golf 8 4motion	VW Golf VIII R-line	VW Golf 2019
VW Golf 8 bazar	VW Golf 8 tipcars	VW Golf VIII review	VW Golf VIII test	VW Golf VIII press
VW Golf 8 zu verkaufen	VW Golf 8 prodej	VW Golf VIII skladove vozy	VW Golf VIII grey	VW Golf VIII white
VW Golf VIII gebrauchtwagen	VW Golf VIII Auto Basar	VW Golf VIII artikel	VW Golf VIII bildchen	VW Golf 8 a vendre
VW Golf 8 aufbewahrungswagen	VW Golf 8 bazos.cz	VW Golf 8 road	VW Golf 8 camion di stoccagio	VW Golf 8 vendita

Tabulka 5.1: Příklady dotazů pro vyhledávání VW Golf.

Na příkladu z tabulky 5.1 lze vysledovat, jak jsem formuloval dotazy, za účelem maximalizace počtu a různorodosti výsledků. Jmenovitě jde o následující pozorování.

- Použití cizích jazyků zvyšuje šanci hledání zahraničních výsledků.
- Použití různých specifikací – barva, typ motoru, typ karoserie, speciální edice a jiné. Dobrým zdrojem informací o specifikacích daného automobilu jsou konfigurátory na oficiálních stránkách značky automobilu.
- Použití typických slovních spojení pro daný automobil. Příkladem může být Fiat 500, který je velmi populárním vozem v Itálii, lze tedy využít vyhledávání v italském jazyce, zmínit italská města či provincie. Jiné automobily jsou preferované třeba v Asii nebo Americe, pro takové je možné najít jiná typická hesla v příslušném jazyce.

- Použití slovních spojení typických pro prodej aut. Může jít o slovní spojení typická pro nové i použité vozy. Velice často totiž prodejní stránky obsahují velké množství fotografií.
- Použití slovních spojení typických pro novinářské články či propagační materiály.
- V neposlední řadě lze zkusit dotazy zmiňující prostředí, ve kterém se má automobil na obrázku nacházet. Příkladem mohou být slova jako "dálnice", "město", "příroda" a mnoho dalších. Možné je i vyhledávání takových hesel v cizích jazycích. Typicky jsou tyto vyhledávací dotazy méně účinné, na druhou stranu často nabízejí unikátní snímky automobilu z poptávaného prostředí.

5.4 Scraping Bing obrázků

Scraping obrázků z vyhledávače Bing je o něco snažší než v případě Google obrázků. Za tímto účelem jsem využil knihovnu *bing-image-downloader* (v době psaní práce šlo o verzi 1.1.2) [34]. Opět se zde projevuje stejný problém jako u vyhledávače Google, a sice že počet nalezených obrázků pro jeden dotaz se pohybuje kolem 150. Dalším problémem byla samotná knihovna, ve které jedna z funkcí obsahovala jako vstupní parametr počet obrázků, které si uživatel přeje najít, přičemž pokud vyčerpala veškeré zdroje z vyhledávače Bing, pokračovala dále v hledání "na prázdno". Často se však stává, že pro některé dotazy Bing vyhledávač najde i 200 zdrojů, pro jiné naopak třeba jen 40, což znamenalo, že nastavením požadovaného množství obrázků na, řekněme, 140 se skript velmi často zasekl na hledání zdrojů, které již Bing nevracel. V knihovně jsem tedy funkci modifikoval a přidal parametr času. Po zadané době neúspěšného hledání se funkce ukončí. Velmi pozitivní je naopak možnost použít identickou množinu dotazů jako v minulém případě. Podobně jako v případě Google obrázků lze pomocí vyhledávače Bing při použití 100 různých dotazů nalézt a uložit okolo 15 000 obrázků.

5.5 Scraping sociální sítě Pinterest

Pinterest je sociální síť umožňující jejím uživatelům vytvářet tématická alba a kolekce obrázků. Jedná se tak o zdroj s velkým potenciálem k nalezení uživatelských fotek z reálného prostředí, jež jsou málokdy k nalezení prostřednictvím klasických webových vyhledávačů. K usnadnění vyhledávání a ukládání obrázku z Pinterestu lze opět využít knihovnu, v tomto případě knihovnu nazývající se *Pinscrape* (v době psaní práce šlo o verzi 3.0.3) [27]. Počet nalezených fotek na Pinterestu je silně závislý na vyhledávaných dotazech, obecnější dotazy mají z mé zkušenosti lepší šanci nalézt větší počet obrázků, naopak pro specializovanější a komplexnější dotazy často nenajde jediný obrázek. Při použití stejné množiny dotazů jako tomu bylo v případě posledních dvou technik lze z Pinterestu získat mezi 3 000 až 6 000 obrázky.

5.6 Scraping inzertních stránek

Poslední používanou technikou je scraping inzertních webových stránek. Bazarové stránky jsou zdrojem tisíců obrázků velmi dobře kategorizovaných automobilů. Velmi často jsou automobily nafoceny detailně z různorodých úhlů. Je tedy poměrně výhodné tento zdroj využít. Opět zde ovšem platí, že převážná část nejznámějších inzertních stránek je chráněná před strojovým přístupem, a to zejména prostřednictvím ochrany proti automatizovanému prohlížení *reCAPTCHA* (společnosti *Google*). Nejenom, že je přinejmenším velmi obtížné tuto ochranu obejít, ale primárně by se jednalo o jednání v rozporu se zásadami a podmínkami dané stránky. Možností by bylo použití oficiální API knihovny (z anglického *application programming interface*), ty ovšem v případě inzertních stránek buď neexistují, nebo se jedná o placenou službu. Jediným východiskem je tedy hledat takovou stránku, která není chráněná proti strojovému přístupu, a zároveň se nejedná o úkon proti zásadám užívání stránky. Takovou inzertní platformou se pro mě stal web `sauto.cz` společnosti *Seznam*.

K vyhledávání a ukládání fotek ze stránek `sauto.cz` jsem použil knihovnu *Scrapy* (v době psaní práce se jednalo o verzi 2.9.0) [42]. Pomocí této knihovny lze vytvořit takzvaného *spidera*, což je třída popisující, jak se má daná stránka nebo skupina stránek skenovat a parsovat, případně které URL odkazy následovat. K extrakci dat z HTML stránky se používají selektory, které podle CSS nebo XPath výrazů na stránce najdou a extrahují požadované informace.

Ve své práci jsem použil dva spidery. Jeden slouží k vyhledání požadovaného typu automobilu na stránkách `sbazar.cz` a uložení URL odkazů na všechny nabídky. Po otevření první stránky výsledků se pomocí CSS a XPath výrazů naleznou a uloží nabídky. Jakmile jsou všechny nabídky nalezeny a uloženy, tak se obdobným způsobem extrahuje URL odkaz na další stranu. Stejným způsobem potom probíhá prohledávání dalších stránek do té doby, než skript nenarazí na stránku poslední. Druhý spider otevře a přečte JSON soubor, v němž jsou uloženy URL adresy všech nabídek extrahované prvním spiderem. Iterativně tyto stránky otevírá a opět užitím CSS a XPath výrazů extrahuje adresy všech obrázků v daném inzerátu, ze kterých ihned poté obrázky ukládá.

Oproti 3 výše zmíněným technikám je scraping inzertních stránek jednoznačně nejrychlejší. například, nalezení a stažení všech obrázků ze všech zhruba 800 nabídek vozu *VW Golf* trvalo jen pár jednotek minut, zatímco scraping *Google* či *Bing* obrázků zabere i několik hodin (čas je vztažen k mým výpočetním podmínkám). Počet obrázků, jež lze touto metodou získat, je primárně ovlivněn typem hledaného vozu. Intuitivně bude počet nabídek závislý na počtu prodaných kusů za posledních přibližně 20 let. Například, dlouhodobě nejprodávanější vůz v ČR – Škoda Octavia – se vyskytuje na zhruba 7 000 nabídkách, oproti tomu vůz dodávkového typu – Mercedes-Benz V – je předmětem pouhých asi 300 nabídek. V závislosti na typu automobilu je tak scrapingem stránek `sauto.cz` možné získat rozličné množství obrázků, přesněji od pár desítek až po více jak 50 000 v případě Škody Octavie. Jedním z parametrů pro výběr typu automobilu do datasetu tak byla informace o počtu prodaných kusů za poslední dekádu. Tímto způsobem výběru pak bylo možné z inzertní

stránky `sauto.cz` získat okolo 12 000 až 15 000 obrázků.

5.7 Eliminace duplicitních snímků

Již víme, že kombinací výše zmíněných metod lze získat okolo 50 000 obrázků pro každý jednotlivý typ automobilu. Tyto snímky však ani vzdáleně nelze prohlásit za připravené k použití ve finálním datasetu. Prvním řešeným problémem jsou snímky duplicitní.

Z teorie neuronových sítí vyplývá, že několik duplicitních snímků by nemělo znatelně ovlivnit trénink sítě. Jak se ale brzy ukázalo, čtyřmi výše zmíněnými metodami se v nepřetříděných obrázcích nacházejí i vyšší stovky jednoho totožného snímku, což už by mohlo mít negativní vliv při tréninku sítě. Všechny kategorie jsem se tak rozhodl zbavit duplicit.

Hrubou silou (metoda "brutal-force") by bylo velmi implementačně jednoduché srovnávat obrázky pixel po pixelu a následně mazat ty, které jsou identické. Toto řešení má však dva výrazné problémy. Prvně, srovnání pixel po pixelu by znamenalo, že jakákoliv malá změna ve světelnosti, sytosti barev, rozlišení, či oříznutí obrázku by obrázky v očích této metody odlišovala, ač v pojetí neuronových sítích se jeví jako duplikáty. Druhým problémem je časová náročnost: srovnat každý obrázek s každým na vzorku s 50 000 obrázky o velikostech průměrně zhruba 500×500 pixelů jednoduše není reálné.

Tímto se dostáváme k metodě, jež byla v této práci použita pro eliminaci duplicit, a sice hashovací algoritmy určené pro obrazová data. K tomuto účelu byla použita knihovna *ImageHash* (v době psaní práce šlo o verzi 4.3.1) [2]. Knihovna nabízí vícero obrazových hashovacích algoritmů. Dle výsledků článků [6] [41] jsem se rozhodl použít hashovací metodu *Difference hashing* (dHash). Nejdříve je obrázek převeden do nebarevného spektra odstínů šedi, poté je lineární interpolací zmenšen na předem daný rozměr. Z mé zkušenosti nejlepších výsledků dosahovalo zmenšení na 5×5 pixelů, jedná se o dobrý kompromis mezi obecnou přesností a tolerancí malých změn v obrázku. Dále se na zmenšeném obrázku, na doporučení autorů knihovny, provede Z-transformace. Poté se již aplikuje samotná hashovací funkce, která v případě dHash metody a rozměrů 5×5 pixelů postupuje dle následující trojice kroků:

1. Pro každý řádek se určí odpovídající hash kód. Algoritmus se posouvá v řádku pixel po pixelu, kde v každém pixelu zjišťuje, zda je hodnota sousedícího pixelu větší nebo rovna. Pokud tomu tak je, vrátí algoritmus hodnotu 1, v opačném případě vrátí 0. Pro každý řádek 5 pixelů se tak vytvoří 4 bity kódu.

2. Stejným způsobem jako v předešlém kroku se určí hash kód pro každý sloupec pixelů (algoritmus postupuje odshora dolu).

3. V posledním kroku se sloučí hash kódy z řádků a sloupců do jednoho. Pro vstup o rozměrech 5×5 tak vzniká hash kód délky 32 bitů.

Tímto zahashováním obrázků se razantně zjednodušila úloha srovnání 500×500 pixelů na úlohu srovnání 32bitového kódu. Krom toho budou jako duplicity brány i velmi podobné obrázky. I přesto však srovnání 50 000 32bitových kódů stylem "každý s každým" na mé výpočetní technice trvalo i několik desítek hodin. Další optimalizací je tedy využití datové struktury jazyka Python *set*, která využívá stromové datové struktury. Operace, která hledá daný hash kód v datové struktuře *set*, má tak konstantní složitost. Navíc je tato operace prováděna automaticky při přidávání prvků do datové struktury *set*. Tímto způsobem lze eliminaci duplicit na vzorku o 50 000 fotek provést v řádu minut.

Na souboru obrázků získaných výše zmíněnými web scraping technikami se po eliminaci duplicit jejich počet snížil přibližně na polovinu. Z typických 50 000 obrázků jich tak po tomto kroku zbývá zhruba 25 000.

5.8 Filtrace exteriérů automobilů

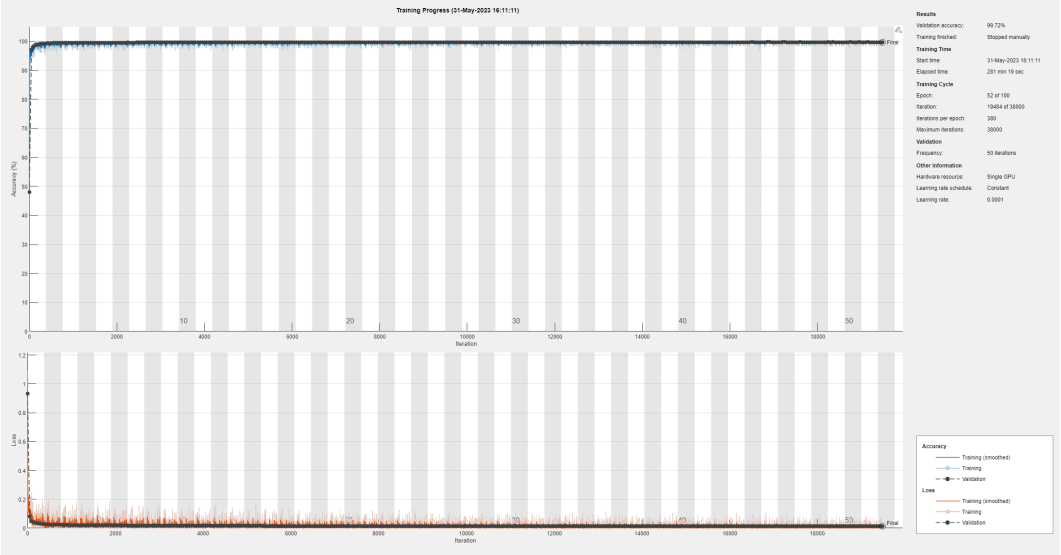
Vlivem web scrapingových metod bylo nechtěně uloženo mnoho obrázků neodpovídajících zadání datasetu. Připomeňme, že hledáme pouze exteriéry předem určených automobilů. Mezi uloženými snímky však často bývají interiéry, náhradní součástky, pneumatiky nebo jiné předměty se souvislostí s hledaným vozem. Tyto snímky lze samozřejmě vyfiltrovat ručně, ovšem procházení 25 000 fotek je časově náročná a nezáživná činnost, která by mohla být automatizována. Z tohoto důvodu jsem se rozhodl natrénovat konvoluční neuronovou síť na binární výstup – jedná se o exteriér automobilu, či nikoliv?

K tomuto úkolu jsem se rozhodl využít před-trénované sítě, neboli *transfer learning* (viz kapitola 6.1.1). Manuálně jsem vytvořil dataset o 20 275 snímků exteriérů automobilů a 3 901 snímků ostatních. Jako před-trénovanou konvoluční neuronovou síť jsem použil *GoogLeNet* (viz kapitola 6.1.3). Hyperparametry tréninku byly nastaveny následovně:

- Dataset – náhodně rozdělen na trénovací, testovací a validační data v poměru $80 : 10 : 10$.
- Metoda hledání minima – algoritmus MBGD.
- Learning rate – konstatní s hodnotou 0,0001.
- Zastavovací kritérium – 100 epoch.
- Momentum – 0,9.
- L2 regularizace – 0,0001.
- Velikost mini-batche – 128.
- Exekuční prostředí – *GPU NVIDIA GTX A5500 Mobile*.

Přesnost na testovacích datech	99,67 %
Chyba na testovacích datech	0,0142

Tabulka 5.2: Výsledky tréninku sítě GoogLeNet klasifikující exteriéry automobilů.



Obrázek 5.1: Graf tréninku sítě GoogLeNet klasifikující exteriéry automobilů.

Výsledky tréninku jsou zaneseny v tabulce 5.2 a příslušný graf na obrázku 5.1.

Využitím transfer learningu se podařilo rychle natrénovat síť (z důvodu stagnace chybovosti byl trénink ukončen manuálně již po 52 epochách), která s přesností 99,67 % dokáže rozpoznat exteriéry automobilů. Manuální činnost zabírající několik hodin se tímto způsobem zautomatizovala a zkrátila na několik minut. Typický soubor obrázků po eliminaci duplicit čítající okolo 25 000 snímků se filtrací exteriérů redukuje opět přibližně na polovinu – na výsledných 12 000 až 15 000 obrázků.

5.9 Manuální kontrola

Z přibližných 50 000 uložených obrázků se eliminací duplicit a filtrací exteriérů počet obrázků redukuje na přibližně 12 000 až 15 000. Posledním krokem je manuální kontrola, kde jsou odstraněny snímky neodpovídající podmínkám datasetu, typicky se jedná o snímky jiného typu, či snímky havarovaných vozidel. Finální dataset pro jedinou kategorii tak typicky obsahuje mezi 7 000 až 12 000 snímky.

Vybrané snímky z datasetu jsou přiloženy na obrázcích 5.2, 5.3, 5.4, 5.5, 5.6 a 5.7, konkrétně jde o vůz Škoda Octavia. Diagram aktivit pro tvorbu datasetu je na obrázku 5.8.

5.10 Struktura datasetu

V tabulce 5.3 jsou přiloženy počty snímků v každé kategorii a detailní popis začleněných modelů. Typy *VW Golf 8* a *VW Golf 5-7* jsou záměrně rozdělené na dvě kategorie, a to ze 2 důvodů:

1. Lze sledovat, jak ovlivní klasifikaci, pokud bude ve dvou kategoriích přibližně polovina obrázků než v ostatních. Tyto dvě kategorie lze následně spojit do jedné a porovnávat výsledky.

2. Automobily v těchto 2 kategoriích jsou si velmi podobné, jde o stejný typ, ale jiné generace. Lze tak sledovat, jak budou sítě vyhodnocovat právě tyto dvě kategorie, zda bude mezi nimi více chybných klasifikaci oproti ostatním či nikoliv.

typ	Počet obrázků	Modely
VW Golf 8	3 795	Hatchback, Combi
VW Golf 5-7	4 715	Hatchback, Combi
Škoda Octavia	9 291	2. až 4. generace, Liftback, Combi
Škoda Karoq	13 013	Doposud existuje jen jeden model
Toyota Corolla	7 166	E120, E130, E140, E150, E160, E170, E180, E210; Hatchback, Sedan, Estate, Verso
Fiat 500	7 890	2007 až současnost; veškeré modely
Dacia Duster	10 984	1. i 2. generace; veškeré modely včetně Renault Duster a Nissan Terrano
BMW řady 3	6 783	E90 - E93, F30 - F35, G20, G21, G28; včetně M modelů
Mercedec-Benz V-class	6 275	2. a 3. generace; včetně modelů Vito, Viano, EQV a Marco Polo

Tabulka 5.3: Struktura datasetu.

Obrázek 5.2: Škoda Octavia – ilustrační snímek 1.

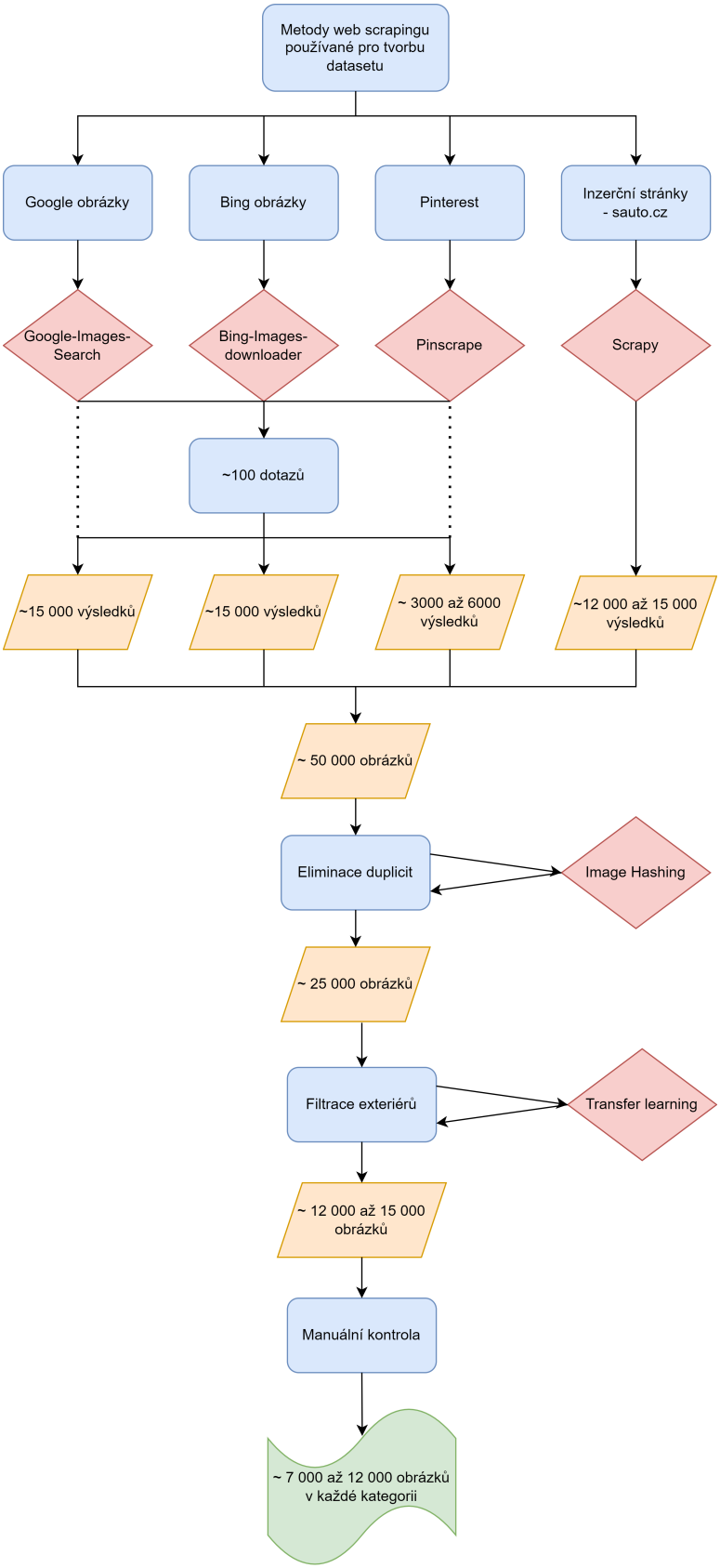
Obrázek 5.3: Škoda Octavia – ilustrační snímek 2.

Obrázek 5.4: Škoda Octavia – ilustrační snímek 3.

Obrázek 5.5: Škoda Octavia – ilustrační snímek 4.

Obrázek 5.6: Škoda Octavia – ilustrační snímek 5.

Obrázek 5.7: Škoda Octavia – ilustrační snímek 6.



Obrázek 5.8: Diagram aktivit pro tvorbu datasetu (počet obrázků je uveden pro jednu kategorii).

Kapitola 6

Architektury používaných konvolučních neuronových sítí

Pro konstrukci, trénink a prezentaci výsledků byl použit jazyk a vývojové prostředí MATLAB.

6.1 Předtrénované CNN

6.1.1 Transfer learning

Transfer learning je přístup tréninku konvolučních neuronových sítí, kdy se využívá již existujících a předtrénovaných sítí, které následně přetrénují na jiný, ač podobný, problém. Využití transfer learningu je typicky mnohem jednodušší a rychlejší metodou než konstrukce a trénink sítě zcela nové. Mezi hlavní výhody transfer learningu patří:

- Trénink sítí na menším datasetu, neboť parametry používaných sítí již byly natrénovány na často velkém množství podobných dat – v případě sítě GoogLeNet šlo o více jak milion obrázků [36].
- Rychlost učení, neboť váhy a biasy se nebudou během tréninku drasticky měnit.
- Využití velmi kvalitních architektur. Sítě jako GoogLeNet nebo ResNet byly vytvořeny vědeckou komunitou v oboru *deep learningu*. Uživatel tedy nemusí být profesionálem v oboru, aby dosáhl velmi vysoké přesnosti. Mimo jiné jde o znatelnou časovou úsporu absencí konstrukce architektury a optimalizace hyperparametrů.

Předtrénované sítě budou typicky obsahovat jinak nastavených posledních pár vrstev. Před zahájením tréninku je proto nutné vyměnit klasifikační a případnou plně

propojenou vrstvu a nové nakonfigurovat na počet kategorií daného problému. Důležité je také prozkoumat vstupní vrstvy, kde bude nastavená fixní velkost vstupů, zamýšlená data je nutné před tréninkem na tuto velikost modifikovat. [38]

6.1.2 AlexNet

Konvoluční neuronová síť AlexNet [20] byla představena v souvislosti s *LSVRC2012 - ImageNet Large-Scale Visual Recognition Challenge 2012* – každoročně pořádaná mezinárodní soutěž, kde účastníci testují své CNN na datasetu *ImageNet*. *ImageNet* je celosvětově uznávaný dataset čítající 1000 kategorií a více než 14 miliónů obrázků [4]. Tuto soutěž AlexNet pro rok 2012 vyhrál.

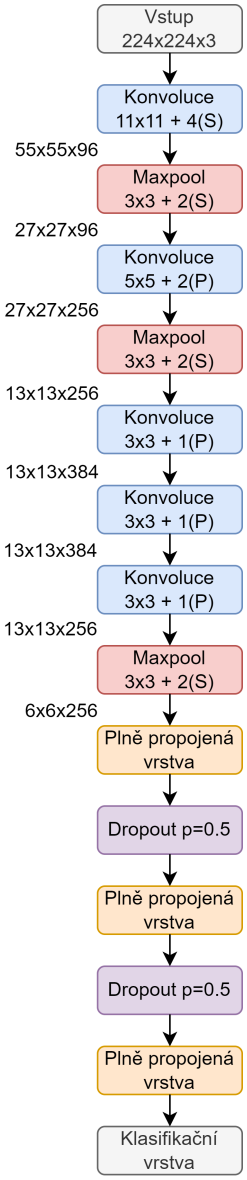
AlexNet obsahuje 5 konvolučních vrstev a 3 plně propojené vrstvy. Za každou konvoluční vrstvou, krom té poslední, je použita ReLU aktivační vrstva a za poslední konvolucí se nachází Softmax vrstva. AlexNet byla mimo jiné první velkou sítí prezentující výhody užití aktivačních funkcí ReLU. Celkem má tato síť více jak 60 miliónů parametrů, což jí vystavuje velké tendenci k přeučení. K zmírnění tohoto jevu byla využita metoda umělého rozšíření datasetu (přesněji šlo o translaci, horizontální reflexi a modifikace barevného spektra) a dvě dropout vrstvy nacházející se za první a druhou plně propojenou vrstvou. Z důvodů pramenících z užití aktivační funkce ReLU byla zavedena *LRN – local response normalization*.

Obor hodnot ReLU funkce není oproti sigmoidní nebo tanh funkci shora omezený. Z tohoto důvodu bylo nutné zavést normalizaci hodnot v aktivaci. Mimoto byla právě LRN použita k podnícení efektu laterální inhibice – schopnost neuronu snižovat aktivitu neuronů sousedních. V původním článku se uvádí, že laterální inhibice "vytváří konkurenci pro aktivace vysokých hodnot mezi výstupy neuronů vypočtených pomocí různých kernelů" [20] (citace přeložena z angličtiny). LRN je dána vztahem

$$\overline{a^i_{x,y}} = a^i_{x,y} \,/\, \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} \left(a^j_{x,y}\right)^2 \right)^{\beta}, \tag{6.1}$$

kde $a^i_{x,y}$ je aktivace z i-té feature mapy v bodě (x, y), $\overline{a^i_{x,y}}$ je analogická aktivace po normalizaci, n je hyperparametr udávající počet sousedů, které se při normalizaci budou brát v potaz, N je celkový počet feature map – hloubka, k je hyperparametr zabraňující dělení nulou, α je normalizační konstanta a β je takzvaná kontrastní konstanta. AlexNet byl trénován s hyperparametry LRN zadanými jako $(k, \alpha, \beta, n) = (0, 1, 1, N)$, přičemž takto zadaným hyperparametrům LRN říkáme standardní normalizace.

Náčrt pseudo-architektury sítě AlexNet je přiložen na obrázku 6.1.



Obrázek 6.1: Pseudo-architektura konvoluční neuronové sítě AlexNet.

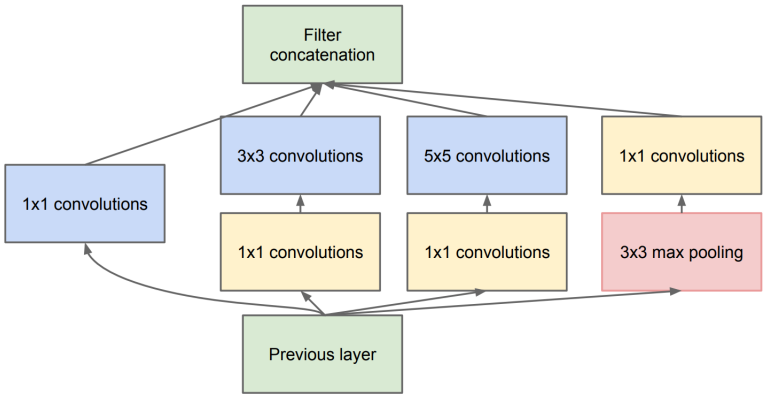
6.1.3 GoogLeNet

Takzvaná "inception"konvoluční neuronová síť GoogLeNet [36] vznikla v roce 2014 týmem vědců pracujících ve společnosti *Google*. Představena byla opět v souvislosti se soutěží LSVRC, tentokrát pro 2014, přičemž tento ročník GoogLeNet vyhrál. Tým vědců rozšířil a aplikoval myšlenky z článku *Network in network* [23], který se jako první, v návaznosti na slavné internetové meme “we need to go deeper” [39] zmiňuje o architektuře s názvem Inception.

Obrázek 6.2: "We need to go deeper" meme z filmu Inception. [39]

Inception CNN je hluboká neuronová síť obsahující několik inception modulů. Každý inception modul (obrázek 6.3) se skládá z následujících vrstev:

- 1×1 konvoluční vrstva.
- 1×1 & 3×3 konvoluční vrstvy.
- 1×1 & 5×5 konvoluční vrstvy.
- Max pooling vrstva & 1×1 konvoluční vrstva.
- "Concatenation" vrstva.



Obrázek 6.3: Inception modul. [36]

V každém bodě se nachází 1×1 konvoluční vrstva, jejíž hlavní funkce je redukce dimenze dat. Zřejmě 1×1 konvoluce není schopna rozpoznávání prostorově strukturních jevů, využívá se k redukci počtu feature map na vstupu, aniž by se informace z těchto feature map ztratily – zhušťuje vícero hodnot do jedné. Snížením počtu feature map však snižuje výpočetní náročnost, přičemž se učí jevy v daném bodě

přes všechny feature mapy. 1×1 konvoluční vrstva je do jisté míry podobná pooling vrstvám – pooling vrstvy redukují velikost obrazu, kdežto 1×1 konvoluční vrstvy snižují počet feature map.

V druhém a třetím bodě jsou konvoluční vrstvy s velikostí kernelů 5×5 a 5×5. Podobně jako u nerozvrstvených[1] CNN se tyto konvoluce učí prostorově strukturní jevy na odlišných velikostech receptivního pole. Výhodou modůlu inception je kombinace tří různých velikostí kernelů. Touto technikou se tak eliminuje problém optimalizace velikostí kernelů v jednotlivých vrstvách. Podobně je tomu i v případě maxpoolingu, který se provádí v inception modulu souběžně se ostatními zmíněnými konvolucemi. Všimněme si, že parametry operace maxpool jsou nastaveny tak, aby nezmenšovaly rozměry feature map (přesněji velikost kroku 1 a velikost ohraničení 1).

Posledním bodem je poté takzvaná "concatenation" vrstva, ta využívá faktu, že velikosti feature map jsou ze všech 4 vrstev inception modulu stejné a lze je tak všechny sloučit v jeden výstup čítající počet feature map rovný součtu feature map všech 4 vrstev inception modulu[2]. Typicky se celkový počet feature map v síti směrem k výstupní (potažmo klasifikační) vrstvě zvětšuje, i v tomto případě tomu tak je. Totiž platí, že počet feature map na výstupu inception modulu je větší než na vstupu.

Neokomentovanou částí je důvod použití 1×1 konvolučních vrstev v každé větvi inception modulu. Odůvodnění budu prezentovat na příkladu. Předpokládejme konvoluční vrstvu 5×5 z obrázku 6.4 s 32 kernely a velikostí kroku 1. Dále předpokládejme vstup o rozměrech $28 \times 28 \times 192$, kde poslední rozměr značí počet feature map (hloubku). V takovém případě můžeme jednoduše dojít počtu multiplikačních operací (označme jej M) potřebných pro průchod touto konvoluční vrstvou

$$M = (28 \times 28 \times 192) \times (5 \times 5) \times 32 = 120\,244\,400\,. \tag{6.2}$$

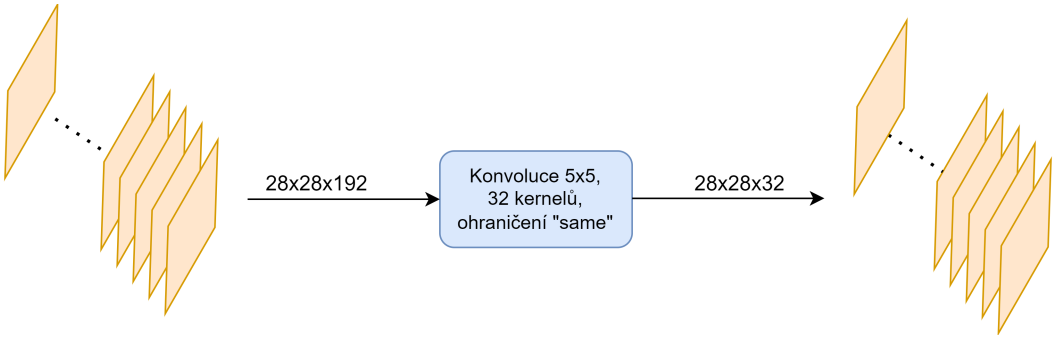
Tedy více jak 120 miliónů operací pouze pro jedinou konvoluční vrstvu. Právě z důvodu snížení výpočetní složitosti jsou před konvoluce vloženy redukční 1×1 konvoluce. Použijeme-li výše uvedený příklad s přidáním $1 \times 1 \times 16$ konvoluční vrstvy, dostáváme pro počet multiplikačních operací

$$M = (28 \times 28 \times 192) \times (1 \times 1) \times 16 + (28 \times 28 \times 16) \times (5 \times 5) \times 32 = 12\ \ 443\ \ 648\,. \tag{6.3}$$

V tomto případě je vypočtení náročnost oproti předchozímu případu přibližně desetinásobně nižší. Z tohoto velmi dobrého důvodu a v návaznosti na velkém množství konvolucí v GoogLeNet síti jsou v každém inception modulu zavedeny 1×1 redukční konvoluce.

[1] Rozvrstvenou CNN je myšlena architektura, kde každá vrstva je napojena pouze na jednu vrstvu další.

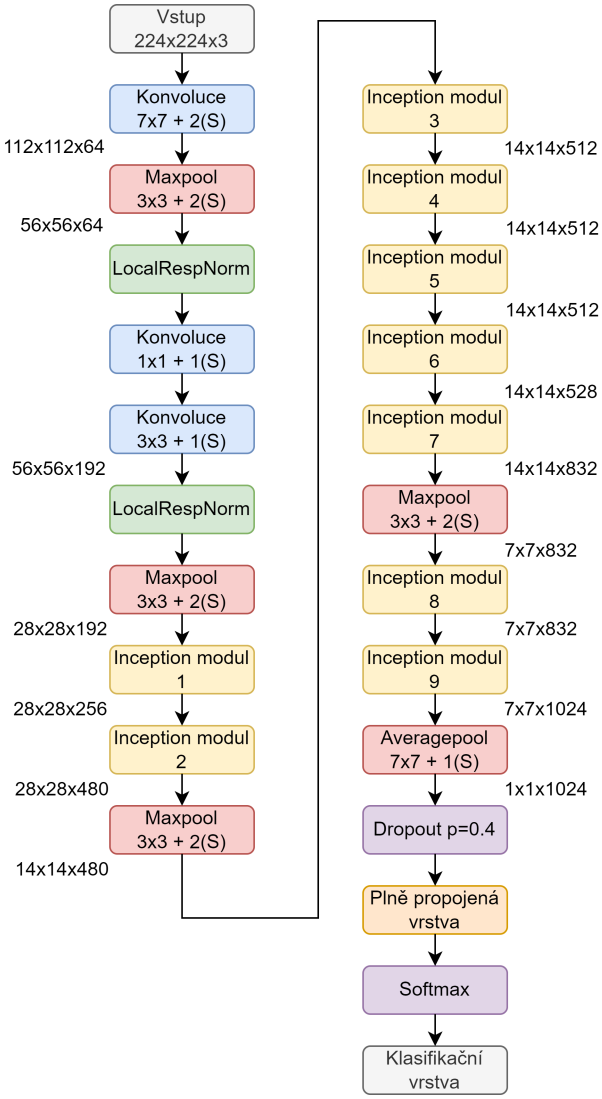
[2] 4 vrstvami inception modulu jsou myšleny ony 3 větvě konvolučních vrstev a jedna větev maxpool vrstvy.



Obrázek 6.4: Ilustrační 5×5 konvoluce.

Povšimněme, že před maxpooling vrstvou redukční 1×1 konvoluce chybí a naopak je za ní. Maxpooling totiž neprovádí takové množství multiplikačních operací, přesněji neprovádí vůbec žádnou multiplikaci. Je tedy výpočetně mnohem méně náročný a není potřeba počet feature map před maxpool vrstvou redukovat. Redukční 1×1 konvoluce je umístěna za maxpooling vrstvou proto, aby počet feature map na jeho výstupu nebyl příliš vysoký ve srovnání s počtem feature map 3 zmíněných konvolucí v inception modulu.

GoogLeNet je sestaven z 9 takovýchto inception modulů, náčrt pseudo-architektury je přiložen na obrázku 6.5. Podobně jako AlexNet, i GoogLeNet používá LRN vrstvy.



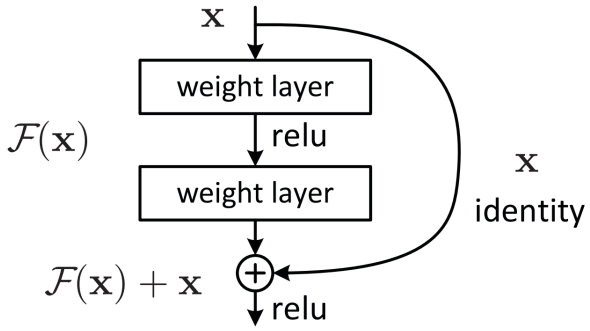
Obrázek 6.5: Pseudo-architektura konvoluční neuronové sítě GoogLeNet.

6.1.4 ResNet50

ResNet50 [11] se řadí mezi takzvané reziduální konvoluční neuronové sítě. Publikován byl v roce 2015 týmem vědců z programu *Microsoft Research* a hned v témže roce dosáhla síť několika úspěchů včetně prvního místa v LSVRC soutěže pro rok 2015. Hlavním cílem sítě ResNet byla redukce jevu degradace gradientu, jako hlavní překážkou v konstrukci velmi hlubokých CNN. Toho bylo dosaženo zavedením reziduálních bloků. Touto metodou je následně možné zkonstruovat desítky, či stovky vrstev hluboké CNN a využít tak jejich komplexitu k dosažení lepších výsledků.

reziduální blok je v jiném slova smyslu zavedení spojů nesoucí identitu a obcházející jednu nebo několik vrstev, aby se následně tato identita přidala do výstupu z přeskočených vrstev. Spojení nesoucí identitu a obcházející určitý počet vrstev se často říká *skip connections*. Ilustrace reziduálního bloku převzata z původního článku, která zobrazuje skip connection přes dvě konvoluční vrstvy, je přiložena na

obrázku 6.6. Z kapitoly 3.11 víme, že aplikací algoritmu zpětného šíření chyby roste



Obrázek 6.6: Ilustrace reziduálního bloku. [11]

potenciál pro mizející, či explodující gradienty s rostoucím počtem parametrů. Protože přidáním každé další vrstvy roste počet parametrů exponenciálně, je zřejmé, že hlubší sítě mají vetší tendenci projevu těchto jevů. V článku je komentováno, že problém mizejícího, či explodujícího gradientu je často řešen normalizací různých forem, zároveň je však v článku na reálných datech ukázáno, že velmi hluboké sítě nepřinášejí tížené zlepšení přesnosti a snížení chyby i když je normalizace různých forem v síti použita. Toto zjištění vede k zaměření na problém degradace gradientu. Ač není zcela zřejmé, co degradaci gradientu způsobuje, je na reálných datech ukázáno, že tento jev přetrvává bez ohledu na použití normalizačních technik. Mimoto je v článku ukázáno, že obohacení mělké sítě o identitní mapování, a tím pádem konstrukci hluboké sítě vyústí neintuitivně v horší výsledky než použití pouze té mělké sítě. Tímto je spekulováno, že za snížení výkonosti hlubokých sítí může právě jejich hloubka způsobující jev degradace gradientu. Článek tedy prezentuje nový způsob zmírnění degradace gradientu právě reziduálními bloky.

Myšlenkou metody reziduálních sítí je přimět vrstvy, aby se místo učení mapování podstaty (v článku *underlying mapping*) začaly učit reziduální mapování (v článku *residual mapping*). Jinými slovy, nechť $H(x)$ je funkce kterou se snaží fittovat určité množství vrstev, potom budeme chtít přimět tyto vrstvy, aby místo fittování funkce $H(x)$ začaly fittovat funkci $F(x)$ zadanou jako

$$F(x) := H(x) - x\,, \tag{6.4}$$

což dává

$$H(x) := F(x) + x\,. \tag{6.5}$$

$F(x)$ je tedy forma reziduální funkce - rozdíl mezi zamýšleným výsledkem a vstupem. Předpokládejme reziduální blok z obrázku 6.6, potom pro aktivaci na výstupu reziduálního bloku platí

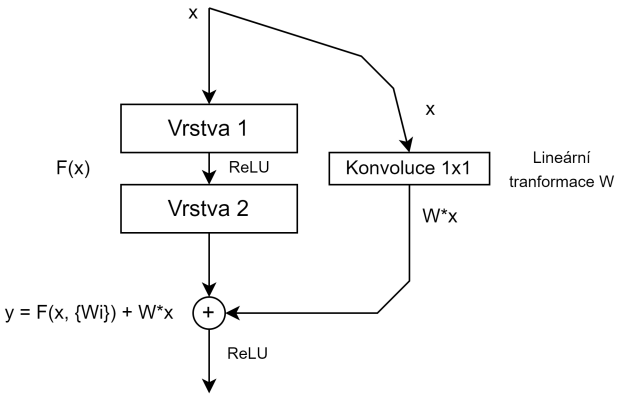
$$A^{l+2}_{outResid} = \mathrm{ReLU}\left(z^{l+2} + A^{l}\right)\,, \tag{6.6}$$

$$z^{l+2} = W^{l+2}A^{l+2} + b^{l+2}\,, \tag{6.7}$$

kde A^l je ona identita, která obchází dvě konvoluční vrstvy, z^{l+2} jsou naučené váhy a biasy z reziduálního bloku, A^{l+2} značí aktivaci před přidáním identity a

$A^{l+2}_{out_resid}$ naopak aktivaci po přidání identity. Zřejmě platí, že pokud $z^{l+2} \to 0$, potom $A^{l+2}_{outResid} = \text{ReLU}\left(A^l\right)$, přičemž pokud $A > 0$, pak $A^{l+2}_{outResid} = A^l$, což odpovídá identitnímu mapování. Skip connections tedy zachovávají výstupy z předchozích vrstev a beze změny je přičítají do výstupu o několik vrstev dále.

Výše zmíněný příklad platí jen v případě, kdy rozměry vstupů do reziduálního bloku odpovídají rozměrům výstupu z vrstev uvnitř reziduálního bloku. Článek však zavádí i metodu pro případ, kdy se počet feature map uvnitř reziduálního bloku změní. V takové případě se do skip connection větvě reziuálního bloku přidá lineární transformace v podobě 1×1 konvoluce (popsané v sekci 6.1.3) viz obrázek 6.7. Velice



Obrázek 6.7: Ilustrace reziduálního bloku s lineární transformací

jednoduše jde ukázat, že reziduální bloky do určité míry potlačují jev mizejícího gradientu. Zde je však nutné dodat, že mizející gradient není podstata problému degradujícího gradientu. Fakt, že reziduální sítě mírní jev mizejícího gradientu neimplikuje zmírnění jevu degradujícího gradientu, panuje pouze hypotéza pramenící z empirických testů, že reziduální bloky do jisté míry jev degradujícího gradientu omezují. Předpokládejme síť reziduálních bloků bez aktivačních funkcí mezi bloky. Potom vstup do reziduálního bloku $l + 1$ je dán vztahem

$$z^{l+1} = F(z^l) + z^l \,. \tag{6.8}$$

Pro další reziduální bloky tedy platí

$$z^{l+2} = F(z^{l+1}) + z^{l+1} = F(z^{l+1}) + F(z^l) + z^l \,, \tag{6.9}$$

$$\implies z^L = z^l + \sum_{i=l}^{L-1} F(z^i) \,. \tag{6.10}$$

Pro gradienty počítané v rámci algoritmu zpětného šíření chyby tedy dostáváme

$$\frac{\partial C}{\partial z^l} = \frac{\partial C}{\partial z^L}\frac{\partial z^L}{\partial z^l} \tag{6.11}$$

$$= \frac{\partial C}{\partial z^L}\left(1 + \frac{\partial}{z^l}\sum_{i=l}^{L-1} F(z^i)\right) \tag{6.12}$$

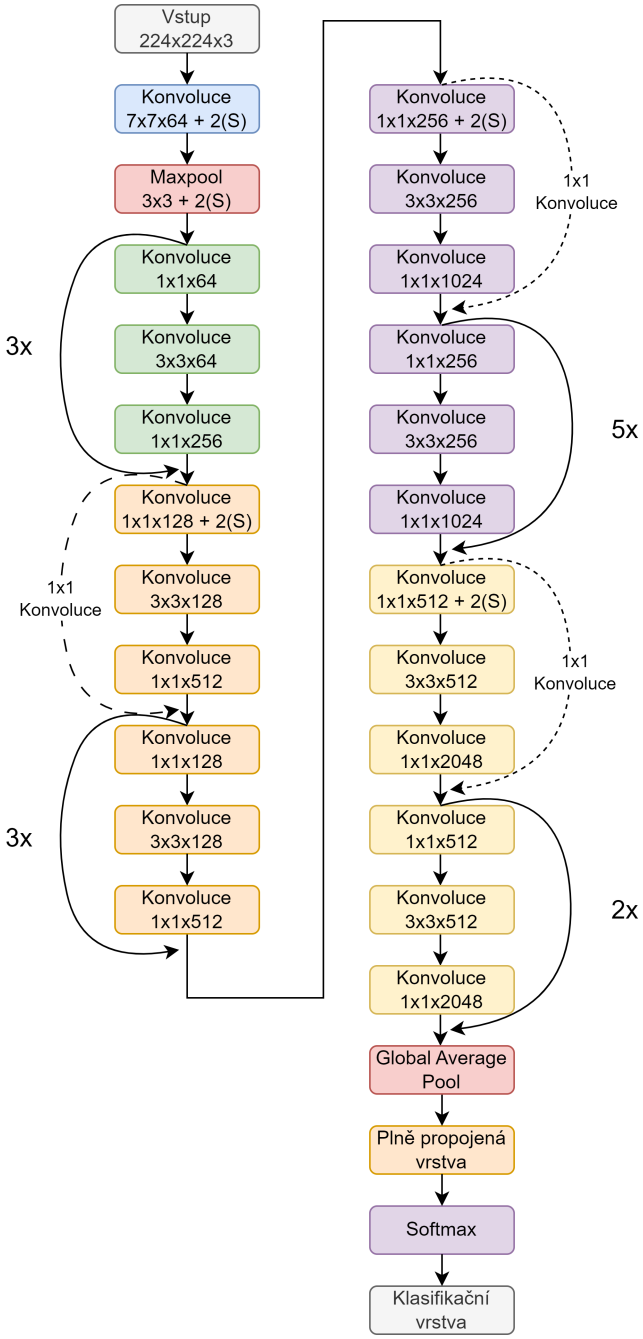
$$= \frac{\partial C}{\partial z^L} + \frac{\partial C}{\partial z^L}\frac{\partial}{\partial z^l}\sum_{i=l}^{L-1} F(z^i) \,, \tag{6.13}$$

což ukazuje, že gradient $\frac{\partial C}{\partial z^l}$ se bude vždy skládat i z gradientů předchozích vrstev $\frac{\partial C}{\partial z^L}$ (předchozí ve smyslu zpětného průchodu). Tedy pokud by gradienty funkcí $F(z^i)$ klesaly k nule, a tedy mizely, nevyústí to v mizející gradient $\frac{\partial C}{\partial z^l}$.

ResNet50 se skládá z následujících vrstev (Vynechány jsou ReLU aktivace a batch normalizace, které se vyskytují po konvolucích):

- 7×7 konvoluce s 64 kernely a velikostí kroku 2.
- Maxpool vrstva s velikostí kroku 2.
- 3 reziduální bloky následujících konvolucí:
  - Konvoluce 1×1 s 64 kernely.
  - Konvoluce 3×3 s 64 kernely.
  - Konvoluce 1×1 s 256 kernely.
- 4 reziduální bloky následujících konvolucí:
  - Konvoluce 1×1 s 128 kernely.
  - Konvoluce 3×3 s 128 kernely.
  - Konvoluce 1×1 s 512 kernely.
- 6 reziduálních bloků následujících konvolucí:
  - Konvoluce 1×1 s 256 kernely.
  - Konvoluce 3×3 s 256 kernely.
  - Konvoluce 1×1 s 1024 kernely.
- 3 reziduální bloky následujících konvolucí:
  - Konvoluce 1×1 s 512 kernely.
  - Konvoluce 3×3 s 512 kernely.
  - Konvoluce 1×1 s 2048 kernely.
- Global Average Pool vrstva.
- Plně propojená vrstva.
- Softmax vrstva.

Pseudoarchitektura sítě ResNet50 je přiložena na obrázku 6.8.



Obrázek 6.8: Pseudo-architektura konvoluční neuronové sítě ResNet50.

6.1.5 SqueezeNet

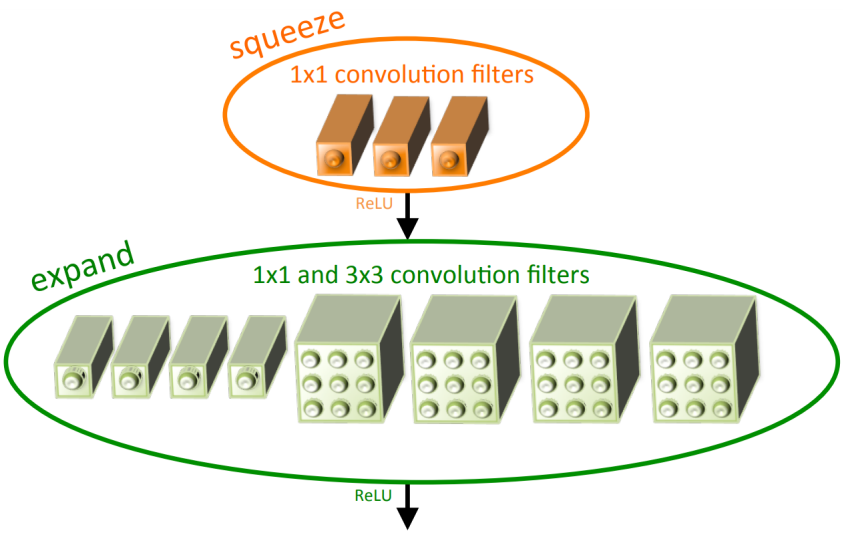
SqueezeNet[15] je konvoluční neuronová síť vyvinuta na univerzitách UC Berkeley a Stanford v roce 2016. Cílem tohoto výzkumu bylo zkonstruovat síť velmi dobrých kvalit a současně minimalizovat velikost sítě ve smyslu místa, které zabírá na disku. Přímo v názvu původního článku je uvedeno, že SqueezeNet síť dosahuje přesnosti podobné AlexNet sítě, přičemž obsahuje 50x méně parametrů s velikostí modelu menší jak 0.5MB, což je asi 510x méně než velikost sítě AlexNet. Autoři uvádějí 3 hlavní výhody malých neuronových sítí:

- Efektivnější distribuovaný trénink sítě. Komunikace mezi servery je limitujícím faktorem škálovatelnosti distribuovaného tréninku, kde komunikační režie mezi servery je přímo úměrná počtu parametrů sítě.
- Méně dat při rozesílání nově natrénovaných sítí klientům. Příkladem - dnešní automobily často používají takzvané *over-the-air updates*, tedy nové verze softwaru, které jsou do počítače v autě posílány přes datové služby. Menší sítě tak mohou benefitovat z rychlejšího přenosu do automobilu, čímž pádem se mohou odesílat i menší, ač třeba zásadní bezpečnostní vylepšení častěji.
- Možnost nasazení na FPGA, které často nemají více, jak 10MB paměti.

K snížení celkové velikosti sítě využívá SqueezeNet 3 metody:

- Nahrazení kernelů velikost 3×3 kernely o velikosti 1×1, neboť kernel velikosti 1×1 má 9x méně parametrů oproti kernelu 3×3.
- Snížení počtu feature map na vsutpu konvolucí s kernely velikosti 3×3. K tomuto jsou v síti zavedeny takzvané squeeze vrstvy.
- Výrazné zmenšování rozměrů aktivačních map koncentrované až ke konci sítě. Malé rozměry aktivačních map mohou vést k horším výsledkům sítě (usuzování z empirického sledování). Záměr je tedy provádět konvoluce na větších aktivačních mapách a výrazné zmenšování (anglicky *downsampling*) koncentrovat spíše ke konci sítě. Zachování velikostí aktivačních map je jednoduše dosaženo užitím konvolucí s velikostí kroku 1.

Stavebním kamenem SqueeezeNetu jsou *fire* moduly obsahující *squeeze* a *expand* konvoluční vrstvy (viz obrázek 6.9 z původního článku [15]). *Squeeze* vrstvami jsou

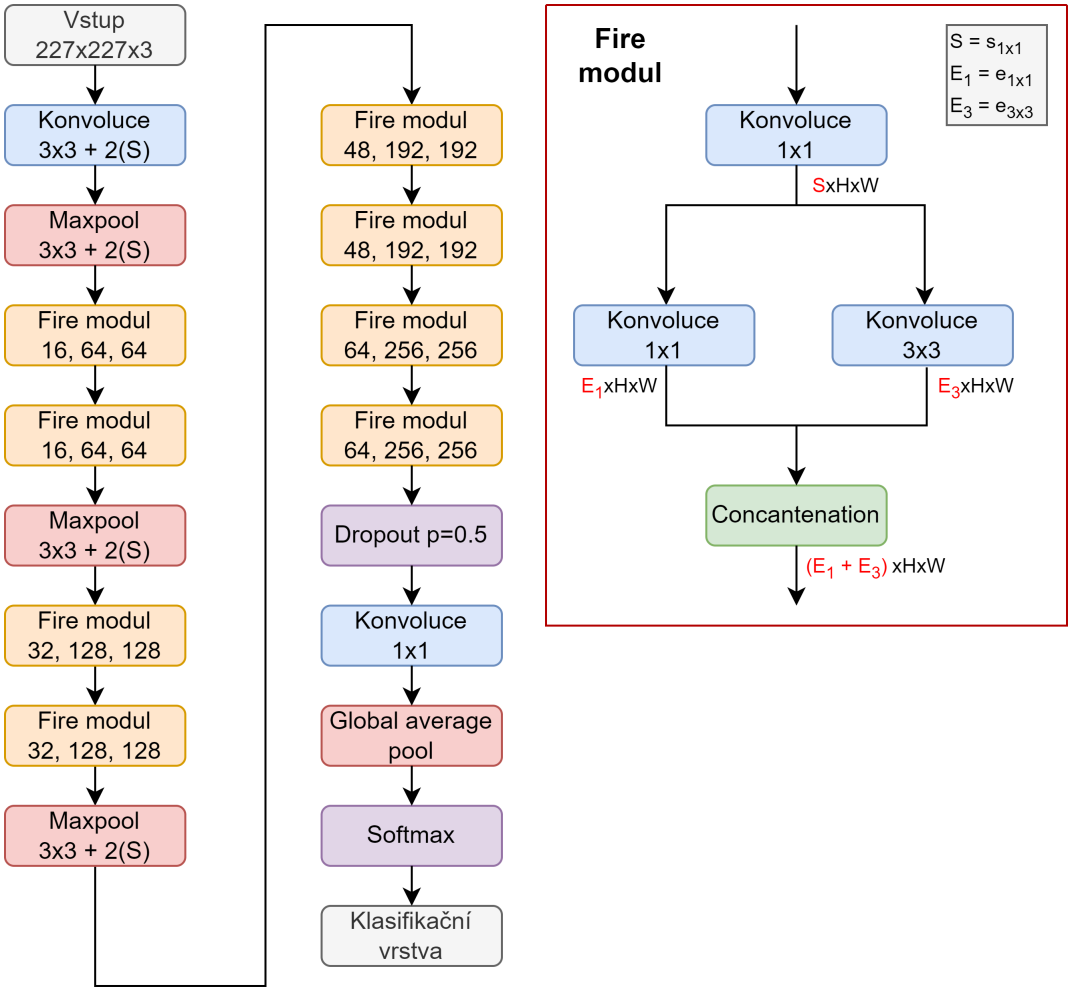


Obrázek 6.9: SqueezeNet fire modul. [15]

nazývány redukční 1×1 konvoluční vrstvy z *fire* modulu, *expand* vrstvami pak kombinace 1×1 a 3×3 konvolučních vrstev nacházejících se za *squeeze* vrstvami

taktéž ve *fire* modulu. Zároveň se definují 3 hyperparametry - $s_{1\times1}$, $e_{1\times1}$ a $e_{3\times3}$, kde $s_{1\times1}$ je počet kernelů konvoluce 1×1 ve *squeeze* vrstvě, $e_{1\times1}$ je počet kernelů konvoluce 1×1 v *expand* vrstvě a $e_{3\times3}$ analogicky. Pro tyto hyperparametry musí při konstrukci *fire* modulu platit $s_{1\times1} \leq (e_{1\times1} + e_{3\times3})$ - tím je zajištěno, že *squeeze* vrstva bude redukovat počet feature map jdoucích do *expand* vrstvy.

Na obrázku 6.10 je přiložena pseudo-architektura sítě SqueezeNet. Aktivační funkcí v síti je ReLU.



Obrázek 6.10: Pseudo-architektura sítě SqueezeNet.

6.1.6 DarkNet19

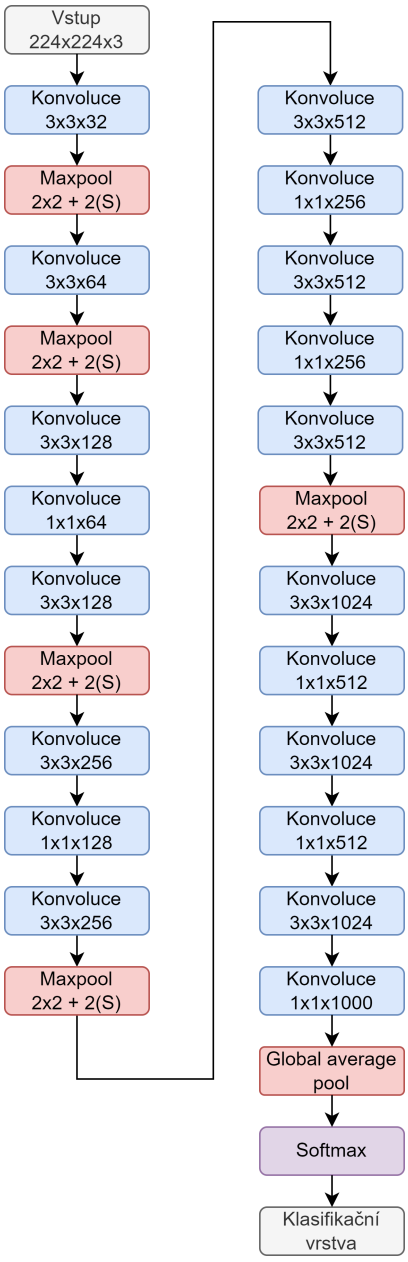
DarkNet19[28] je Konvoluční neuronová síť uvedena v roce 2016 a používaná v technologii *YOLOv2* - technologie rozpoznávající objekty v reálném čase. Z tohoto důvodu je její přední vlastností rychlá konvergence a nízký počet výpočetních operací k zpracování vstupního obrazu.

DarkNet19 převážně využívá 3×3 konvoluční vrstvy a zdvojnásobuje počet feature map po každé pooling vrstvě. Dále navázáním na článek Network in Network

[23] využívá jako poslední pooling vrstvu *Global average pool* (z důvodu lokalizace objektů v obraze, což je jedna z funkcí *YOLOv2* technologie). Podobně jako předchozí zmíněné sítě, i DarkNet19 využívá redukční 1×1 konvoluce před konvolucemi 3×3. V neposlední řadě implementuje batch normalizaci, která zajišťuje stabilizaci tréninku, zrychlení konvergence a regularizaci modelu.

Síť obsahuje 5 max-pooling vrstev a, jak již název napovídá, 19 konvolučních vrstev. Pseudo-architektura DarkNetu19 je přiložena na obrázku 6.11, přičemž jsou v ní opomenuty ReLU aktivační funkce.

Pro testování této sítě bude mezi GAP a Softmax vrstvu přidána vrstva plně propojená, a to z důvodu nekompatibilního počtu feature map a klasifikačních kategorií (viz sekce 4.3).



Obrázek 6.11: Pseudo-architektura sítě DarkNet19.

6.2 Vlastní CNN architektury

V této sekci budou popsány vlastní architektury CNN, které budou předmětem testování. Architektury jsou rozdělené do dvou kategorií - základní, které slouží jako podklad k optimalizaci hyperparametrů a dále komplexní, přesněji residual-inception CNN, na kterých nebude optimalizace hyperparametrů prováděna. Tímto způsobem bude umožněno porovnat dvě metody přístupů, a sice přístup důkladné optimalizace parametrů a přístup konstrukce komplexních architektur.

6.2.1 Základní CNN

Základními CNN jsou myšleny architektury několika konvolučních vrstev střídající se s poolingovými vrstvami a navíc neobsahuje rozvětvení. Takovýmto architekturám se někdy říká *plain* CNN. Cílem bude postupnou optimalizací hyperparametrů najít takovou architekturu, která bude na mnou vytvořeným datasetem dosahovat nejlepších výsledků.

Slovem optimalizace je myšlena optimalizace částečná úměrná mým výpočetním kapacitám. Zřejmě nelze prozkoumat celý prostor zmíněným hyperparametrů. Příkladem uvažme, že každý zkoumaný hyperparametr budeme optimalizovat pro 10 hodnot, potom dostáváme miliardy tréninků, které by bylo nutné provést. Z tohoto důvodu budou zkoumány jen vybrané hodnoty parametrů a jejich optimalizace bude rozdělena na několik experimentů. Postupným vyhodnocováním hyperparametrů bude dosaženo jejich částečné optimalizace. Není však vyloučeno, že existuje kombinace hyperparametrů, pro něž nebyl trénink proveden, a které by dosahovaly lepších výsledků. Prezentované závěry tak nelze považovat za optimalizaci v pravém slova smyslu, ač to bude optimalizací nazýváno. Výsledky slouží spíše jako představa o obecných pravidlech a zákonitostech při určování hyperparametrů.

Pro sekvenční spouštění tréninků sítí různých hyperparametrů architektur byl využit *Matlab Experiment Manager*. Celkem bylo provedeno 291 tréninků v rámci 7 experimentů. Každý experiment vycházel z jedněch tréninkových parametrů, těmi jsou:

- Architektura obsahující vstupní vrstvu, určitý počet konvolučních modulů sestavených z konvoluční vrstvy, aktivační funkce a maxpooling vrstvy, za kterými se nachází softmax vrstva, plně propojená vrstva a klasifikační vrstva. Konvoluční vrstvy v jednotlivých konvolučních modulech (KM) obsahují postupně kernely velikostí:
  - KM1 – 3×3.
  - KM2 – 3×3
  - KM3 – 5×5
  - KM4 – 7×7
  - KM5 – 11×11
  - KM6 – 3×3

- Výsledná síť bude ta s nejnižší validační chybou.
- Identický tréninkový, testovací a validační dataset v poměru $80 : 10 : 10$.
- Velikosti vstupu – $256 \times 256 \times 3$.
- Optimalizační metoda – Adam.
- Aktivační funkce – ReLU.
- Learning rate – 0,001.
- Parametr L_2 regularizace – 0,0001.
- Normalizační technika – Batch normalizace aplikovaná po každé konvoluční vrstvě.
- Velikost mini batche – 128.
- Zastavovací kritéria – 100 epoch nebo 30 souvisle neklesajících validačních chyb, přičemž validace se provádí po každých 50 iteracích.
- Exekuční prostředí – jediná grafická karta *NVIDIA GTX A5500 Mobile*.

V jednotlivých experimentech byly vždy některé z těchto parametrů modifikovány. Změnami oproti počátečním parametrům pro každý experiment jsou:

- **Experiment 1** – celkem 90 tréninků.
  - Počet konvolučních modulů v rozmezí 2 až 6.
  - Normalizační techniky po každé konvoluční vrstvě – žádná; LRN normalizace; Batch normalizace.
  - Plně propojené vrstvy v počtu 1 až 3.
  - Variabilní learning rate modifikován faktorem 1 (konstantní learning rate) nebo 0,5 (dále faktor poklesu) po každých 10 epochách (dále frekvence poklesu).
- **Experiment 2** – celkem 24 tréninků.
  - Počet konvolučních modulů – $5; 6$.
  - Dropout vrstva – žádná; za plně propojenou vrstvou.
  - Parametr dropout vrstvy – 0,5
  - Normalizační techniky po každé konvoluční vrstvě – Batch normalizace.
  - Variabilní learning rate
    - Faktor poklesu – $0{,}3; 0{,}5; 0{,}7$.
    - Fekvence poklesu – $5; 10$.
- **Experiment 3** – celkem 12 tréninků.

  - Počet konvolučních modulů – 6.
  - Dropout vrstva – žádná; po každé konvoluční vrstvě a zároveň po plně propojené vrstvě.
  - Parametr dropout vrstev – $0{,}3; 0{,}5; 0{,}7$.
  - Normalizační techniky po každé konvoluční vrstvě – Batch normalizace.
  - Velikost mini batche – 64; 128.

- **Experiment 4** – celkem 12 tréninků.
  - Počet konvolučních modulů – 6.
  - Optimalizační metody – SGDM; RMSprop, ADAM.
  - Normalizační techniky po každé konvoluční vrstvě – Batch normalizace.
  - Velikosti vstupu – $64 \times 64 \times 3; 128 \times 128 \times 3; 256 \times 256 \times 3; 320 \times 320 \times 3;$.

- **Experiment 5** – celkem 45 tréninků.
  - Počet konvolučních modulů – 6.
  - Normalizační techniky po každé konvoluční vrstvě – Batch normalizace.
  - Variabilní learning rate
    * Learning rate – $0{,}01; 0{,}001; 0{,}0001$.
    * Faktor poklesu – $0{,}3; 0{,}5; 0{,}7; 0{,}9; 1$.
    * Frekvence poklesu – $5; 10; 20$.

- **Experiment 6** – celkem 12 tréninků.
  - Počet konvolučních modulů – 6.
  - Parametr L_2 regularizace – $0{,}1; 0{,}01; 0{,}001; 0{,}0001$.
  - Normalizační techniky po každé konvoluční vrstvě – Batch normalizace.
  - Aktivační funkce – ReLU; tanh; sigmoid.

- **Experiment 7** – celkem 96 tréninků.
  - Počet konvolučních modulů – 6.
  - Normalizační techniky po každé konvoluční vrstvě – Batch normalizace.
  - Velikosti kernelů konvolučních vrstev v jednotlivých konvolučních modulech (KM)
    * KM1 – 3×3; 5×5.
    * KM2 – 3×3; 7×7.
    * KM3 – 3×3; 7×7.
    * KM4 – 3×3; 7×7.
    * KM5 – 3×3; 7×7; 11×11.
    * KM6 – 3×3; 11×11.

6.2.2 Inception reziduální CNN

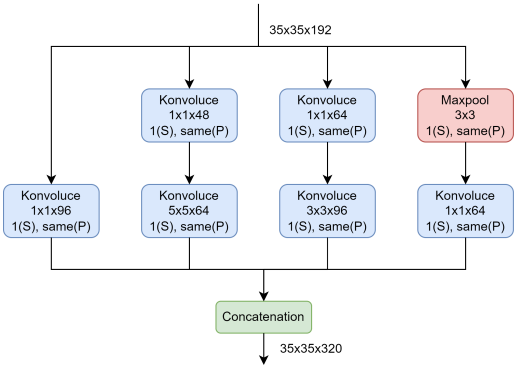
Komplexní architektury mají o několik řádů více hyperparametrů oproti základním architekturám. Protože s narůstajícím počtem hyperparametrů roste prostor, ve kterém probíhá jejich omptimalizace, exponenciálně, tak náročnost takového problému daleko přesahuje možnosti této práce. Namísto optimalizace hyperparametrů tak komplexní neuronové sítě budou sloužit k porovnání výkonosti se základními CNN architekturami, na kterých byla provedena částečná optimalizace hyperparametrů.

Ke zkonstruování této komplexní sítě jsem vyšel z myšlenek sítě GoogLeNet a ResNet. První zmiňovaná využívá inception moduly, které kombinují výstupy z několika kernelů různých velikostí a poolingu do jedné sady feature map. Tato metoda tak dokáže v každém kroku zaznamenat prostorově strukturní jevy na různých velikostech receptivního pole a do jisté míry se tak vyhýbá složité optimalizaci velikostí kernelů v jednotlivých konvolučních vrstvách. Druhá zmiňovaná využívá reziduální bloky, které mají potenciál zmírnit degradaci gradientu, jež je častou limitací při konstrukci velmi hlubokých sítí. Rozhodl jsem se tedy vytvořit síť kombinující reziduální a inception bloky.

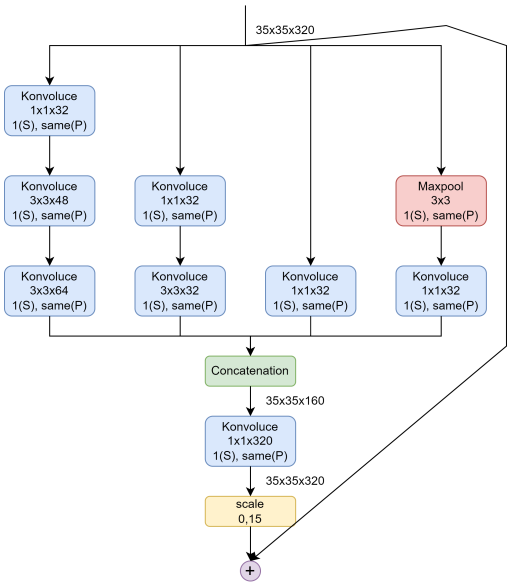
Za posledních několik let bylo vydáno mnoho článků zaměřujících se na sítě kombinující inception a reziduální bloky. Často citovaným článkem tohoto zaměření je [37] prezentující síť s názvem *Inception-ResNet-v4*. Ten mimo kombinaci dvou zmíněných bloků popisuje další využívané metody, přesněji faktorizaci větších konvolucí menšími, faktorizaci do asymetrických konvolucí a vrstvu škálování.

- Faktorizace větších konvolucí menšími – typicky se využívá u kernelů velikosti větší než 3x3. Myšlenka spočívá v nahrazení jednoho většího kernelu několika menšími. Příkladem kernel velikosti 5×5 lze aproximovat dvěma kernely velikostí 3×3 a současně snížit výpočetní náročnost přibližně o třetinu.
- Faktorizace do asymetrických konvolucí – velice jednoduše nahrazuje jeden kernel velikosti $S \times S$ dvěma kernely o velikostech $1 \times S$ a $S \times 1$, přičemž se opět snižuje výpočetní náročnost. Příkladem konvoluce s kernely 1×3 a 3×1 je oproti jedné konvoluci s kernelem 3×3 opět přibližně o třetinu výpočetně méně náročná.
- Vrstva škálování – aplikací vrstvy škálování se všechny hodnoty aktivace násobí předem určeným parametrem. Autoři uvádějí, že škálování reziduálních bloků přechází "úmrtí" sítě, které bylo pozorováno v případech, kdy počet feature map byl větší jak 1000.

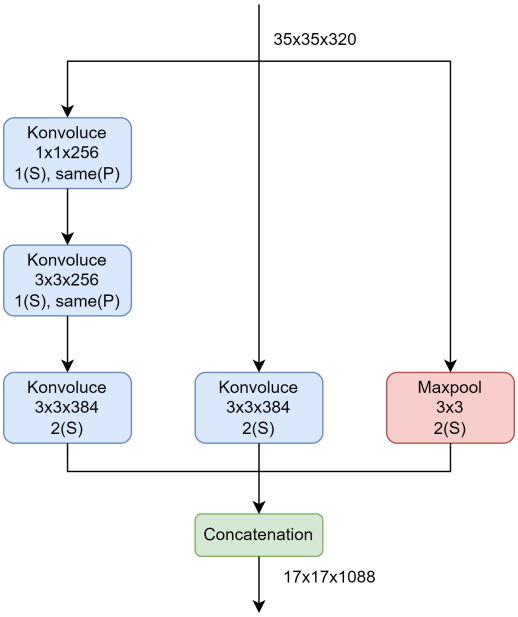
Výše zmíněné techniky jsem se rozhodl v síti použít taktéž. Níže je přiloženo celkem 6 bloků, ze kterých je síť složena (obrázek 6.12, 6.13, 6.14, 6.15, 6.16, 6.17). Bloky jsou řazeny tak, jak se vyskytují v síti a rozměry aktivací tak odpovídají pouze pro místo, kde se blok v síti nachází, nikoliv obecně. Výsledná pseudo-architektura je na obrázku 6.18. Aktivační funkcí byla použita funkce ReLU, před ní se navíc vždy nachází Batch normalizační vrstva. Celkem má tato síť přibližně 38,9 miliónů parametrů v 581 vrstvách, ž čehož je 173 vrstev konvolučních. [1]



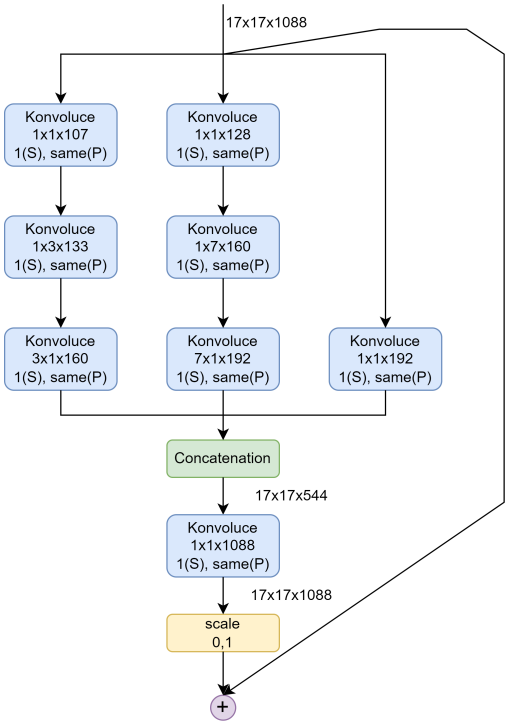
Obrázek 6.12: Inception blok 1.



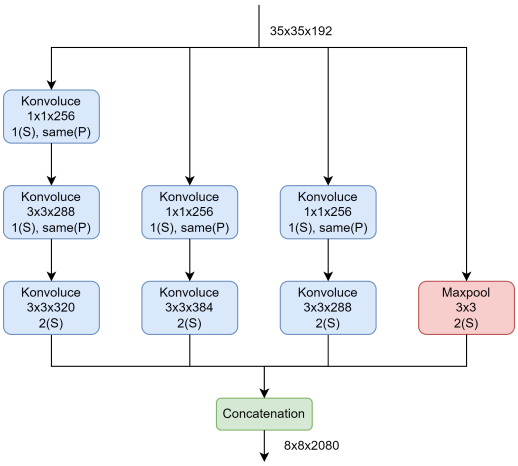
Obrázek 6.13: Inception reziduální blok 1.



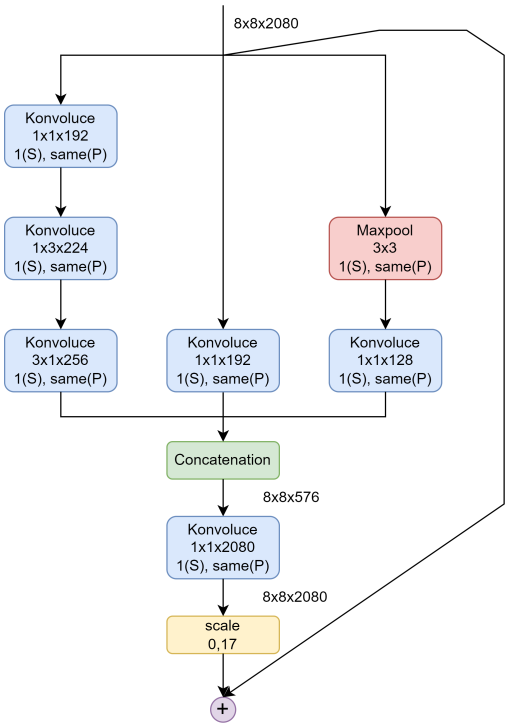
Obrázek 6.14: Inception blok 2.



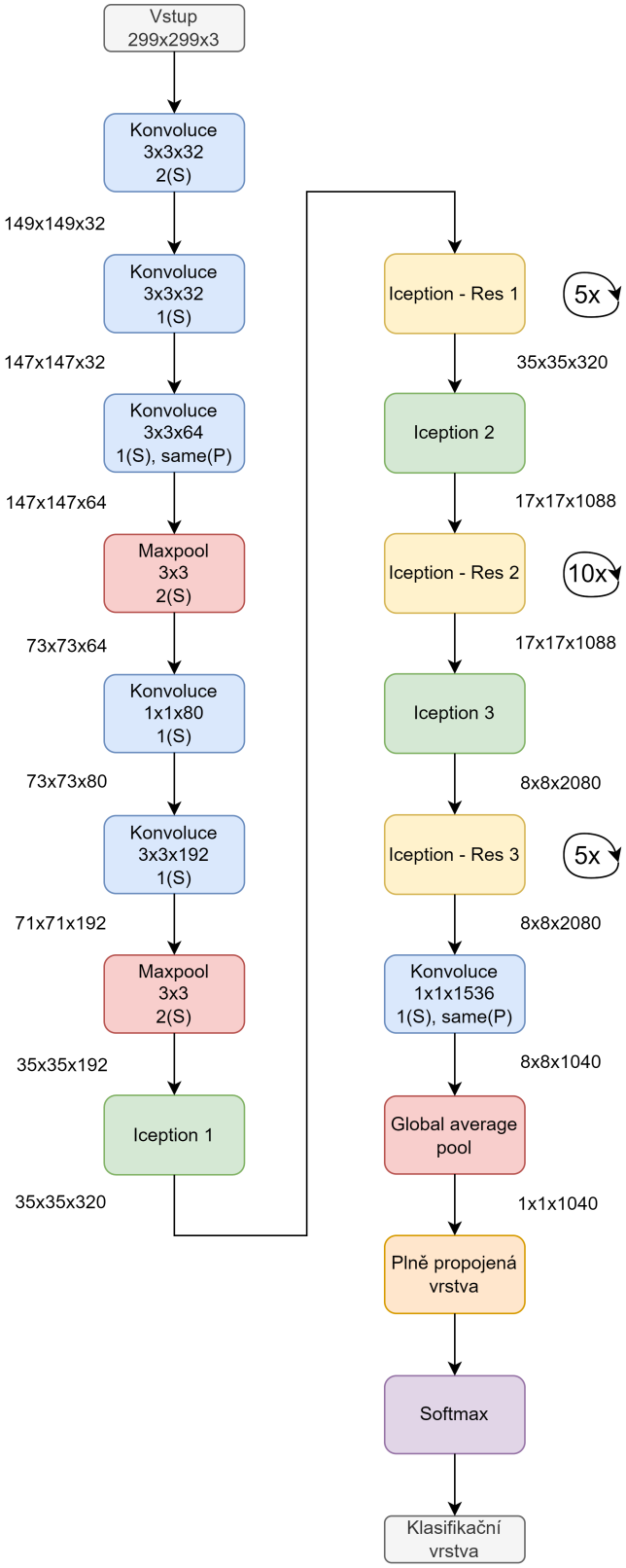
Obrázek 6.15: Inception reziduální blok 2.



Obrázek 6.16: Inception blok 3.



Obrázek 6.17: Inception reziduální blok 3.



Obrázek 6.18: Pseudo-architektura inception-residual CNN.

Kapitola 7

Výsledky

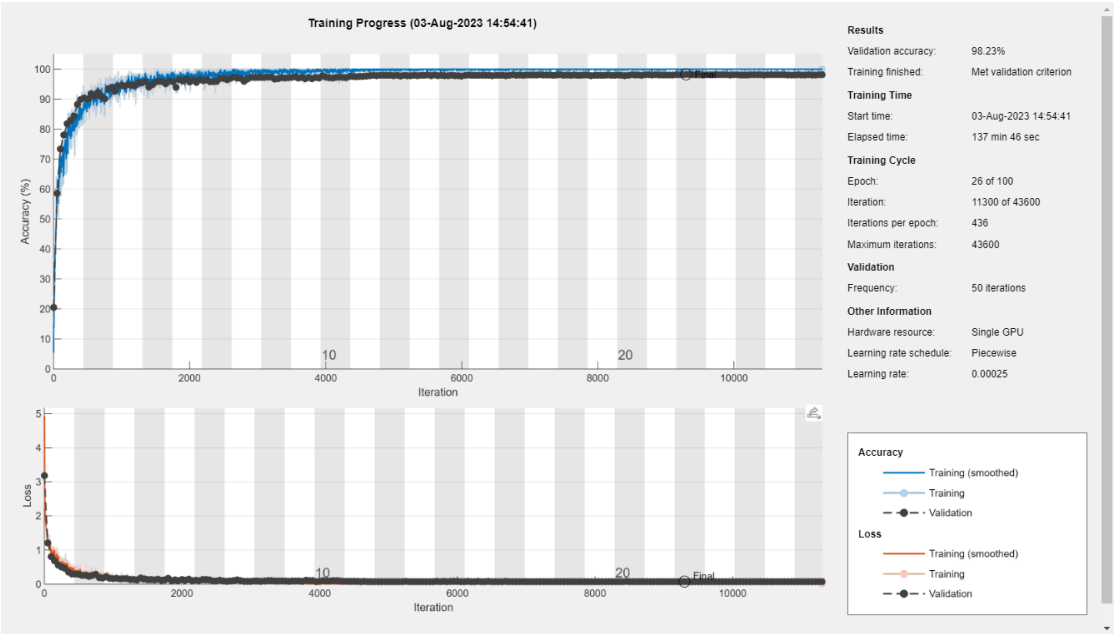
7.1 Předtrénované CNN

Parametry transfer learningu.

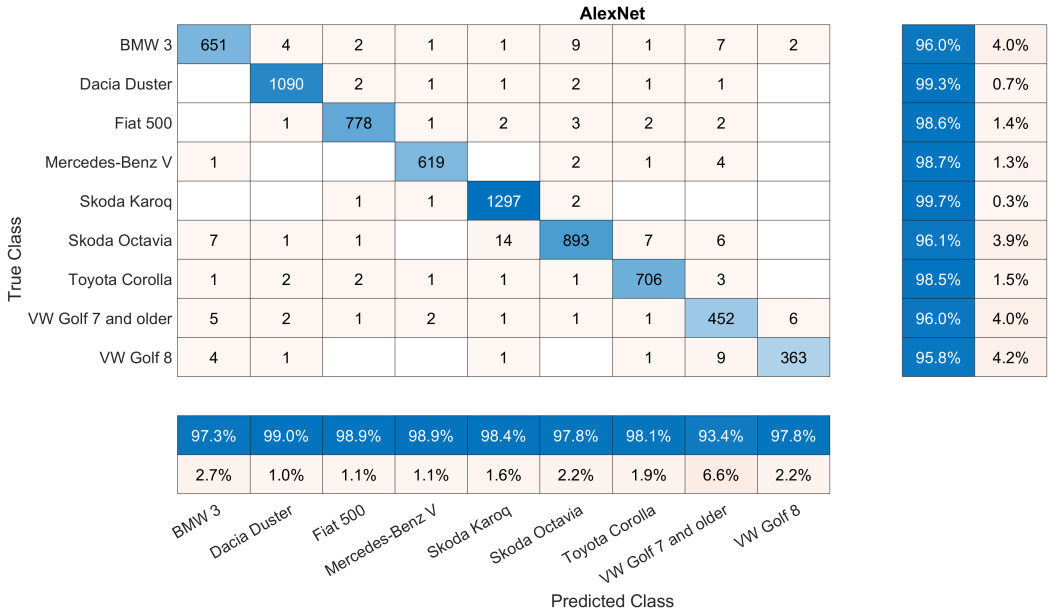
AlexNet

Přesnost na testovacích datech	98,00 %
Chyba na testovacích datech	0,069

Tabulka 7.1: Výsledky tréninku předtrénované sítě AlexNet.



Obrázek 7.1: Graf tréninku předtrénované sítě AlexNet.

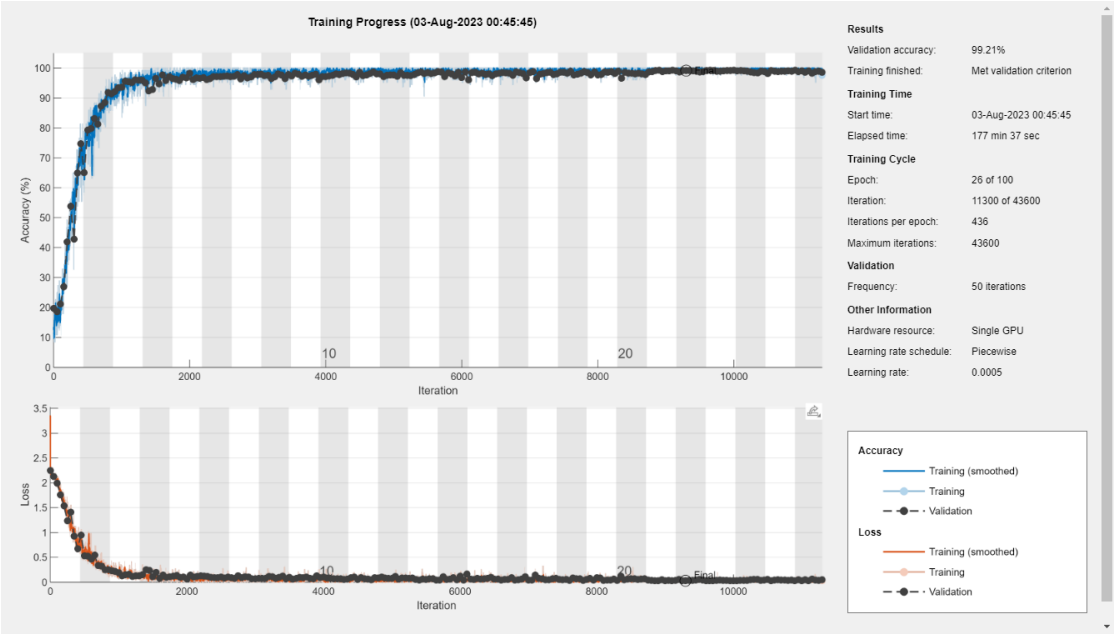


Obrázek 7.2: Matice záměn předtrénované sítě AlexNet.

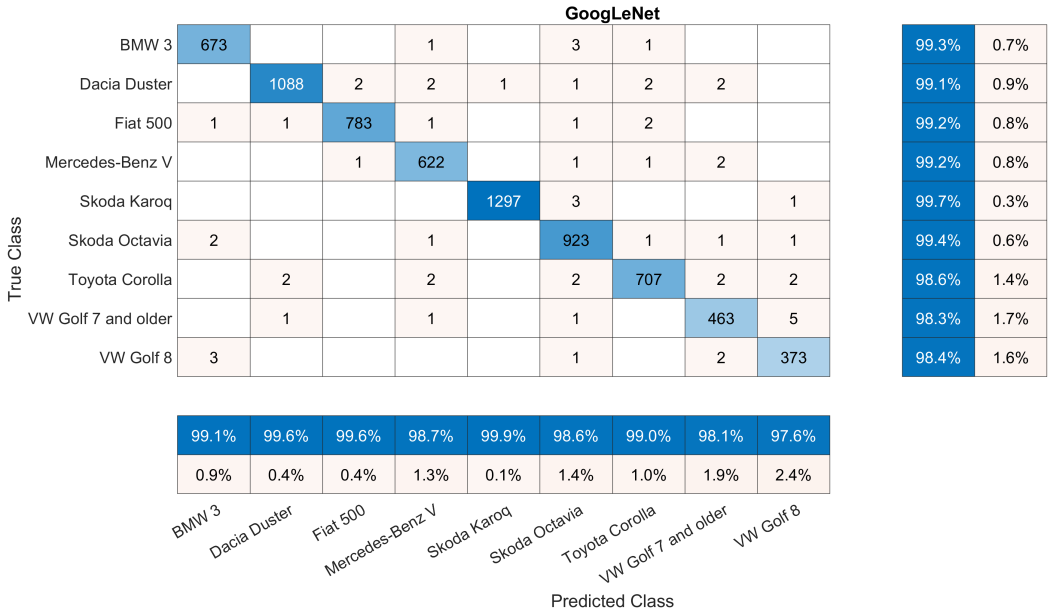
GoogLeNet

Přesnost na testovacích datech	99,14 %
Chyba na testovacích datech	0,026

Tabulka 7.2: Výsledky tréninku předtrénované sítě GoogLeNet.



Obrázek 7.3: Graf tréninku předtrénované sítě GoogLeNet.

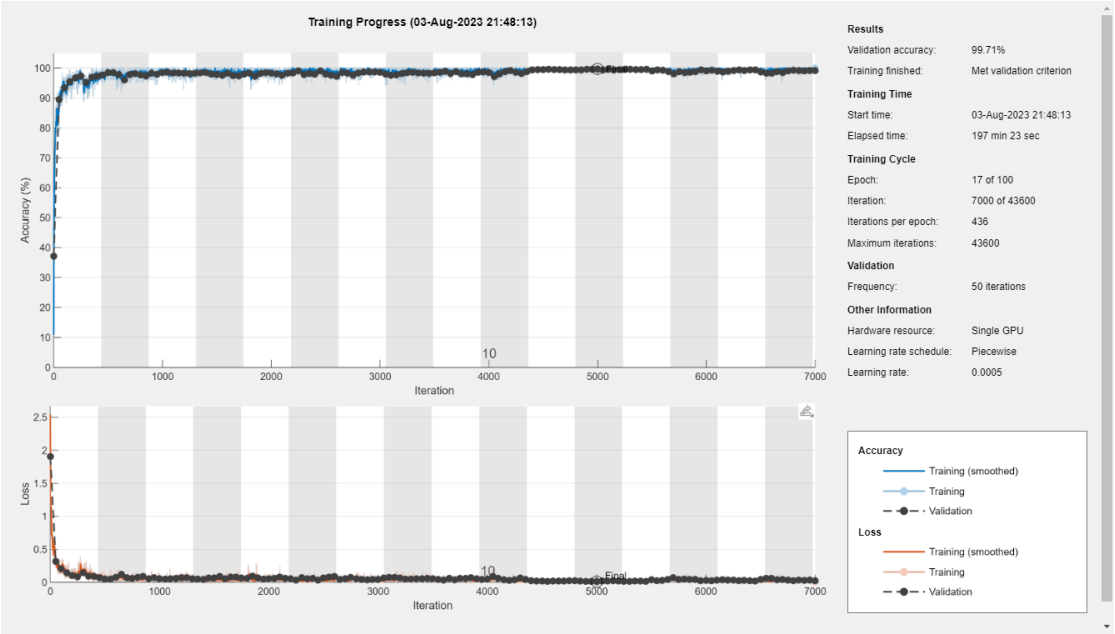


Obrázek 7.4: Matice záměn předtrénované sítě GoogLeNet.

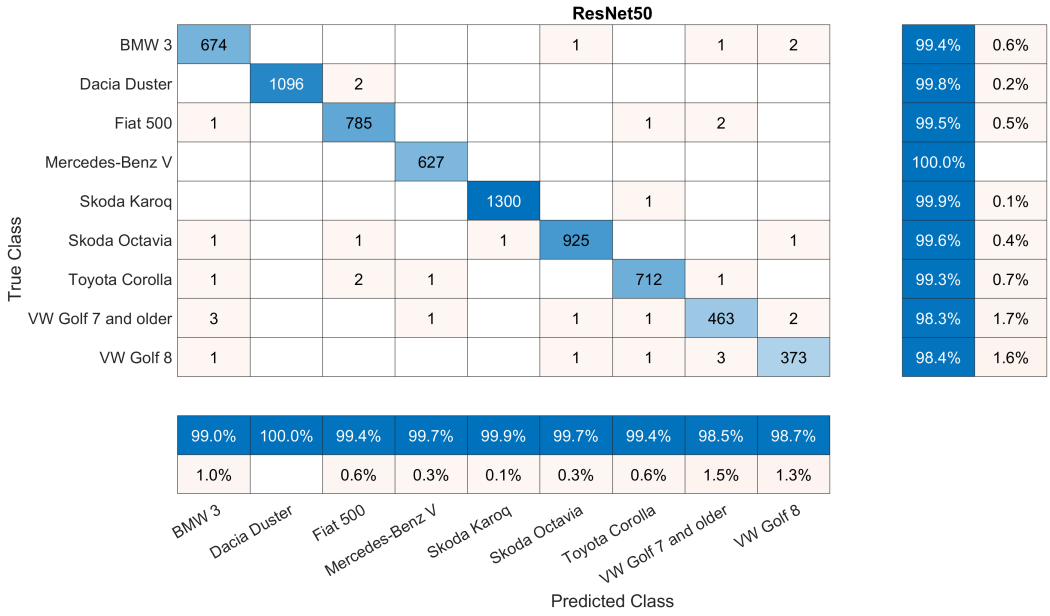
ResNet50

Přesnost na testovacích datech	99,51 %
Chyba na testovacích datech	0,009

Tabulka 7.3: Výsledky tréninku předtrénované sítě ResNet50.



Obrázek 7.5: Graf tréninku předtrénované sítě ResNet50.

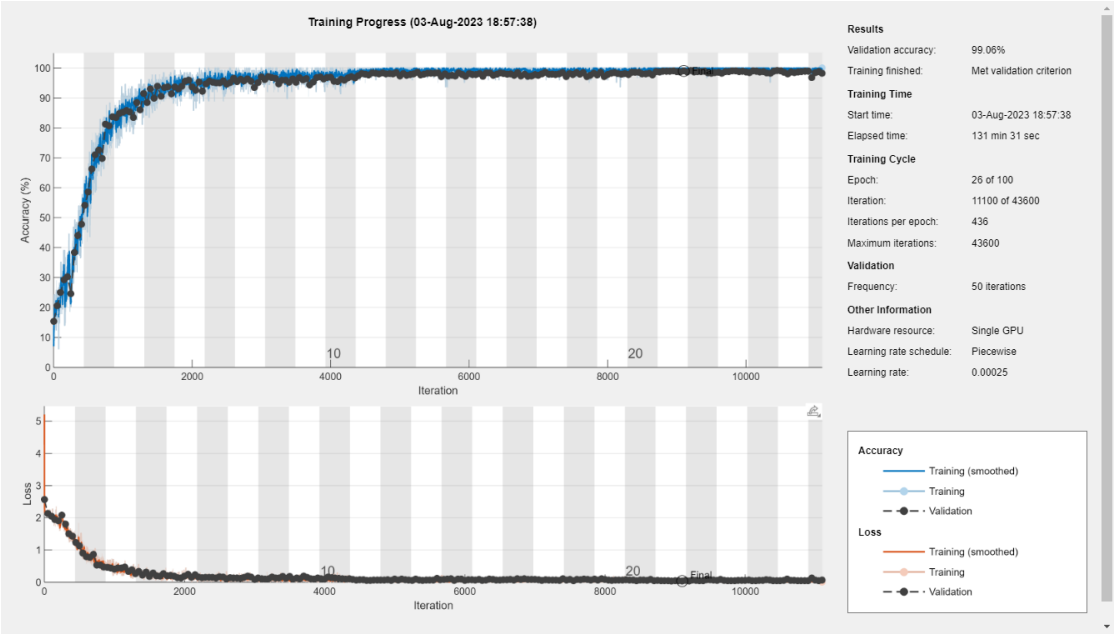


Obrázek 7.6: Matice záměn předtrénované sítě ResNet50.

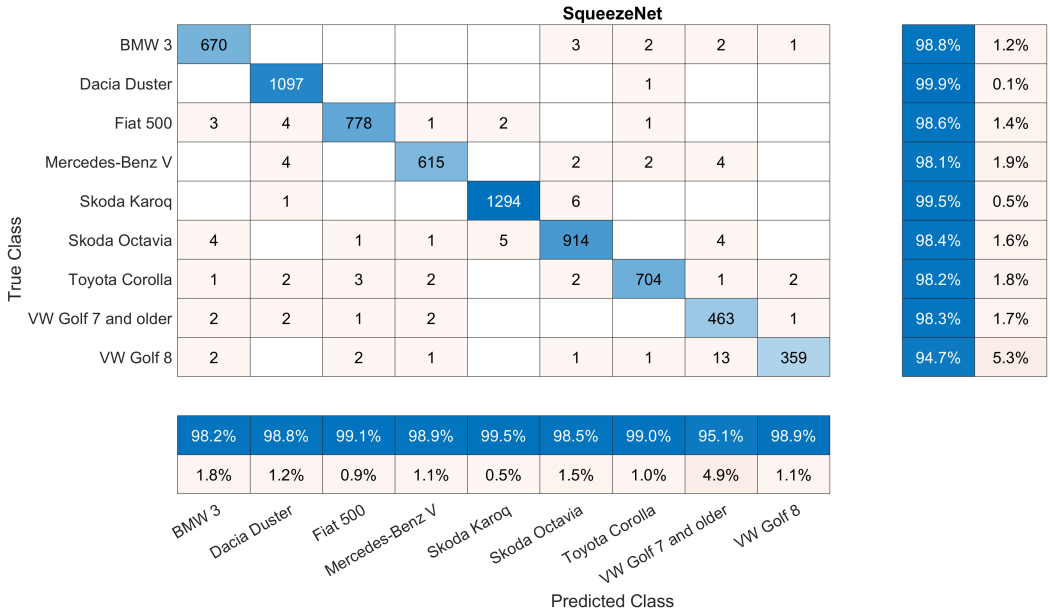
SqueezeNet

Přesnost na testovacích datech	98,64 %
Chyba na testovacích datech	0,054

Tabulka 7.4: Výsledky tréninku předtrénované sítě SqueezeNet.



Obrázek 7.7: Graf tréninku předtrénované sítě SqueezeNet.

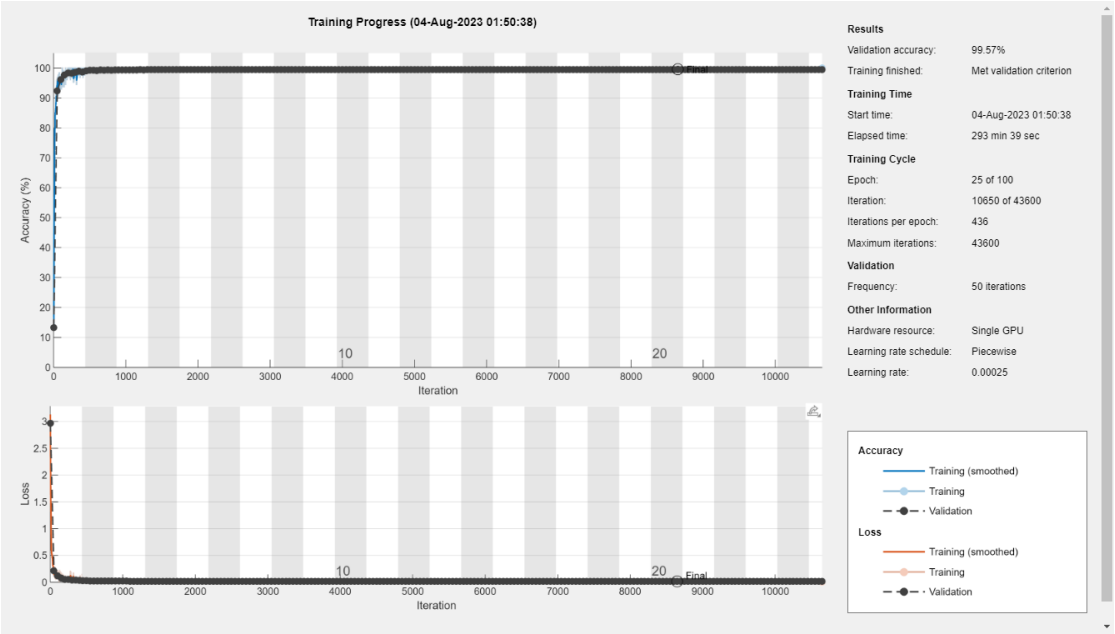


Obrázek 7.8: Matice záměn předtrénované sítě SqueezeNet.

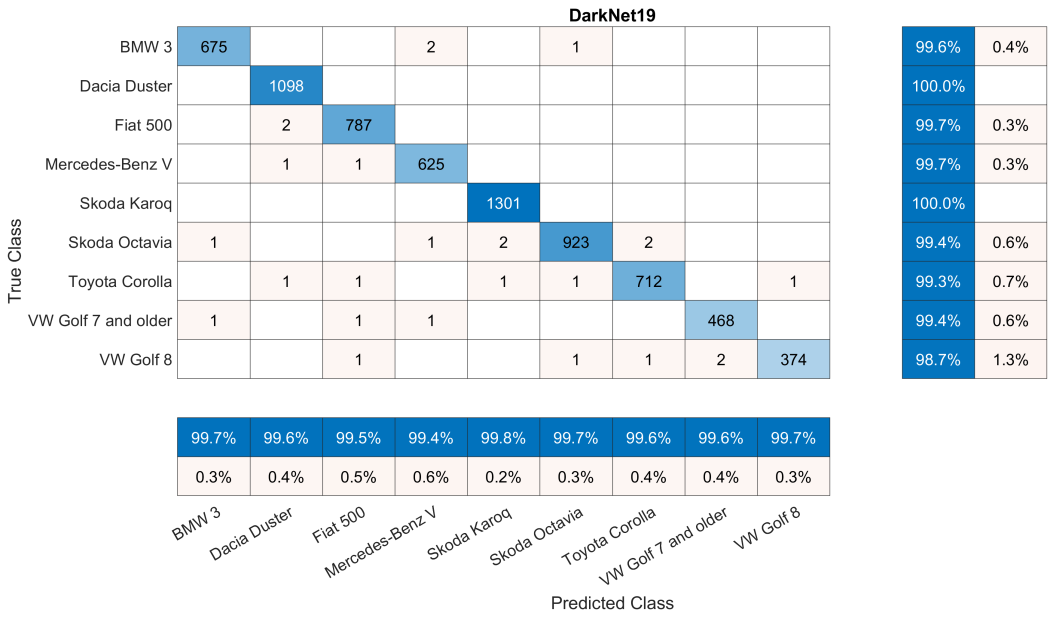
DarkNet19

Přesnost na testovacích datech	99,63 %
Chyba na testovacích datech	0,014

Tabulka 7.5: Výsledky tréninku předtrénované sítě DarkNet19.



Obrázek 7.9: Graf tréninku předtrénované sítě DarkNet19.



Obrázek 7.10: Matice záměn předtrénované sítě DarkNet19.

7.1.1 Komentář k výsledkům předtrénovaných CNN

Umístění	Síť	Přesnost	Chyba	Epocha	Čas [HH:MM]
1	DarkNet19	99,63 %	0,014	20	4:53
2	ResNet50	99,51 %	0,009	12	3:17
3	GoogLeNet	99,14 %	0,026	22	2:57
4	SqueezeNet	98,64 %	0,054	21	2:11
5	AlexNet	98,00 %	0,069	22	2:17

Tabulka 7.6: Výsledky předtrénovaných sítí.

První místo na základě přesnosti na testovacích datech obsadil DarkNet19, ač se spíše jedná o sdílené vítězství se sítí ResNet50, která na DarkNet19 ztrácí 12 setin procentního bodu, přičemž chyby dosáhla síť ResNet50 vůbec nejnižší ze všech testovaných sítí. Je však zajímavé sledovat síť s 19 konvolučními vrstvami dosahovat stejných výsledků jako síť s vrstvami 50. Navíc z grafu tréninku je jasně patrné, že DarkNet19 konverguje k výsledku rapidně. Ač síť DarkNet19 dosáhla nejlepších výsledků až po 20 epochách, již v první epoše její přesnost překročila 99 %.

Z matic záměn se také dají učinit určité závěry. Nejvíce chybných predikcí dosáhl *VW Golf 7 a starší*, u kterého například síť SqueezeNet měla úspěšnost správné predikce pouze 95,1 %. Nepříliš překvapivě byl vůz nejčastěji zaměňován s vozem *VW Golf 8*, tedy s jeho nejnovější generací. Z tohoto pozorování vyplývá i fakt, že oba vozy byly nejvíce zaměňovány za ostatní, přesněji *VW Golf 8* byl celkem zaměněn

za jiné vozy v 2,8 % případů a *VW Golf 7 a starší* v 1,9 % případů. Naopak nejlépe předpovídanými vozy byla *Škoda Karoq* s průměrnou přesností 99,5 % a *Dacia Duster* s přesností 99,4 %. Úspěšnost byla celkově horší pro automobily podobného stylu karoserie, přesněji střední třída vozidel byla průměrně klasifikována s úspěšností 98,4 % a ostatní vozy s úspěšností 99,3 %. S jedinou výjimkou, a tou byla síť DarkNet19, která klasifikovala automobily střední třídy lehce lépe než ostatní, přesněji s úspěšností 99,66 % pro vozy střední třídy a 99,58 % pro vozy ostatní.

7.2 Optimalizace parametrů základních CNN

Výsledky a závěry se vztahují pouze na aplikaci CNN na dataset zavedený v kapitole 5. Tabulky jsou z důvodu velikosti omezeny pouze na vybrané výsledky experimentů.

Legenda k tabulkám výsledků 7.7 až 7.13:

- Normalizační techniky: 0 – žádná; LRN – *Local response normalization*; BN – Batch normalizace.
- KM – Konvoluční moduly.
- PPV – Plně propojená vrstva.
- Dropout vrstvy: 0 – žádná; DPPV – Dropout vrstva po plně propojené vrstvě; DKVPPV – Dropout vrstva po každé konvoluční vrstvě a zároveň po plně propojené vrstvě.
- PDV – Parametr dropout vrstvy/vrstev.
- C_1 až C_6 – velikost kernelů konvoluční vrstvy v konvolučním modulu 1 až 6.

Experiment 1

Počet KM	Normalizace	Počet PPV	Faktor poklesu	Přesnost	Chyba
2	0	1	0,5	55,9 %	1,39
2	N	1	0,5	59,9 %	1,24
2	BN	1	0,5	71,1 %	0,89
3	BN	1	0,5	86,7 %	0,41
4	BN	1	0,5	89,3 %	0,29
5	BN	1	0,5	91,2 %	0,27
6	0	1	0,5	85,0 %	0,49
6	N	1	0,5	88,2 %	0,33
6	BN	1	0,5	95,2 %	0,15
6	0	2	0,5	65,8 %	1,03
6	0	3	0,5	43,1 %	1,79
6	BN	1	1	94,3 %	0,17

Tabulka 7.7: Vybrané výsledky experimentu 1.

Experiment 2

Počet KM	Frekvence poklesu	Faktor poklesu	Dropout vrstva	Přesnost	Chyba
6	5	0,3	DPPV	94,2 %	0,17
6	20	0,3	DPPV	95,8 %	0,13
6	10	0,3	0	96,6 %	0,11
6	10	0,3	DPPV	96,8 %	0,11
6	10	0,7	0	95,2 %	0,15
6	10	0,7	DPPV	95,7 %	0,14
6	5	0,7	0	95,4 %	0,16
6	5	0,7	DPPV	96,1 %	0,15
5	10	0,3	0	91,6 %	0,28
5	10	0,3	DPPV	92,1 %	0,22

Tabulka 7.8: Vybrané výsledky experimentu 2.

Experiment 3

Počet KM	Dropout vrstva	PDV	Velikost mini batche	Přesnost	Chyba
6	DPPV	0,3	64	95,5 %	0,15
6	DPPV	0,5	64	95,7 %	0,15
6	DPPV	0,7	64	94,8 %	0,17
6	DPPV	0,5	128	95,5 %	0,14
6	DKVPPV	0,5	64	52,7 %	1,28

Tabulka 7.9: Vybrané výsledky experimentu 3.

Experiment 4

Počet KM	Optimalizační metoda	Velikost vstupu	Čas tréninku [HH:MM]	Přesnost	Chyba
6	ADAM	128×128	1 : 15	86,7 %	0,47
6	SGDM	128×128	1 : 15	83,1 %	0,56
6	RMSprop	128×128	1 : 08	86,2 %	0,47
6	ADAM	256×256	2 : 02	96,2 %	0,13
6	SGDM	256×256	2 : 08	94,5 %	0,19
6	RMSprop	256×256	2 : 03	95,8 %	0,15
6	ADAM	320×320	2 : 36	97,3 %	0,08
6	SGDM	320×320	3 : 24	95,6 %	0,15
6	RMSprop	320×320	2 : 44	96,4 %	0,12
6	ADAM	64×64	0 : 42	52,1 %	1,19

Tabulka 7.10: Vybrané výsledky experimentu 4.

Experiment 5

Počet KM	Learning rate	Frekvence poklesu	Faktor poklesu	Čas tréninku [HH:MM]	Přesnost	Chyba
6	0,01	5	0,3	2 : 01	86,7 %	0,47
6	0,01	10	0,5	2 : 11	83,1 %	0,56
6	0,01	20	0,9	2 : 58	86,2 %	0,47
6	0,001	10	0,3	2 : 14	96,2 %	0,13
6	0,001	10	0,7	2 : 31	94,5 %	0,19
6	0,001	5	0,7	1 : 57	95,8 %	0,15
6	0,0001	20	0,9	4 : 01	97,3 %	0,08
6	0,0001	10	0,9	4 : 41	95,6 %	0,15
6	0,0001	5	0,3	Ukončeno po 5-ti hodinách tréninku.		

Tabulka 7.11: Vybrané výsledky experimentu 5.

Experiment 6

Počet KM	Aktivační funkce	Parametr L_2 regularizace	Přesnost	Chyba
6	ReLU	0,0001	95,9 %	0,14
6	ReLU	0,001	95,5 %	0,15
6	ReLU	0,01	93,6 %	0,18
6	ReLU	0,1	90,0 %	0,34
6	tanh	0,0001	91,4 %	0,27
6	tanh	0,001	92,4 %	0,24
6	sigmoid	0,0001	92,8 %	0,25
6	sigmoid	0,001	91,8 %	0,28

Tabulka 7.12: Vybrané výsledky experimentu 6.

Experiment 7

Počet KM	C_1	C_2	C_3	C_4	C_5	C_6	Přesnost	Chyba
6	5×5	3×3	3×3	7×7	3×3	11×11	96,7 %	0,13
6	5×5	3×3	7×7	7×7	7×7	11×11	96,6 %	0,12
6	5×5	3×3	7×7	3×3	7×7	11×11	96,5 %	0,13
6	5×5	3×3	3×3	7×7	7×7	11×11	96,5 %	0,12
6	5×5	3×3	7×7	7×7	3×3	3×3	96,5 %	0,11
6	3×3	7×7	3×3	3×3	7×7	11×11	94,1 %	0,21
6	3×3	3×3	3×3	3×3	3×3	3×3	94,1 %	0,22
6	3×3	7×7	3×3	3×3	7×7	3×3	93,7 %	0,21
6	3×3	7×7	3×3	3×3	3×3	11×11	93,3 %	0,22
6	3×3	7×7	3×3	3×3	3×3	3×3	93,2 %	0,23

Tabulka 7.13: Vybrané výsledky experimentu 7.

7.2.1 Komentář k experimetům 1-7

V tabulce 7.7 experimentu 1 je zřejmá korelace mezi počtem konvolučních modulů a přesností. Rozdíl mezi sítí 2 a 6 konvolučních modulů je při stejných hyperparametrech přibližně 15 % bodů. V příštích experimentech tak bude primárně zkoumána přesnost sítí s 6 konvolučními moduly. Dále je znatelná technika normalizace aktivací, kde mírně lepších výsledků dosahuje batch normalizace nad LRN. Naopak razantní pokles úspěšnosti je spojen s přidáním jedné či dvou plně propojených vrstev, kde síť se třemi plně propojenými vrstvami dosahuje zhruba poloviční úspěšnosti oproti síti s jednou plně propojenou vrstvou. Faktor poklesu v tomto experimentu nebyl zásadním hyperparametrem a na základě výsledků nelze konstatovat, jak, a zda vůbec, ovlivňuje úspěšnost tréninku. Bude tak předmětem dalšího testování.

V tabulce 7.8 experimentu 2 se lze přesvědčit o pozorování z tabulky 7.7, a sice, že použití 6 konvolučních modulů vykazuje stabilně lepší výsledky. Hlavním cílem bylo pro tento experiment zmapovat přínos dropout vrstvy. Ač se nejedná o přesvědčivě rozdílné výsledky, tak mohu konstatovat, že použití dropout vrstvy po plně propojené vrstvě konzistentně vykazovalo lehce lepší výsledky, v převážném počtu případů šlo o zlepšení v rámci jednoho procentního bodu. Faktor a frekvence poklesu jsou stále velice těžce uchopitelné, neboť je nutno na ně nahlížet jako na dvojici úzce spjatých parametrů. Nejlepších výsledků dosahuje zpravidla několik různých kombinací. V tomto experimentu dosáhla nejlepších výsledků kombinace 10, 0,3 pro frekvenci respektive faktor poklesu. Toto pozorování však rozhodně není obecné, z mých zkušeností vykazují nejlepších hodnot různé kombinace těchto dvou parametrů pro různě nastavené ostatní parametry. Mohu jen odhadovat, že dobrou univerzální hodnotou pro většinu parametrů je kombinace 10, 0,5 pro frekvenci respektive faktor poklesu.

Experiment 3 (Tabulka 7.9) rozšiřuje pozorování z experimentu 2 o modifikaci parametru dropout vrstvy. Výsledky nejsou v tomto případě příliš přesvědčivé. Přesněji pro hodnoty parametru 0,3 a 0,5 byly výsledky téměř totožné a pro parametr 0,7 šlo o pokles přibližně jednoho procentního bodu. Dále byl zkoumán vliv velikosti mini batche. Ani zde nešlo o výraznější změny. Typicky se rozdíl v přesnosti sítě v závislosti na velikosti mini batche pohyboval v rozmezí půl procentního bodu a rozdíl v chybě v jednotkách setin. Naopak velmi výrazný pokles přesnosti nastává při použití dropout vrstvy po každé konvoluční vrstvě, nehledě na ostatní hyperparametry tréninky končily úspěšností v oblasti okolo 50 %.

Experiment 4 (Tabulka 7.10) se zabývá velikostí vstupů a optimalizačními metodami. Z výsledků je zřejmý znatelný nárůst přesnosti se zvětšující se velikostí vstupu. Současně však s velikostí vstupu narůstá i celkový čas tréninku. Z optimalizačních technik dosahuje nejlepších hodnot algoritmus ADAM, s náskokem nad algoritmem RMSprop v jednotkách desetiny procentního bodu. Větší pokles přesnosti je pak patrný u algoritmu SGDM, a to v rozmezí přibližně dvou procentních bodů. Navíc algoritmus SGDM konvergoval k výsledku nejpomaleji. Rozdíly v rychlosti konvergence mezi algoritmy ADAM a RMSprop byl nepatrný.

Experiment 5 (Tabulka 7.11) se opět snaží mapovat vliv frekvence a faktoru poklesu. Výsledky však svojí nekonzistentností korespondují s experimentem 1 a 2. Tentokrát jsou zkoumané parametry rozšířeny o inicializační learning rate, díky němuž lze vysledovat alespoň některé závislosti. Typicky platí, že se snižujícím se parametrem learning ratu je výhodné zvýšit frekvenci a faktor poklesu. Extrémem je potom poslední řádek tabulky, který znázorňuje nízké všechny 3 zkoumané parametry, a který nakonec vyústil v manuální zastavení tohoto tréninku po 5 hodinách. Na času tréninku lze vysledovat znatelné rozdíly, kde nízké hodnoty všech 3 parametrů mají tendenci výrazně navyšovat čas tréninku, aniž by přinášely zlepšení výkonosti. Podobně jako v případě experimentu 2 mohu, na základě testování, pouze odhadovat, že dobrými univerzálními hodnotami těchto 3 parametrů pro mé data je learning rate velikosti 0,001, frekvence poklesu velikosti 10 a faktor poklesu velikosti 0,5.

Experiment 6 (Tabulka 7.12) se zabývá vlivem výběru aktivačních funkcí a parametrů L_2 regularizace. Zde je znatelný nárůst výkonosti při použití ReLU namísto

tanh a sigmoid aktivačních funkcí. ReLU dosahovala v každém tréninku o několik procentních bodů vyšší přesnost než zbývající 2 funkce. Parametry L_2 regularizace vykazují nejlepší přesnosti při použití dvou nejnižších hodnot – 0,0001 a 0,001 – pro vyšší hodnoty klesá úspěšnost o několik procentních bodů.

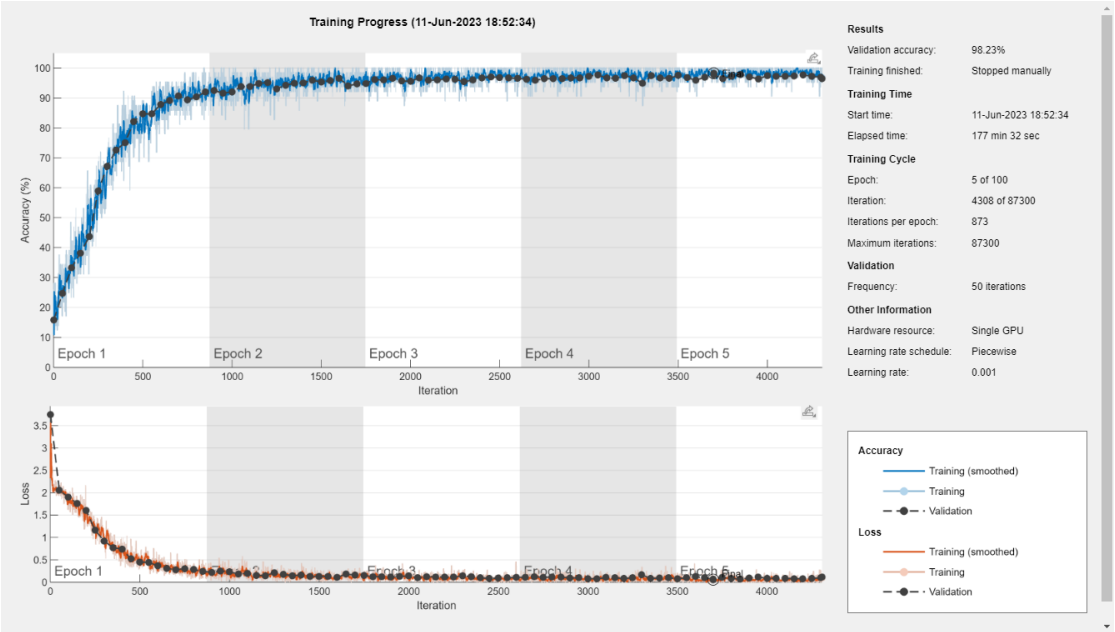
Experiment 7 (Tabulka 7.13) zkoumá vliv velikostí kernelů jednotlivých konvolučních vrstev. Z výsledků však neplynou přesvědčivé závěry. Na celkovém počtu 96 tréninků byl rozdíl mezi nejvyšší a nejnižší přesností přibližně 3 procentní body. Protože je velice složité cokoliv z výsledků vypozorovat, obsahuje tabulka 5 nejlepších a 5 nejhorších kombinací velikostí kernelů. Na prvních příčkách se umístili sítě s velikostí kernelu první konvoluční vrstvy 5×5, naopak na posledních příčkách se jednalo o velikosti 3×3, což může signalizovat potenciál pro vyšší výkonost sítí při použití kernelů větších rozměrů u první konvoluční vrstvy. To lze mimo jiné pozorovat i u předtrénovaných sítí - například GoogLeNet i ResNet obsahuje první konvoluční vrstvu s velikostí kernelu 7×7, AlexNet kernel o velikosti 11×11. Druhá konvoluční vrstva potom obsahuje typicky menší velikost kernelů, což je jev, který lze vysledovat i z mého testování.

Zcela nejlepších hodnot pak dosáhly 2 sítě – z experimentu 4 a z experimentu 5 – a to přesnosti 97,3 %.

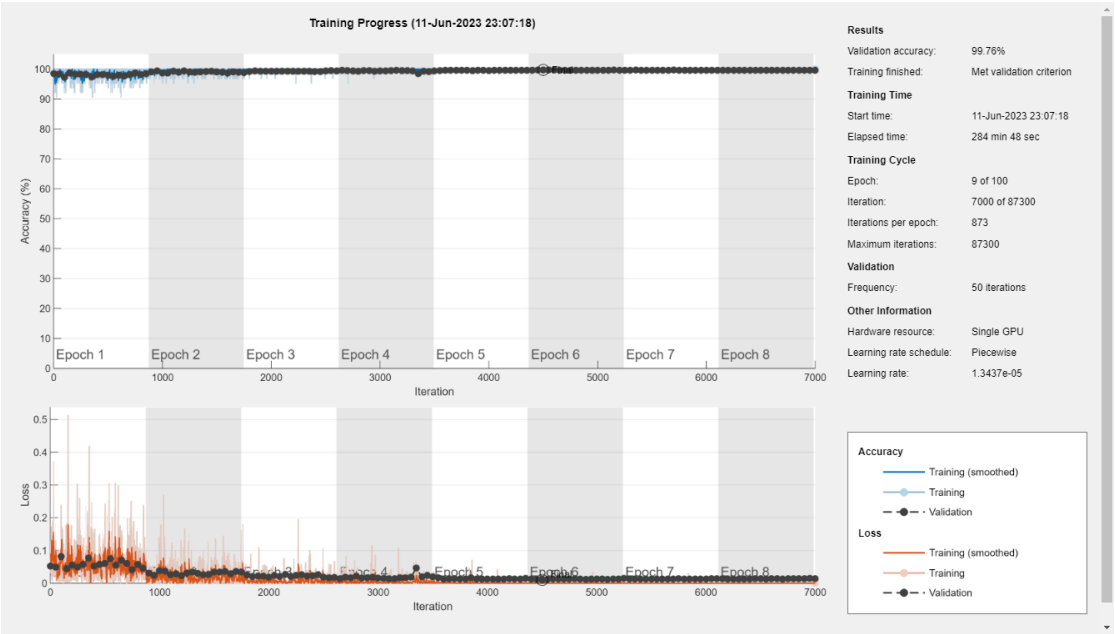
7.3 Inception reziduální CNN

Přesnost na testovacích datech	99,76 %
Chyba na testovacích datech	0,009

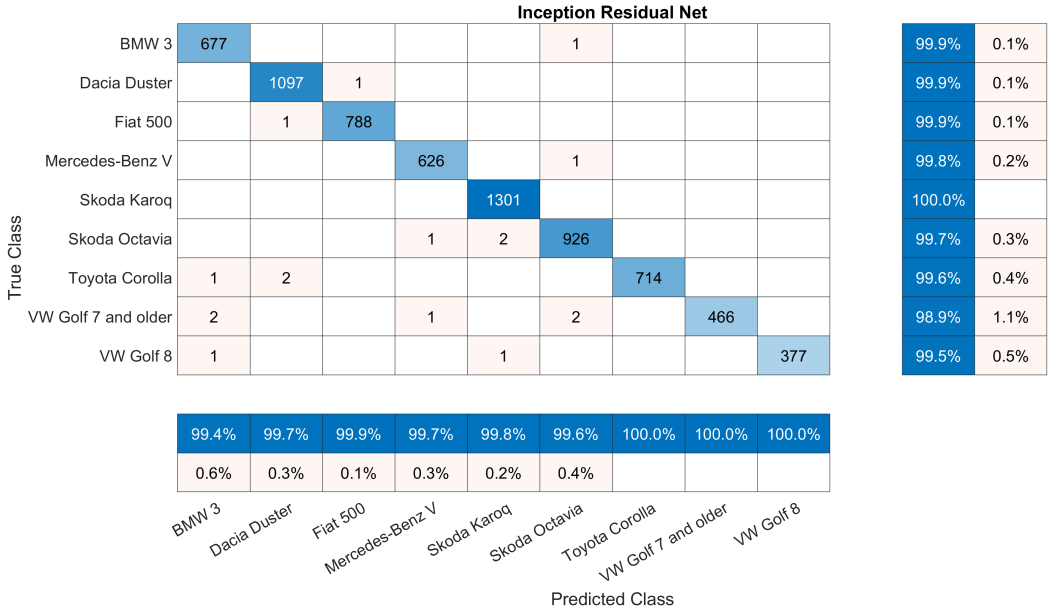
Tabulka 7.14: Výsledky tréninku mé inception reziduální architektury.



Obrázek 7.11: Graf prvního tréninku mé inception reziduální architektury.



Obrázek 7.12: Graf druhého tréninku mé inception reziduální architektury.



Obrázek 7.13: Matice záměn mé inception reziduální architektury.

Tato inception reziduální síť dosáhla velice dobrých výsledků. Po druhém tréninku se sníženým learning ratem na hodnotu 0,0001 dosáhla úspěšnosti na testovacích datech 99,76 %, což je vyšší hodnota než u všech testovaných předtrénovaných sítí. Chybu měla tato síť identickou se sítí ResNet50, a to 0,009. Nutno ale dodat, že předtrénované sítě byly trénovány pouze jednou a s fixními hyperparametry. Velmi zajímavý je fakt, že tato síť zcela nejlépe klasifikovala vozy, které předtrénované sítě

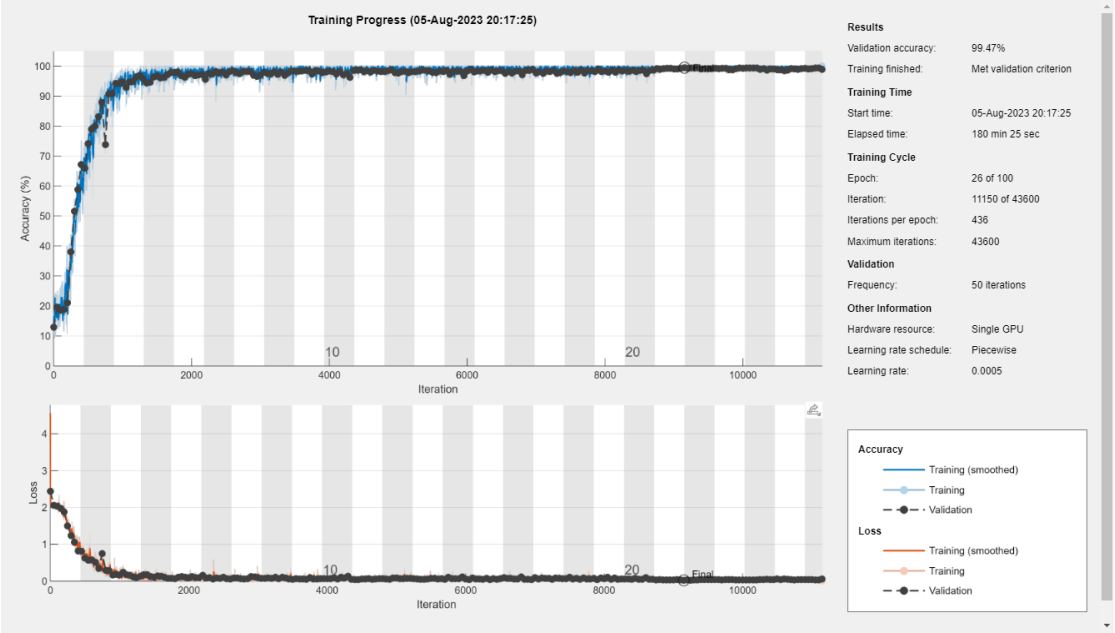
klasifikovaly nejhůře, a sice vozy *Toyota Corolla*, *VW Golf 7 a starší* a *VW Golf 8*, které byly klasifikovány s přesností 100 %. Naopak podobně jako u předtrénovaných sítí byl vůz *VW Golf 7 a starší* nejvíce zaměňován za jiné, přesněji v 1,1 % případů. Síť se také v porovnání s předtrénovanými učila nejdéle - v prvním tréninku po 177 minutách nedosáhla ani 6-té epochy a v druhém tréninku trvalo 8 epoch 284 minut.

7.4 Další testování a komentáře

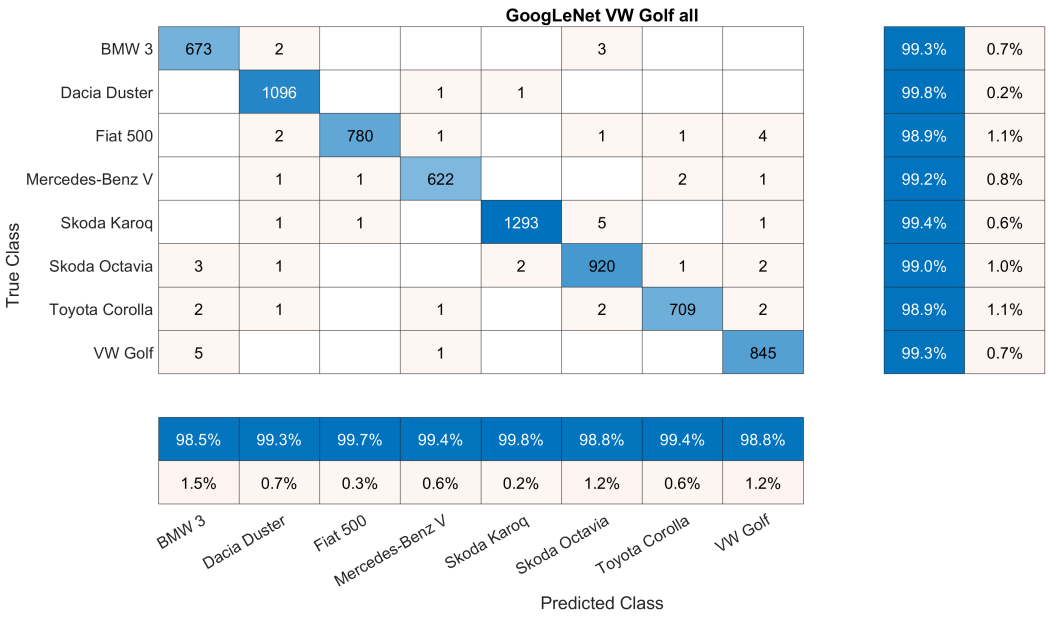
V sekci 5.10 bylo uvedeno, že se v datasetu nachází záměrně rozdělené kategorie *VW Golf 8* a *VW Golf 7 a starší*. Jedná se tedy o stejný vůz jiných generací. V sekci 7.1.1 bylo pozorováno, že nejvíce chybných predikcí se vyskytovalo právě mezi těmito dvěma kategoriemi, což potvrzuje intuitivní předpoklad. Nabízí se tak otázka, zda budou sítě lépe klasifikovat, pokud budou zmíněné dvě kategorie spojené v jednu. Pro toto srovnání bude využita síť GoogLeNet, výsledky se nacházejí v tabulce 7.15 a na obrázcích 7.14 a 7.15.

Přesnost na testovacích datech	99,42 %
Chyba na testovacích datech	0,024

Tabulka 7.15: Výsledky tréninku předtrénované sítě GoogLeNet s modifikovaným datasetem.



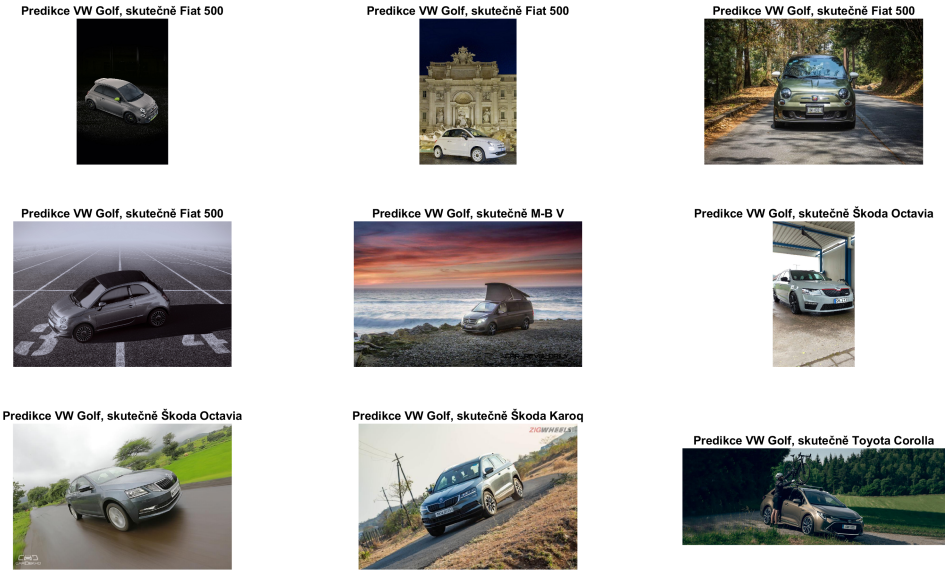
Obrázek 7.14: Graf tréninku předtrénované sítě GoogLeNet s modifikovaným datasetem.



Obrázek 7.15: Matice záměn předtrénované sítě GoogLeNet s modifikovaným datasetem.

Síť vskutku dosáhla lepších výsledků, přesněji úspěšnosti 99,42 % oproti 99,14 % a chyby 0,024 oproti 0,026. Přesnost predikce rozdělených kategorií byla 98,1 % respektive 97,6 %, zatímco přesnost predikce sloučené kategorie je 98,8 %.

Všimněme si, že síť nesprávně klasifikovala 10 obrázků vozu *VW Golf* jako automobily jiné. Zkusil jsem si tedy nechat vykreslit 9 z těchto špatně klasifikovaných obrázků (viz obrázek 7.16).



Obrázek 7.16: Chybně klasifikované obrázky.

Z obrázků však není zjevné, proč byly sítí klasifikovány jako *VW Golf*. U čtyř z nich by se dalo spekulovat, že jsou až příliš širokoúhlé a jejich konverze na velikost 224×224 způsobí silné distorze v obraze. U ostatních 6 však osobně nevidím podobnosti s vozem *VW Golf*, což jen dokládá, jak obtížné je pochopit, jaké jevy konvoluce v obraze zachytávají.

Dále jsem zkusil na síti otestovat 8 vlastních fotek, které nejsou dostupné on-line, a tak nemohli být součástí datasetu. jedná se o 4 fotky vozu *BMW 3* a 4 fotky vozu *Mercedes-Benz V* (ukázky na obrázcích 7.17, 7.18, 7.19, 7.20).

Obrázek 7.17: Vlastní snímek 1 – *BMW 3*.

Obrázek 7.18: Vlastní snímek 2 – *BMW 3*.

Obrázek 7.19: Vlastní snímek 3 – *Mercedes-Benz V*.

Obrázek 7.20: Vlastní snímek 4 – *Mercedes-Benz V*.

Fotky jsem klasifikoval inception reziduální sítí. Všech osm fotek bylo klasifikováno správně, navíc s velmi vysokou pravděpodobností. Na obrázku 7.21 přikládám tabulku pravděpodobností, se kterými síť jednotlivé obrázky klasifikovala. Z tabulky vyplývá, že klasifikací si bylo síť "jistá", neboť pravděpodobnosti pro ostatní kategorie jsou velmi blízké 0.

		Predikce								
		BMW 3	Dacia Duster	Fiat 500	Mercedes-Benz V	Škoda Karoq	Škoda Octavia	Toyota Corolla	VW Golf 7 a starší	VW Golf 8
Skutečnost	BMW 3	1,0000E+00	1,7003E-07	5,5722E-09	2,3898E-11	1,6528E-11	5,5526E-08	1,3953E-09	2,1303E-13	2,5167E-12
	BMW 3	9,9994E-01	5,6518E-08	1,3053E-08	8,7907E-08	8,7616E-08	5,5284E-05	5,4471E-08	1,8607E-07	4,4573E-07
	BMW 3	1,0000E+00	6,8415E-08	5,0928E-09	1,7665E-08	4,3989E-10	4,2470E-08	1,8621E-07	1,9064E-08	3,2748E-09
	BMW 3	9,9999E-01	7,6146E-06	1,7036E-09	9,0398E-10	6,2095E-11	4,2972E-08	3,8382E-06	9,7485E-12	4,6514E-09
	Mercedes-Benz V	9,3525E-07	1,1111E-07	1,7472E-05	9,9998E-01	1,7488E-06	2,1431E-06	1,7982E-07	5,8592E-07	1,0697E-07
	Mercedes-Benz V	9,0204E-11	4,6156E-12	3,0123E-13	1,0000E+00	7,7253E-12	5,4889E-12	5,1272E-12	1,0007E-12	1,1021E-11
	Mercedes-Benz V	9,3525E-07	1,1111E-07	1,7472E-05	9,9998E-01	1,7488E-06	2,1431E-06	1,7982E-07	5,8592E-07	1,0697E-07
	Mercedes-Benz V	3,2979E-14	1,0042E-13	2,1967E-09	1,0000E+00	5,8531E-12	2,6458E-13	4,9271E-14	2,7852E-12	2,1878E-14

Obrázek 7.21: Tabulka pravděpodobností klasifikace vlastních obrázků.

V neposlední řadě zbývá zodpovědět otázku nastolenou v sekci 6.2, a sice, zda je pro podobné problémy výhodnější přístup užití základních architektur s optimalizací hyperparametrů či užití komplexních architektur s hyperparametry nastaveny na nějaké všeobecně doporučované hodnoty. Z mého testování vyplývá, že vhodnějším je druhý uvedený přístup. Nejenže zástupce mé komplexní architektury dosáhl přesnosti 99,76 % a chyby 0,009 oproti nejlepšímu zástupci základních architektur 97,3 % a chyby 0,08, ale navíc je optimalizace hyperparametrů velmi zdlouhavým procesem. Čistý čas všech 291 tréninků trval, s mojí výpočetní kapacitou, okolo 600 hodin, tedy okolo 25 dnů. Na druhou stranu zdaleka nebyla prozkoumána velká část prostoru hyperparametrů, a je tak pravděpodobné, že při zevrubnější optimalizaci by bylo dosaženo lepších výsledků. Z experimentů navíc zcela jistě vyplývá silná kladná korelace mezi přesností klasifikace a počtem konvolučních vrstev. Je tak velice pravděpodobné, že použitím více jak 6-ti konvolučních vrstev by se jednoduše přesnost základních architektur zlepšila. Jinými slovy, výpočetní zdroje byly často

neefektivně užity na průzkum hyperparametrů, které nepříliš ovlivňují celkovou přesnost sítě. Vhodnějším zacílením výpočetních zdrojů se dá předpokládat výraznější zlepšení výsledků.

Závěr

Závěrem mohu konstatovat, že cíl práce byl splněn. Byla zevrubně prostudována teorie neuronových a konvolučních neuronových sítí. Dále byl pečlivě vybrán výtah z teorie a zpracován první polovině práce. Výtah byl vybrán primárně s ohledem na teorii použitou v praktické části práce. Cílem teoretické části bylo nabídnout přehlednou, úplnou, dobře popsanou a jednoduše pochopitelnou teorii neuronových sítích tak, aby práce mohla sloužit jako učební pomůcka. Z tohoto důvod byla velká část věnována příkladům a komentářům, aniž by byly opomenuty matematické definice. Důkazy definic byly použity pouze tam, kde významně přispěly k pochopení celé tématiky.

V další části byla popsána tvorba datasetu. Přes veškeré počáteční pochybnosti byl vytvořen dataset o vyšších jednotkách tisíců snímků pro každou kategorii. Popsané byly celkem 4 webscrapingové metody, které umožnili takové množství snímku jedné kategorie obstarat. Dále byly popsány úspěšné metody a techniky eliminace chybných či nechtěných snímků. Těmito metodami byla významně zkrácena doba manuální kontroly jednotlivých obrázků.

6-tá kapitola byla věnována popisu architektur použitých konvolučních neuronových sítí. Důkladně popsány a srovnány byly předtrénované sítě. Čerpal jsem pouze z originálních článků a vždy se snažil poukázat a komentovat techniky, které jsou v sítích použity a proč. Druhá část této kapitoly se věnuje popisu vlastních architektur. Krom všech provedených experimentů je popsána motivace, použité vrstvy a samotná architektura inception reziduální konvoluční neuronové sítě. Všechny architektury sítí jsou jednotným stylem ilustrovány, což výrazně přispívá celkové přehlednosti.

V poslední kapitole jsou prezentovány výsledky. Byly porovnány všechny aspekty tréninku předtrénovaných sítí. Komentována byla nejen přesnost a chybovost sítí, ale i časová náročnost, rychlost konvergence, úspěšnost predikce v rámci jednotlivých kategorií. Popis byl vždy podpořen přehlednými grafy a maticemi záměn. Dále byly komentovány výsledky experimentů optimalizací hyperparametrů základních CNN. Bylo dosaženo částečného zmapování prostoru hyperparametrů a zjištěn vliv jednotlivých hyperparametrů na výkonost sítě. Další část této kapitoly je věnována inception reziduální síti. Za úspěch považuji vysokou přesnost klasifikace, kterou tato síť dosáhla. V neposlední řadě bylo prezentováno několik menších pozorování praktického charakteru, které pomáhají dokreslovat obrázek o celkové výkonnosti konovlučních neuronových sítí.

Práce nabízí velké množství rozšíření. Atraktivním tématem jsou rekurentní nebo mělké neuronové sítě, které se stávají stále populárnější. Dále by bylo zajímavé

aplikovat statistické a heuristické metody v rámci optimalizace hyperparametrů sítí. V neposlední řadě je tu také velký prostor k zdokonalení a automatizaci tvorby datasetu.

Touto prací jsem převážně splnil úkol sám pro sebe – prostudoval jsem teorii neuronových sítí, získal jsem znalosti o jejich funkci a tvorbě jejich architektur. Osvojil jsem si používání CNN v programu MATLAB a naučil se srovnávat jejich výkonnost. Oblast neuronových sítí mě vděk této práci nadchla a rád bych se jí věnoval i nadále, a to ať už v akademickém nebo komerčním prostředí.

Literatura

[1] ALOM, M.Z., M. HASAN, C. YAKOPCIC, T.M. TAHA a V.K. ASARI. Improved inception-residual convolutional neural network for object recognition [online]. Dayton, OH, USA, 2018, 04.8.2018, 32, 279–293 [cit. 2023-07-30]. Dostupné z: doi:https://doi.org/10.1007/s00521-018-3627-6

[2] BUCHNER, J. ImageHash 4.3.1 [online]. [cit. 2023-08-06]. Dostupné z: https://pypi.org/project/ImageHash/

[3] Custom Search JSON API. Programmable Search Engine: Guides [online]. [cit. 2023-08-01]. Dostupné z: https://developers.google.com/custom-search/v1/overview

[4] DENG, J., W. DONG, R. SOCHER, L. LI, K. LI a F. LI. ImageNet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition. Miami, FL, USA: IEEE, 2009, s. 248-255. ISBN 978-1-4244-3992-8. Dostupné z: doi:https://doi.org/10.1109/CVPR.2009.5206848

[5] DENG, L. The MNIST Database of Handwritten Digit Images for Machine Learning Research. IEEE Signal Processing Magazine [online]. IEEE, 2012, 29(6), 141-142 [cit. 2023-08-01]. ISSN 1558-0792. Dostupné z: doi:https://doi.org/10.1109/MSP.2012.2211477

[6] FITAS, R., B. ROCHA, V. COSTA a A. SOUSA. Design and Comparison of Image Hashing Methods: A Case Study on Cork Stopper Unique Identification. Imaging [online]. 2021, 7(48) [cit. 2023-07-26]. Dostupné z: doi:https://doi.org/10.3390/jimaging7030048

[7] GILSON, A., C. SAFRANEK, T. HUANG, V. SOCRATES, L. CHI, R. TAYLOR a D. CHARTASH. How Does ChatGPT Perform on the United States Medical Licensing Examination? The Implications of Large Language Models for Medical Education and Knowledge Assessment. JMIR Med Educ [online]. (9), e45312 [cit. 2023-08-01]. ISSN 2369-3762. Dostupné z: doi:https://doi.org/10.2196/45312

[8] GLOROT, X. a Y. BENGIO. Understanding the difficulty of training deep feedforward neural networks. Journal of Machine Learning Research - Proceedings Track [online]. 2010, 9, 249-256 [cit. 2023-07-22]. Dostupné z: https://www.researchgate.net/publication/215616968_Understanding_the_difficulty_of_training_deep_feedforward_neural_networks

[9] GLOROT, X., A. BORDES a Y. BENGIO. Deep Sparse Rectifier Neural Networks. Journal of Machine Learning Research [online]. 2010, 15 [cit. 2023-07-22]. Dostupné z: https://www.researchgate.net/publication/215616967_Deep_Sparse_Rectifier_Neural_Networks

[10] GOODFELLOW, I., Y. BENGIO a A. COURVILLE. Deep Learning [online]. MIT Press, 2016 [cit. 2023-08-01]. Dostupné z: http://www.deeplearningbook.org

[11] HE, K., X. ZHANG, S. REN a J. SUN. Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA: IEEE, 2016, s. 770-778. ISBN 978-1-4673-8851-1. Dostupné z: doi:https://doi.org/10.1109/CVPR.2016.90

[12] HERCULANO-HOUZEL, Suzana. The Human Brain in Numbers: A Linearly Scaled-up Primate Brain. Frontiers in human neuroscience [online]. 2009, 3(31) [cit. 2023-07-14]. Dostupné z: doi:https://doi.org/10.3389/neuro.09.031.2009

[13] HUXLEY, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. The Journal of Physiology [online]. 28.8.1952, 117(4), 500-544 [cit. 2023-07-13]. Dostupné z: doi:https://doi.org/10.1113/jphysiol.1952.sp004764

[14] HUXLEY, A. F. The components of membrane conductance in the giant axon of Loligo. The Journal of Physiology [online]. 28.4.1952, 116(4), 473-496 [cit. 2023-07-13]. Dostupné z: doi:https://doi.org/10.1113/jphysiol.1952.sp004718

[15] IANDOLA, F., M. MOSKEWICZ, K. ASHRAF, S. HAN, W. DALLY a K. KEUTZER. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and textless1MB model size [online]. DEEPSCALE. 2016 [cit. 2023-07-30]. Dostupné z: doi:https://doi.org/10.48550/arXiv.1602.07360

[16] ITO, Takuya, Tim KLINGER, Doug SCHULTZ, Michael COLE a Mattia RIGOTTI. Compositional generalization through abstract representations in human and artificial neural networks. Curran Associates, Inc.: Advances in Neural Information Processing Systems [online]. 2022, 35, 32225-32239 [cit. 2023-07-21]. Dostupné z: https://proceedings.neurips.cc/paper_files/paper/2022/file/d0241a0fb1fc9be477bdfde5e0da276a-Paper-Conference.pdf

[17] JARRETT, Kevin, Koray KAVUKCUOGLU, Marc'Aurelio RANZATO a Yann LECUN. What is the best multi-stage architecture for object recognition?. 2009 IEEE 12th International Conference on Computer Vision [online]. Kyoto, Japan: IEEE, 2009, 2146-2153 [cit. 2023-07-22]. ISSN 2380-7504. Dostupné z: doi:https://doi.org/10.1109/ICCV.2009.5459469

[18] KRAUS, David. Concepts in Modern Biology. New York: Globe Book Company, 1969. ISBN 0835948390.

[19] KRAUSE, J., M. STARK, J. DENG a L. FEI-FEI. 3D Object Representations for Fine-Grained Categorization. In: 2013 IEEE International Conference on Computer Vision Workshops. Sydney, NSW, Australia: IEEE, 2013, s. 554-561. ISBN 978-1-4799-3022-7. Dostupné z: doi:https://doi.org/10.1109/ICCVW.2013.77

[20] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey E. HINTON. ImageNet classification with deep convolutional neural networks. Communications of the ACM [online]. New York, NY, United States: Association for Computing Machinery, 2017, 2012, 60(6), 84–90 [cit. 2023-07-22]. ISSN 0001-0782. Dostupné z: doi:https://doi.org/10.1145/3065386

[21] KOLEN, John F. a Stefan C. KREMER. Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies. In: A Field Guide to Dynamical Recurrent Networks. Wiley-IEEE Press, 2001, s. 237-243. ISBN 9780780353695. Dostupné z: doi:https://doi.org/10.1109/9780470544037.ch14

[22] Lecun, Y., L. Bottou, Y. Bengio a P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE [online]. 1998, 86(11), 2278-2324 [cit. 2023-07-21]. Dostupné z: doi:https://doi.org/10.1109/5.726791

[23] LIN, M., Q. CHEN a S. YAN. Network In Network. CoRR [online]. 2013 [cit. 2023-07-28]. Dostupné z: doi:https://doi.org/10.48550/arXiv.1312.4400

[24] MYERS G., David. Psychology. 4th Edition. New York: Worth Publishers, 1995. ISBN 9780716764281.

[25] MWANDAU, Brian. Investigating Keystroke Dynamics as a Two-Factor Biometric Security. 2018. Disertační práce. Strathmore University. Dostupné z: https://www.researchgate.net/publication/325870973_Investigating_Keystroke_Dynamics_as_a_Two-Factor_Biometric_Security

[26] NIELSEN, M.A. Neural Networks and Deep Learning [online]. Determination Press, 2015 [cit. 2023-08-01]. Dostupné z: http://neuralnetworksanddeeplearning.com/

[27] Pinterest data scraper [Python library] [online]. [cit. 2023-08-01]. Dostupné z: https://pypi.org/project/pinscrape/

[28] REDMON, J. a A. FARHADI. YOLO9000: Better, Faster, Stronger. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Honolulu, HI, USA: IEEE, 2017, s. 6517-6525. ISBN 978-1-5386-0457-1. Dostupné z: doi:https://doi.org/10.1109/CVPR.2017.690

[29] ROSENBLATT, F. The perceptron - A perceiving and recognizing automaton. Cornell Aeronautical Laboratory. Ithaca, New York, 1957, Technický report 85-460-1.

[30] RUMELHART, D., G. HINTON a R. WILLIAMS. Learning representations by back-propagating errors. Nature [online]. 9.10.1986, 323, 533–536 [cit. 2023-07-18]. Dostupné z: doi:https://doi.org/10.1038/323533a0

[31] SANTURKAR, S., D. TSIPRAS, A. ILYAS a A. MADRY. How Does Batch Normalization Help Optimization? NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems [online]. 2018, 2488–2498 [cit. 2023-08-02]. Dostupné z: doi:https://doi.org/10.48550/arXiv.1909.09139

[32] SEBASTIAN, S. An overview of gradient descent optimization algorithms [online]. NUI Galway, 2017 [cit. 2023-08-03]. Dostupné z: doi:https://doi.org/10.48550/arXiv.1609.04747

[33] SHETTY, H.U., T.A. ROSENBERGER a A.D. PURDON. Energy Consumption by Phospholipid Metabolism in Mammalian Brain. Neurochem Research [online]. 2002, 27, 1641–1647 [cit. 2023-07-14]. Dostupné z: doi:https://doi.org/10.1023/A:1021635027211

[34] SINGH, P. Bing Image Downloader [Python library] [online]. [cit. 2023-08-01]. Dostupné z: https://pypi.org/project/bing-image-downloader/

[35] SZANDA, T., BHOI, A., P. MALLICK, C.M. LIU a V.E. BALAS, ed. Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. In: Bio-inspired Neurocomputing. Vol 903. Singapore: Springer Singapore, 2021, 203–224. ISBN 978-981-15-5495-7. Dostupné z: doi:https://doi.org/10.1007/978

[36] SZEGEDY, Ch., L. WEI, J. YANGQING, et al. Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Boston, MA, USA: IEEE, 2015, s. 1-9. ISBN 978-1-4673-6964-0. Dostupné z: doi:https://doi.org/10.1109/CVPR.2015.7298594

[37] SZEGEDY, Ch., S. IOFFE, V. VANHOUCKE a A. ALEMI. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. Proceedings of the AAAI Conference on Artificial Intelligence [online]. 2017, 31(1) [cit. 2023-08-02]. Dostupné z: doi:https://doi.org/10.1609/aaai.v31i1.11231

[38] THE MATHWORKS INC. What Is Transfer Learning?. MathWorks Discovery - Transfer Learning [online]. Natick, Massachusetts, United States: The MathWorks [cit. 2023-07-27]. Dostupné z: https://www.mathworks.com/discovery/transfer-learning.html

[39] We Need To Go Deeper [online]. 2010 [cit. 2023-07-28]. Dostupné z: https://knowyourmeme.com/memes/we-need-to-go-deeper

[40] ZEILER, M.D. a R. FERGUS, FLEET, D., T. PAJDLA, B. SCHIELE a T. TUYTELAARS, ed. Visualizing and Understanding Convolutional Networks. In: Computer Vision – ECCV 2014. Vol 8689. Cham: Springer International Publishing, 2014, 818–833. ISBN 978-3-319-10590-1. Dostupné z: doi:https://doi.org/10.1007/978-3-319-10590-1_53

[41] ZHONG, W., K. QIFA, M. ISARD a S. JIAN. Bundling features for large scale partial-duplicate web image search. In: 2009 IEEE Conference on Computer

Vision and Pattern Recognition. Miami, FL: IEEE, 2009, s. 25-32. ISBN 978-1-4244-3992-8. Dostupné z: doi:https://doi.org/10.1109/CVPR.2009.5206566

[42] ZYTE GROUP LIMITED. Scrapy: An open source and collaborative framework for extracting the data you need from websites. In a fast, simple, yet extensible way. [online]. [cit. 2023-08-01]. Dostupné z: https://scrapy.org/