VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Jiří Beneš

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

UNÁRNÍ KLASIFIKÁTOR OBRAZOVÝCH DAT
UNARY CLASSIFICATION OF IMAGE DATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Jiří Beneš

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Karel Horák, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Jiří Beneš ***ID:*** 186029

Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Unární klasifikátor obrazových dat

POKYNY PRO VYPRACOVÁNÍ:

Práce se zabývá architekturami klasifikátorů vhodných pro unární (jednotřídní) klasifikaci obrazových dat:

1. Nastudujte oblast klasifikačních metod pro unární, binární a obecnou multi-class klasifikaci.
2. Sestavte seznam architektur vhodných pro unární klasifikaci se zevrubným popisem funkce a příklady použití.
3. Dle pokynů vedoucího pořiďte vlastní anotovaný dataset průmyslového výrobku čítající 1000+ realizací.
4. Pro vybraný mechanismus provedte (re)implementaci a uzpůsobení datům v jazyce Python, tj. sestavte jednoúčelovou aplikaci pro ověření funkce klasifikátoru.
5. Pomocí vytvořené aplikace proveďte na pořízeném datasetu sadu trénovacích a testovacích experimentů s různým nastavením hyperparametrů a dělením datasetu.
6. Výsledky experimentů vyhodnoťte formou matice záměn.

DOPORUČENÁ LITERATURA:

1. One-class classification: taxonomy of study and review of techniques. URL: https://doi.org/10.1017/S026988891300043X.

2. One-class classification - Concept-learning in the absence of counter-examples. URL: http://homepage.tudelft.nl/n9d04/thesis.pdf

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 4.8.2021

Vedoucí práce: Ing. Karel Horák, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

Abstrakt

Práce se zabývá úvodem do klasifikačních algoritmů. Následně rozděluje klasifikátory na unární, binární a multi-class a popisuje jednotlivé typy klasifikátorů. Práce srovnává jednotlivé klasifikátory a jejich oblasti použití. Pro unární klasifikátory jsou v práci uvedeny praktické příklady a seznam využívaných architektur. Práce obsahuje kapitolu zaměřenou na srovnání vlivů hyperparametrů na kvalitu unární klasifikace pro jednotlivé architektury. Součástí odevzdání práce je potom praktický příklad implementace unárního klasifikátoru.

Klíčová slova

unární klasifikátor, obrazová data, klasifikace, binární klasifikátory, multi-class klasifikátory, implementace unárního klasifikátoru, neuronové sítě, hyperparametry

Abstract

The work deals with an introduction to classification algorithms. It then divides classifiers into unary, binary and multi-class and describes the different types of classifiers. The work compares individual classifiers and their areas of use. For unary classifiers, practical examples and a list of used architectures are given in the work. The work contains a chapter focused on the comparison of the effects of hyperparameters on the quality of unary classification for individual architectures. Part of the submission is a practical example of implementation of the unary classifier.

Keywords

unary classifier, image data, classification, binary classifiers, multi-class classifiers, implementation of unary classifier, neural networks, hyperparameters

Bibliografická citace

Citace tištěné práce:

BENEŠ, Jiří. *Unární klasifikátor obrazových dat.* Brno, 2021. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/136692. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Karel Horák.

Citace elektronického zdroje:

BENEŠ, Jiří. *Unární klasifikátor obrazových dat* [online]. Brno, 2021 [cit. 2021-07-30]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/136692. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Karel Horák.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	Bc. Jiří Beneš
VUT ID studenta:	186029
Typ práce:	Diplomová práce
Akademický rok:	*2020/21*
Téma závěrečné práce:	Unární klasifikátor obrazových dat

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: 1. srpna 2021 ----------

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Karlu Horákovi, Ph. D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

Děkuji Ing. Tomáši Zemčíkovi, za odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: 1. srpna 2021 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Na úvod této práce je důležité si definovat pojem strojové učení, jeho oblasti použití a vývoj. Citace: „Strojové učení lze definovat jako nauku o nástrojích, které umožňují učení umělých objektů.“ [1]. Tuto definici lze chápat tak, že strojové učení popisuje matematické nástroje pro vytvoření modelu dat na základě získaných zkušeností, s cílem co nejvíce zlepšit výkonnost daného modelu.

Strojové učení bylo vytvořeno za účelem zdokonalení a zrychlení zpracovávání dat. Se strojovým učením je možné se v dnešní době nejčastěji setkat v oblastech syntézy či komprimaci obrazu, rozpoznávání řeči, u hlasové syntézy, kde se využívá pro tvorbu vyhledávacích algoritmů, případně pro třídění dat [2]. Další oblastí využití strojového učení jsou počítačové hry. Zde má strojové učení za cíl konkurovat člověku. Pro tuto oblast bylo strojové učení původně vyvinuto. V roce 1959 Arthur Samuel ze společnosti IBM, poprvé aplikoval strojové učení na hraní dámy [3]. V následujících letech došlo k rozšíření strojového učení do spousty odvětví. V dnešní době se s ním můžeme setkat například u prohlížeče Google (obrazový vyhledávač Google). Společnost IBM vyvíjí strojové učení pro lékařské účely, určení anamnézy na základě diagnózy, nebo pro optimalizaci léčebných postupů. Spousta firem uvažuje v blízké budoucnosti o masivním nasazení strojového učení i do běžných lidských úkonů. Autonomní řídící asistenti od společnosti Tesla, průmysl 4.0 nebo implementace strojového učení do řízení podniku. Je tedy patrné, že jediné, co brání masivnímu využívání strojového učení, je potřebný výpočetní výkon a výkonná bezdrátová konektivita.

Tato práce se zabývá klasifikačními algoritmy a definováním jednotlivých typů unárních klasifikátorů. První kapitola práce se tak zabývá definováním klasifikační funkce a jednotlivým typům unární klasifikace.

Druhá kapitola se věnuje metodám unární klasifikace. V této kapitole jsou metody unární klasifikace rozděleny na metody hustoty, hraniční metody a rekonstrukční metody unární klasifikace.

Třetí kapitola práce následně definuje jednotlivé vrstvy neuronových sítí, jednotlivé funkce a hyperparametry. Tato kapitola práce poskytuje teoretický základ pro praktickou část práce. Tato kapitola také obsahuje seznam nejčastěji používaných CNN architektur spolu s příkladem použití.

Cílem poslední kapitoly této práce je ověření funkčnosti dané aplikace a ověření vlivu hyperparametrů na kvalitu klasifikace. Hyperparametry jsou takové parametry, které mají vliv na výslednou přesnost klasifikace, ale nelze dopředu stanovit jejich hodnotu.

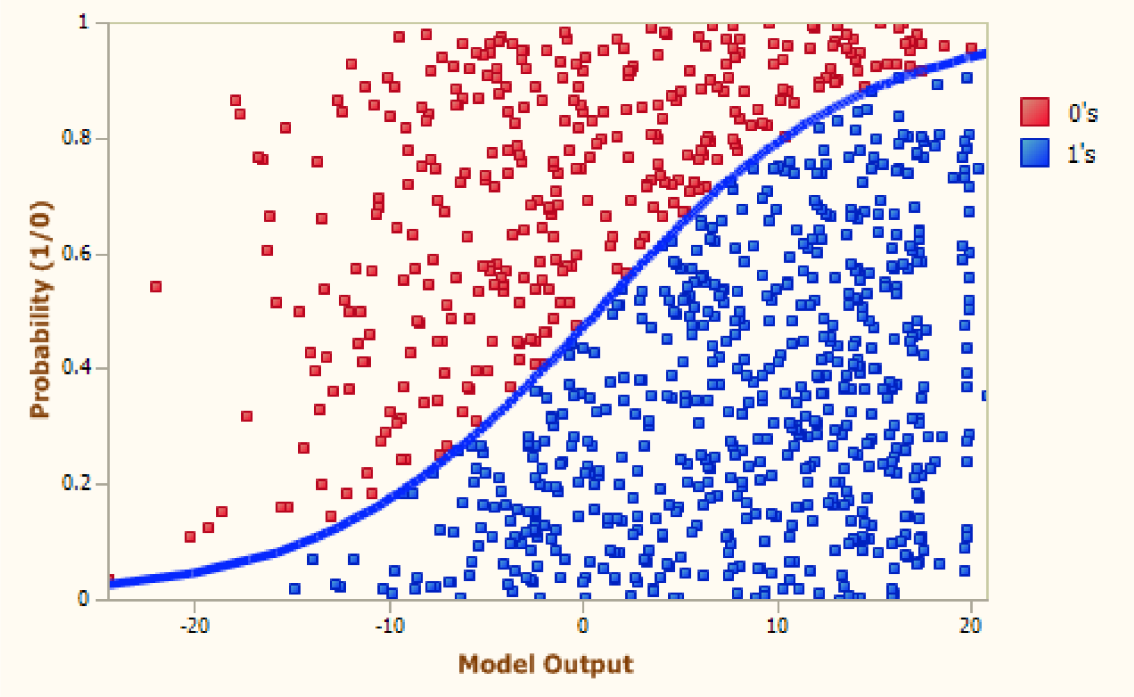
Praktickým cílem práce je vytvoření anotovaného obrazového datasetu reálného průmyslového výrobku a aplikace pro trénink a vyhodnocení přesnosti klasifikace CNN sítí. Výsledná přesnost klasifikace každé natrénované CNN sítě je vyhodnocena pomocí matice záměny.

1. KLASIFIKÁTOR

1.1 Definice klasifikátoru

Jednotlivé klasifikační metody lze rozdělit několika způsoby, nejčastěji se však setkáme s rozdělením na unární, binární a multi-třídní klasifikátory. Tímto způsobem lze rozdělit klasifikátory na základě množství výstupních klasifikovaných množin. Před samotným popisem jednotlivých klasifikátorů je nutné klasifikátor definovat.

Na obrázku 1 je možné vidět obecný příklad klasifikace. Jak je patrné z obrázku 1, tak klasifikátor rozděluje klasifikovaný prostor na dvě části pomocí klasifikační funkce. V první části se nachází pozitivně klasifikované objekty (1), zatímco v druhé části se nachází ostatní objekty (0). Klasifikační funkci lze definovat jako funkci f(x,w), kde x je sloupcový vektor popisující klasifikované objekty a parametr w definuje aktuální nastavení vah klasifikátoru. [4]



Obrázek 1: Rozdělení klasifikovaného prostoru pomocí klasifikační funkce [4]

1.2 Vyhodnocení kvality klasifikace

Samotné vyhodnocení přesnosti klasifikace lze provést několika metodami, vždy se vyhodnocuje rozdíl mezi výstupem modelu a skutečnou výstupní hodnotou pro daný objekt. Těchto metod potom lze využít při trénování modelu, kdy hledané parametry klasifikační funkce $f(x_i,w)$ volíme tak, aby výsledná chyba klasifikace byla co nejmenší. Samotná klasifikační funkce $f(x_i,w)$ je závislá na vstupech (x_i) a aktuálním natavení vah modelu (w).

První možností výpočtu klasifikační chyby je diskrétní klasifikace. Chyba diskrétní klasifikace ($\varepsilon_{0\text{-}1}$) je rovna jedné pokud se výstup klasifikační funkce ($f(x_i,w)$) nerovná skutečné hodnotě výstupu (y_i) :

$$\varepsilon_{0-1}(f(x_i,w),y_i) = \begin{cases} 0, & pokud\ f(x_i,w) = y_i \\ 1, & jinak \end{cases} \quad [1](1.1)$$

Nejběžnější chybou pro klasifikaci kvantitativní výstupní veličiny je střední kvadratická chyba (MSE). Střední kvadratická chyba (ε_{MES}) je rovna druhé mocnině rozdílu výstupu klasifikační funkce ($f(x_i,w)$) a skutečného výstupu (y_i):

$$\varepsilon_{MSE}(f(x_i,w),y_i) = (f(x_i,w) - y_i)^2 \quad [1](1.2)$$

Třetím typem výpočtu klasifikační chyby je křížová entropie, porovnává se rozdíl mezi modelem ($f(x_i,w)$) a skutečným rozložením (y_i), kde skutečná hodnota výstupu nabývá hodnot $y_i = \{0, 1\}$:

$$\varepsilon_{CE}(f(x_i,w),y_i) = f(x_i,w)^{y_i}(1 - f(x_i,w))^{1-y_i} \quad [1](1.3)$$

Minimalizací chyby na tréninkových datech se snažíme najít ideální parametry modelu. Tato metoda však představuje nový problém. Sada tréninkových dat může být zadána velice specificky. Například může obsahovat jenom velice malý počet dat nebo velké odchylky, a to až do takové míry, že nelze stanovit přesný model. Čím méně dat je obsaženo v tréninkové sadě, tím výraznější je tento problém. Cílem návrhu modelu tedy není vytvořit model, který si dokáže nejlépe poradit s daty obsaženými v tréninkové sadě, ale model, který si dokáže poradit s novými daty, které chceme klasifikovat. Hlavním cílem tedy je najít model, který vykazuje dobré zobecnění (anglicky „generalization“). Takovýto model dokáže správně klasifikovat i data neobsažená v tréninkové sadě. Chceme-li odhadnou, jak dobře model zobecňuje, je nutné jej testovat novou sadou dat, která nebyla použita při tréninku. Tím že použijeme rozdílná data také zjistíme skutečnou hodnotu přesnosti klasifikace modelu. Sada dat, která je použita pro odhad chyby klasifikátoru, se nazývá testovací sada. [5]

1.3 Zobecnění

Jak bylo uvedeno v předchozí kapitole, zobecnění je vlastnost modelů, které popisuje jejich schopnost klasifikovat data, která nebyla obsažena v tréninkové sadě. Pro vyhodnocení kvality zobecnění je použito Bayesovo rozhodovací pravidlo. Toto pravidlo přiřadí každému objektu (x) v tréninkových vstupních datech hodnotu na základě pravděpodobnosti (p(ω|x)). Kde pravděpodobnost (p(ω|x)) definuje s jakou pravděpodobností patří objekt (x) do dané třídy (ω). Pro Bayesovskou funkci potom platí (f_{Bayes})[1]:

$$f_{Bayes} = \begin{cases} +1, & když \quad p(\omega_1|x) \geq p(\omega_2|x) \\ -1, & když \quad p(\omega_1|x) < p(\omega_2|x) \end{cases} \qquad [1]\ (1.4)$$

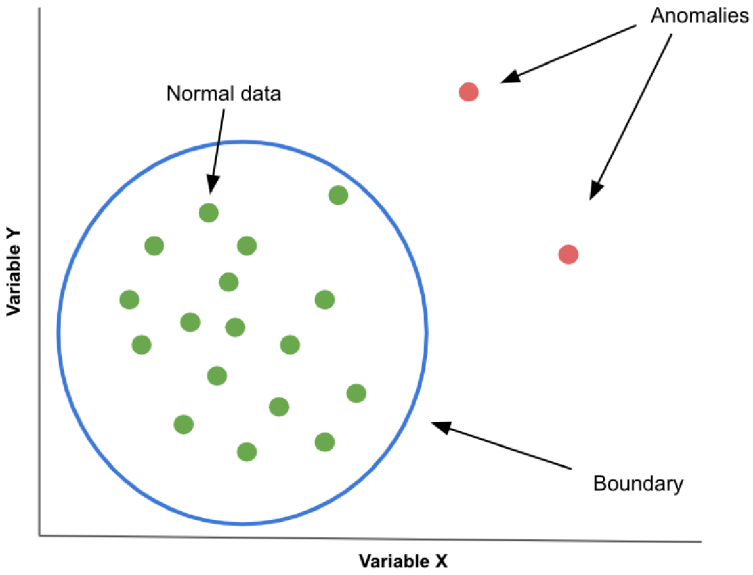
Bayesovo pravidlo je teoreticky nejoptimálnější klasifikační pravidlo za předpokladu, že všechny chybně klasifikované objekty mají stejnou váhu. Problém je, že Bayesovo pravidlo potřebuje skutečnou pravděpodobnost všech tříd pro všechny testované objekty (x). V praxi není skutečná distribuce často známá a jsou k dispozici pouze příklady této distribuce (může se také jednat jenom o atypické příklady a nemáme způsob, jak je ověřit). S výjimkou použití uměle vygenerovaných dat, u kterých známe distribuci a rozdělení tříd, nelze Bayesovo rozdělení v praxi použít. V praxi je skutečná chyba (ε_{true}) nahrazena aproximací empirické chyby tréninkových dat (ε_{emp}). Pro tuto aproximaci je nutné použít dostatečně velký vzorek tréninkových dat (N), která mají shodnou distribuci chyby se skutečnými daty ($\varepsilon(f(x_i, w), y_i)$). Optimalizací empirické chyby (ε_{emp}) není zaručena optimalizace skutečné chyby (ε_{true}), empirickou chybu lze stanovit jako:

$$\varepsilon_{emp}(f, w, X^{tr}) = \frac{1}{N}\sum_i \varepsilon(f(x_i, w), y_i) \qquad [1]\ (1.5)$$

Na závěr lze tedy stanovit, že cílem je za pomocí tréninkových dat vytvořit model, který bude vykazovat dobré zobecnění pro všechna data přes celý datový prostor. V praxi se potom setkáme s postupným návrhem modelu, kdy se nejprve navrhne jednoduchá funkce, u které dochází ke změně jejích parametrů a je měřena celková chyba klasifikace, pokud se ukáže změna parametrů jako nedostatečná je zvolena složitější funkce a celý postup se opakuje.

1.4 Unární klasifikátory

Prvním zde definovaným klasifikátorem je unární klasifikátor. Tento typ klasifikátoru rozlišuje pouze jednu třídu objektů tzv. cílové objekty (zelené body na obrázku 2) a všechny ostatní objekty jsou označena za odlehlé objekty (červené body na obrázku 2). Plná čára zde reprezentuje možný model, který rozlišuje mezi cílovými a odlehlými objekty. [6]

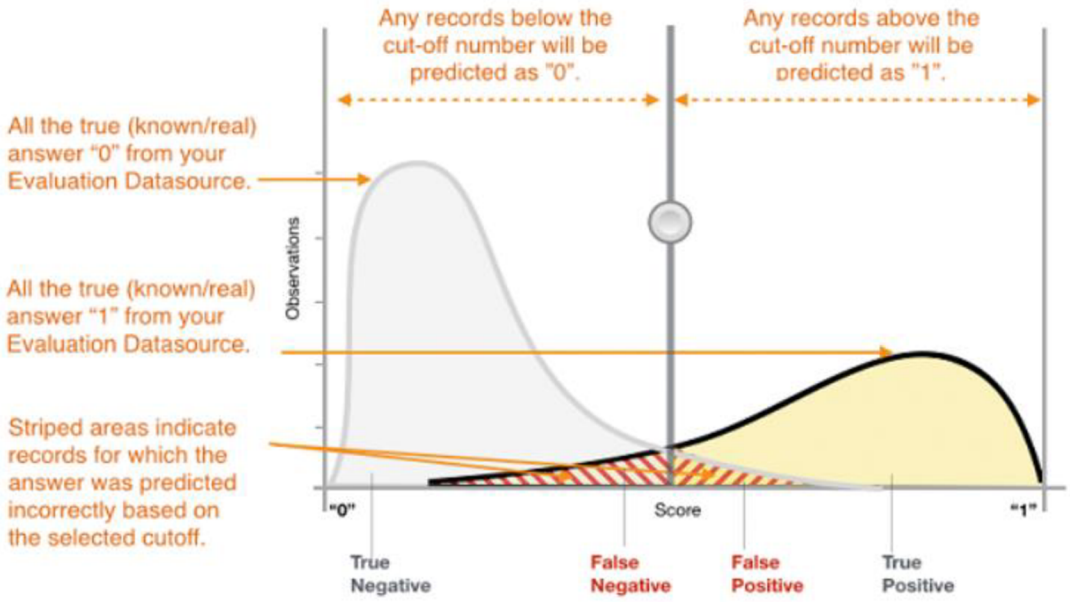


Obrázek 2: Rozdělení objektů na cílové a odlehlé pomocí unární klasifikace [6]

V anglicky psané literatuře můžeme najít několik různých názvů („outlier detection“, „novelty detection“ nebo „concept learning“), svůj název odvozují podle oblasti použití. Všechny pracují na stejném principu, snaží na detekovat neobvyklé objekty v klasifikovaných datech. Tato neobvyklá data mohou být následkem chyb měření, to má za následek výrazný rozdíl oproti zbylým hodnotám v tréninkové sadě. Obecně potom lze stanovit, že trénovaný model dokáže poskytnou spolehlivý výstup pouze pro data, která se podobají tréninkové sadě. V případě, že chceme klasifikovat odlehlá data, můžeme použít například neuronové sítě, které jsou schopny odhadnou pravděpodobnost a jsou tedy schopny poskytnout vysoce spolehlivé výstupy i pro objekty, které jsou odlehlé od tréninkové sady. Občas může dojít ke stavu, kdy v jedné třídě objektů je dostatečný počet dat, zatímco v druhé třídě je jich nedostatek. Může se jednat o měření, kdy během normálního provozu máme dostatečný počet měření, zatímco během poruchových stavů jsme schopni získat pouze omezený počet dat. Navíc pokud bychom chtěli klasifikovat všechny poruchové stavy, museli bychom tyto poruchové stavy vyvolat, což v řadě případu není možné. Proto se používá právě metoda unární klasifikace, kdy unární klasifikátor je natrénován na datech z normálního provozu, pokud tedy následně detekuje odlehlí objekt, dojde k vyhodnocení chybového stavu a už není potřeba zjišťovat, jaký stav přesně nastal. [6]

1.5 Binární klasifikátory

Druhým typem klasifikace je binární klasifikace. Na rozdíl od unární klasifikace je zde model možné definovat z obou stran (ne jenom ze strany cílových objektů, žlutá oblast na obrázku 3). Toho můžeme využít při definování modelu a při jeho vyhodnocení.

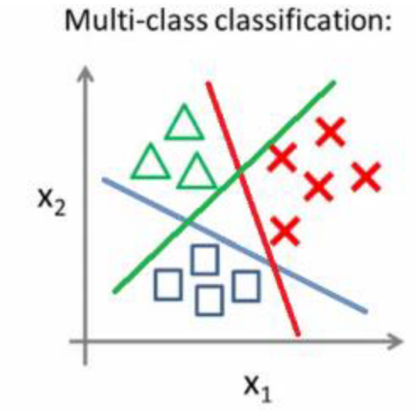


Obrázek 3: Binární klasifikátor – rozložení výsledků a příklad chyby klasifikace [7]

Na obrázku 3 je možné si ukázat princip binární klasifikaci. Jednotlivým objektům je zde přiřazena binární hodnota 0 nebo 1. Tato hodnota je zde objektům přiřazována z důvodu možného výpočtu chyby. Při binární klasifikaci potom dochází k falešné negativitě (chybně klasifikované pozitivní objekty) a falešné pozitivní (chybně klasifikované negativní objekty), jak je patrné z obrázku 3 (červeně čerchované oblasti). Tyto chyby zde vychází z metody binární klasifikace, protože nás zajímá, kolik cílových objektů nebylo za cílové objekty označeno a také naopak, kolik odlehlých objektů bylo označeno za cílové objekty. Pomocí těchto hodnot lze stanovit, jestli je daná klasifikační funkce vhodná či nikoliv. Při vyhodnocování kvality klasifikace binárních klasifikátorů lze využít podobných principů jako pro unární klasifikaci. Je tedy patrné, že binární klasifikátory se od unárních liší pouze na základě počtu vstupních tříd objektů (mají informaci i o odlehlých hodnotách na rozdíl od unárních klasifikátorů, kde tréninková data obsahují pouze cílové objekty). Nevýhodou zde může být jenom částečná znalost odlehlých objektů v tréninkové sadě. Sice lze s jistotou prohlásit, že se jedná o odlehlé hodnoty, ale už nelze stanovit, zda se jedná o typickou ukázku nebo náhodnou odchylku. Tato neznalost potom může způsobit, že ve výsledku navržená klasifikační funkce nebude dosahovat požadované kvality klasifikace. Vliv těchto chyb lze částečně minimalizovat zvětšením tréninkové sady.[7]

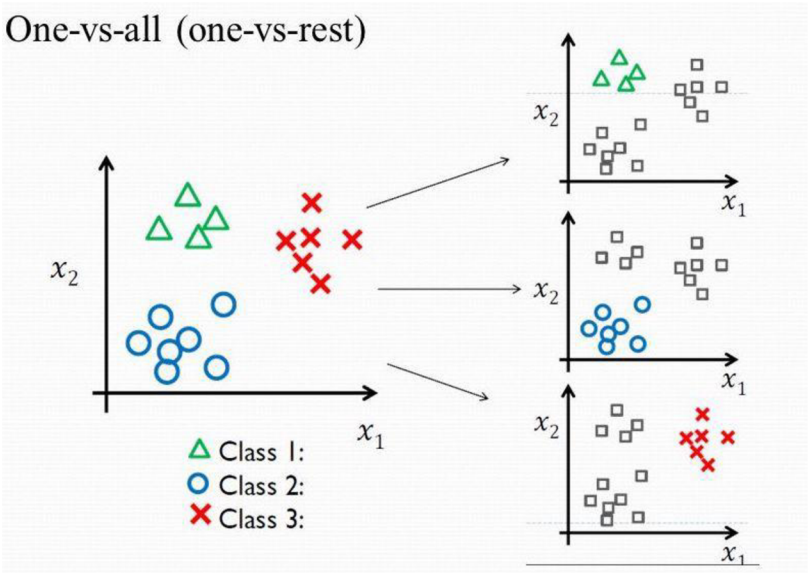
1.6 Multi-třídní klasifikátory

Třetím a posledním typem klasifikátorů je multi-třídní klasifikátor. Jak jeho název napovídá je určen pro několik vstupních množin dat. Jediným rozdílem oproti binárnímu klasifikátoru je zde přítomnost n-množin (zelená, červena a modrá množina na obrázku 4) namísto dvou množin.[8]



Obrázek 4: Rozdělení klasifikovaného prostoru multi-třídním klasifikátorem [8]

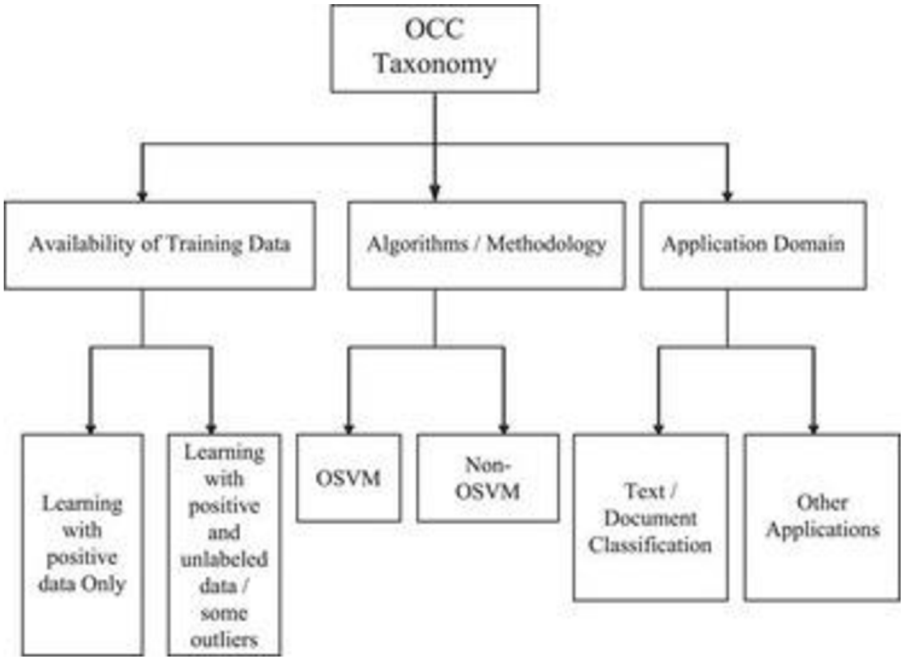
Zajímavou modifikací je potom tzv. one-vs-all klasifikátor. Úprava spočívá v rozdělení vstupních množin na cílovou množinu a odlehlé objekty (za cílovou množinu je vždy zvolena jedna z množin, jak je vidět na obrázku 5). Výhodou tohoto klasifikátoru je, že pokud objekt není součástí cílové skupiny, můžeme s jistotou určit, že se jedná o odlehlou hodnotu. Opět zde platí stejné principy klasifikace jako u unárních klasifikátorů. Nicméně správným využitím znalosti o odlehlých objektech zde lze výrazně zvýšit kvalitu klasifikace, za předpokladu, že trénovací objekty byly typickými zástupci dané třídy objektů.[8]



Obrázek 5: Modifikace multi-klasifikace klasifikace na one-vs-all klasifikátor [8]

2. KLASIFIKAČNÍ METODY

2.1 Rozdělení klasifikačních metod



Obrázek 6: Rozdělení unárních klasifikačních metod [9]

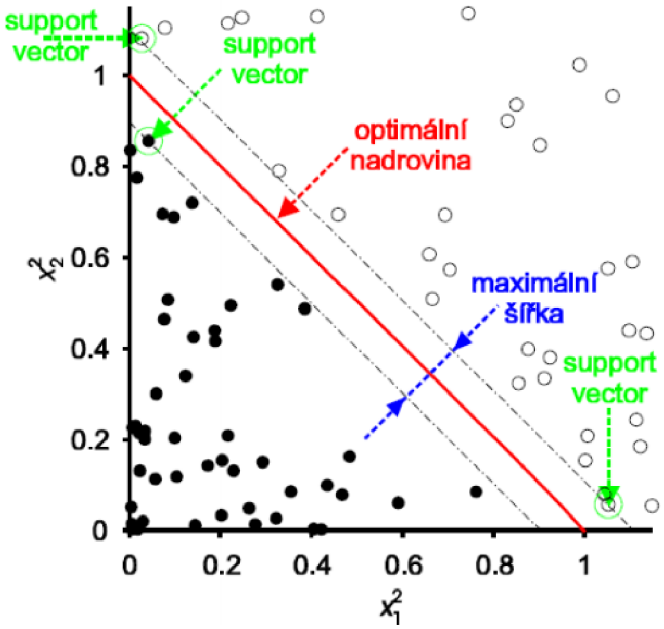
Klasifikační metody unární klasifikace (OCC = one class classification) můžeme rozdělit na základě znalosti jejich metod, dostupnosti tréninkových dat a v neposlední řadě podle místa aplikace, jak je vidět na obrázku 6.

Dělení podle dostupnosti dat:

První podskupinu tvoří klasifikátory trénované pouze z pozitivní sady tréninkových dat. Pokud by byl tento způsob učení použit například na problém klasifikace ovoce, trénování by probíhalo pouze na tréninkovém datasetu obsahující jeden druh ovoce, například jablka. U této metody dochází k problémům s hraničními hodnotami, jelikož při tréninku nejsou k dispozici negativní příklady, nelze vytvořit optimální model. Při tomto způsobu učení se předpokládá, že tréninkový dataset je tvořen typickými příklady cílové třídy. Druhou skupinou je potom učení na pozitivních příkladech s částečnou znalostí odlehlých hodnot. Tato částečná znalost může být tvořena znalostí několika typických odlehlých hodnot, nebo znalostí uměle vygenerovaných odlehlých hodnot. S touto metodou se pojí několik problémů. Typický problémem bývá neznalost kompletního rozložení odlehlých hodnot, takže nemůžeme s jistotou stanovit, zda jsou uvedené příklady typickými zástupci.[9]

Podle využití podpůrného vektoru dat:

Metody lze dělit na základě využívání/nevyužívání podpůrného vektoru dat (OSVM). Základem metody OSVM je lineární binární klasifikátor. Cílem úlohy je nalézt nový prostor, který původní prostor příznaků optimálně rozděluje tak, že tréninková data, náležející odlišným třídám, leží v opačných poloprostorech. Optimální rovina je taková rovina, kde hodnota minim vzdáleností bodů od roviny je co největší. V literatuře je tato hodnota označována jako maximální šířka (modře označená šířka na obrázku 7). Na popis roviny stačí pouze body, ležící na okraji tohoto pásma, a těch je ve většině případů velmi málo (zelené body na obrázku 7). Tyto body se nazývají podpůrné vektory (v angličtině „support vectors“) a odtud tedy název metody. Na druhou stranu algoritmy NON-OSVM nevyužívají podpůrný vektor dat, nejčastěji se tak jedná o neuronové sítě nebo rozhodovací stromy.[10]



Obrázek 7: Rozdělení klasifikovaného prostoru s pomocí podpůrného vektoru dat [10]

Podle aplikační domény:

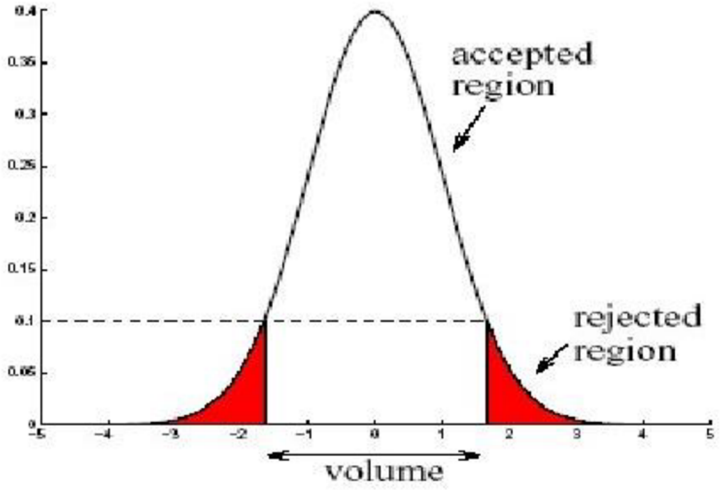
Posledním z kritérií, podle niž je možné dělit unární klasifikátory, je oblast použití. Toto dělení vychází hlavně z historického pohledu na rozdělení klasifikátorů, v dnešní době se příliš nepoužívá. S rozvojem klasifikačních metod a jejich vývojem se dají tyto metody rozdělit do dvou hlavních skupin použití, první takovouto množinou je klasifikace textu (email, dokumenty, textové zprávy …) a druhou skupinou jsou klasifikátory pro obrazovou a další klasifikaci. [9]

2.2 Metody hustoty

Nejpřímější metodou k získání klasifikátoru jedné třídy je odhadnout hustotu tréninkových dat a nastavit prahovou hodnotu modelu pro tuto hustotu. Tato práce je zaměřena na tři modely hustoty, normální model, směs Gaussianů a Parzen hustotu. Pokud je velikost tréninkového datasetu dostatečně velká a použije se model s flexibilní hustotou (pro příklad odhadu hustoty Parzen), fungují tyto modely velmi dobře. Tyto metody však pro svoje fungování vyžadují velké množství dat. Tento problém lze částečně odstranit snížením složitosti modelu, pak ale model vykazuje zkreslení. Cílem je tedy najít optimum mezi složitostí modelu a množstvím dat. Pokud se podaří stanovit vhodný poměr mezi složitostí modelu a množstvím dat vykazují tyto metody velice dobré výsledky.

2.2.1 Gaussův model

Nejjednodušším z modelů je normální nebo Gaussovo rozložení hustoty. Za použití centrálního limitního teorému, lze stanovit, že objekty z jedné třídy dat mají podobnou hustotu rozložení a zároveň obsahují malý počet nezávislých chybových dat (červeně označené regiony na obrázku 8). Cílem metody je nalezení prahové hodnoty, která rozdělí klasifikovaná data na základě hustoty rozložení (čárkovaná čára na obrázku 8). [11]



Obrázek 8: Gaussův model, definující hranici hustoty pro cílová data [1]

Pravděpodobnost distribuce pro d-dimenzionální objekt je dána vztahem:

$$p_n(z,\mu,\sum) = \frac{1}{(2\pi)^{\frac{d}{2}}|\sum|^{\frac{1}{2}}} \exp\{-\frac{1}{2}(z-\mu)^T \sum^{-1}(z-\mu)\} \quad [1](2.1)$$

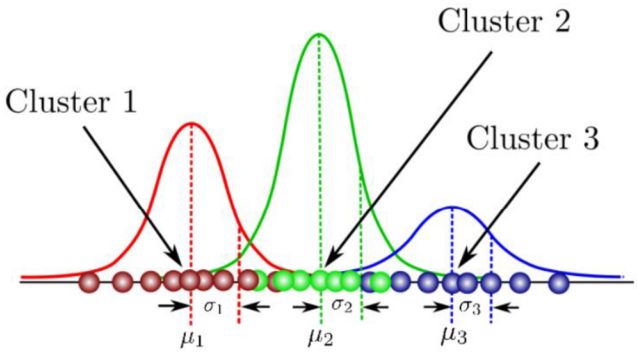
kde $p_N(z,\mu,\sum)$ je pravděpodobnost distribuce pro d-rozměrný objekt, μ je vektor průměrů dat, $\sum$ je kovariantní matice a z je sloupcový vektor reprezentující testovaný objekt. Metoda vyžaduje unimodální (očekávané rozdělení pravděpodobnosti má pouze jeden vrchol) a konvexní rozložení dat (funkce pravděpodobnosti se nachází nad svojí tečnou). Počet volných parametrů v Gaussově modelu je tedy:

$$n_{freeN} = d + \frac{1}{2}d(d-1) \qquad [1](2.2)$$

kde počet volných parametrů (n_{freeN}) závisí pouze na rozměru vstupních objektů (d).

2.2.2 Směs Gaussianů

Gaussova distribuce předpokládá velmi silný model dat. Tento model by měl být unimodální a konvexní. U většiny datových souborů jsou tyto předpoklady porušeny. Pro získání pružnější metody hustoty lze normální rozdělení rozšířit na směs Gaussianů. [12]



Obrázek 9: Směs Gaussianů, se třemi datovými soubory a jejich funkcemi hustoty [12]

Na obrázku 9 lze vidět jednotlivé tréninkové sady a jejich funkce hustoty na základě rozložení dat. Ve výsledku je potom celková hustota určena vhodnou kombinací jednotlivých Gaussianů podle směšovacích koeficientů. [12]

Kdy směs Gaussianů je lineární kombinací normálních distribucí, pravděpodobnost rozložení hustoty této směsi Gaussianů je:

$$p_{MoG}(x) = \frac{1}{N_{MoG}} \sum_j \alpha_j p_N(x, \mu_j, \sum_j \quad) \qquad [1]\ (2.3)$$

kde pravděpodobnost rozložení směsi Gausianů (p_{Mog}) je dána počtem Gausianů (N_{Mog}), směšovacím koeficientem (α_j) a kombinací normálových rozložení (p_N). Kdy jednotlivá normálová rozložení jsou definována svými průměry (μ_j) a svými kovariantními maticemi ($\sum$).

Celkový počet volných parametrů ve směsi Gaussianů je:

$$n_{freeMoG} = \left(d + \frac{d(d+1)}{2} + 1\right) N_{MoG} \quad \text{[1] (2.4)}$$

kde počet volných parametrů ($n_{freeNoG}$) závisí na rozměru vstupních objektů (d) a na počtu Gausiánů (N_{Mog}).

2.2.3 Odhad hustoty Parzen

Třetí metodou je odhad hustoty Parzen, tato metoda je opět rozšířením předchozí metody. Odhadovaná hustota je směsí nejčastějších Gaussovských jader, soustředěných na jednotlivé tréninkové objekty (x) s často diagonálními kovariantními maticemi $\Sigma_i = hI$. [1]

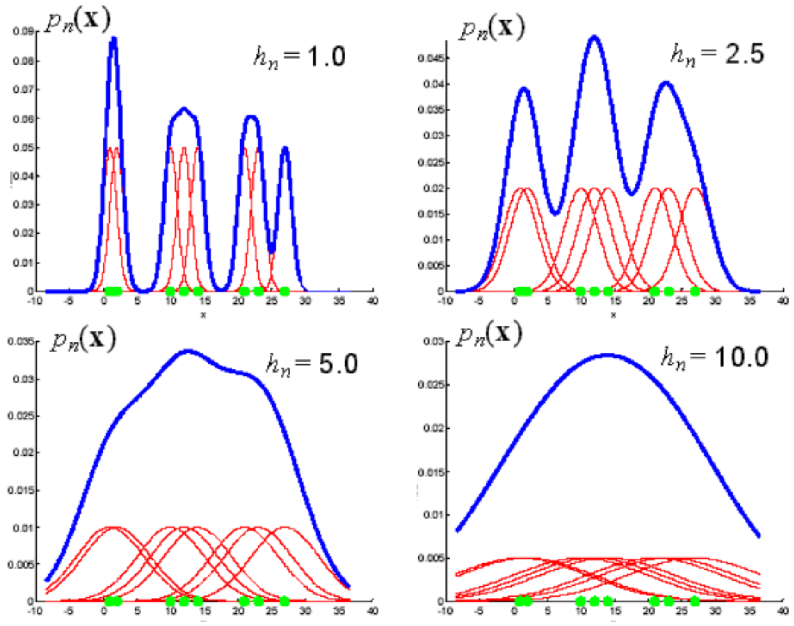
Pravděpodobnost rozložení hustoty Parzen (p_p):

$$p_p(x) = \frac{1}{N}\sum_i p_N(x, x_i, hI) \quad \text{[1] (2.5)}$$

je dána počten tréninkových objektů (N) a normálním nebo Gaussovským rozdělení (p_N), které je charakterizované průměrem (μ) a kovariantní maticí (Σ). Stejná šířka jádra (h) v každém směru prvku znamená, že Parzenův odhad hustoty předpokládá stejně vážené objekty v tréninkové sadě, proto bude citlivý na změnu velikosti hodnot objektů trénovací sady, zejména pro menší velikosti datasetu. Trénink metody Parzen se skládá z určení jediného parametru, optimální šířky jádra (h).

Počet volných parametrů je tedy roven 1: [1]

$$n_{freep} = 1 \quad \text{[1] (2.6)}$$



Obrázek 10: Vliv šířky jádra na odhad hustoty Parzen [13]

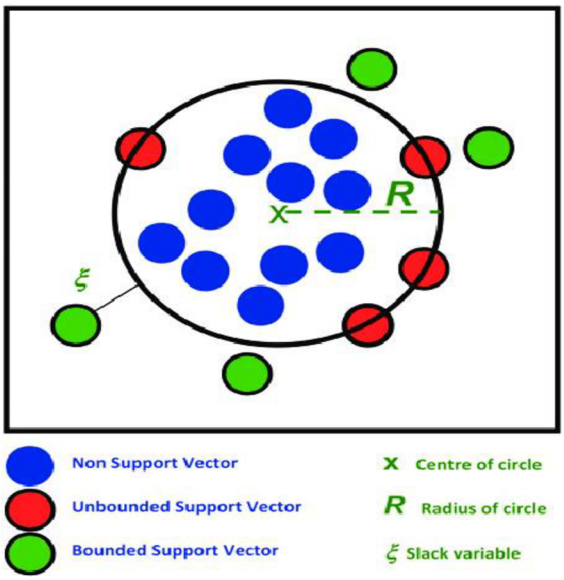
Jak je vidět na obrázku 10, je výsledná přesnost odhadu ($p_n(x)$) přímo závislá na přesném stanovení parametru šířky jádra (h). Tím, že přesnost modelu závisí pouze na jednom parametrů, je velice snadné tento parametr nastavit. [13]

2.3 Hraniční metody

Tyto hraniční metody se zaměřují na případy, kdy je k dispozici pouze omezený počet dat v tréninkové sadě. Cílem tedy není odhadnout celou hustotu rozložení, ale pouze hranici oddělující cílové a odlehlé objekty. Tato práce je zaměřena na metody SVDD, k-průměr a NN-d hraniční metody. Všechny tyto metody pracují na podobném principu, tedy na určení vzdálenosti mezi jednotlivými objekty v tréninkové sadě. Tento postup je značně citlivý na změnu velikosti vstupních dat, ale na druhou stranu vyžaduje mnohem méně dat v tréninkovém datasetu než metody hustoty. Výstupem těchto metod už není pravděpodobnost, ale výstupem je v tomto případě funkce, která dokáže zachytit většinu z cílových objektů.

2.3.1 SVDD „Support Vector Data Description“ klasifikátory

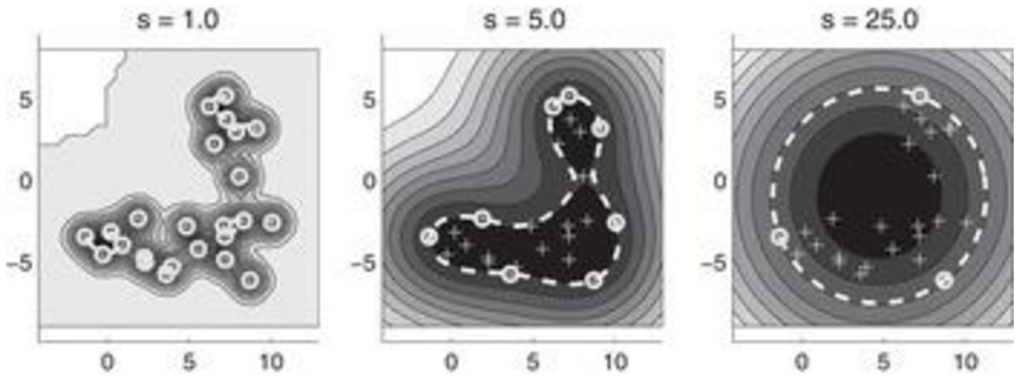
Na obrázku 11 vidíme SVDD klasifikátor s poloměrem (R) se středem v bodě (x). Odlehlé hodnoty potom tvoří chyby při klasifikaci pomocí SVDD klasifikátoru. Takovýchto hodnot je možné se zbavit, pokud je zahrneme do klasifikované sféry za pomoci změny poloměru nebo vytvořením druhé klasifikační sféry. [14]



Obrázek 11: Rozdělení cílových a odlehlých hodnot pomocí metody SVDD [14]

Klasifikátor SVDD odmítne daný testovací bod jako odlehlý (zelené body na obrázku 11), pokud spadne mimo hyper-sféru (kruhová hyper-sféra na obrázku 11), tato hyper-sféra je definována hraničními body (červenými body na obrázku 11). SVDD však může odmítnout určitou část pozitivně označených dat, pokud je objem hyper-sféry snížen. Hyper-sférický model SVDD může být flexibilnější zavedením funkcí jádra. Pro většinu uvažovaných datových souborů funguje Gaussovo jádro lépe, ve srovnání s polynomiálním jádrem.[14]

Lze také používat různé hodnoty šířky jádra. Čím větší je šířka jádra, tím méně podpůrných vektorů je vybráno a popis se stává sféričtějším, jak je vidět na obrázku 12. Použití Gaussovského jádra místo polynomiálního vede k přesnějším popisům, ale je vyžadováno více dat, aby došlo k vytvoření pružnější hranice. Metoda nefunguje dobře, pokud existují velké rozdíly v hustotě mezi objekty cílové třídy. V takovém případě začne metoda odmítat cílové hodnoty s nízkou místní hustotou jako odlehlé hodnoty. [14]



Obrázek 12: Příklad snížení počtu jader SVDD metody při zvětšení poloměrů hyper-sfér [1]

Na obrázku 12 jsou data rozložena do oblého tvaru, to zabraňuje jejich snadné klasifikaci. V jednotlivých krocích zde dochází k navyšování poloměru jednotlivých klasifikačních sfér, kdy každá sféra má střed v klasifikovaných datech. Při poloměru pět jsou již všechny data uvnitř sféry, zatím co pokud zvolíme poloměr deset stačí pouze jedna sféra pro klasifikaci všech dat. Ve výsledku je tedy vždy důležité správně zvolit, zda je důležitější rychlejší klasifikace za pomocí jedné sféry nebo kvalitnější klasifikaci za pomocí několika sfér. Takováto pomocná sférická data se nazývají Gaussovské jádro. Zvětšováním poloměru těchto jader se stává sféričtějším a je potřeba méně těchto jader. Pro vyhodnocení těchto klasifikátorů se používají binární funkce, které vrací 1 uvnitř sfér a –1 pro body mimo sféru. Pro výpočet chyby SVDD metody (ε_{struct}) potom lze použít vztah [1]:

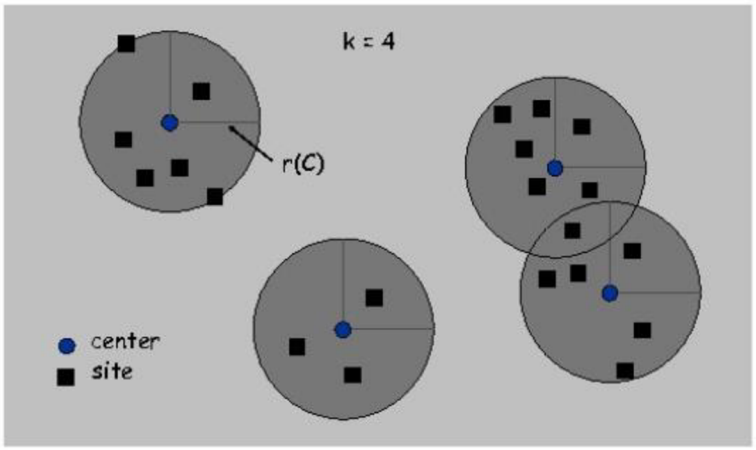
$$\varepsilon_{struct}(R, a) = R^2 \qquad [1]\ (2.7)$$

kde chyba metody SVDD (ε_{struct}) je dána parametry druhé mocniny poloměru hyper-sféry (R). Chyba SVDD metody (ε_{struct}) musí být minimalizována následujícím omezením. Poloměr hyper-sféry (R) musí být menší nebo roven druhé mocnině absolutní vzdálenosti středu hyper-sféry (a) a testovaného objektu (X_i):

$$||x_i - a||^2 \leq R^2 \qquad [1]\ (2.8)$$

2.3.2 K-střed

Tato metoda pokrývá trénovací dataset malými sférickými oblastmi se stejným poloměrem. Tato metoda se používá pro popis složitosti tréninkové sady. Středy jednotlivých sfér jsou umisťovány tak, aby maximum všech minim vzdáleností mezi tréninkovými objekty a středy jednotlivých sfér byla co nejmenší. Uživatel musí zadat počet sfér (k) i maximální počet opakování pro nalezení optimálního minima.



Obrázek 13: Klasifikace metodou k-střed při použití čtyř sfér [15]

Na obrázku 13 je potom vidět výsledek této metody pro čtyři zvolené sféry (šedé kruhové oblasti se středem v modrých bodech), tyto středy musí ležet na některém z testovaných objektu. Je patrné, že středy jednotlivých sfér jsou umístěny tak, aby vzdálenost ke všem příslušným objektům (černé body na obrázku 13) byla co nejmenší.[15] Výsledná chyba klasifikace ($\varepsilon_{k\text{-}centr}$) je potom dána, maximem minim rozdílů polohy středů sfér (μ_k) a polohy testovaných objektů (x_i) [1]:

$$\varepsilon_{k-centr} = max_i(min_k||x_i - \mu_k||^2) \qquad [1] (2.9)$$

Metoda k-střed používá strategii dopředného vyhledávání, která začíná náhodnou inicializací. Poloměr ($d_{k\text{-}centr}$) je určen maximální vzdáleností k objektům, které by měla příslušná sféra zachytit. Potom je možné určit vzdálenost testovaného objektu (z) k objektu tréninkové sady (μ_k), tato vzdálenost je definována jako [1]:

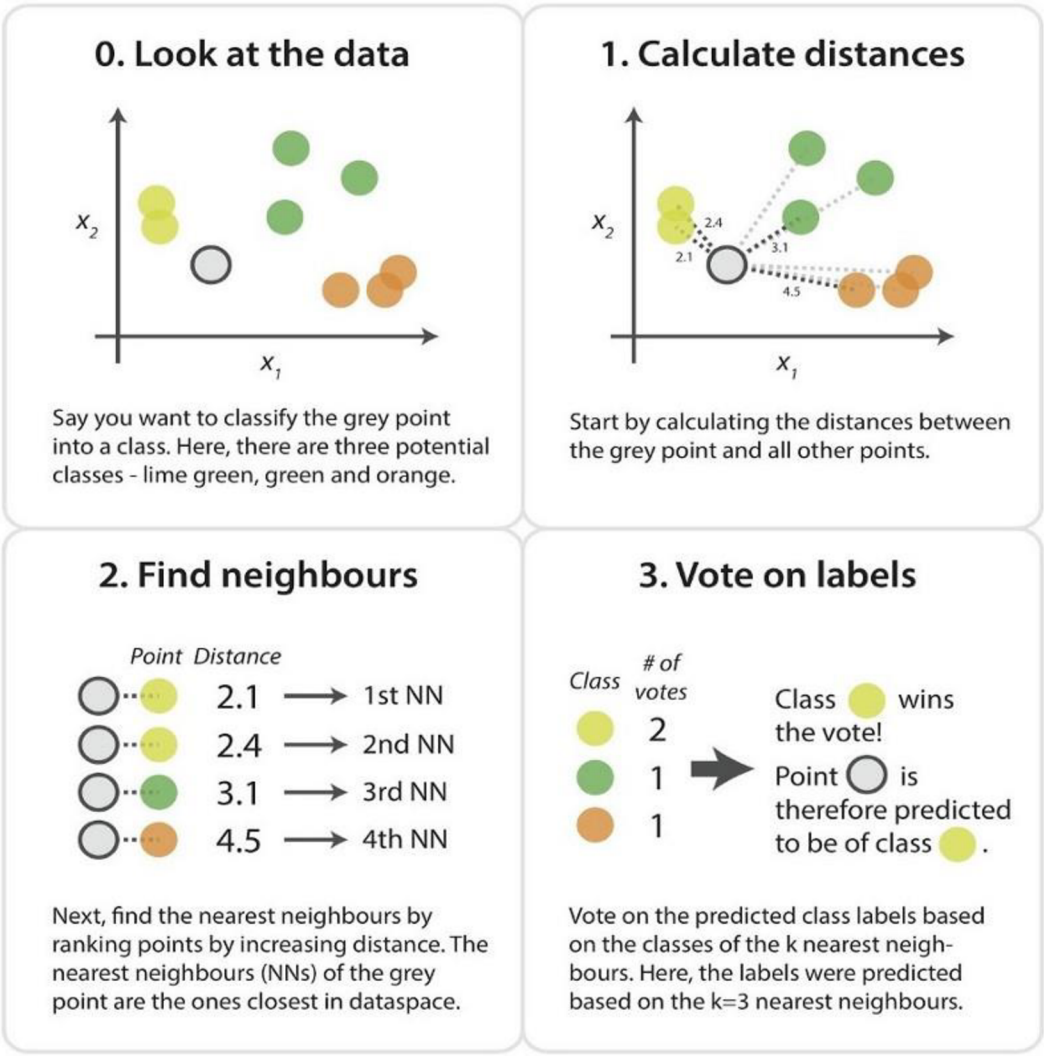
$$d_{k-centr}(z) = min_k||z - \mu_k||^2 \qquad [1] (2.10)$$

Aby bylo zabráněno neoptimálnímu řešení během tréninku, je vhodné vyzkoušet několik náhodných inicializací a použít nejlepší řešení (to s nejmenší chybou $\varepsilon_{k\text{-}centr}$). Počet volných parametrů je tedy dán počtem sfér (k) [1]:

$$n_{freek-c} = k \qquad [1] (2.11)$$

2.3.3 Metoda nejbližšího souseda

Další z hraničních metod je metoda nejbližšího souseda, NN-d. Může být odvozena z odhadu místní hustoty klasifikátorem nejbližšího souseda. Metoda se vyhýbá výslovnému odhadu hustoty a používá pouze vzdálenosti k prvnímu nejbližšímu sousedovi. Při odhadu hustoty nejbližšího souseda je buňka, často hyper-sféra v dimenzích d, soustředěna kolem testovaného objektu (z). Objem této buňky narůstá, dokud nezachytí několik objektů z tréninkové sady (k). Postup klasifikace je možné vidět na obrázku 14, zde je testovaný objekt přiřazen k třídě, ke které je nejblíže. Jinak řečeno objekt je přiřazen k té třídě, která po přiřazení testovaného objektu dosahuje nejvyšší místní hustoty. [16]



Obrázek 14: Princip výpočtu metody nejbližšího souseda [16]

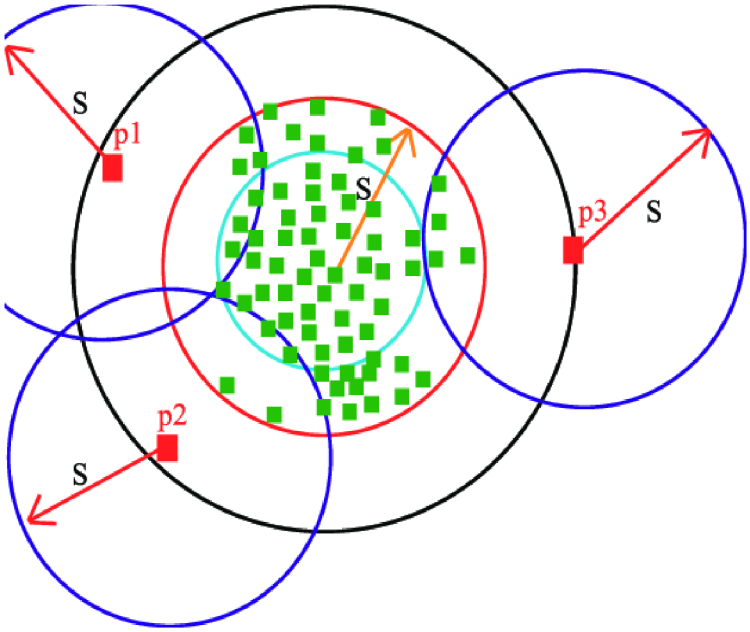
Místní hustota (P_{nn}) se pak vypočítá podle:

$$p_{nn}(z) = \frac{k/N}{V_k(||z - NN_k^{tr}(z)||)} \qquad [1] (2.12)$$

kde místní hustota (P_{nn}) závisí na počtu nejbližších sousedů (k) objektu (z) v tréninkové sadě (N), dále na objemu buňky (V_k) obsahující tento objekt (z) a na k-tém nejbližším sousedovi (NN^{tr}_k) testovaného objektu (z). U unární klasifikace je testovaný objekt (z) přijat, pokud je lokální hustota větší nebo rovna hustotě nejbližšího souseda z tréninkové sady. Pro lokální hustotu k=1 potom tedy objekt přijat pokud [1]:

$$f_{NNtr}(z) = I\left(\frac{V(||z - NN^{tr}(z)||)}{V(||NN^{tr}(z) - NN^{tr}(NN^{tr}(z)||)} \leq 1\right) \qquad [1] (2.13)$$

Jednoduchou úpravou této metody potom dostáváme metodu „Local Outlier Factor“ (LOF), která je zkonstruována k hledání odlehlých hodnot ve velkých databázích. U této metody jsou všechny vzdálenosti k nejbližšímu sousedovi zprůměrovány. Dále vzdálenost objektu k jeho nejbližším sousedům, je nahrazena robustnější definicí vzdálenosti. Pokud jsou objekty velmi blízko cílových dat, použije se vzdálenost k-tého nejbližšího souseda, místo vzdálenosti k prvnímu nejbližšímu sousedovi. Toto robustní opatření ztěžuje rozlišení mezi objekty, které jsou blízko hranice nebo které jsou hluboko v těsném shluku cílových objektů. Dále tento algoritmus vyžaduje, aby uživatel definoval počet sousedů, kteří jsou používáni, ve většině případů je volena hodnota v rozsahu od 10 do 50. Ačkoli je metoda LOF je velmi vhodná k nalezení odlehlých hodnot ve velkých souborech dat, je velice těžké porovnat ji s ostatními metodami na malých souborech dat, bez jasných odlehlých hodnot (objekty na hranici nejsou odmítnuty). [17]



Obrázek 15: Ukázka principu klasifikace LOF metody na základě místní hustoty [17]

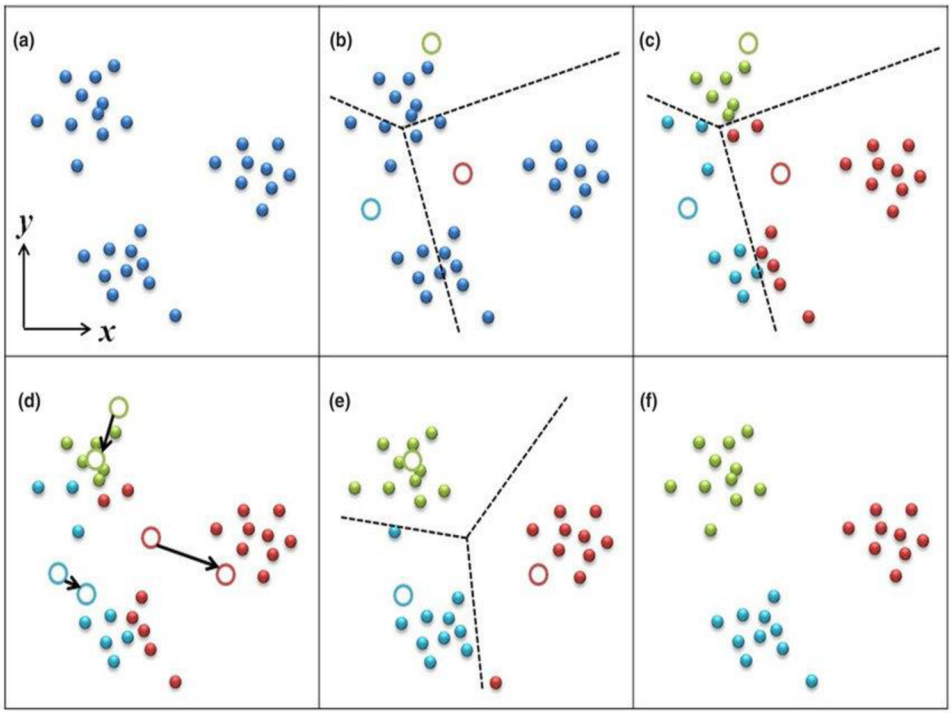
Na obrázku 15 jsou vidět tři odlehlé objekty (červené body p1, p2 a p3). Tyto objekty lze prohlásit za odlehlé, protože jejich lokální hustota (modře zvýrazněné oblasti) je výrazně nižší než u cílových dat (zelené body).

2.4 Rekonstrukční metody

Rekonstrukční metody nejsou primárně tvořeny pro unární klasifikaci, ale spíše pro modelování dat. Využitím znalostí o datech a jejich generování, lze vybraný model přizpůsobit datům. Testované objekty mohou být nyní popsány na základě stavu modifikovaného modelu. Cílem tohoto postupu je snaha o dosažení kompaktnějšího popisu dat. Tyto metody pracují s klastrováním dat nebo jejich distribuci v podprostorech. Jednotlivé metody se potom liší v definici podprostorů, definováním rekonstrukční chyby a optimalizační rutinou. Při použití rekonstrukčních metod se předpokládá, že odlehlé objekty nesplňují předpoklady o cílové distribuci. Odlehlé hodnoty jsou tedy rekonstruovány s velkou chybou na rozdíl od cílových dat.

2.4.1 K-průměr

V metodě k-průměr dochází k inicializaci středů metody v náhodně zvoleném místě (b na obrázku 16). Každý testovaný objekt je přiřazen právě k takovému středu, od kterého je nejblíže (c na obrázku 16). V následujícím kroku dojde k přemístění středu, tak aby průměrná vzdálenost přiřazených objektů od středu byla co nejmenší (d na obrázku 16). Následně dojde ke kontrole jednotlivých vzdáleností objektů od jednotlivých středů, v případě že některý z objektů se nyní nachází nejblíže k jinému středu dojde k jeho pře-klastrování (e na obrázku 16). Tyto kroky jsou opakovány, dokud není splněna podmínka o minimální střední hodnotě vzdáleností od jednotlivých středů (f na obrázku 16).



Obrázek 16: Postup při klasifikaci pomocí metody K-průměr [18]

Metoda k-průměr se podobá metodě k-střed, ale důležitým rozdílem je chyba, vůči které je minimalizována. Metoda k-střed se zaměřuje na objekty v nejhorším případě (tj. objekty s největší chybou rekonstrukce) a snaží se optimalizovat středy a poloměry sfér tak, aby přijímala všechna data. V metodě k-průměr jsou vzdálenosti všech objektů průměrovány, proto je metoda odolnější vůči vzdáleným odlehlým hodnotám. Navíc v metodě k-střed jsou středy umístěny podle definice na některé z tréninkových objektů, zatímco v metodě k-průměr jsou všechny středové pozice umísťovány volně. [18]

Chyba metody k-průměr je potom stanovena jako suma minim rozdílu testovaného objektu (x_i) a k-tého středu (μ_k):

$$\varepsilon_{k-m} = \sum_i (min_k ||x_i - \mu_k||^2) \qquad \text{[1] (2.14)}$$

Vzdálenost objektu (d_{k-m}) cílové sady je poté definována jako minimální čtvercová vzdálenost daného objektu (z) k nejbližšímu středu (μ_k):

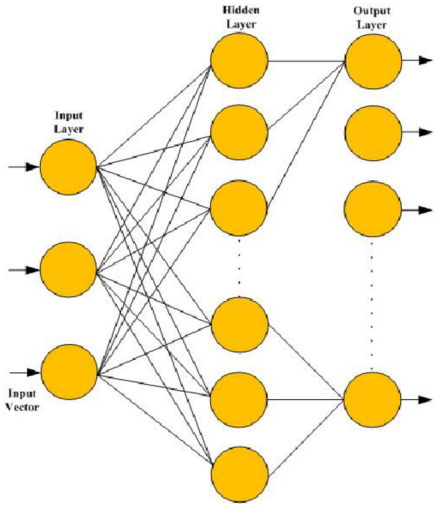
$$d_{k-m}(z) = min_k ||z - \mu_k||^2 \qquad \text{[1] (2.15)}$$

2.4.2 LVQ

Algoritmus LVQ je super-vizovaná verze klastrování metody k-průměr a používá se hlavně pro klasifikační úkoly. Pro každý z tréninkových objektů (x_i) je k dispozici štítek (y_i) označující, ke kterému klastru by měl patřit. LVQ je trénován jako konvenční neuronová síť (na obrázku 17 je možné vidět ukázku LVQ), s tím rozdílem, že každá skrytá jednotka je prototyp, kde pro každý prototyp (μ_k) je definován štítek třídy (y_k). Metoda LVQ vyžaduje, aby uživatel definoval rychlost učení (η). Cvičný algoritmus aktualizuje pouze klastr nejblíž cvičnému objektu (x_i) [19]:

$$\Delta\mu_k = \begin{cases} +\eta(x_i - \mu_k) & když\ y_i = y_k \\ -\eta(x_i - \mu_k) & jinak \end{cases} \qquad \text{[1] (2.16)}$$

kde nejbližší klastr je aktualizován ($\Delta\mu_k$) pokud se shoduje definovaný štítek testovaného objektu (y_i) se štítkem, který je na výstupu skryté vrstvy (y_k).



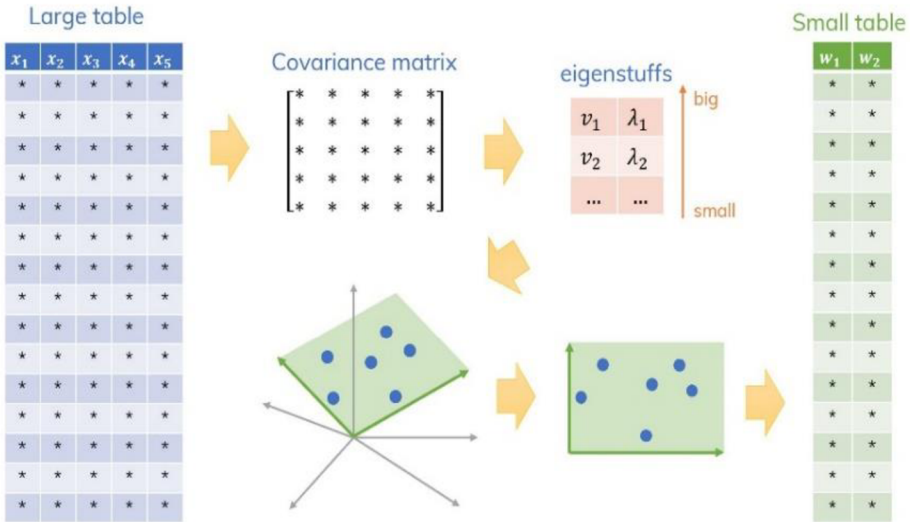
Obrázek 17: Ukázka struktury LVQ metody, s jednou skrytou vrstvou [19]

V metodě k-průměr i v LVQ musí být počet potřebných sfér odhadnut. Počet volných parametrů (n_{free}) závisí na počtu klastru (k) a velikosti vstupních dat (d):

$$n_{free} = n_{freeLVQ} = kd \qquad [1] (2.17)$$

2.4.3 Principal Component Analysis (PCA)

Analýza hlavních komponent (PCA) se používá pro data distribuovaná v lineárním podprostoru. Mapování PCA najde ortonormální podprostor, který co nejlépe zachytí rozptyl v datech (ve smyslu čtvercové chyby). Cílem metody je nalezení takového podprostoru, který nám umožní popsat data pomocí menšího množství dat. [20]



Obrázek 18: Snížení objemnosti dat pomocí PCA algoritmu [20]

Nejjednodušší postup optimalizace používá k výpočtu vlastních vektorů cílové kovariantní matice a rozklad vlastních čísel (viz. obrázek 18). Vlastní vektory s největšími vlastními hodnotami jsou hlavní osou d-dimenzionálních dat a ukazují ve směru největšího rozptylu. Tyto vektory se používají jako základní vektory pro mapovaná data. Rekonstrukční chyba je definována jako čtverec vzdáleností originálního objektu (z) a mapované verze objektu ($(WW^T)z$):

$$d_{PCA}(z) = ||z - (WW^T)z||^2 \qquad [20](2.18)$$

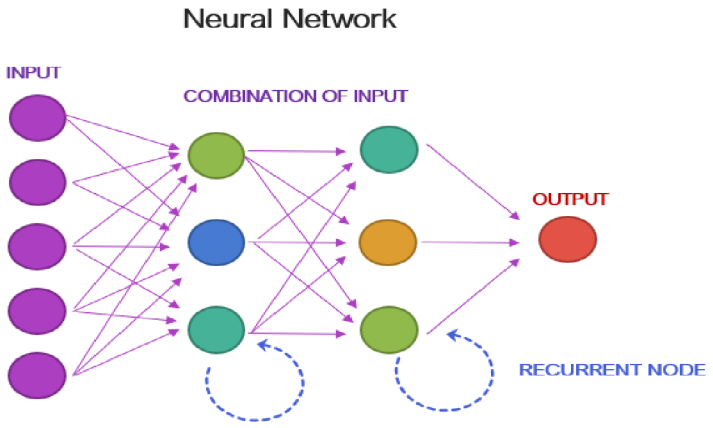
kde bázový vektor (W) je tvořen maticí d × M, tato matice závisí na počtu bázových vektorů (M).

PCA funguje dobře v případě, kdy je k dispozici jasně definovaný lineární prostor, ve kterém jsou obsažena data. Výhodou této metody je, že dobře pracuje s malým množstvím dat. Mezi největší nevýhody však patří neschopnost snížit rozměrnost dat, pokud jsou data rozptýlená ve všech směrech prostoru funkce, nebo když jsou data distribuována v samostatných podprostorech. PCA potom vytvoří průměrný podprostor, který může velmi špatně popisovat data v každém z původních podprostorů. Metoda PCA je relativně citlivá na změnu velikosti funkcí, kdy tato změna přímo ovlivňuje odchylky funkcí. Škálování mění pořadí směrů velkých rozptylů a tím i základ PCA. V neposlední řadě se PCA zaměřuje pouze na rozptyl cílového souboru (soubor obsahující cílové objekty) a metoda PCA není schopna zahrnout do tréninku negativní příklady. [20]

2.5 Neuronové sítě

2.5.1 Klasifikace pomocí neuronové sítě

Metoda klasifikace pomocí neuronové sítě je založena na návrhu jednoduché neuronové sítě pro filtraci vstupních dat. Tato neuronová síť je učena na malém počtu pozitivních příkladů. [22]

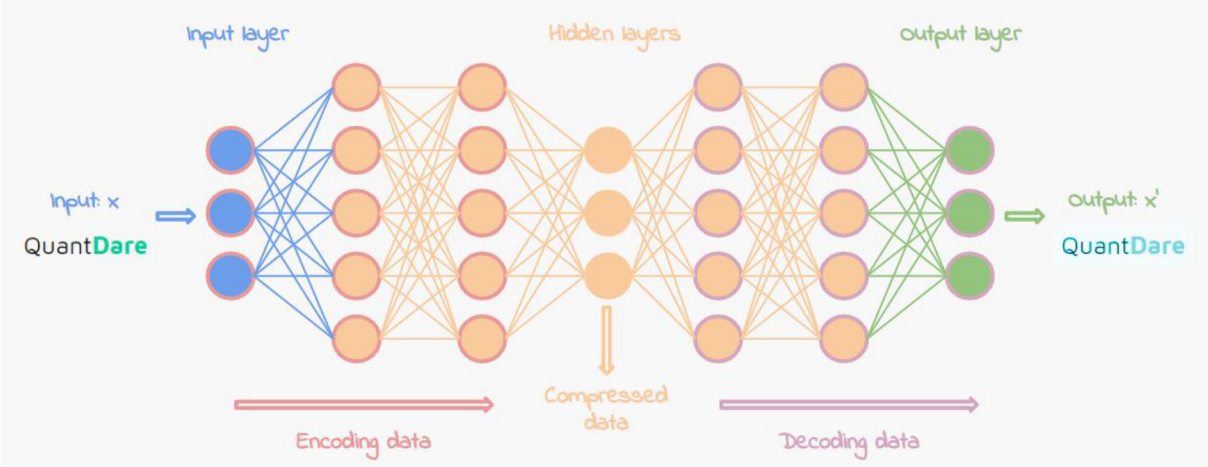


Obrázek 19: Schématická ukázka neuronové sítě s dvěma skrytými vrstvami [22]

Neuronové sítě jsou používány pro klasifikaci z důvodů zlepšení klasifikačních výsledků po přidání skryté vrstvy neuronové sítě. Většinou je volena minimálně tří úrovňová neuronová síť s jednou vstupní vrstvou (fialová vrstva na obrázku 19), s M výstupními neurony (poslední červená vrstva na obrázku 19) a s K skrytými neurony (druhá a třetí vrstva na obrázku 19). Neuronová síť je trénována pomocí standardního algoritmu zpětného šíření, který trénuje klasifikační funkci pouze na pozitivních příkladech. Takto navržená neuronová síť se může naučit klasifikovat data, která jsou podobná datům použitých při tréninku. Neuronová síť potom označí tato data jako cílová, zatímco všechna ostatní data jsou označena za odlehlá. Varianta dopředné neuronové sítě může trénovat jak na pozitivních, tak negativních datech. V konvenčním dopředném binárním klasifikátoru konvoluční neuronové sítě jsou pozitivní příklady označeny jako 1 a negativní příklady jako 0. Výstup sítě představuje pravděpodobnost, že neznámý příklad patří do cílové třídy s prahovou hodnotou 0,5. V tomto případě, protože neoznačená data mohou obsahovat některé neoznačené pozitivní příklady, může být výstup trénované neuronové sítě menší nebo roven skutečné pravděpodobnosti. Pokud lze uvažovat, že definované pozitivní příklady adekvátně představují pozitivní množinu, lze předpokládat že neuronová síť bude schopna určit hranici mezi negativními a pozitivními daty. [22]

2.5.2 Auto-enkodér a Diabolo sítě

Jedná se o variantu klasických klasifikátoru s neuronovou sítí. Obě metody jsou trénovány pro reprodukci vstupních vzorů na jejich výstupní vrstvě. Mezi těmito sítěmi lze rozlišovat podle počtu skrytých vrstev a velikostí jednotlivých vrstev. V architektuře auto-enkodér je přítomna pouze jedna skrytá vrstva s velkým počtem skrytých jednotek. V síti Diabolo se používají tři skryté vrstvy s nelineární sigmoidní přenosovou funkcí, druhá vrstva obsahuje velmi nízký počet skrytých jednotek. Této vrstvě se říká zúžená vrstva. Další dvě vrstvy mohou mít libovolný počet skrytých jednotek, počet jednotek musí být větší než v zúžené vrstvě. Princip kódování a dekódování dat za pomocí neuronové sítě se zobrazen na obrázku 20. Data jsou zde přeneseny ze vstupní vrstvy pomocí přenosových funkcí do skryté vrstvy. Přenosem dat do zúžené vrstvy dojde k jejich zakódování. Následující skryté vrstvy jsou schopny data opět dekódovat, jestliže jsou rekonstruována data na výstupu shodná se vstupem jsou označena za cílová. [23]



Obrázek 20: Princip kódování a dekódování dat pomocí neuronové sítě [23]

Předpokládá se, že cílové objekty budou rekonstruovány s menší chybou než odlehlé objekty. Vzdálenost mezi původním objektem (z) a rekonstruovaným objektem (f(z,w)) je pak měřítkem vzdálenosti (d) objektu od cílových dat [1]:

$$d(z) = ||\mathrm{f}(z, w) - z||^2 \qquad [1]\ (2.19)$$

Pokud se v síti auto-enkodéru použije jen jedna skrytá vrstva, je nalezen lineární typ řešení hlavní komponenty. To znamená, že síť auto-enkodéru má tendenci najít popis dat, který se podobá popisu trénovací sadě dat. Na druhou stranu malý počet neuronů v zúžené vrstvě sítě Diabolo funguje jako informační kompresor. Pro získání malé chyby rekonstrukce pro cílovou sadu, je síť nucena trénovat kompaktní mapování dat do podprostoru kódovaného těmito skrytými neurony. Počet skrytých jednotek v nejmenší vrstvě dává rozměrnost tohoto

podprostoru. Vzhledem k nelineárním přenosovým funkcím neuronů v ostatních vrstvách se tento podprostor může stát nelineárním. Pokud se velikost tohoto podprostoru shoduje s podprostorem v původních datech, může síť diabolo dokonale odmítnout objekty, které nejsou v cílovém datovém podprostoru. Pokud je podprostor stejně velký jako původní prostor funkcí, nelze očekávat žádný rozdíl mezi cílovými a odlehlými daty. V aplikaci auto-enkodéru a sítí diabolo jsou stejné problémy jako při konvenční aplikaci neuronových sítí na klasifikační problémy. Metody jsou velmi flexibilní, ale vyžadují od uživatele předem definovaný počet vrstev a neuronů, krok učení a kritérium zastavení od uživatele. Výhoda těchto modelů je, že mohou být optimalizovány na daný problém, pro který mohou získat velmi dobré výsledky. Počet volných parametrů může být velmi vysoký jak pro auto-enkodér, tak pro síť Diabolo. Počet vstupních a výstupních neuronů je dán rozměrností dat (d). [23]

Celkový počet volných parametrů v auto-enkodéru je dán [1]:

$$n_{free\_auto} = (2d + 1)h_{auto} + d \qquad [1](2.20)$$

kde h_{auto} je počet skrytých jednotek v auto-enkodéru.

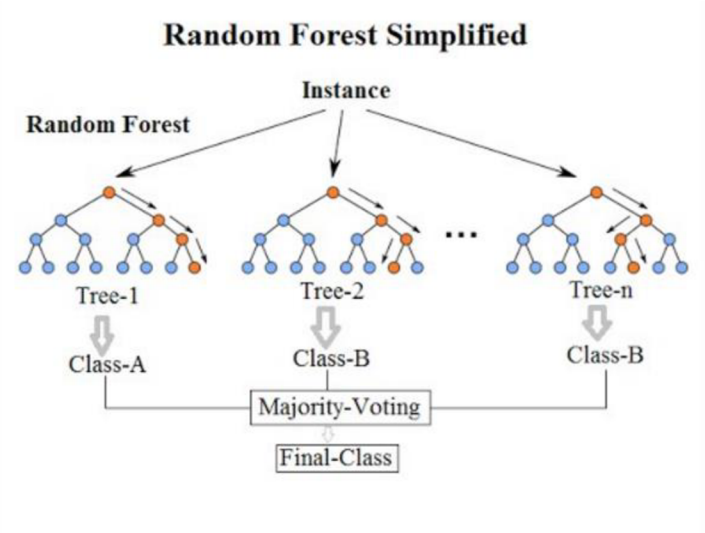
Pro síť Diabolo je počet volných parametrů roven [1]:

$$n_{free\_diab} = h_{diab}(4d + 4h_{diab} + 5) + d \qquad [1](2.21)$$

kde h_{diab} počet neuronů v zúžené vrstvě.

2.6 Stromové struktury

Klasifikační algoritmus založený na principu rozhodovacího stromu pracuje s řadou pozitivních trénovacích dat, podle kterých je schopen sestavit rozhodovací strom pro zjištění pozitivních vstupních dat. Výhodou této metody je možnost, za pomocí dostatečného počtu vstupních testovacích vzorků a jejich anotací, vytvořit rozhodovací strom i pro několik typů pozitivních dat. Tuto metodu lze použít i na data, která by byla jinak velice obtížně klasifikovatelná, většinou se jedná o velké objemy dat, kontinuální funkce nebo pro data s řídkými instančními prostory [21].



Obrázek 21: Ukázka klasifikace pomocí rozhodovacího stromu [21]

Princip tvorby rozhodovacího stromu

Pro tvorbu rozhodovacího stromu je používán cyklický návrh pro nalezení optimálního řešení:

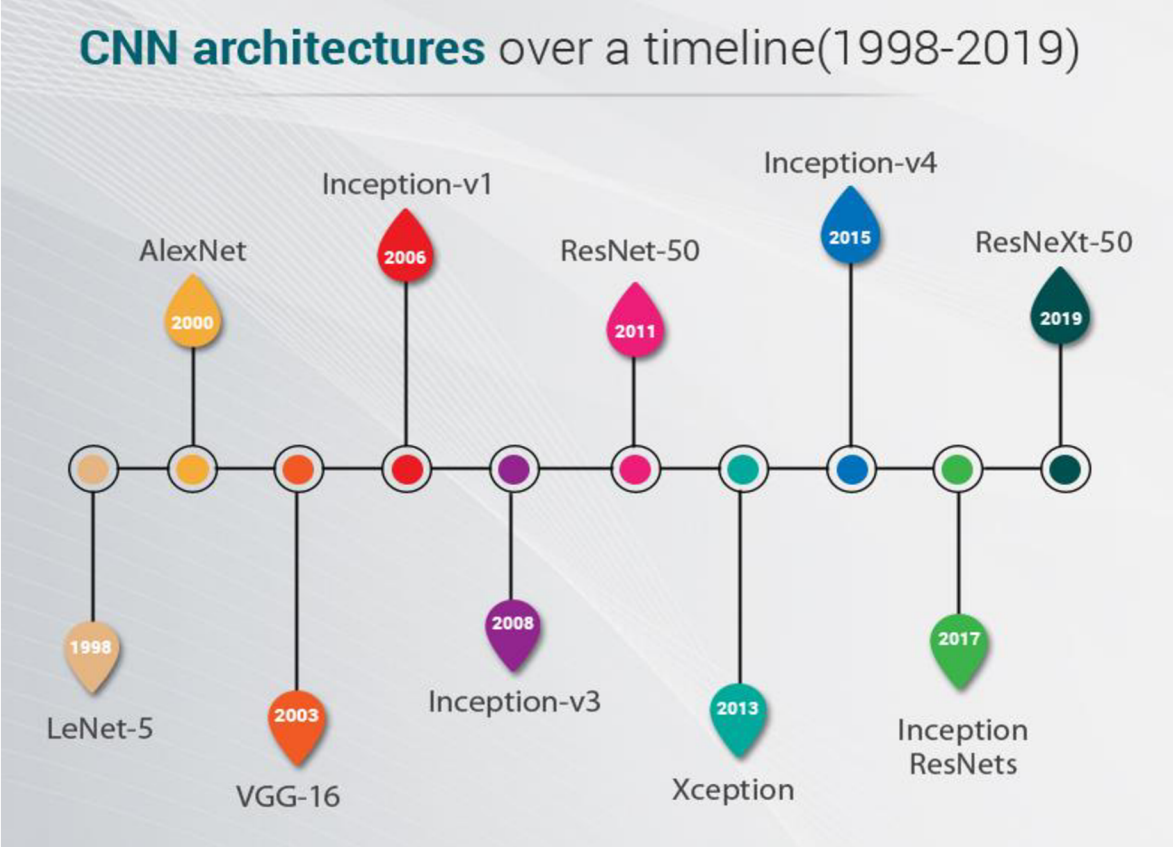
1. Získání informací o uzlu
2. Rozhodni o uzlu, zda bude dál dělen
3. Vyber nejlepšího atributu na větvení
4. Rozděl data do nových uzlů

Prořezání stromu – Na rozhodovací stromy lze uplatnit metodu Occamova ostří, tedy nejjednodušší rozhodovací strom konzistentní s trénovacími daty je pravděpodobně ten nejvhodnější. [21]

Ukončovací podmínky – O ukončení daného rozhodovacího stromu nebo samostatné větve lze rozhodnout na základě rozhodovacích podmínek. Ukončovací podmínky mohou být následující: dosažení maximální hloubky, dosažení maximálního počtu uzlů, dosažení požadované přesnosti nebo nedostatečný počet trénovacích dat. [21]

3. CNN ARCHITEKTURY

V hlubokém učení je konvoluční neurální síť (CNN nebo ConvNet) třídou neuronových sítí, která se nejčastěji používá k analýze vizuálních obrazů. Tyto sítě mají uplatnění v rozpoznávání obrazu a videa, optimalizačních systémech, klasifikaci obrazu, segmentaci obrazu, analýza lékařského obrazu, zpracování jazyka, řešení rozhraní mozek-počítač a analýze finanční a časové řady. Tyto sítě mají celou řadu architektur, nicméně tyto architektury mají společné rysy. Každá tato architektura se skládá ze vstupní, výstupní vrstvy a několika skrytých vrstev. Za skryté vrstvy jsou považovány ty vrstvy, u kterých jsou vstupy a výstupy maskovány aktivační funkcí a konvolucí.[24][25]



Obrázek 22: Historický vývoj CNN architektur [26]

Na obrázku 22 je možné vidět historický vývoj CNN architektur. Pro popis následujících architektur byly použity schémata viz. kapitola 3.5. Pro popis je nutné definovat funkce jednotlivých funkčních bloků. Obecná funkce jednotlivých bloků zůstává stejná ve všech uvedených případech.

3.1 Definování jednotlivých funkčních vrstev CNN architektur:

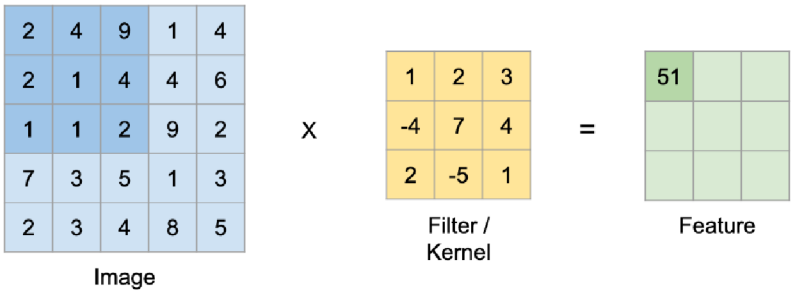
Každá CNN architektura se skládá z jednotlivých vrstev. Tyto vrstvy mohou mít v různých architekturách rozdílné parametry, nicméně základní funkce je ve všech případech stejná. Mezi nejtypičtější používané vrstvy je potom možné zařadit konvoluční vrstvu, sdružovací vrstvu a aktivační funkci.

3.1.1 Konvoluční vrstva

Každá konvoluční vrstva v CNN architekturách má tři parametry, jsou to Kernel, Stride a Pooling. Tyto parametry definují požadavky na výpočet konvoluční vrstvy.

Kernel

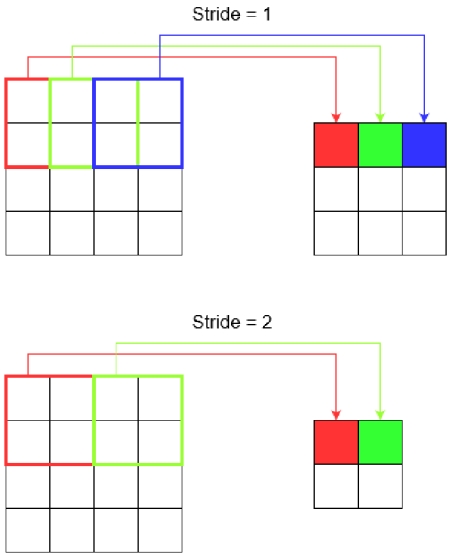
Tento parametr udává velikost konvolučního filtru, pro CNN architektury je zadáván ve čtvercové velikosti NxN, nejčastěji se používá ve velikostech 1x1, 3x3 a 5x5. Jak vidíme na obrázku 23, konvoluce se provádí postupnou aplikací filtru na jednotlivé části vstupní mapy, následně je výsledek zapsán do odpovídající buňky výstupní mapy. Filtry různých velikostí jsou schopny detekovat prvky různých velikostí, proto se u některých CNN architektur můžeme setkat s paralelní aplikací několika velikostí konvolučních vrstev.[26]



Obrázek 23: Princip kernel funkce v konvoluční vrstvě [26]

Stride

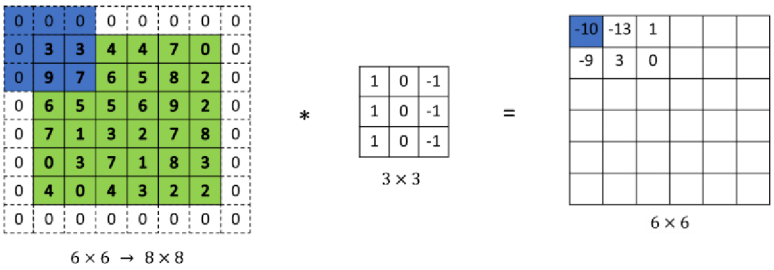
Stride parametr česky označovaný jako krok. Je parametr udávající velikost posunu konvolučního filtru mezi jednotlivými operacemi. Tato hodnota je primárně nastavena na jedna, nicméně ne pro všechny operace je tato hodnota vhodná. Celkově dokáže tento parametr ovlivnit výstupní velikost mapy. Tento parametr je vhodné využít u hlubokých architektur, kde za pomocí hlubší znalosti můžeme vyloučit případnou ztrátu dat způsobenou kompresí výstupu. Srovnání posunu konvolučního filtru je možné vidět na obrázku 24, na tomto obrázku je srovnáván posun filtru pro krok 1 a 2. [26]



Obrázek 24: Srovnání posunu konvolučního filtru pro dvě hodnoty kroku [26]

Padding

Česky označován jako výplň, je parametr umožňující manuální zvětšení vstupní mapy přidáním nulových hodnot na kraje vstupní mapy, jak lze vidět na obrázku 25. Tento parametr je vhodné nastavovat u velmi hlubokých sítí, kde hrozí velmi velké zmenšení mapy funkcí.[26]



Obrázek 25: Zvětšení vstupní mapy konvoluce pomocí výplně [26]

Speciální využití konvoluce

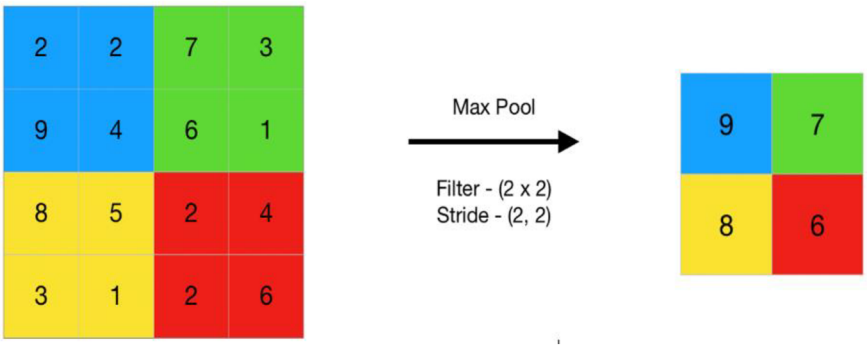
Speciálním případem je konvoluce 1x1. Takto navržená konvoluční vrstva nemá za cíl snížit rozměrnost dat standartním způsobem. Místo aby snižovala rozměrnost dat, tak snižuje hloubku těchto dat. V praxi potom může být na data o rozměrech 56x56 s hloubkou 64 aplikována konvoluce 1x1 s 32 filtry, výsledkem jsou potom data o rozměrech 56x56 s hloubkou 32.[27]

3.1.2 Sdružovací vrstva

Sdružovací vrstva je v konvolučních sítích obecně používána pro zmenšení objemu dat a zrychlení výpočtu. Ke zmenšení dat dojde na základě filtrace vstupních dat filtrem dané velikosti. Existují dva typy sdružování.

Maximální sdružování

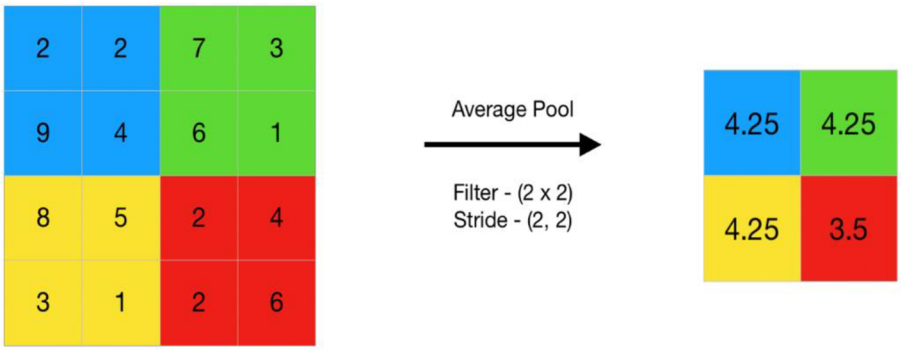
Tato operace sdružování, vybírá maximální prvek z oblasti mapy prvků pokryté filtrem. Výstupem po vrstvě maximálního sdružování je tedy mapa funkcí obsahující nejvýznamnější prvky vstupní mapy. Princip maximálního sdružování je na obrázku 26.



Obrázek 26: Aplikace maximální sdružovací vrstvy [28]

Průměrné sdružování

Tato operace sdružování vypočítá průměr prvků přítomných v oblasti mapy prvků pokryté filtrem. Zatímco operace maximálního sdružování poskytuje nejvýznamnější prvek z konkrétní vstupní mapy funkcí, průměrné sdružování poskytuje průměr prvků přítomných v dané oblasti mapy prvků. Princip průměrného sdružování je na obrázku 27.



Obrázek 27: Aplikace průměrné sdružovací vrstvy [28]

3.1.3 Slučovací vrstva

Jedná se o vrstvu v CNN sítích umožňující slučovat více větví CNN sítí dohromady. V CNN sítích není neobvyklé rozdělení sítě do několika větví, každá z těchto větví potom na data aplikuje jiné filtry nebo funkce. Cílem jednotlivých větví je nalezení rozdílných vzorů v datech. Například tyto větve umožňují srovnávat objekty, které jsou na snímcích jinak zvětšeny za pomocí různě velkých filtrů, tento princip lze nalézt v Inceptions modulu. Slučovací vrstva může obsahovat jednu ze dvou slučovacích funkcí:

Slučovací funkce – „Add“

Slučovací funkce je schopna přičíst k tensoru konstantní hodnotou, nebo sčítat tensor s tensorem. Funkce má dva parametry, vstupní tensor (input) a přičítané číslo nebo druhý vstupní tensor (other). Funkce má dále dva argumenty, prvním je číslo násobící druhý vstupní tensor (alpha) a výstupní tensor (out). [29]

Tensor sčítaný s číslem:

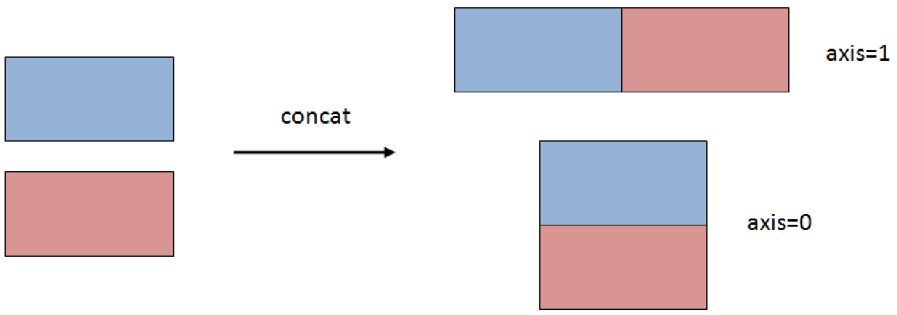
$$out = input + other \qquad [29]\ (3.1)$$

Tensor sčítaný se tensorem:

$$out = input + alpha * other \qquad [29]\ (3.2)$$

Slučovací funkce – „Concad“

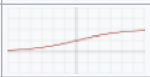
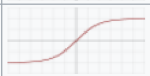
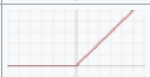
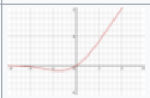
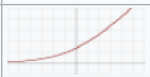
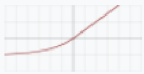
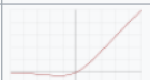
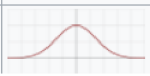
Slučovací funkce je schopna spojovat tensory podél definované osy. Funkce má dva parametry, matici vstupních tensorů (tensor) a parametr pro definování osy sloučení (dim). Funkce umožňuje tensory sloučit v řádcích (dim = 1) nebo ve sloupcích (dim = 0). [30]



Obrázek 28: Ukázka slučování pomocí CONCAT [30]

3.1.4 Aktivační funkce

V základu existují dva typy aktivačních funkcí, a to lineární a nelineární. Základním rozdílem mezi těmito funkcemi je omezení výstupního rozsahu, lineární funkce nikterak neomezují svůj výstup, zatímco u nelineárních funkcí dochází určitým způsobem k omezení výstupu. Nelineární funkce na druhou stranu umožňuje modelu se lépe zobecňovat nebo se přizpůsobit různým datům na vstupu.[31]

Name	Plot	Function, $f(x)$
Identity		x
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$[1]
tanh		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU)[7]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU)[4]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$
Softplus[8]		$\ln(1 + e^x)$
Exponential linear unit (ELU)[9]		$\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α
Scaled exponential linear unit (SELU)[10]		$\lambda\begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$
Leaky rectified linear unit (Leaky ReLU)[11]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$
Parameteric rectified linear unit (PReLU)[12]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter α
Sigmoid linear unit (SiLU,[4] Sigmoid shrinkage, [13] SiL,[14] or Swish-1[15])		$\frac{x}{1 + e^{-x}}$
Mish [16]		$x \tanh(\ln(1 + e^x))$
Gaussian		e^{-x^2}

Obrázek 29: Srovnání aktivačních funkcí [32]

Mezi nejvíce používané aktivační funkce v dnešních neuronových sítích lze zařadit Sigmoid, Tanh, ReLu a Leky-ReLu. Na obrázku 29 je uvedeno grafické srovnání jednotlivých aktivačních funkcí a jejich průběhů.[32]

Sigmoid

Hlavním důvodem využívání této aktivační funkce je hodnota jejího výstupu pohybující se v rozmezí mezi 0 a 1, což umožňuje výstup této funkce považovat za rozložení pravděpodobností. V dnešní době je často nahrazována funkcí Softmax.[31]

Tanh

Funkce velice podobná funkci Sigmoid, nicméně výstup je v rozsahu od -1 do 1, toto rozmezí má tu výhodu, že záporné vstupy jsou mapovány záporně, zatímco nulové vstupy jsou mapovány nulou. Funkce Tanh je používaná hlavně při klasifikaci dvou tříd. [31]

ReLu

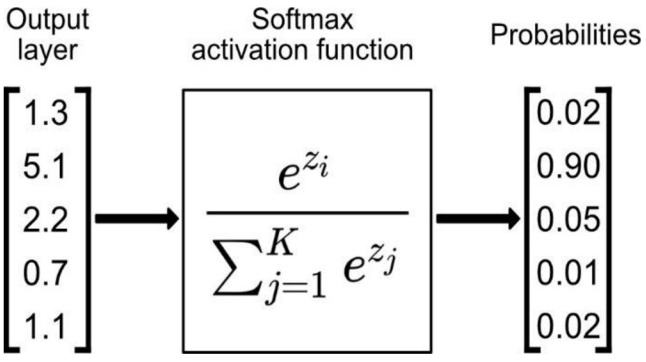
V současné době pravděpodobně nejpoužívanější aktivační funkce. Výstup funkce je potom hodnota od 0 po nekonečno s tím že, všechny záporné vstupy jsou mapovány nulovou hodnotou.[31]

Leaky Relu

Jedná se o zmodifikovanou funkci ReLu. Změna spočívá v úpravě mapování záporných hodnot, místo striktně nulové hodnoty jsou namapovány velice malou hodnotou obvykle 0,01.[31]

Softmax

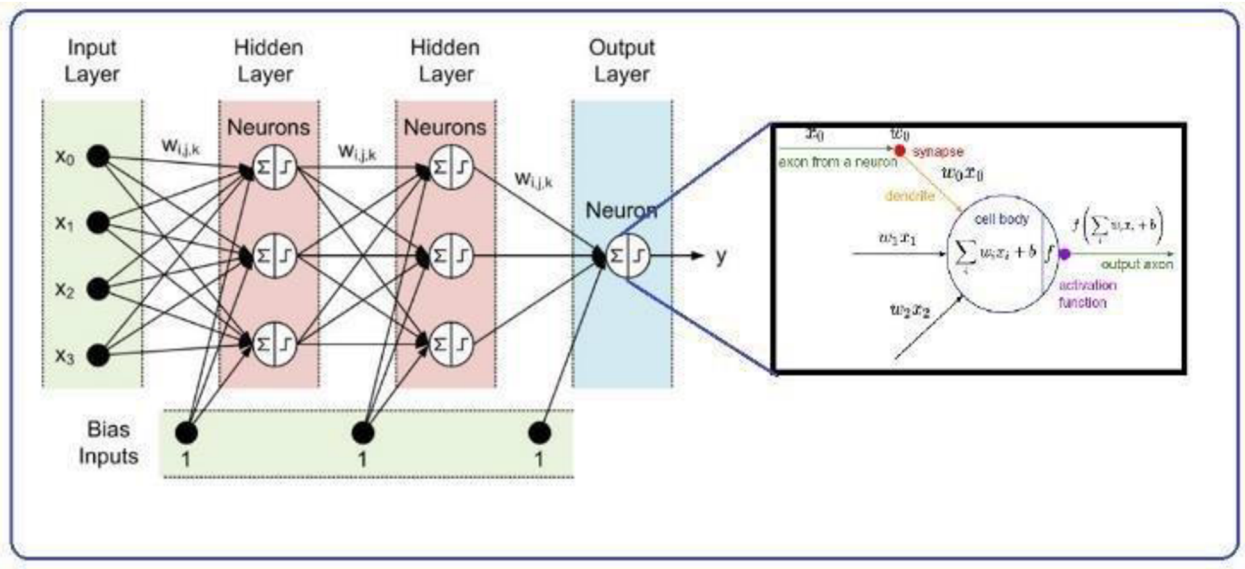
Poslední vrstva v CNN sítích je většinou tvořena funkcí Softmax. Tato funkce má na starosti normalizaci výstupu. Tedy před aplikováním funkce Softmax mohou některé vektorové komponenty nabývat hodnot větších než jedna nebo záporných hodnot. Funkce Softmax zajistí na výstupu hodnoty v rozmezí 0-1, takže je lze interpretovat jako hodnoty pravděpodobnosti. Tato funkce také zajistí, že celkový součet složek výstupního vektoru je roven 1. Tento princip je zobrazen na obrázku 30.[32]



Obrázek 30: Normalizace výstupu pomocí Softmax aktivační funkce [33]

3.1.5 Plně připojená vrstva

Tento typ vrstvy slouží pro finální klasifikaci. Každá takováto vrstva se skládá z určeného počtu neuronů. Neurony v plně propojené vrstvě mají spojení se všemi prvky v předchozí vrstvě, toto propojení je zobrazeno na obrázku 31. Na tomto obrázku je možné vidět nejen propojení jednotlivých neuronů, ale i detail výstupu neuronu v závislosti na aktivační funkci a vstupech tohoto neuronu. Tyto vrstvy jsou potom obecně zodpovědné za klasifikaci vstupních dat.



Obrázek 31: Propojení neuronů a skrytých vrstev v CNN architektuře [34]

3.2 Hyperparametry

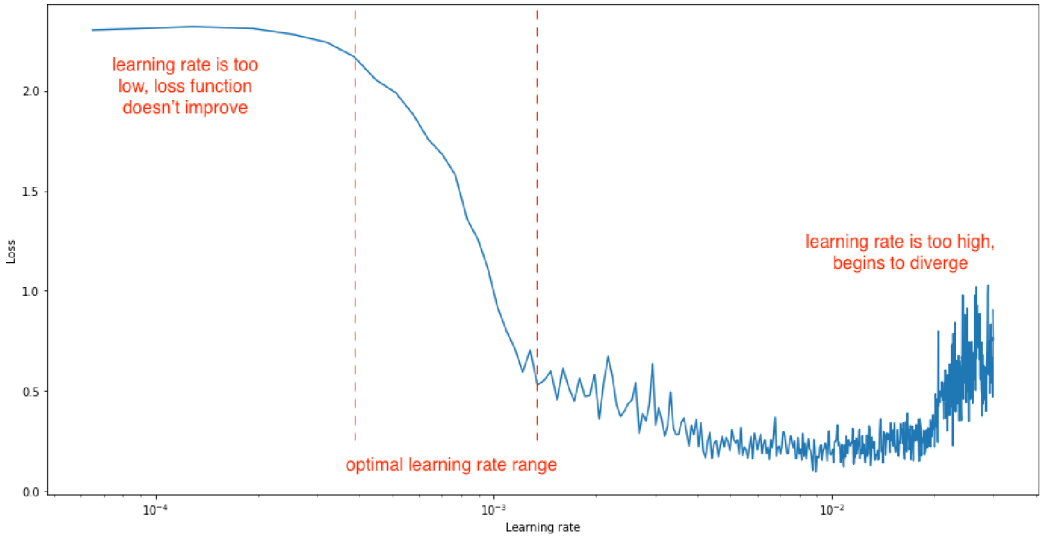
Kromě parametrů jednotlivých vrstev CNN sítí je nutné nastavit hyperparametry. Za hyperparametry lze považovat jakýkoliv parametr, který ovlivní výslednou přesnost klasifikace. Hodnoty jednotlivých hyperparametrů nelze dopředu přesně určit a je tedy nutné je určovat experimentálně. Pro jednotlivé hyperparametry existuje řada teoretických poznatků, které lze využít při jejich nastavování.

3.2.1 Krok učení

Hyperparametr je v anglické literatuře označován jako „learning rate“. Učení neuronových sítí je postaveno na principu optimalizace sestupu stochastického gradientu. Na tuto metodu má vliv celá řada hyperparametrů, za nejdůležitější hyperparametr je obecně považován krok učení. Tento hyperparametr má významný vliv na rychlost učení. Správná volba tohoto hyperparametru je velice obtížná. Tréninkový proces je pro nízký krok učení velmi dlouhý, zatímco pokud je zvolen velký krok učení, může dojít k neoptimální úpravě sady parametrů modelu. Velikost tohoto hyperparametru je volena mezi 0 a 1.[35]

Hledání optimální kroku učení

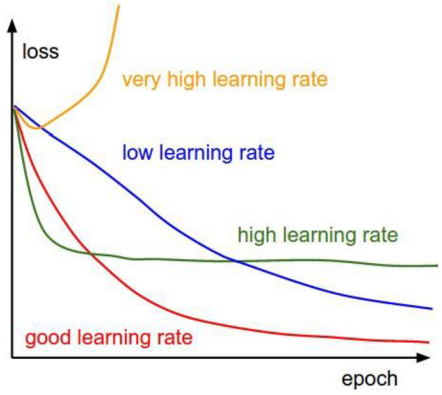
Hodnota tohoto hyperparametru je pro každý problém klasifikace jiná. Pro nalezení optimální hodnoty lze postupovat následovně. Postupně měnit hodnotu kroku učení od řádově 10^{-4} až po 10^{-2}, optimální krok učení se pro většinu klasifikačních úloh nachází právě v tomto rozsahu. Nevýhodou této metody je její časová náročnost, kdy malý krok učení vyžadují větší počet cyklů pro trénink. Modifikovaná varianta této metody je označována jako žíhání rychlosti učení, v anglické literatuře označována jako „learning rate annealing“. Modifikace spočívá ve volbě relativně velkého kroku učení a ten je následně snižován. Tato modifikace zajistí rychlý výpočet pro velké hodnoty kroku učení a následný rychlý přechod k nižším hodnotám. [35]



Obrázek 32: Vývoj chyby klasifikace v závislosti na velikosti kroku učení [35]

Jak je vidět na obrázku 32, lze krok učení rozdělit do tří oblastí podle jeho vlivu na celkovou chybu klasifikace. Oblast s malým krokem učení, v této oblasti jsou jednotlivé parametry modelu ovlivňovány krokem učení pouze minimálně. V této oblasti tedy nedochází ke snížení chyby klasifikace nebo pouze k malému snížení. Využití hodnoty kroku učení z této oblasti může vést k zaseknutí tréninkového procesu v lokálním minimu. Na druhou stranu lze dosáhnou vysoké přesnosti klasifikace i s malou hodnotou kroku učení, za předpokladu použití dostatečně velkého počtu tréninkových cyklů. Druhou část potom tvoří oblast optimálního kroku učení. Ve většině případů klasifikace je snaha volit krok učení právě z této oblasti. Správnou volbou hodnoty kroku učení je možné docílit rychlé konvergence k velké přesnosti klasifikace, při použití poměrně malého počtu cyklů. Poslední oblast je oblast s velkou hodnotou kroku učení. V této oblasti dochází k rychlé konvergenci k přesnému řešení. Problém s velkou hodnotou kroku učení spočívá v nestálosti výsledků. Tato nestálost se může projevit jako rychlá změna přesnosti klasifikace i při nepatrné změně kroku učení.

Na obrázku 33 je grafické srovnání vývoje chyby klasifikace pro různé velikosti kroku učení. V tomto grafickém srovnání je průběh chyby klasifikace s ideální velikost kroku učení zobrazen červeně, chyba klasifikace pro ideální krok učení klesá logaritmicky k minimální hodnotě chyby klasifikace.

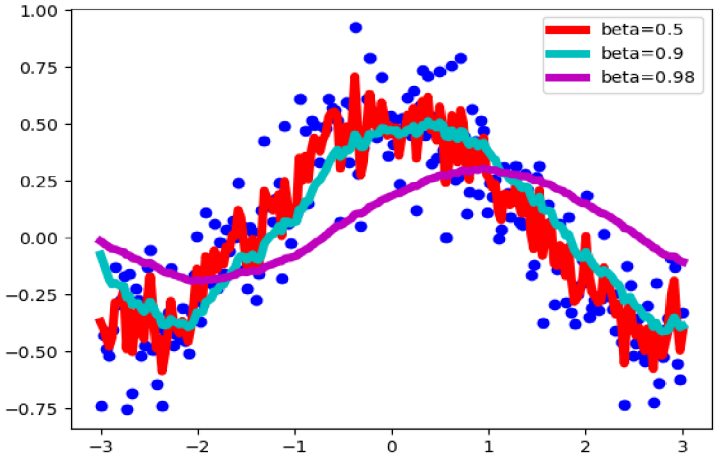


Obrázek 33: Průběh chyby klasifikace pro různé velikosti kroku učení [36]

3.2.2 Hybnost

Hyperparametr je v anglické literatuře označován jako „momentum“. Vliv hybnosti na strojové učení je obdobou vlivu hybnosti ve fyzikálních úlohách. Tento hyperparametr má tedy za cíl zohlednit vliv gradientu z minulých kroků v aktuálním kroku výpočtu gradientu. Správnou volbou tohoto hyperparametru lze zamezit zaseknutí tréninku v lokálních minimech a zamezení oscilace tréninkového procesu. [37]

Využití hybnosti si lze popsat na následujícím obrázku 34. Jak je vidět graf obsahuje tři datové řady. Tyto datové řady jsou generovány na základě vstupních dat (modré body). S rostoucí velikostí hybnosti dochází k lepšímu vyhlazení funkce dat za cenu většího posunu výsledné momentové funkce vůči originální funkci dat. Ve většině případů se lze setkat s hybností volenou buďto 0.5, 0.9 nebo 0.98. [37]



Obrázek 34: Srovnání posunu funkce pro různé hodnoty hybnosti [37]

3.2.3 Úbytek hmotnosti

V reálných příkladech klasifikace se lze setkat s velice velkými objemy dat. Velké objemy dat vedou ke složitým modelům, složité modely nemusí být vždy vhodné a je tedy nutné stanovit metody, které jsou schopny složitost modelů snížit. První a intuitivním řešením je omezení velikosti vstupních dat. Tento postup nemusí být ten nejvhodnější, protože pokud se sníží objem vstupních dat, sníží se i množství interakcí těchto dat v modelu a tím se omezí velikost nelinearity. Tyto nelinearity umožňují modelu řešit komplexní problémy. Druhým způsobem je právě úbytek hmotnosti. Tato metoda snižuje složitost modelu na základě aktualizace vah jednotlivých parametrů modelu. Problémem je, že některé parametry jsou pozitivní a některé negativní. Aby byl odstraněn tento nedostatek, je používána druhá mocnina těchto vah. V případě, že by chyba klasifikace byla příliš velká, je nezbytné velikost mocnin vah snížit. Parametr, který snižuje velikost vah, se nazývá úbytek hmotnosti. Z principu je patrné, že je nezbytné volit tento hyperparametr v intervalu od 0.01 do 0.1. Tato volba vychází z teoretického předpokladu, pokud je zvolen příliš velký úbytek hmotnosti model nikdy zcela neodpovídá realitě, zatímco pro nízké hodnoty úbytku hmotnosti lze natrénovat model relativně přesně.[38]

3.2.4 Ztrátová funkce

Ztrátová funkce se používá pro vyhodnocení kvality klasifikace pro aktuální nastavení vah. Při tréninku CNN sítí je snaha minimalizovat chybu klasifikace. Pro správný výpočet chyby klasifikace při tréninku sítě je nezbytné zvolit optimální ztrátovou funkci. To může být náročný proces z důvodu velkého množství parametrů, které je třeba zohlednit.[39]

MSE

Ztráta střední kvadratické chyby zkráceně MSE (ε_{MSE}) je počítána jako rozdíl čtverců skutečné (y_i) a předpovězené hodnoty (f(x_i,w)), kde predikována hodnota závisí na vstupu (x) a aktuálních vahách sítě (w). Výsledek je vždy pozitivní bez ohledu na znaménko predikovaných nebo skutečných hodnot. Cílem klasifikace je potom minimalizovat ztrátovou funkci MSE na hodnotu 0. Střední kvadratická chyba se počítá jako:

$$\varepsilon_{MSE} = \sum_i (y_i - f(x_i, w))^2 \quad [40](3.3)$$

CrossEntropy

V unární nebo binární klasifikaci je nejčastěji použita ztrátová funkce CrossEntropy. Tato funkce definuje klasifikační problém jako odhad pravděpodobnosti přítomnosti objektu v daná třídě 0 nebo 1. V praktické části práce jsou ve třídě 1 pozitivní průmyslové výrobky, zatímco třída 0 obsahuje všechny ostatní výrobky nebo znehodnocené pozitivní průmyslové výrobky.

Funkce CrossEntropy počítá skóre, které určuje rozdíl mezi skutečnou (y_i) a předpovězenou hodnotou (f(x_i,w)) pro predikci třídy 1. Cílem klasifikace je potom minimalizovat ztrátovou funkci. Pro ideální případ klasifikace je ztrátová funkce rovna 0.
Chyba metody CrossEntropy je rovna:

$$\varepsilon_{CE}(f(x_i, w), y_i) = f(x_i, w)^{y_i}(1 - f(x_i, w))^{1-y_i} \quad [40](3.4)$$

kde f(x_i,w) je predikována hodnota a y_i je skutečná hodnota.

3.2.5 Optimalizační funkce

Optimalizační funkce jsou algoritmy umožňující změnu atributů nebo parametrů modelu neuronové sítě s cílem snížit chybu klasifikace.

Gradient sestupu

Jedná se jednu se základních optimalizačních funkcí. Nejčastěji se používá v lineárních algoritmech. Tato funkce je závislá na první derivaci ztrátové funkce. Díky zpětnému šíření dochází k aktualizaci vah modelu na základě ztrátové funkce. Výhodou této metody je snadné použití. Největší nevýhodami této funkce jsou její malá rychlost pro velké datasety, uváznutí v lokálních minimech a paměťová náročnost výpočtu. Váhy modelu jsou aktualizovány:

$$\theta = \theta - \epsilon_k * \nabla_\theta \quad [41](3.5)$$

kde váhy modelu (θ) jsou aktualizovány v záporném směru o výslednou chybu klasifikace (∇_θ) násobenou krokem učení (ϵ_k).

SGD

SGD funkce aktualizuje váhy modelu v záporném směru přechodu (g) převzetím dávky dat o dané velikosti (m). Neuronová síť je zde reprezentována funkcí f(x_i, θ), kde x(i) jsou tréninková data a y(i) jsou štítky těchto dat. Gradient ztráty (L) je počítán s ohledem na váhy modelu (Θ). Krok učení (ϵ_k) definuje velikost kroku, který algoritmus provede (pro minimalizaci v záporném směru, pro maximalizaci v pozitivním směru). Váhy modelu (Θ) jsou aktualizovány následujícím způsobem:

$$g = \frac{1}{m} * \nabla_\theta \sum L(f(x^i; \theta), y^i) \quad [41](3.6)$$

$$\theta = \theta - \epsilon_k * g \quad [41](3.7)$$

AdaGrad

Jedná se o adaptivní metodu pro nastavitelnou velikost kroku učení. Lze uvažovat dva případy. Pro malou velikost gradientu (L) by bylo nutné zvolit velký krok učení (ε_k) pro dosažení optima. Zatímco pro velký gradient (L) je nutné zvolit malý krok učení (ε_k). Pokud dojde ke zvolení velkého kroku učení bude výsledek oscilovat kolem optima. Funkce AdaGrad tento problém řeší akumulací čtverců použitých přechodů (s) a dělením kroku učení druhou odmocninou tohoto součtu, váhy modelu (Θ) jsou aktualizovány následujícím způsobem:

$$g = \frac{1}{m} * \nabla_\theta \sum L(f(x^i; \theta), y^i) \quad [41](3.8)$$

$$s = s + g^T g \quad [41](3.9)$$

$$\theta = \theta - \epsilon_k * g / \sqrt{s + eps} \quad [41](3.10)$$

kde eps je volitelný parametr s doporučenou hodnotou eps = 1e-8.[41]

RMSprop

RMSprop upravuje funkci AdaGrad. Již nedochází k akumulaci gradientu (g), ale je počítán exponenciální vážený klouzavý průměr (s). Pro výpočet je nezbytné zadat hyperparametr úbytek hmotnosti (weight_dacay).

Váhy modelu (Θ) jsou aktualizovány následujícím způsobem:

$$g = \frac{1}{m} * \nabla_\theta \sum L(f(x^i; \theta), y^i) \quad [41](3.11)$$

$$s = Weight\_decay * s + (1 - Weight\_decay) g^T g \quad [41](3.12)$$

$$\theta = \theta - \epsilon_k * g / \sqrt{s + eps} \quad [41](3.13)$$

Adam

Adam je funkce odvozená od kombinace modelu RMSprop a hybnosti. V dnešní době se jedné o nejčastěji využívanou optimalizační funkci. Tato optimalizační funkce dosahuje nejvyšší přesnosti klasifikace pro většinu aplikací. Úprava spočívá v nahrazení stochastického přechodu (g) vyhlazenou funkcí stochastického přechodu (m, s). Výsledný mechanismus je doplněn o korekci zkreslení. Váhy modelu (Θ) jsou potom aktualizovány následujícím způsobem:

$$g = \frac{1}{m} * \nabla_\theta \sum L(f(x^i; \theta), y^i) \quad [41](3.14)$$

$$m = \beta_1 m + (1 - \beta_1) * g \quad [41](3.15)$$

$$s = \beta_2 s + (1 - \beta_2) g^T g \quad [41](3.16)$$

$$\theta = \theta - \epsilon_k * m / \sqrt{s + eps} \quad [41](3.17)$$

kde doporučená hodnota parametrů je: beta1 = 0.9, beta2 = 0.999 a eps = 1e-8.[41]

AdamW

Jedná se o variantu optimalizační funkce Adam doplněnou o úbytek hmotnosti. Jedná se o optimalizační funkci, kterou zavádí knihovna PyTorch. Pokles hmotnosti u této optimalizační metody je nastaven na hodnotu 0,01.[42]

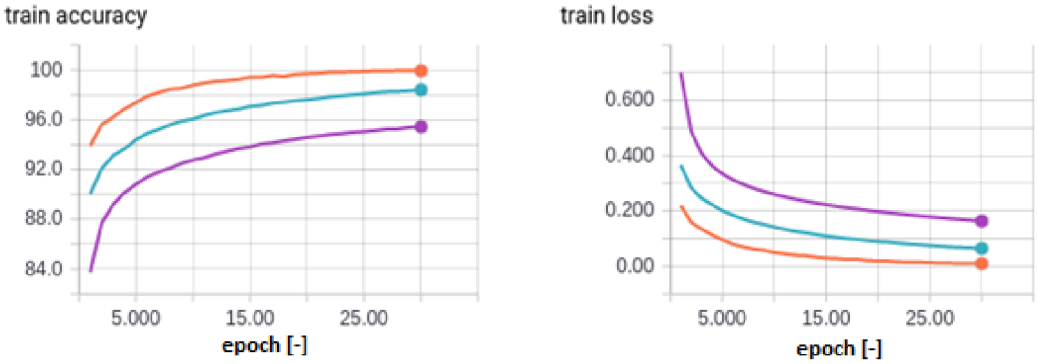
3.2.6 Rozložení datasetu

Rozložení datasetu je jedním z méně významných hyperparametrů. Volby ovlivňuje počet dat použitých pro trénink a počet dat použitých pro testování přesnosti modelu. Většina literatury považuje za minimální použitelné rozdělení datasetu na poloviny. Musí platit, že jak tréninkový, tak testovací dataset musí obsahovat objekty ze všech množin dat. Za ideální rozdělení datasetu jsou potom považována rozdělení 70/30 % nebo 80/20 %, při rozdělení tréninková/testovací data. Problém s rozdělením datasetu se stává závažnějším při vícetřídní klasifikaci. Kvalitu klasifikace lze podpořit anotací datasetu. [43]

3.2.7 Velikost dávky

Velikost dávky určuje počet objektů zpracovaných před aktualizací vah modelu. Velikost dávky se volí mezi 1 a celkovým počtem vzorků. Nejčastěji je velikost dávky volena 16,32 nebo 64. Jeden cyklus proběhne po zpracování všech dávek. Velikost dávky ovlivňuje rychlost učení a stabilitu tohoto procesu. Výsledný gradient, který je použit pro výpočet ztrát a aktualizaci vah modelu, je vypočten z gradientu jednotlivých dávek.[44].

Vliv velikosti dávky na přesnost klasifikace je možné vidět na obrázku 35. Pro srovnání byly použity tři velikosti dávky (oranžová = 64, modrá = 256, fialová = 1042). Jak je vidět změnou velikosti dávky lze výrazně ovlivnit přesnost klasifikace.[45]



Obrázek 35: Ukázka vlivu velikosti dávky na přesnost klasifikace a chybu klasifikace [45]

3.3 Vyhodnocení přesnosti klasifikace

Pro vyhodnocení přesnosti klasifikace CNN sítí existuje řada metod, většina těchto metod pracuje na základě matice záměny. Matice záměny popisuje počet správně a chybně klasifikovaných objektů. Pro sestavení matice je potřeba znát následující parametry matice:

Skutečně pozitivní

V anglické literatuře označován „True Positive“ - (TP), udává počet objektů, které byly klasifikovaný jako pozitivní a jsou skutečně pozitivní.

Skutečně negativní

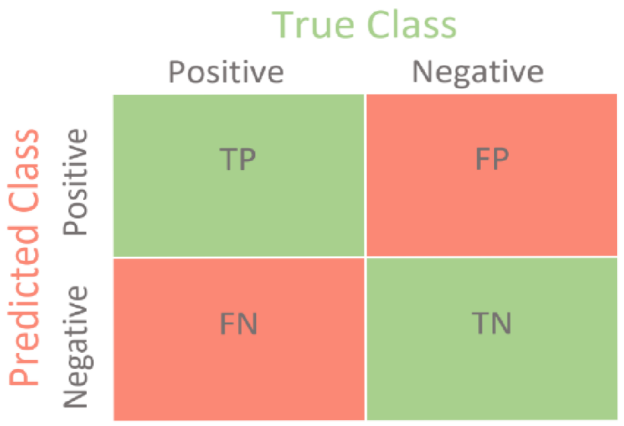
V anglické literatuře označován „True Negative“ - (TN), udává počet objektu, které byly klasifikovány jako negativní a jsou skutečně negativní.

Falešně pozitivní

V anglické literatuře označován „False Positive“ - (FP), udává počet objektů, které byly nesprávně klasifikovány jako pozitivní, ve skutečnosti patří do negativní třídy.

Falešně negativní
V angličtině označován „False Negative“ - (FN), udává počet objektů, které byly nesprávně klasifikovány jako negativní, ve skutečnosti patří do pozitivní třídy.

Na obrázku 36 je příklad matice záměny sestavené z těchto parametrů. Na základě této matice záměny jsou sestaveny matice záměny v praktické části práce.



Obrázek 36: Matice záměny a definování jejích parametrů [46]

Mezi nejčastěji používané metody pro určení přesnosti modelu lze zařadit přesnost, míru nesprávné klasifikace, preciznost, citlivost a F1 skóre případně Top-N.

Přesnost:
V anglické literatuře označována jako „accuracy“. Přesnost udává počet správně klasifikovaných objektů vůči celkovému počtu klasifikovaných objektů. Pro reálné případy klasifikace bude přesnost téměř vždy nižší než jedna.

$$Acc = \frac{(TP+TN)}{(TP+TN+FP+FN)} \qquad [46](3.18)$$

Míra nesprávné klasifikace:
V anglické literatuře označována jako „misclassification rate“. Míra nesprávné klasifikace udává počet chybně klasifikovaných objektů vůči celkovému počtu klasifikovaných objektů.

$$Misclassification\ Rate = \frac{(FP+FN)}{(TP+TN+FP+FN)} \qquad [46](3.19)$$

Pokud je známá přesnost klasifikace lze dopočítat míru nesprávné klasifikace:

$$Misclassification\ Rate = 1 - Acc \quad [46](3.20)$$

Preciznost

V anglické literatuře označována jako „precision“. U nerovnoměrně rozloženého datasetu nemusí být přesnost nejvhodnější metrikou, proto může být nahrazena preciznosti. Udává počet správně klasifikovaných pozitivních dat vůči celkovému počtu pozitivně klasifikovaných dat.

$$Precision = \frac{TP}{(TP+FP)} \quad [46](3.21)$$

Citlivost

V anglické literatuře označována jako „sensitivity“. Jedná se o modifikaci předchozího výpočtu. U této metody je zohledněn parametr FN namísto parametru FP.

$$Sensitivity = \frac{TP}{(TP+FN)} \quad [46](3.22)$$

Specifičnost:

V anglické literatuře označována jako „specificity“. Udává počet správně klasifikovaných negativních dat vůči všem negativním datům.

$$Specifity = \frac{TN}{(TN+FP)} \quad [46](3.23)$$

F1 skóre:

V anglické literatuře označována jako „F1-score“. V ideálním případě klasifikace chceme, aby parametry FN a FP byly nulové. Proto je potřeba zavést metodu která zohlední jak preciznost, tak citlivost.

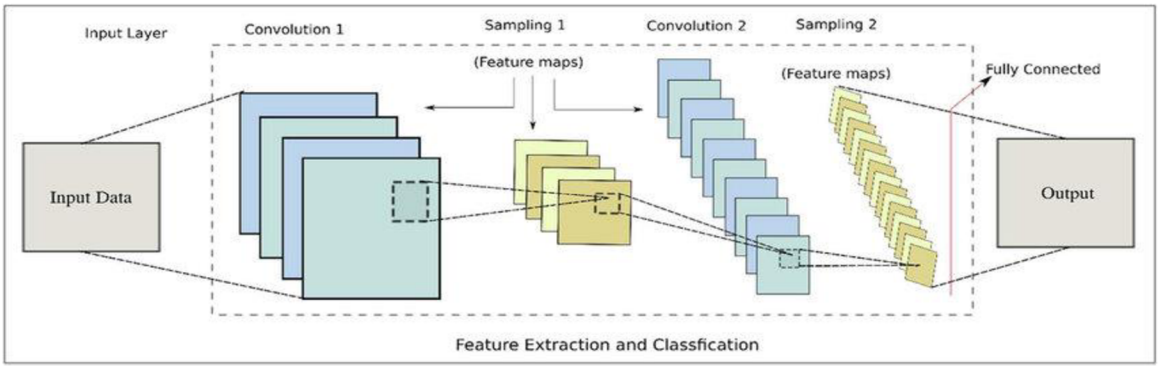
$$F1 = \frac{(2*TP)}{(2*TP+FP+FN)} = 2 * \frac{Precision*Recall}{Precision+Recall} \quad [46](3.24)$$

Top-N:
Jedná se o způsob vyhodnocení přesnosti klasifikace na základě kontroly daného počtu predikovaných hodnoty s cílovým štítkem. Nejčastěji používané metriky jsou Top-1 a Top-5. [46]

Top-1: Měří podíl příkladů, u kterých se predikovaný štítek shoduje s jediným cílovým štítkem.
Top-5: Považuje klasifikaci za správnou, pokud se některá z pěti předpovězených hodnot shoduje s cílovým štítkem.

3.4 Základní CNN architektura

Všechny CNN architektury jsou sestaveny na stejném principu. CNN architektura se skládá s několika druhů vrstev, jak je vidět na obrázku 37, prvním druhem je konvoluční vrstva. V této vrstvě probíhá operace konvoluce mezi vstupními daty a filtrem nastavené velikosti MxM. Výsledkem této operace je potom bodový produkt mezi filtrem a částí vstupního obrázku vzhledem k velikosti filtru. Výstupem potom je „Feature map“, která obsahuje informace o obrázku, například rohy a hrany. Další použitou vrstvou je sdružovací vrstva (v angličtině Pooling). Tato vrstva má za cíl snížit celkový objem dat a tak urychlit výpočet. K tomuto účelu používá nejčastěji funkci „Max Pooling“, tato funkce přebírá pouze největší prvek v předdefinované oblasti dat. A jedním z nejdůležitějších parametrů je potom aktivační funkce. Tato funkce rozhoduje o tom, které informace se mají použít a zároveň přidává do sítě nelinearitu. Tímto umožňuje vytvářet aproximace mezi jednotlivými složitými vztahy a mezi jednotlivými proměnnými v síti. [47]



Obrázek 37: Příklad jednotlivých vrstev CNN architektury [47]

3.5 Architektury vhodné pro unární klasifikaci

V následujících kapitolách jsou uvedeny některé z typických architektur vhodných pro unární klasifikaci obrazových dat. Pro lepší přehlednost je každá architektura popsána graficky, každé funkci v architektuře je přiřazeno barevné označení a jsou u ní uvedeny poznámky pro konstrukci dané vrstvy.

červeně – jsou zde označeny konvoluční operace, velikost dané operace je vždy uvedena v daném bloku

šedě - jsou zde označeny sdružovací funkce, jestli se jedná o průměrnou nebo maximální sdružovací funkci udává popisek (avg-pool = average pooling, max-pool = maximum pooling)

fialově - jsou zde uvedeny slučovací operace, jednotlivé architektury potom používají dva druhy slučování, a to add a concat

modře – jsou zde označeny nelineární vrstvy, obecně se jedná o vrstvy podobné lineárním vrstvám, ale výsledek je zde předán aktivační funkcí

Aktivační funkce:

T- Tanh aktivační funkce

R- Relu aktivační funkce

Ostatní funkce:

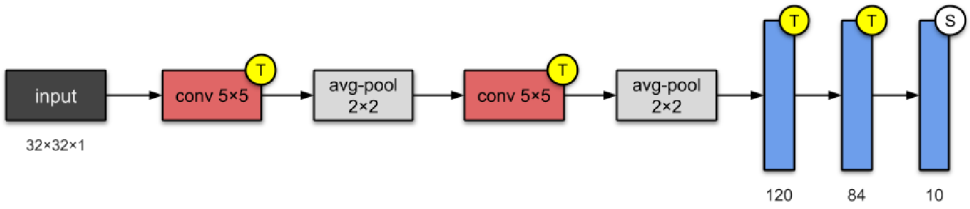
B – Batch normalizace

S – Softmax

x2 – Pokud jsou některé funkce opakovaně řazeny za sebe je toto opakování uvedeno pod daným blokem.

3.5.1 LeNet-5

Lenet-5 je jedním z prvních předem vyškolených modelů navržených Yannem LeCunem a dalšími v roce 1998. Tuto architekturu použili k rozpoznávání ručně psaných a strojově vytištěných znaků. Hlavním důvodem popularity tohoto modelu je jeho jednoduchá a přímá architektura. Jedná se o vícevrstvou konvoluční neuronovou síť pro klasifikaci obrazu. V době svého vzniku dosahoval LaNet-5 skvělých výsledků a byl převážně používán k rozpoznávání textu a čísel na bankovkách, některé bankomaty ještě dnes využívají tuto architekturu pro klasifikaci bankovek.[48]



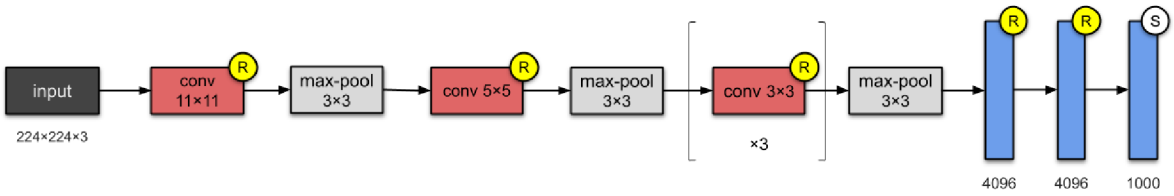
Obrázek 38: Ukázka struktury architektury LaNet-5 [25]

Tabulka 1: Parametry jednotlivých vrstev pro LaNet-5 [49]

Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

3.5.2 AlexNet

Architektura vznikla z potřeby klasifikovat objemné datové soubory o milionech obrázků. Vznikla tak architektura, která je velice rychle trénovatelná na velkém množství snímků, zároveň jsou však kladeny požadavky na co nejmenší rozlišení daných snímků. Vhodnou vstupní datovou sadou může být například ImageNet, jedná se o datovou sadu obsahující 14 milionů obrázků spolu s anotacemi. Architekturu AlexNet lze použít například pro detekci v počítačovém vidění s umělou inteligencí. [50]



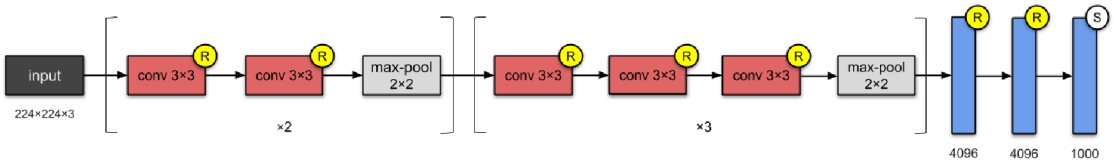
Obrázek 39: Ukázka struktury architektury AlexNet [25]

Tabulka 2: Parametry jednotlivých vrstev pro AlexNet [51]

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-

3.5.3 VGG-16

Jedná se o architekturu odvozenou z architektury AlexNet. Architektura vznikla v roce 2014 na Oxfordské univerzitě a jejími autory byly K. Simoyanem a A. Zissermanem. Opět se jedná o architekturu určenou pro klasifikaci velmi objemných datasetů. Hlavním rozdílem oproti AlexNet je změna velikosti vstupních filtrů řadou po sobě jdoucích filtrů o velikosti 3x3. Mezi další výhody patří možnost zpracovávat i obrázky ve vysokém rozlišení a relativně snadná implementace. Mezi největší nevýhodu patří velmi dlouhá doba tréninku. Architekturu VGG-16 lze použít například pro analýzu rentgenových snímků covid-19 pozitivních pacientů.[52]



Obrázek 40: Ukázka struktury architektury VGG-16 [25]

Tabulka 3: Parametry jednotlivých vrstev pro VGG-16 [53]

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

3.5.4 Inception-v1

Tato architektura vznikla ze základní myšlenky rozdílné velikosti zájmové oblasti v jednotlivých snímcích datasetu, v praxi se s tímto problémem můžeme setkat formou zmenšení/zvětšení klasifikovaného objektu na různých snímcích v datasetu. S tímto problémem je úzce spojen problém se správným výběrem velikosti filtru u konvoluce. Velké filtry jsou vhodnější pro informace distribuované globálně, zatímco menší jádro je potom výhodnější pro lokální informace. Výsledným řešením tohoto problému je potom architektura Inception-v1, jejíž základ tvoří paralelní filtry s různými velikostmi. Grafické znázornění dané architektury je potom uvedeno v příloze A. [54]

Tabulka 4: Parametry jednotlivých vrstev pro Inception v-1 [55]

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

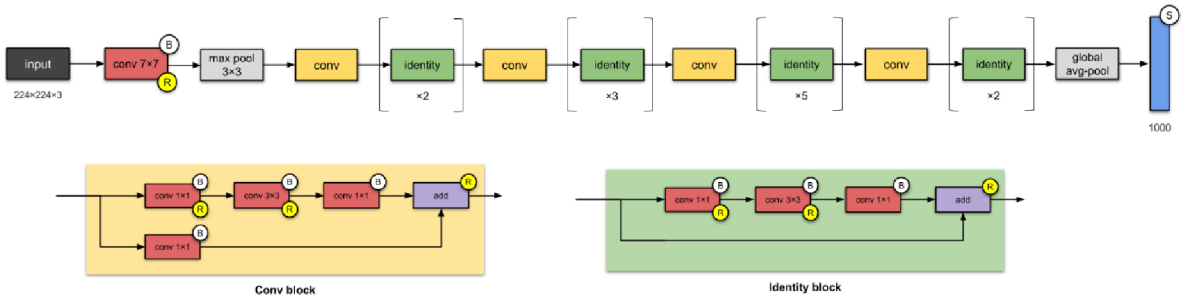
Jedná se o architekturu, do které je doplněn „Inception modul“. Tento modul obsahuje paralelní větve s různou velikostí filtrů. Výsledek modelu je potom dán paralelní kombinací výsledků jednotlivých filtrů, jak je patrné v příloze A. Další úpravou je potom zavedení pomocné klasifikace. Jelikož architektura Inception-v1 je již hodně hluboká architektura může se u ní projevit efekt mizejícího gradient.[55]

3.5.5 Inception-v3

Jedná se o přímé vylepšení předchozí architektury Inception-v1. Jak je možné vidět na modelu architektury v příloze B. Cílem je opět zvýšit výpočetní rychlost architektury a omezit ztrátu informace během výpočtu 5x5 konvolucí. Z matematického hlediska je konvoluce 5x5 2,78krát dražší než konvoluce 3x3, tedy pokud je konvoluce 5x5 nahrazena dvěma konvolucemi 3x3 dostáváme stejný výsledek v kratším čase. Dále architektura pracuje s ekvivalentním výpočtem konvoluce 3x3, kdy dojde k postupnému výpočtu konvoluce 1x3 a následně konvoluce 3x1. Tato metoda opět ušetří třetinu času potřebného pro výpočet. Praktickým příkladem použití může být vyhodnocování snímků ze zdravotnictví. Tyto metody jsou použity k sestavení tří klíčových modulů, ze kterých je následně architektura Inception-v3 tvořena.[56]

3.5.6 ResNet-50

Tento typ architektury se zaměřuje na řešení řady úkolů v počítačovém vidění. Mezi jeho největší předností patří možnost trénování extrémně hlubokých neurálních sítí se 150 i více vrstvami. Verze ResNet-50 je menší verzí klasické verze ResNet-152 a je často využívána jako výchozí bod pro vyhodnocení kvality učení. Tento typ architektur zavádí přeskočené připojení, tedy vstupní mapa dat není přivedena na vstup následující vrstvy, ale může být zavedena do libovolné následující vrstvy. Jedinou podmínkou je, že musí být zavedena do vrstvy odpovídající velikosti. Tímto způsobem lze omezit problém mizejícího gradientu. Model ResNet-50 se skládá z 4 částí, každá se skládá z konvolucí a bloku identity. Každý konvoluční blok má 3 konvoluční vrstvy a každý blok identity má také 3 konvoluční vrstvy. ResNet-50 má více než 23 milionů trénovatelných parametrů.[57]

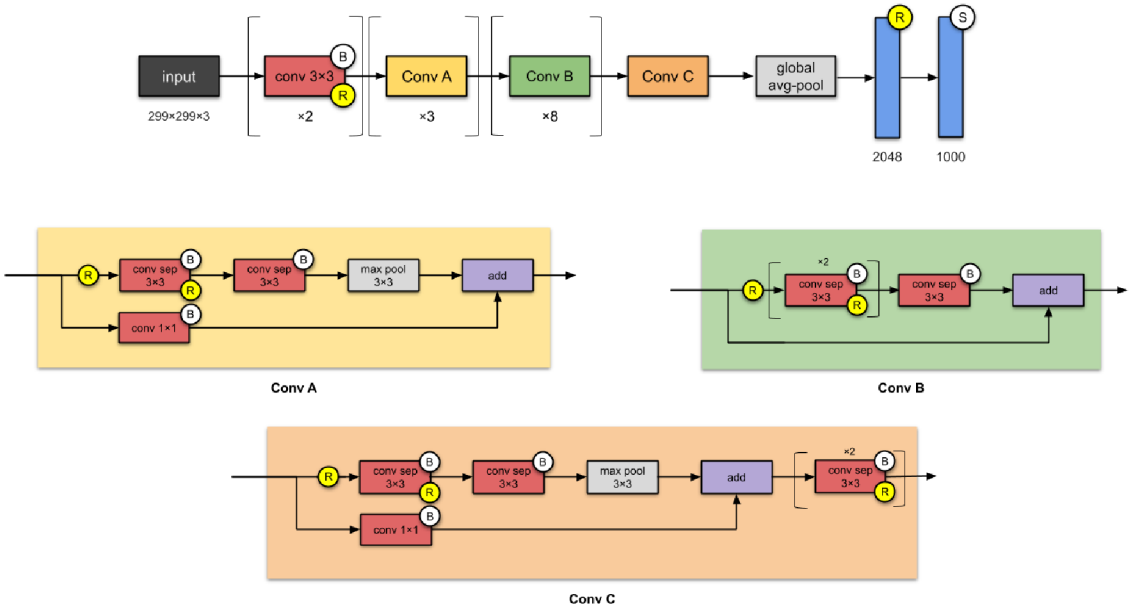


Obrázek 41: Ukázka struktury architektury ResNet-50 [25]

Zajímavou aplikací pro architekturu ResNet-50 je použití této sítě pro diagnostiku Alzheimerovy chorob. Cílem práce bylo natrénování sítě pro rozeznání typických poškození mozku.[58]

3.5.7 Xception

Architektura Xception zavádí několik malých změn oproti architektuře Inception-v3, ze které vychází, a to s cílem zrychlit celkový proces učení. Celá architektura je složená z 36 konvolučních vrstev rozložených do 14 modulů.



Obrázek 42: Ukázka struktury architektury Xception [25]

Typickou ukázkou použití Xception architektury může být opět vyhodnocování lékařských dat. V praxi se potom mohou vyhodnocovat anomálie v rentgenových snímcích pacientů, například metastázy na krční tepně. [59]

3.5.8 Inception-v4

Inception-v4 je konvoluční architektura neuronových sítí, která staví na předchozích iteracích rodiny Inception, zjednodušením architektury a použitím více počátečních modulů než Inception-v3. Celá architektura je potom v příloze C. Hlavní změnou oproti předchozí architektuře Inception-v3 je přidání Staem modulu na vstupu architektury. Přidáním tohoto modulu je dosaženu lepších výsledků sítě, a to hlavně z důvodu prohloubení architektury [25].

Mezi typické příklady použití je možné opět zařadit vyhodnocování lékařských dat, například studie vytvořená na základě architektury Inception-v4 na rozpoznávání rakoviny kůže [60].

3.5.9 Inception-ResNets

Architektura využívá zbytkové připojení (v angličtině residual blocks), toto propojení nepropojuje danou vrstvu s následující vrstvou, místo toho přeskakuje 2 až 3 vrstvy a až zde se opět pomocí slučovací vrstvy napojuje na danou větev. Tento způsob propojení se používá pro snížení složitosti modelu.

Inception-Resnet-v2 je definován na základě kombinace struktury Inception a zbytkového připojení, v bloku Inception jsou konvoluční filtry různých velikostí kombinovány se zbytkovými připojeními. Použití zbytkových spojení nejen předchází problémům s degradací způsobeným hlubokými strukturami, ale také zkracuje dobu tréninku. [25]. Grafické znázornění dané architektury je potom přiloženo v příloze D. Praktické využití má tato architektura například pro vyhodnocování snímků z mikroskopů a následné klasifikaci dat. [61]

3.5.10 ResNeXt-50

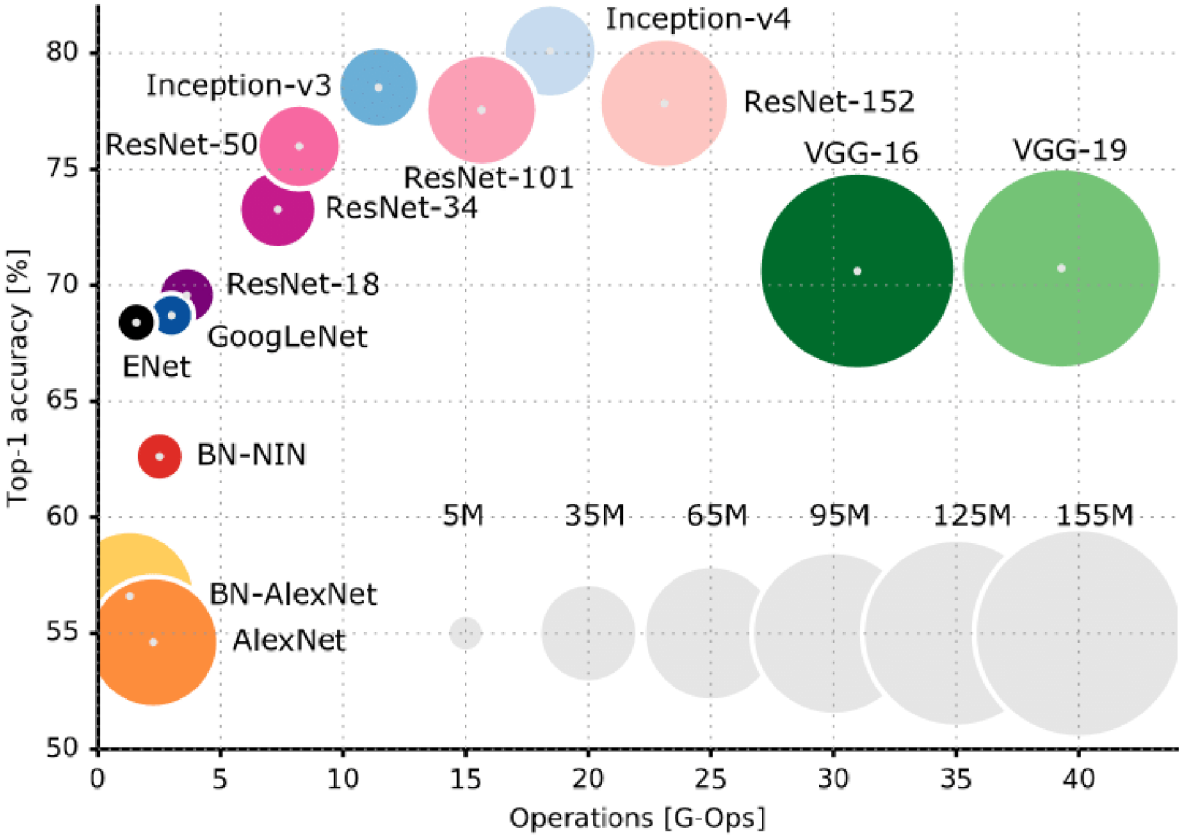
Architektura ResNeXt je rozšířením hluboké zbytkové sítě, která nahrazuje standardní zbytkový blok, blokem, který využívá strategii „split-transform-merge “. To znamená rozvětvení cesty v buňce, která je použita v modelech Inception. Jednoduše namísto provádění konvolucí na mapě plného vstupního prvku se vstup bloku rozdělí na několik samostatných větví sítě, které před sloučením výsledků samostatně používají několik konvolučních filtrů. Tato architektura vychází z rodiny architektur ResNet, ale je doplněna o paralelní bloky Conv a Identity, spojuje tak výhody jednotlivých architektur. Architektura je potom zobrazena v příloze E.[25]

3.5.11 GoogleNet

Architektura postavená na základě Inception modulu, který přebírá z Inception-v1 architektury. Architektura je potom zobrazena v příloze F. Zároveň tento blok doplňuje řadou metod pro snížení množství parametrů architektury a současně dochází k prohloubení architektury. Využitím konvoluce 1x1 dochází k výraznému snížení operací potřebných pro výpočet. V případě provedení konvoluce 5x5 se 48 filtry bez použití konvoluce 1x1, bude celkový počet operací 112,9 miliónu. Při použití mezikroku s konvolucí 1x1 bude celkový počet operací roven 5,3 miliónu. Tato redukce umožňuje výrazně snížit čas pro trénink sítě. U takto hluboké sítě je opět použit princip pomocných výstupů pro trénování sítě.[62] Celková chyba klasifikace je opět počítaná jak z reálného výstupu, tak s pomocí těchto pomocných ztrát. Jednou ze zajímavých aplikací je možnost vyhodnocení stavu pacientů s onemocněním COVID-19 na základě rentgenových snímků [63].

3.6 Srovnání architektur

Srovnání jednotlivých architektur lze provádět několika způsoby. Nejběžnějšími jsou srovnání na základě kvality klasifikace dané architektury. Dalším způsobem je srovnání na základě počtu cyklů potřebných k natrénování sítě dané architektury. Zároveň lze jednotlivé architektury srovnávat na základě jejich velikosti a počtu parametrů.

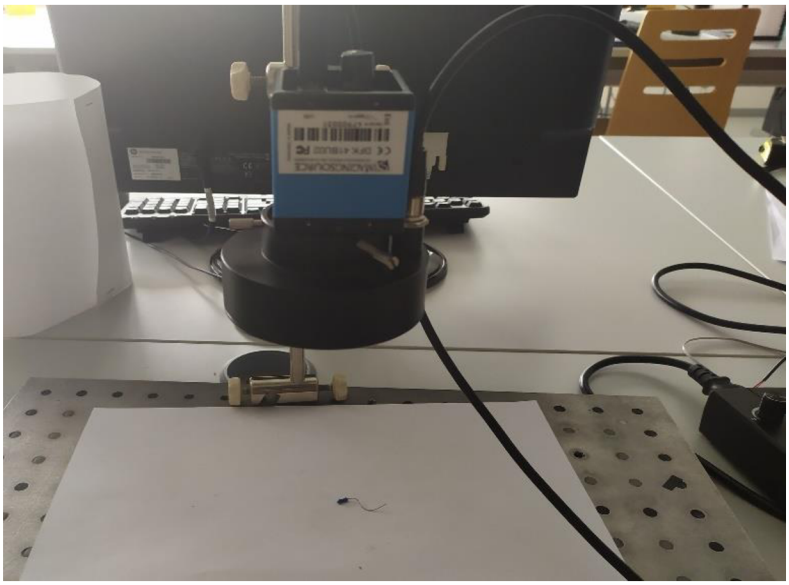


Obrázek 43: Srovnání přesnosti klasifikace jednotlivých CNN architektur [64]

Využitím srovnání z obrázku 43, lze vytvořit několik základních předpokladů, které lze využít při návrhu sítí v praktické části práce. Prvním z těchto předpokladů je přesnost klasifikace jednotlivých architektur. Ze srovnání je zřejmé že přesnost klasifikace jednotlivých architektur se pohybuje mezi 50-80 %. Druhým takovýmto předpokladem je počet cyklů potřebných pro trénování jednotlivých sítí. Jak je vidět na obrázku 43 je 20 cyklů zcela dostatečných pro trénink většiny typů sítí, menším sítím stačí 10 cyklů. A posledním předpokladem je velikost jednotlivých sítí. Tento předpoklad lze využít při nastavování velikosti dávky při tréninku sítě, logicky lze předpokládat, že mnohem větší sítě dovolí pouze menší velikosti dávky pro trénink.

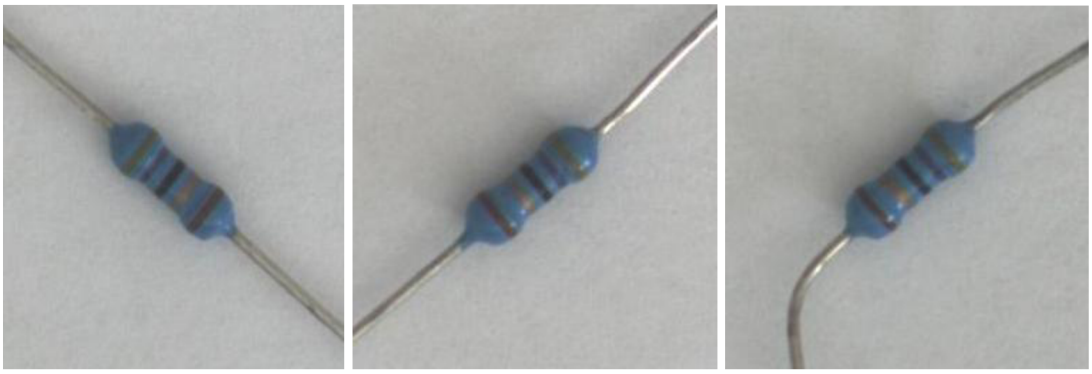
4. ANOTOVANÝ DATASET

Pro získání datasetu bylo použito zařízení z laboratoře optiky SE2.149. Pro nasnímání rezistorů byla použita kamera DFK 41BU02 se sériovým číslem 47900031 od společnosti ImagingSourse, spolu s objektivem Pentacon F1.8/50 mm. Za pozadí snímků byl zvolen standardizovaný bílý papír pro tisk digitálních fotek (A4, 250 g/m^2). Tento podklad byl zvolen z důvodu vysokého kontrastu vůči snímaným rezistorům a minimu nečistot. Pro nasvícení scény bylo použito kruhové světlo připevněné na kameře. Kruhové světlo bylo nastaveno na minimální výkon pomocí připojeného výkonového regulátoru. Zároveň byly snímky osvětlovány venkovním světlem. Toto nasvícení se projevuje změnou jasových podmínek na několika snímcích. Toto nasvícení bylo zvoleno z důvodu zabránění vzniku nežádoucích stínů na snímcích a odlesků na kovových částech rezistorů při použití zářivek v laboratoři. Kamera s objektivem byla umístěna 15 centimetrů nad snímaným povrchem.

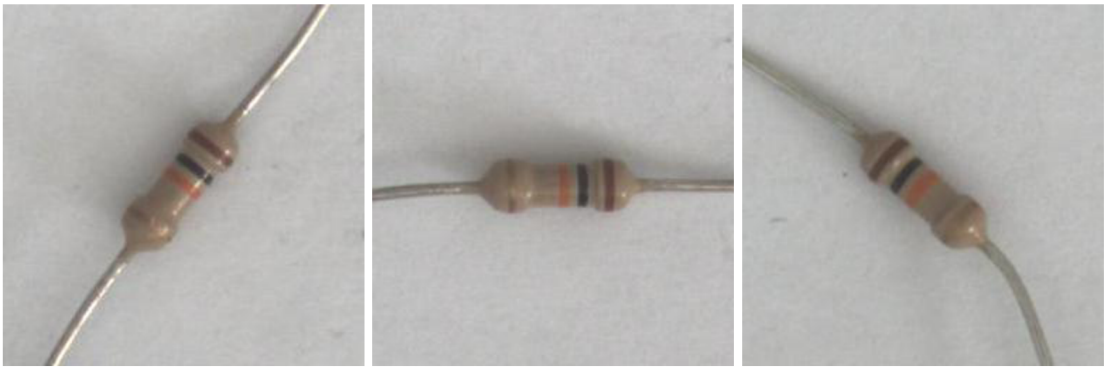


Obrázek 44: Pracoviště pro získání datasetu

Dataset obsahuje 1200 snímků reálných rezistorů a je rozdělen na dvě části. První část, tedy snímek 1-600, je složena ze snímků rezistorů 47 ohmů v modrém pouzdře. Tyto snímky jsou definovány jako pozitivní část datasetu. Druhá část je potom tvořena snímky rezistoru, které jsou definovány jako negativní. Typickými zástupci této třídy jsou snímky rezistorů s jinou barvou pouzdra, poškozenými přívody, rezistory s modrým pouzdrem, ale jiným čárovým označením a z rezistorů s poškozeným čárovým označením. Součástí práce je CSV anotace daného datasetu obsahující seznam snímků a jejich kategorií (1 pro pozitivní příklady a 0 pro negativní). Jednotlivé snímky byly před použitím pro trénink CNN architektur upraveny na velikost 256x256. Na následujících snímcích jsou uvedeny typické ukázky z datasetu:

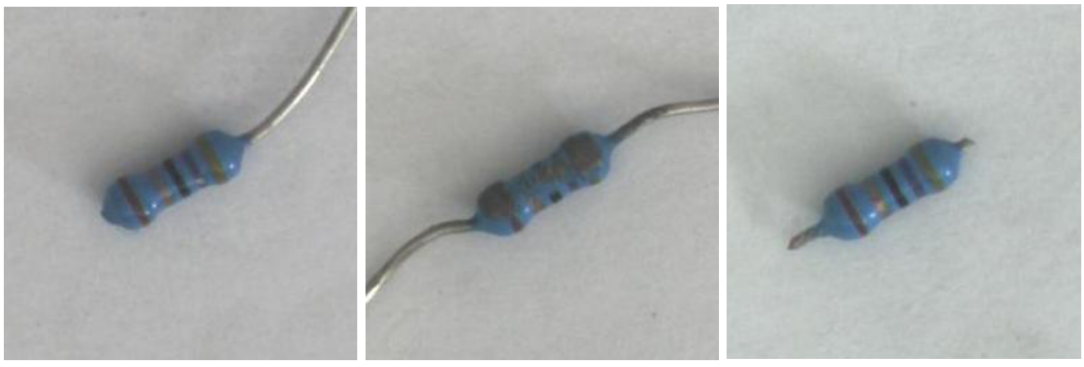


Obrázek 45: Typická ukázka pozitivního datasetu (rezistory 47 Ω v modrém pouzdru)



Obrázek 46: Typická ukázka negativního datasetu (jiná barva pouzdra)

Obrázek 47: Typická ukázka negativního datasetu (chybný nebo poškozený čárový kód)



Obrázek 48: Typická ukázka negativního datasetu (mechanické poškození rezistoru)

5. NÁVRH A VYHODNOCENÍ IMPLEMENTACE

Poslední část práce se zaměřuje na provedení implementace v prostření Python. Cílem je vytvoření jednoduché aplikace pro testování jednotlivých architektur vhodných pro unární klasifikaci spolu s možností nastavování hyperparametrů architektury. Pro samotnou implementaci je používána knihovna PyTorch a rozšíření Cuda. Pro detailní analýzu jednotlivých CNN architektur je použita knihovna TensorBoard.

Cuda

Jedná se o rozšíření pro Python podporující výpočet na grafické kartě. Díky tomuto rozšíření je Python schopen efektivně provádět složité výpočty na grafické kartě při tréninku CNN sítí, jediným omezením se tak stává velikost paměti grafické karty. Pro výpočet na grafické kartě se nejprve projekt nahraje do paměti GPU. Tento krok může být neefektivní v případě jednoduchých operací, kdy samotné nahrávání do paměti GPU zabere řádově mnohem více času, než samotný výpočet na CPU. Program je vytvořen tak, že je možné jej spustit i bez naistalováni knihovny Cuda, kdy celý výpočet bude probíhat na CPU a efektivita bude limitován velikostí pamětí RAM. Během praktického testování práce byla vždy použita grafická karta pro výpočet výsledků.

PyTorch

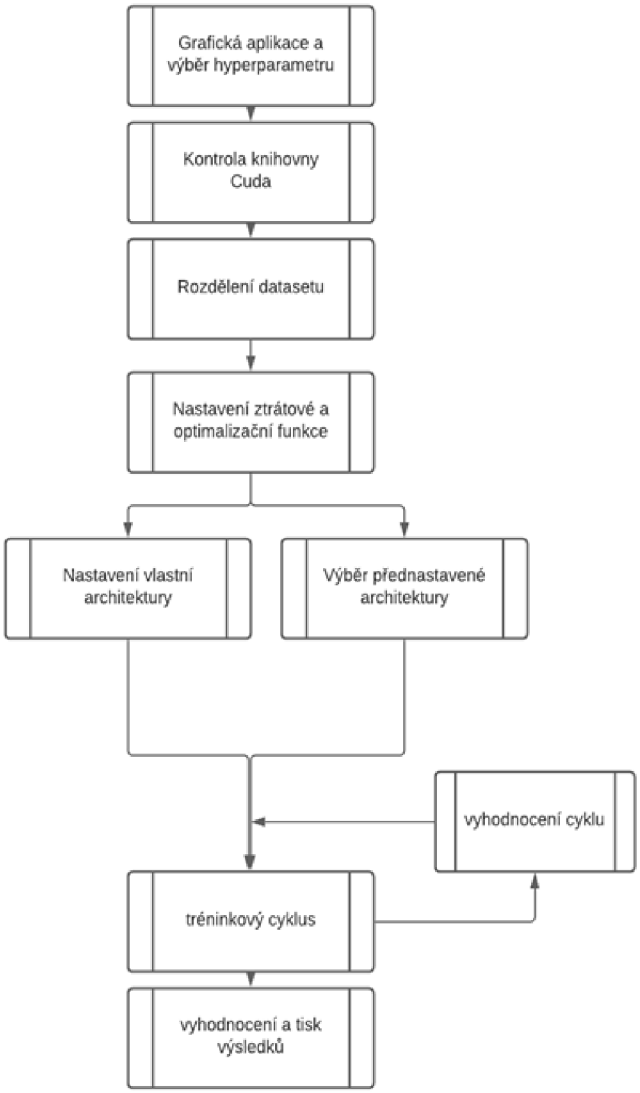
PyTorch je optimalizovaná knihovna tenzorů pro hluboké učení pomocí GPU nebo CPU. Samotná knihovna je volně stažitelná a osahuje všechny funkce pro tvorbu, optimalizaci a vyhodnocení CNN sítí. Tato knihovna je velice dobře optimalizována pro prostředí Python a zároveň je velice jednoduchá z uživatelského hlediska. Základ knihovny tvoří funkce pro výpočet tenzorů, součástí knihovny jsou i předpřipravené modely CNN sítí, které je možné využít. Pokud je daná předpřipravená síť používána poprvé musí se nejprve stáhnou ze serverů Pytorch.

TensorBoard

Jedná se o knihovnu umožňující analyzovat data. Knihovna je určena pro obecnou analýzu CNN sítí, jejích parametrů a umožňuje přidat další požadavky na analýzu. Knihovna TensorBoard umožňuje zobrazit výsledky buďto na off-line serveru nebo pomocí online aplikace TensorBoardDev. Online aplikace se v tuto chvíli nachází v předběžném přístupu, a proto má řadu omezení. Mezi největší omezení lze zařadit omezené místo pro ukládání vlastních dat a nemožnost ukládat obrazová data.

5.1 Rozbor implementace

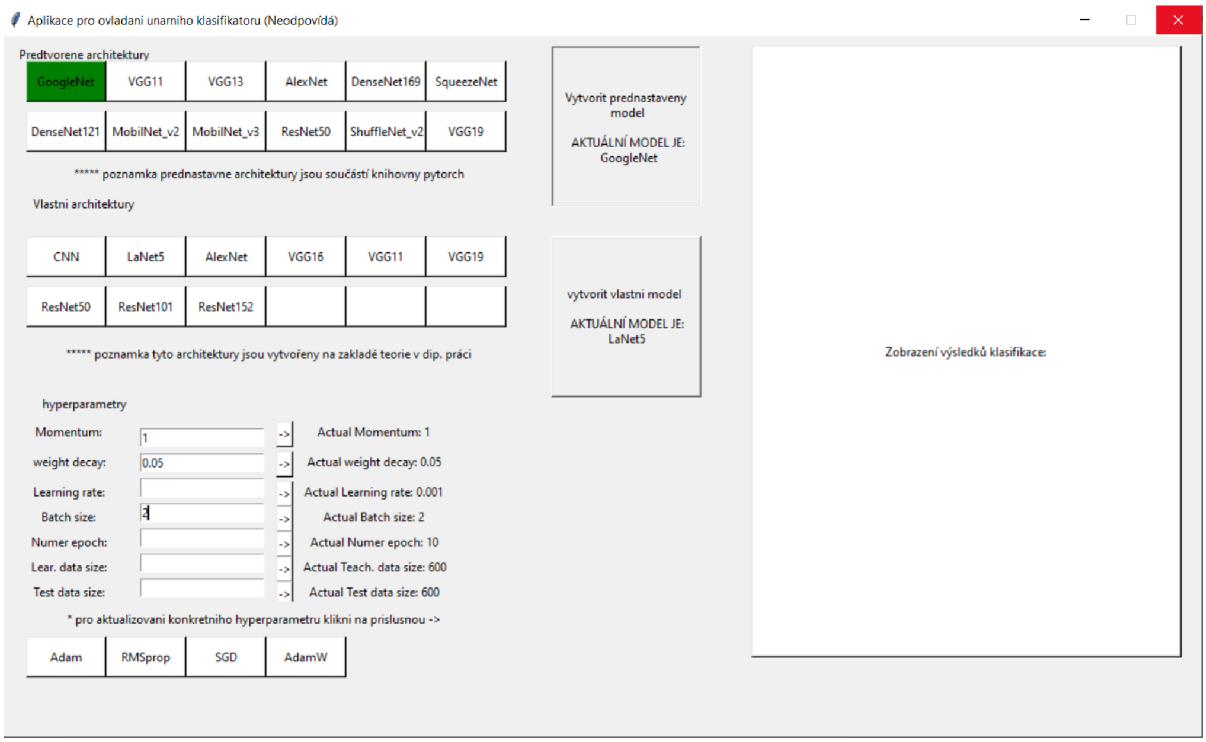
Jednotlivé bloky programu jsou popsány v kapitolách níže. Jednotlivé bloky potom tvoří logickou posloupnost kroků tvořící výsledný program pro trénink sítě a vyhodnocení přesnosti klasifikace. Celkový koncept programu je popsán na obrázku 49.



Obrázek 49: Blokové schéma implementace

5.1.1 Grafická aplikace

V zadání práce je vyžadováno provedení řady měření se změnou jednotlivých hyperparametrů, pro zjednodušení zadávání je součástí práce grafická aplikace sloužící jako vstup programu i jako výstup, pro zobrazení výsledků, samotnou grafickou aplikaci je vidět na obrázku 50. Samotné grafické okno lze rozdělit na několik částí. První část umožňuje volbu požadovaného modelu CNN sítě. Tato část je rozdělena na dvě poloviny, kdy v horní části jsou předpřipravené modely, které jsou součástí knihovny Pytorch a ve spodní části jsou na výběr navržené modely na bázi CNN architektur. Druhá část umožňuje definovat jednotlivé hyperparametry, zadávání není omezeno a je tedy možné zadat libovolnou hodnotu. Jednotlivé hyperparametry a jejich výchozí hodnoty jsou uvedeny v kapitole 3.2. Poslední část potom tvoří pole pro zobrazení výsledků. V této části jsou po ověření kvality klasifikace vypsány hyperparametry modelu, typ sítě, pro který byl model vytvořen a výsledná kvalita klasifikace je prezentovány jak ve formě procentuální přesnosti, tak i pomocí matice záměny.



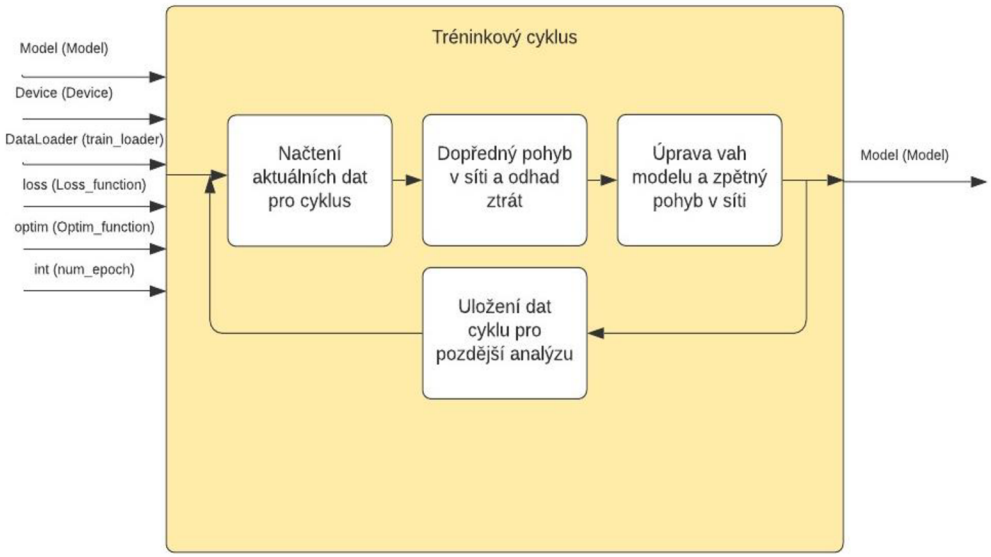
Obrázek 50: Grafická aplikace pro ovládání programu

5.1.2 Aplikace pro trénink a vyhodnocení

Hlavní část práce se zabývá tvorbou a tréninkem CNN sítí. Pro tvorbu a trénink sítí je vytvořen script v prostředí Python s názvem „*klasifikator_main.py*“. Tento script obsahuje funkci “*DPlearnink*“ odpovědnou za trénink sítě. Funkci lze rozdělit do několika po sobě jdoucích logických bloků, které přejímají hyperparametry z grafické aplikace, na jejichž základě trénuje požadovanou síť.

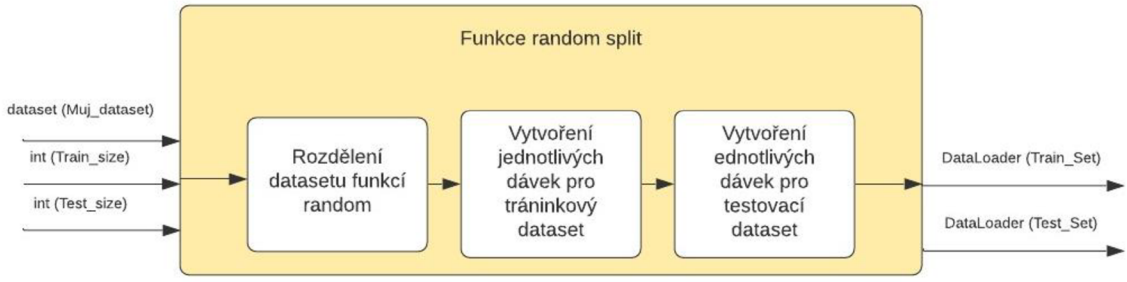
Obrázek 51: Blokové schéma funkce DPlearning

Jak již bylo zmíněno v kapitole 3, samotný tréninkový cyklus pracuje na principu stochastického gradientu sestupu. Před samotným zahájením tréninku sítě je nutné přijmout hyperparametry z grafické aplikace, na jejich základě je nastaven typ architektury modelu, optimalizační funkce, ztrátová funkce a počet tréninkových cyklů. Následně je ještě nezbytné rozdělit dataset na tréninkovou a testovací část. Pro každý cyklus jsou data rozdělena do jednotlivých dávek. Každý cyklus provede aktualizaci vah modelu s cílem minimalizovat chybu klasifikace viz. kapitola 3.2.5. Míra aktualizace vah závisí na používané optimalizační funkci a zadaných hyperparametrech. Pro lepší analýzu kvality klasifikace je funkce „*DPlearnink*“ doplněna o výpočet matice záměny. Matice záměny je počítána na základě počtu správně a chybně klasifikovaných objektů. Tyto návratové hodnoty jsou použity v grafické aplikaci pro zobrazení kvality klasifikace viz. kapitola 5.1.4.



Obrázek 52: Detail tréninkového cyklu

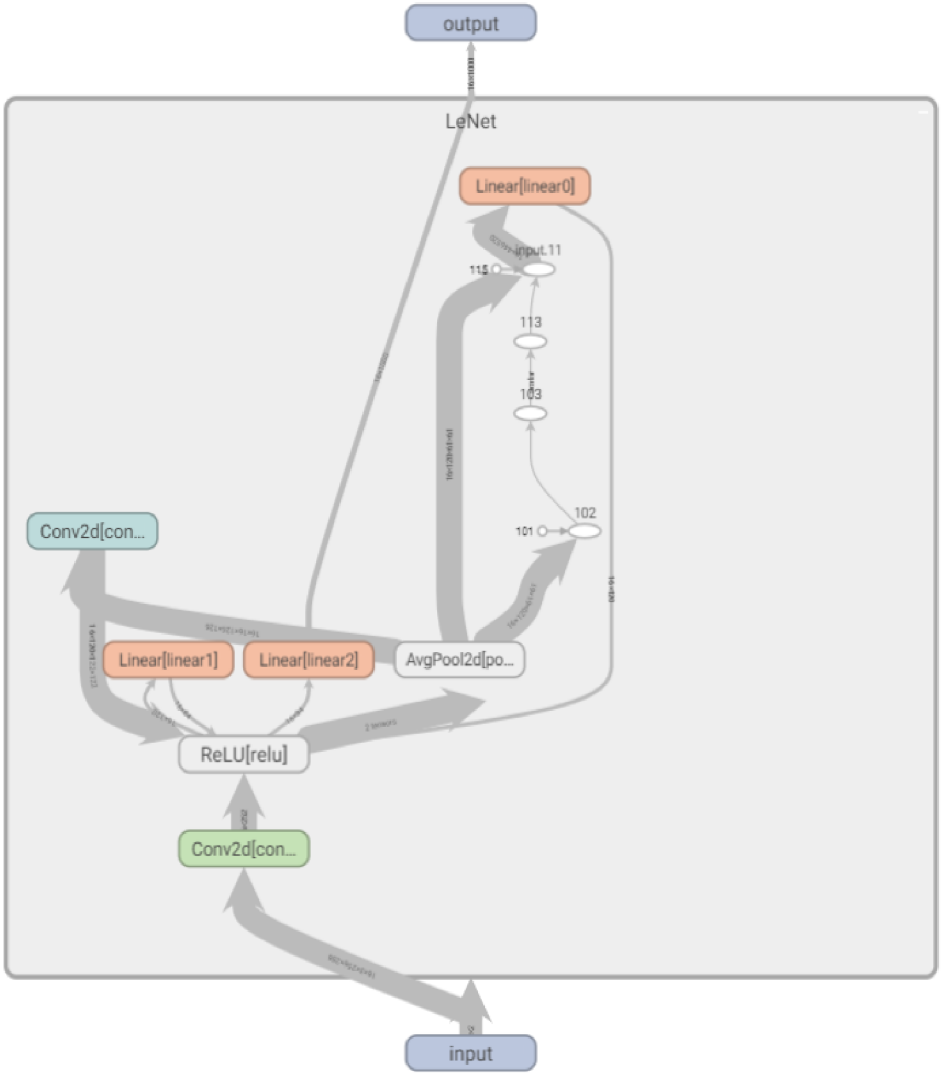
K rozdělení datasetu je použita funkce "*random split*" z knihovny Pytorch, která zajistí náhodné rozdělení datasetu podle zadaných hyperparametrů na tréninkovou a testovací část. Vstupem funkce jsou hyperparametry z grafické aplikace. Při rozdělení datasetu na tréninkovou a testovací část je nutné využít celý dataset. Výstupem jsou potom proměnné ve formátu DataLoader (specializovaný typ proměnných definovaný knihovnou Pytorch, pro uložení datasetu pro trénink CNN sítí). Tento speciální datový typ umožňuje jednotlivé datasety automaticky rozdělit podle požadavku na jednotlivé dávky.



Obrázek 53: Funkce random_split pro rozdělení datasetu

5.1.3 Navržené modely

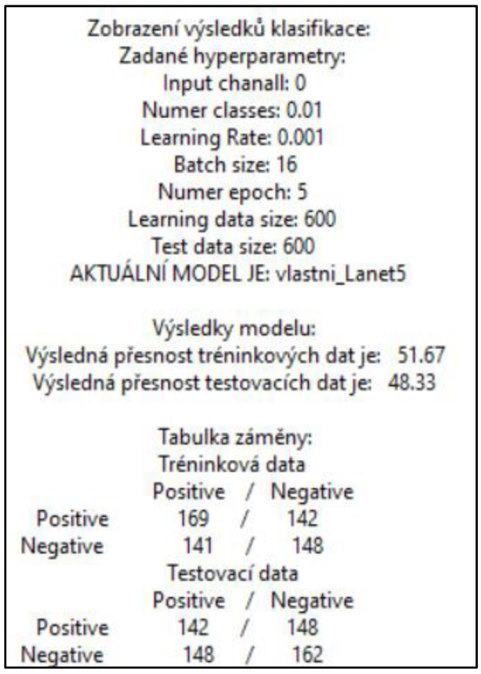
Součástí práce jsou scripty obsahující modely na bázi CNN architektury. Tyto modely vychází z architektur popsaných v kapitole 3.5. Cílem je vytvořit modely na základě CNN architektur, které jsou srovnatelné s předpřipravenými modely z knihovny Pytorch, nicméně zároveň umožňují kompletně upravovat jednotlivé vrstvy nebo aktivační funkce. Každý takovýto model se skládá ze dvou částí, v první je nejprve nutné definovat jednotlivé vrstvy a používané funkce, zatímco v druhé části je nutné definovat posloupnost těchto funkcí tak, aby vytvořily model na základě dané CNN architektury. Práce používá knihovnu TensorBoard pro analýzu jednotlivých vrstev modelu. TensorBoard umožňuje analyzovat jednotlivé vrstvy modelu sítě, propojení jednotlivých vrstev a velikosti vstupů/výstupů těchto vrstev. Jednotlivé výsledky analýzy jsou ukládaný a je tedy možnost později analyzovat jednotlivé provedené pokusy. Na obrázku 54 je vidět model sítě LaNet analyzovaný pomocí TensorBoard, včetně jednotlivých vrstev a jejich propojení.



Obrázek 54: Analýza sítě LaNet pomocí TensorBoard

5.1.4 Vyhodnocení výsledků klasifikace

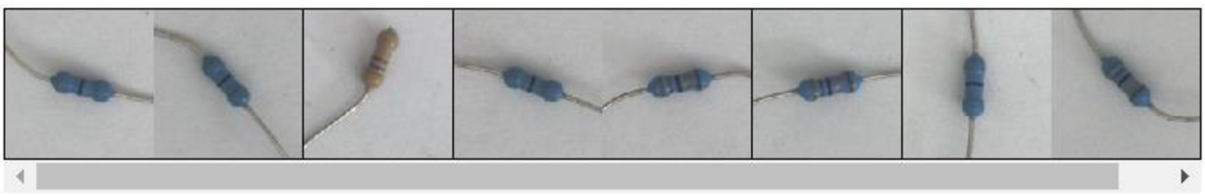
Součástí grafické aplikace je grafické okno pro zobrazení výsledků klasifikace. Samotné okno lze rozdělit do dvou částí: první část zobrazuje nastavené hyperparametry a zvolenou síť pro kterou byl model vytvořen. Druhá část zobrazuje výslednou přesnost klasifikace pomocí procentuální přesnosti a pomocí matice záměny, jak je vidět na obrázku 55.



Obrázek 55: Vyhodnocení výsledků pomocí grafické aplikace

TensorBoard

Pro detailnější analýzu je použita knihovna TensorBoard, tato knihovna umožňuje detailně analyzovat a zaznamenávat jednotlivé hyperparametry, případně jejich vývoj v jednotlivých cyklech. Knihovna TensorBoard dále umožňuje graficky analyzovat použitou CNN architekturu a rozdělení obrazového datasetu do jednotlivých dávek. Při použití TensorBoard dojde ke spuštění off-line verze TensoroardDev na portu 6006.



Obrázek 56: Ukázka grafické analýzy dávky pomocí TensorBoard

5.2 Ověření kvality klasifikace

Vyhodnocení pomocí matice záměn

Pro každý testovaný model byla stanovena odpovídající matice záměny, na základě kvality klasifikace daného modelu. Matice je u každého modelu ve tvaru [TP,FP,TN,FN], kde TP je počet skutečně pozitivních testovaných objektů, FP je počet falešně pozitivních testovaných objektů, TN je počet skutečně negativních testovaných objektů a FN je počet falešně negativních testovaných objektů. Detaily jednotlivých parametrů matice záměny jsou uvedeny v kapitole 3.3. Pro každý model je následně vypočtena přesnost, citlivost, specifičnost klasifikace a F1 skóre.

5.2.1 Ověření maximální velikosti dávky

Velikost dávky ovlivňuje množství paralelně zpracovávaných informací. Větší velikost dávky tedy umožňuje rychlejší trénink modelu sítě. Celková velikost dávky je ovlivněna velikostí modelu sítě a maximální velikostí, kterou Python s knihovnou Cuda může na grafické kartě alokovat. Maximální velikost alokované paměti je závislá na používané grafické kartě a velikosti její paměti, programově tato hodnota nelze zvýšit. Je však možné program optimalizovat pro zvýšení maximální velikosti dávky, toto lze provést například průběžným odstraňováním nepotřebných proměnných nebo optimalizací výpočtu lineárních vrstev v CNN sítích. Pro příklad: lineární vrstva s velikostí 1000 neuronů je spojena s další vrstvou s velikostí 1000 neuronů, v takovémto případě je v paměti alokována matice o velikosti 1000x1000 prvků. U sítí s malou velikostí dávky je doba tréninku poměrně dlouhá, řádově se jedná o deset minut, zatímco sítě o velikosti dávky 64 a větší jsou schopny natrénovat síť během minut.

Tabulka 5: Tabulka maximálních velikostí dávek pro modely z knihovny PyTorch

Architektury z knihovny PyTorch	
Architektura	Maximální velikost dávky [-]
GoogleNet	64
VGG11	8
VGG13	8
AlexNet	512
DenseNet169	16
SqueezeNet	64
DenseNet121	16
MobilNet v2	32
MobilNet v3	16
ResNet50	16
ShufleNet v2	128
VGG19	8

Tabulka 6: Tabulka maximálních velikostí dávek pro navržené modely na základě CNN architektur

Vytvořené modely na bázi architektůry	
Architektura	Maximální velikost dávky [-]
CNN	256
Lanet5	64
AlexNet	512
VGG16	16
VGG11	32
VGG19	4
ResNet50	16
ResNet101	4
ResNet152	4

V tabulce 5 a 6 jsou uvedeny maximální velikosti dávky pro jednotlivé typy CNN architektur. Tato velikost dávky je v dalších kapitolách práce označována jako maximální přípustná velikost dávky pro daný typ architektury. Výše zmíněné výsledky byly získány za použití grafické karty NVIDIA GeForce GTX 1650 Ti s velikostí paměti 4096 MB.

5.2.2 Ověření vlivu velikosti dávky a velikosti kroku učení na kvalitu klasifikace

Z teoretických poznatků o velikosti dávky je zřejmé, že kvalita klasifikace modelu CNN sítě je úzce spojena s velikostí dávky a krokem učení. Pro ověření tohoto předpokladu, lze uvažovat dvě hodnoty kroku učení. Kdy krok učení o velikosti 0,001 je ve většině teoretických případů brán jako referenční. Cílem kapitoly je ověření vlivu velikosti dávky a kroku učení na kvalitu klasifikace. Pro ověření vlivu velikosti dávky na kvalitu klasifikace jsou použity pouze ty modely, které v kapitole 5.2.1 dosáhly maximální velikosti dávky 64. U modelů, které mají menší velikost maximální dávky, nemá smysl tento hyperparametr testovat.

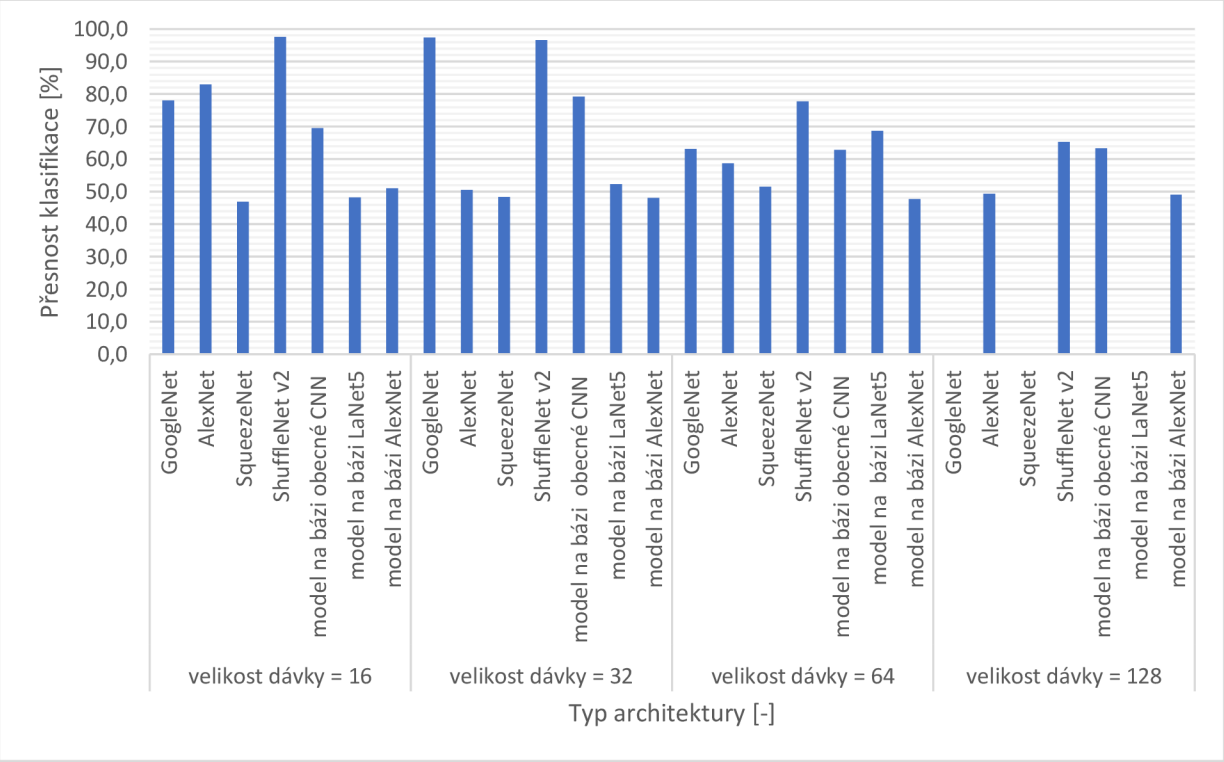
Kapitola je rozdělena na dvě části. V první části je testován vliv velikosti dávky na kvalitu klasifikace s krokem učení 0,001. Druhá část potom testuje vliv velikosti dávky na kvalitu klasifikace s krokem učení 0,0001:

Nastavení hyperparametrů:
Počet cyklů = 5
Rozložení datasetu = 600/600
Ztrátová funkce = CrossEntropy
Optimalizační funkce = Adam

Tabulka 7: Srovnání kvality klasifikace pro různou velikost dávky při kroku učení 0,001

		Krok učení = 0,001							
		TP [-]	FP [-]	TN [-]	FN [-]	Přesnost [%]	Citlivost [%]	Specifičnost [%]	skore F1 [%]
	GoogleNet	225	66	243	66	78,0	77,3	78,6	77,3
	AlexNet	228	50	270	52	83,0	82,0	84,4	81,7
	SqueezeNet	155	164	126	155	46,8	48,6	43,4	49,3
	ShuffleNet v2	283	7	302	8	97,5	97,6	97,7	97,4
	model na bázi CNN	205	95	212	88	69,5	68,3	69,1	69,1
	model na bázi LaNet5	134	155	155	156	48,2	46,4	50,0	46,3
velikost dávky = 16	model na bázi AlexNet	162	132	144	162	51,0	55,1	52,2	52,4
	GoogleNet	283	8	301	8	97,3	97,3	97,4	97,3
	AlexNet	151	152	152	145	50,5	49,8	50,0	50,4
	SqueezeNet	150	161	140	149	48,3	48,2	46,5	49,2
	ShuffleNet v2	279	15	301	5	96,7	94,9	95,3	96,5
	model na bázi CNN	226	72	249	53	79,2	75,8	77,6	78,3
	model na bázi LaNet5	150	137	164	149	52,3	52,3	54,5	51,2
velikost dávky = 32	model na bázi AlexNet	156	156	132	156	48,0	50,0	45,8	50,0
	GoogleNet	193	99	186	122	63,2	66,1	65,3	63,6
	AlexNet	171	116	181	132	58,7	59,6	60,9	58,0
	SqueezeNet	147	147	162	144	51,5	50,0	52,4	50,3
	ShuffleNet v2	240	77	226	57	77,7	75,7	74,6	78,2
	model na bázi CNN	177	120	200	103	62,8	59,6	62,5	61,4
	model na bázi LaNet5	205	91	201	94	68,7	69,3	68,8	68,9
velikost dávky = 64	model na bázi AlexNet	135	179	151	135	47,7	43,0	45,8	46,2
	GoogleNet	*				*	*	*	*
	AlexNet	154	150	142	154	49,3	50,7	48,6	50,3
	SqueezeNet	*				*	*	*	*
	ShuffleNet v2	184	101	208	107	65,3	64,6	67,3	63,9
	model na bázi CNN	175	113	205	107	63,3	60,8	64,5	61,4
	model na bázi LaNet5	*				*	*	*	*
velikost dávky = 128	model na bázi AlexNet	138	156	156	150	49,0	46,9	50,0	47,4

Pokud daný typ architektury nedovoluje natrénovat model s danou velikostí dávky, je v matici záměny označen *.

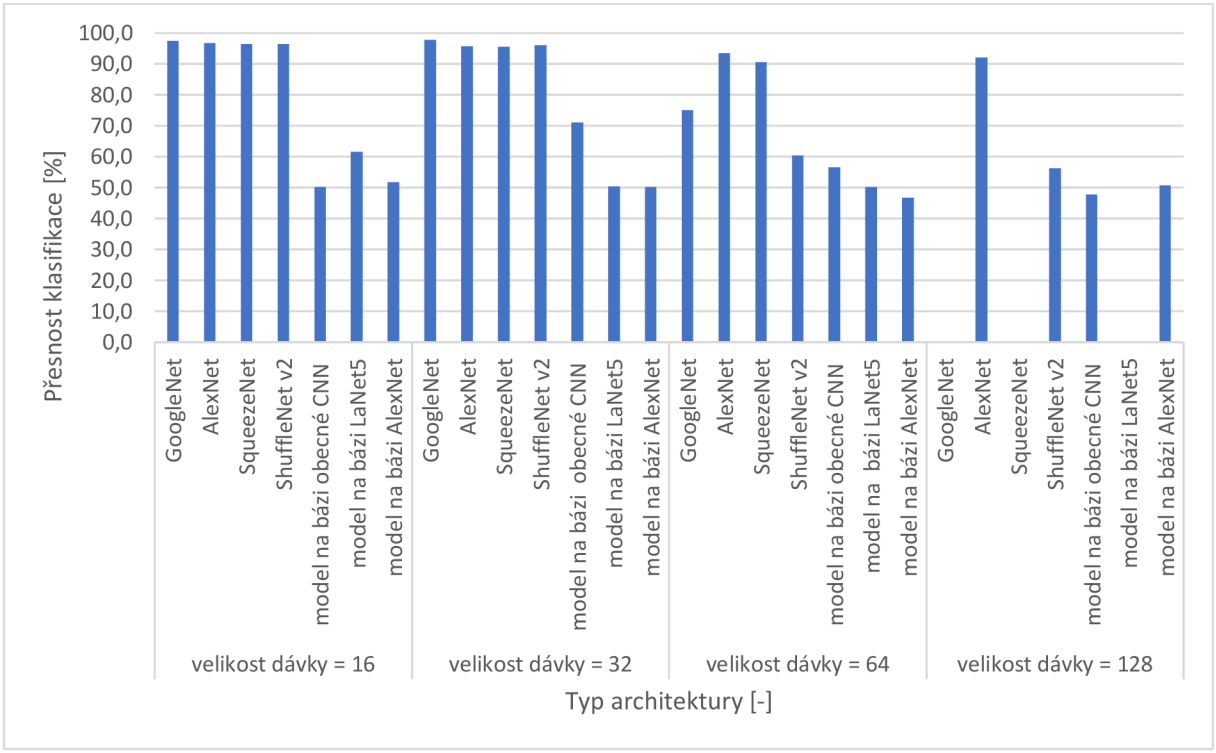


Obrázek 57: Grafické srovnání přesnosti klasifikace pro různé velikosti dávek s krokem učení 0,001

Tabulka 8: Srovnání kvality klasifikace pro různou velikost dávky při kroku učení 0,0001

		Krok učení = 0,0001							
		TP [-]	FP [-]	TN [-]	FN [-]	Přesnost [%]	Citlivost [%]	Specifičnost [%]	skore F1 [%]
	GoogleNet	290	6	295	9	97,5	98,0	98,0	97,5
	AlexNet	190	11	294	5	96,8	94,5	96,4	96,0
	SqueezeNet	293	13	285	9	96,3	95,8	95,6	96,4
	ShuffleNet v2	283	12	296	9	96,5	95,9	96,1	96,4
	model na bázi CNN	145	154	156	145	50,2	48,5	50,3	49,2
	model na bázi LaNet5	176	125	193	106	61,5	58,5	60,7	60,4
velikost dávky = 16	model na bázi AlexNet	147	144	163	146	51,7	50,5	53,1	50,3
	GoogleNet	292	9	295	4	97,8	97,0	97,0	97,8
	AlexNet	301	9	273	17	95,7	97,1	96,8	95,9
	SqueezeNet	294	16	279	11	95,5	94,8	94,6	95,6
	ShuffleNet v2	289	11	287	13	96,0	96,3	96,3	96,0
	model na bázi CNN	191	92	235	82	71,0	67,5	71,9	68,7
	model na bázi LaNet5	146	152	156	146	50,3	49,0	50,6	49,5
velikost dávky = 32	model na bázi AlexNet	137	164	164	135	50,2	45,5	50,0	47,8
	GoogleNet	213	80	237	70	75,0	72,7	74,8	74,0
	AlexNet	279	23	282	16	93,5	92,4	92,5	93,5
	SqueezeNet	277	23	266	34	90,5	92,3	92,0	90,7
	ShuffleNet v2	172	119	190	119	60,3	59,1	61,5	59,1
	model na bázi CNN	158	117	181	144	56,5	57,5	60,7	54,8
	model na bázi LaNet5	149	144	152	155	50,2	50,9	51,4	49,9
velikost dávky = 64	model na bázi AlexNet	152	169	128	151	46,7	47,4	43,1	48,7
	GoogleNet	*				*	*		*
	AlexNet	284	25	269	22	92,2	91,9	91,5	92,4
	SqueezeNet	*				*	*	*	*
	ShuffleNet v2	177	126	160	137	56,2	58,4	55,9	57,4
	model na bázi CNN	138	149	149	164	47,8	48,1	50,0	46,9
	model na bázi LaNet5	*				*	*	*	*
velikost dávky = 128	model na bázi AlexNet	155	149	149	147	50,7	51,0	50,0	51,2

Pokud daný typ architektury nedovoluje natrénovat model s danou velikostí dávky, je v matici záměny označen *.

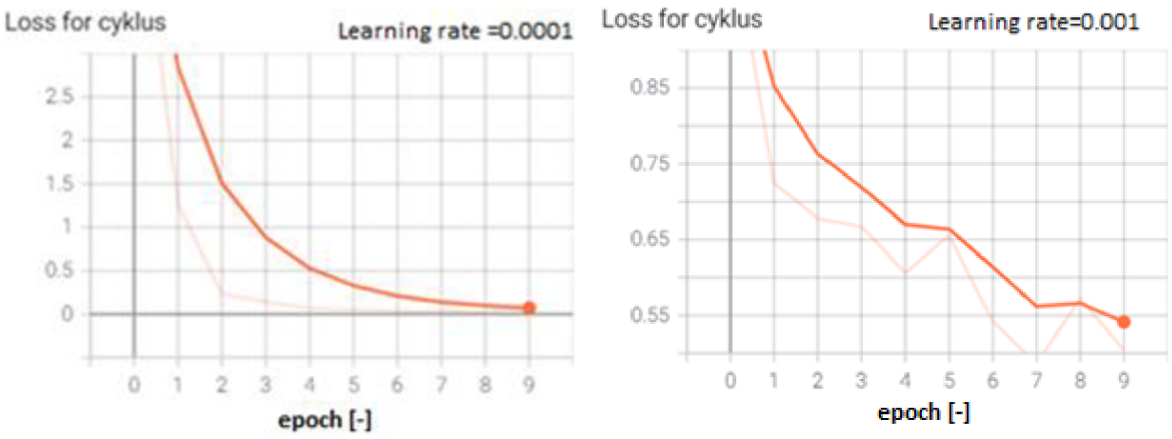


Obrázek 58: Grafické srovnání přesnosti klasifikace pro různé velikosti dávek s krokem učení 0,0001

Z experimentů v této kapitole lze vyvodit následující závěry týkající se nastavování velikosti dávky. Sítě s malou velikosti dávky dosahují dobrých výsledků klasifikace. S rostoucí velikostí dávky dochází ke zhoršování přesnosti klasifikace. Tento závěr lze vyvodit z tabulek 7 a 8.

Tento jev souvisí s principem dávkového učení. Tento typ učení sestavuje výsledný gradient z gradientů získaných na základě každé dávky. U datasetu o velikosti 600 snímků a velikost dávky 16, dojde v tomto případě k rozdělení datasetu na 37 částí o 16 snímcích a poslední 38.část potom bude obsahovat zbývajících 8 snímků. Výsledný gradient je potom sestaven zprůměrováním 38 gradientů dávek. Lze předpokládat, že chyba klasifikace obsažená v jednom z gradientů se po zprůměrování minimalizuje. Pokud je použita velikost dávky 128 dojde k následujícímu rozdělení datasetu. První čtyři dávky obsahují 128 snímků, zatímco poslední 5.dávky obsahuje pouze 88 snímků. Je tedy zřejmé, že při výpočtu finálního gradientu z jednotlivých gradientů dávek má chyba obsažená v posledním gradientu mnohem větší vliv na výslednou přesnost klasifikace. Tato chyba gradientu se tedy projeví výrazněji s větší velikostí dávky.

Krok učení je jedním z nejdůležitějších hyperparametrů. Jak je vidět na obrázcích 57 (krok učení = 0,001) a 58 (krok učení = 0,0001), nižší rychlost učení dovoluje dosáhnout přesnější klasifikace. Lze tedy prohlásit krok učení 0,0001 za ideální nastavení tohoto hyperparametru. Tento závěr lze také vyvodit z průběhu chyby klasifikace. V kapitole 3.2.1 na obrázku 33 jsou teoretické průběhy chyby klasifikace pro různé nastavení kroku učení. Srovnáním s obrázkem 59, lze dojít k závěru že průběh chyby klasifikace pro krok učení = 0,0001 odpovídá průběhu chyby klasifikace při ideálně zvoleném kroku učení.



Obrázek 59: Srovnání chyby klasifikace pomocí TensoBoard pro různý krok učení

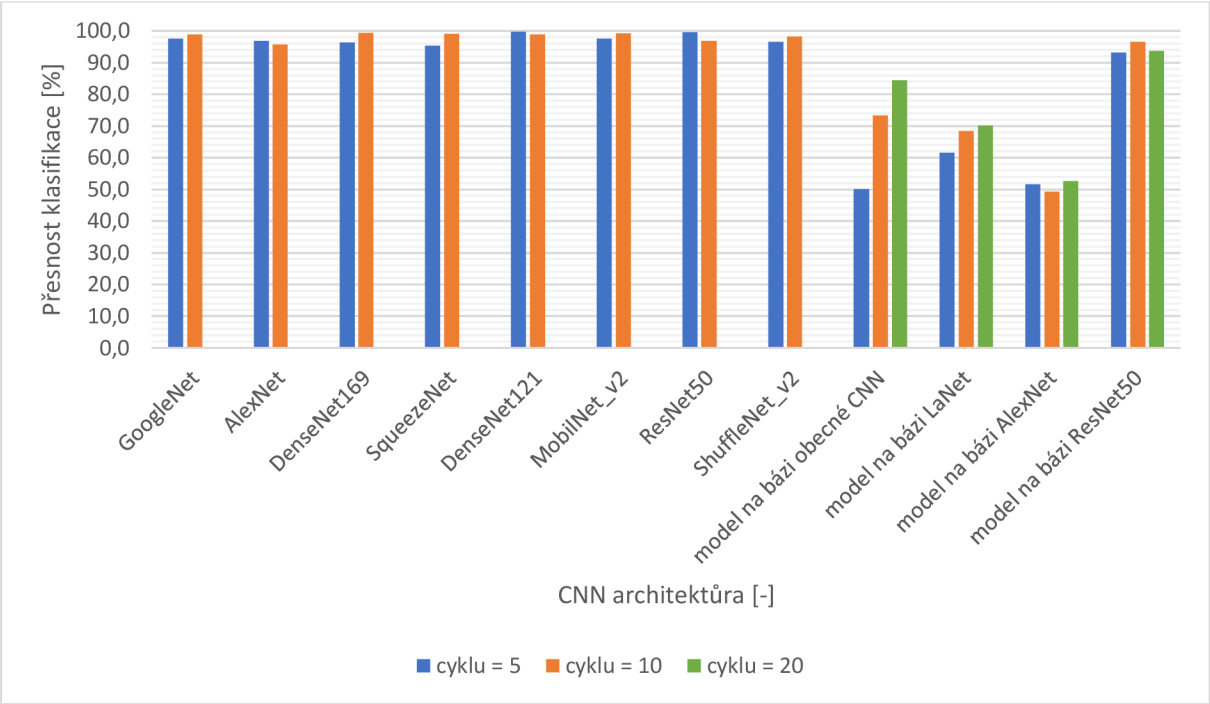
5.2.3 Ověření vlivů počtu cyklu na kvalitu klasifikace

Počet cyklů přímo ovlivňuje celkovou přesnost klasifikace, volby správného počtu cyklů je jedním z důležitějších hyperparametrů. Jak je zmíněno v kapitole 3.4 pro většinu architektur je 20 cyklů dostatečných pro vytvoření sítě s dostatečnou přesností klasifikace. Zároveň platí předpoklad, že složitější architektury vyžadují více cyklů pro trénink modelu. Cílem kapitoly je ověření vlivu počtu cyklu na přesnost klasifikace a zároveň určit ideální počet cyklů pro trénink jednotlivých architektur. Při nastavení hyperparametrů lze vycházet z poznatků získaných v minulé kapitole, velikost dávky je tedy nastavena na hodnotu 16 a krok učení na 0,0001.

Nastavení hyperparametrů:
Rozložení datasetu = 600/600
Ztrátová funkce = CrossEtropy
Optimalizační funkce = Adam
Krok učení = 0,0001
Velikost dávky = 16

Tabulka 9: Srovnání kvality klasifikace v závislosti na počtu tréninkových cyklů

		TP [-]	FP [-]	TN [-]	FN [-]	Přesnost[%]	Citlivost [%]	Specifičnost [%]	skore F1 [%]
	GoogleNet	290	6	295	9	97,5	98,0	98,0	97,5
	AlexNet	190	11	294	5	96,8	94,5	96,4	96,0
	DenseNet169	293	13	285	9	96,3	95,8	95,6	96,4
	SqueezeNet	298	11	274	17	95,3	96,4	96,1	95,5
	DenseNet121	301	1	297	1	99,7	99,7	99,7	99,7
	MobilNet_v2	285	8	300	7	97,5	97,3	97,4	97,4
	ResNet50	292	2	305	1	99,5	99,3	99,3	99,5
	ShuffleNet_v2	283	12	296	9	96,5	95,9	96,1	96,4
	model na bázi CNN	145	154	156	145	50,2	48,5	50,3	49,2
	model na bázi LaNet	176	125	193	106	61,5	58,5	60,7	60,4
	model na bázi AlexNet	147	144	163	146	51,7	50,5	53,1	50,3
počet cyklů = 5	model na bázi ResNet50	300	20	259	21	93,2	93,8	92,8	93,6
	GoogleNet	306	3	287	4	98,8	99,0	99,0	98,9
	AlexNet	289	13	285	13	95,7	95,7	95,6	95,7
	DenseNet169	299	3	297	1	99,3	99,0	99,0	99,3
	SqueezeNet	301	2	293	4	99,0	99,3	99,3	99,0
	DenseNet121	287	3	306	4	98,8	99,0	99,0	98,8
	MobilNet_v2	287	5	308	0	99,2	98,3	98,4	99,1
	ResNet50	288	10	293	9	96,8	96,6	96,7	96,8
	ShuffleNet_v2	289	5	300	6	98,2	98,3	98,4	98,1
	model na bázi CNN	211	77	229	83	73,3	73,3	74,8	72,5
	model na bázi LaNet	216	82	195	107	68,5	72,5	70,4	69,6
	model na bázi AlexNet	154	143	142	161	49,3	51,9	49,8	50,3
počet cyklů = 10	model na bázi ResNet50	273	8	306	13	96,5	97,2	97,5	96,3
	model na bázi CNN	261	46	245	48	84,3	85,0	84,2	84,7
	model na bázi LaNet	224	86	197	93	70,2	72,3	69,6	71,5
	model na bázi AlexNet	164	153	152	131	52,7	51,7	49,8	53,6
počet cyklů = 20	model na bázi ResNet50	296	16	266	22	93,7	94,9	94,3	94,0



Obrázek 60: Grafické srovnání přesnosti klasifikace v závislosti na počtu tréninkových cyklů

Na obrázku 60 je grafické srovnání přesnosti klasifikace jednotlivých modelů CNN architektur v závislosti na počtu tréninkových cyklů. Zvýšením počtu tréninkových cyklů lze zlepšit přesnost klasifikace, jak je vidět na tomto obrázku. U modelů CNN architektur z knihovny PyTorch dochází ke zlepšení přesnosti klasifikace o 1-2 %. U vlastních modelů na bázi CNN architektur došlo ke zlepšení přesnosti klasifikace u modelů CNN a LaNet. Lze tedy prohlásit že 10 cyklů je zcela dostatečných pro trénink modelů z knihovny PyTorch. Přesnost klasifikace těchto modelů s 10 tréninkovými cykly se pohybuje okolo 97 %.

Vlastní modely CNN architektur dosahují mnohem menší přesnosti klasifikace než modely z knihovny Pytorch. Zvýšením počtu tréninkových cyklů na 20 nedošlo k výraznému zvýšení přesnosti klasifikace u těchto modelů, lze tedy prohlásit že 10 tréninkových cyklů je dostatečných i pro vlastní modely na bázi CNN architektur.

Srovnáním modelu ResNet z knihovny Pytorch a vlastního modelu na bázi ResNet je patrné, že vlastní model na bázi ResNet dosahu nižší přesnosti klasifikace pouze o 6 %. Na druhou stranu vlastní model na bázi AlexNet dosahuje přesnosti klasifikace 52 % oproti modelu AlexNet z knihovny Pytorch, který dosahuje přesnosti klasifikace 96,8 %.

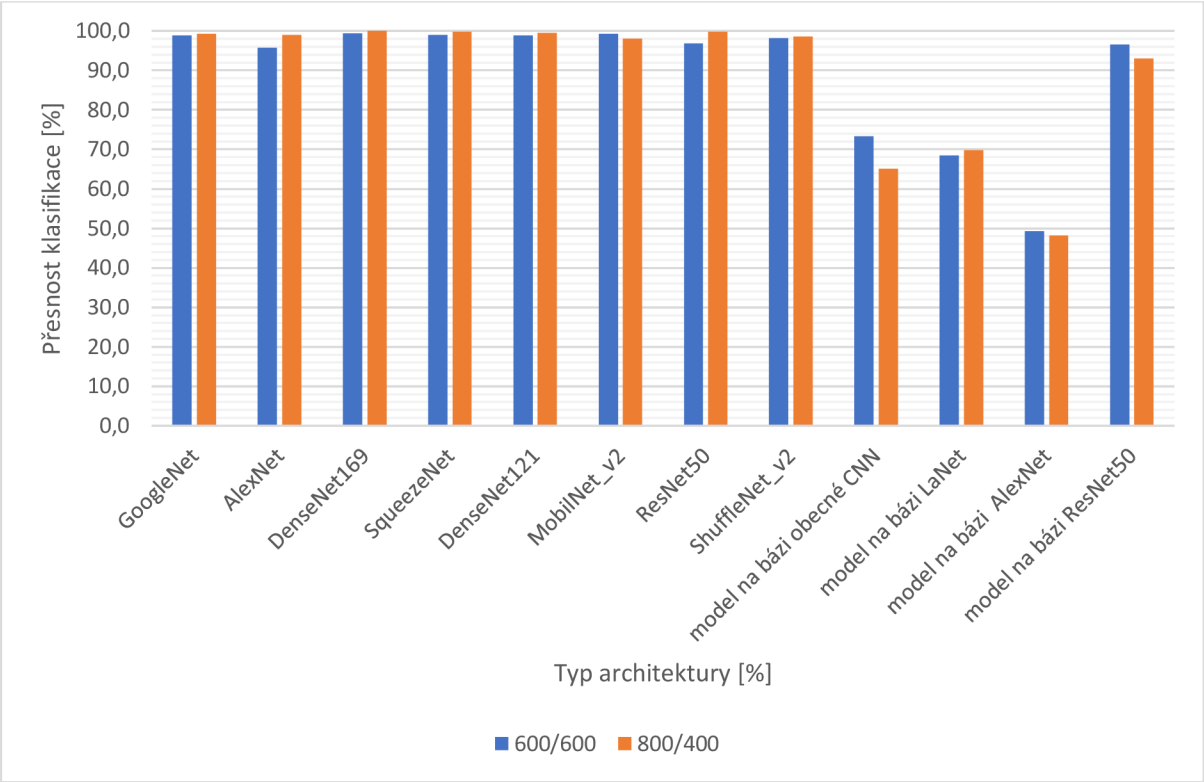
5.2.4 Ověření vlivu rozložení datasetu na kvalitu klasifikace

Rozložením datasetu lze ovlivnit kolik snímků bude použito pro vytvoření sítí a kolik pro její otestování. Cílem tohoto experimentu je určit, zda rozšířením tréninkového datasetu lze zvýšit přesnost klasifikace. O samotné rozdělení datasetu se stará funkce (random_split). Z definice vyplývá, že pro jakékoli rozložení datasetu bude daný dataset obsahovat přibližně 50 % pozitivních a 50 % negativních snímků.

Nastavení hyperparametrů:
Počet cyklů = 10
Ztrátová funkce = CrossEtropy
Optimalizační funkce = Adam
Krok učení = 0.0001
Velikost dávky = 16

Tabulka 10: Srovnání kvality klasifikace v závislosti na velikosti tréninkového datasetu

		TP [-]	FP [-]	TN [-]	FN [-]	Přesnost [%]	Citlivost [%]	Specifičnost [	skore F1 [%]
	GoogleNet	306	3	287	4	98,8	99,0	99,0	98,9
	AlexNet	289	13	285	13	95,7	95,7	95,6	95,7
	DenseNet169	299	3	297	1	99,3	99,0	99,0	99,3
	SqueezeNet	301	2	293	4	99,0	99,3	99,3	99,0
	DenseNet121	287	3	306	4	98,8	99,0	99,0	98,8
	MobilNet_v2	287	5	308	0	99,2	98,3	98,4	99,1
	ResNet50	288	10	293	9	96,8	96,6	96,7	96,8
	ShuffleNet_v2	289	5	300	6	98,2	98,3	98,4	98,1
Rozložení	model na bázi CNN	211	77	229	83	73,3	73,3	74,8	72,5
datasetu	model na bázi LaNet	216	82	195	107	68,5	72,5	70,4	69,6
=	model na bázi AlexNet	154	143	142	161	49,3	51,9	49,8	50,3
600/600	model na bázi ResNet50	273	8	306	13	96,5	97,2	97,5	96,3
	GoogleNet	203	3	194	0	99,3	98,5	98,5	99,3
	AlexNet	172	2	224	2	99,0	98,9	99,1	98,9
	DenseNet169	200	0	200	0	100,0	100,0	100,0	100,0
	SqueezeNet	190	0	209	1	99,8	100,0	100,0	99,7
	DenseNet121	203	2	195	0	99,5	99,0	99,0	99,5
	MobilNet_v2	208	3	184	5	98,0	98,6	98,4	98,1
	ResNet50	201	0	198	1	99,8	100,0	100,0	99,8
	ShuffleNet_v2	203	4	191	2	98,5	98,1	97,9	98,5
Rozložení	model na bázi CNN	120	65	140	75	65,0	64,9	68,3	63,2
datasetu	model na bázi LaNet	136	66	143	55	69,8	67,3	68,4	69,2
=	model na bázi AlexNet	90	103	103	104	48,3	46,6	50,0	46,5
800/400	model na bázi ResNet50	184	15	188	13	93,0	92,5	92,6	92,9



Obrázek 61: Grafické srovnání přesnosti klasifikace v závislosti na velikosti tréninkového datasetu

V tomto experimentu lze vycházet z teoretických doporučení týkajících se rozložení datasetu. V kapitole 3.2.6. je stanoveno teoretické pravidlo, toto pravidlo doporučuje rozložit dataset v poměru 70/30 % (tréninková data/ testovací data). Pro dataset čítající 1200 snímku by potom toto pravidlo splňovalo rozložení 840/360 snímkům. Rozložení blížící se této hodnotě je potom na obrázku 61 znázorněno červeně. Z experimentů provedených v této kapitole, lze stanovit, že použitím takto rozděleného datasetu nedochází ke zlepšení kvality klasifikace oproti rozložení datasetu 600/600. Pro tento případ unární klasifikaci je tedy 600 tréninkových objektů dostačujících.

5.2.5 Ověření vlivu optimalizační funkce na kvalitu klasifikace

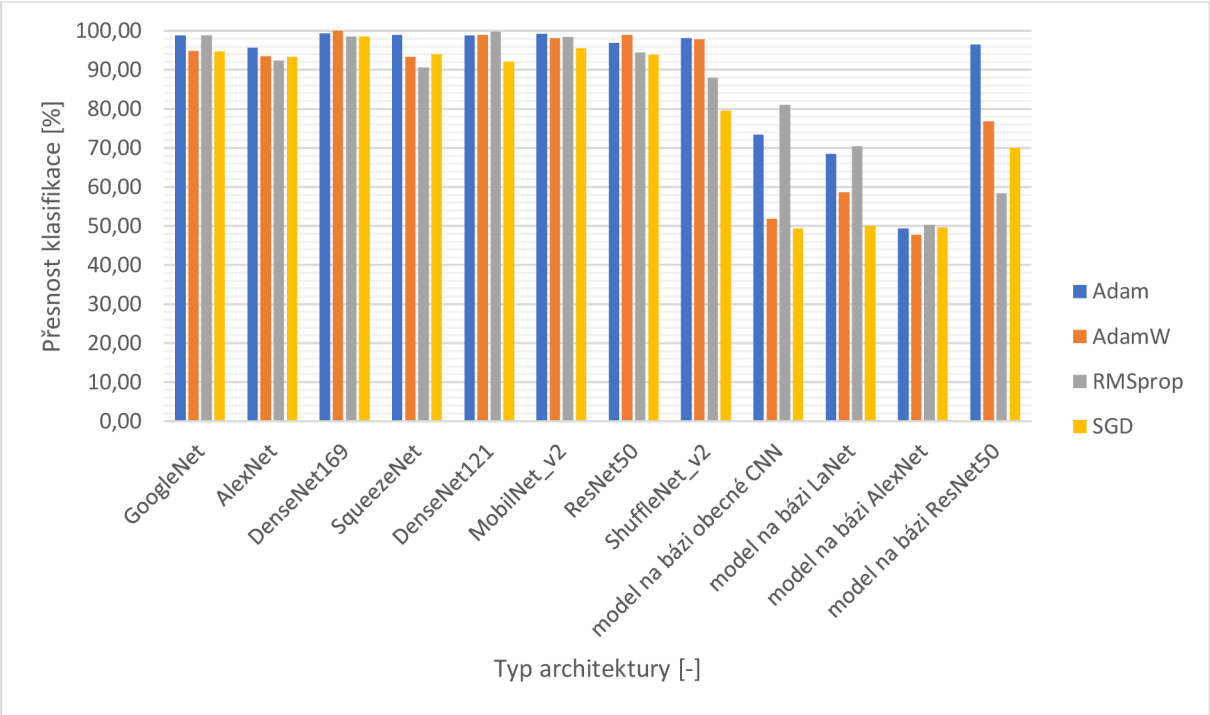
U CNN architektur se používá optimalizační metoda klesajícího gradientu. V posledních několika letech bylo vyvinuto několik alternativ ke klasickým algoritmům klesajícího gradientu. Teorie definuje, že nejlepší optimalizační funkcí je funkce Adam. Pro následnou klasifikaci byly použity nastavení hyperparametrů vycházející ze závěru předchozích kapitol. Nastavení momentu a poklesu hmotnosti jednotlivých optimalizačních funkcí vychází z teoretických předpokladů. [65]

Nastavení hyperparametrů:
Počet cyklů = 10
Ztrátová funkce = CrossEtropy
Velikost dávky = 16

Nastavení jednotlivých optimalizačních funkcí:
Adam: krok učení =0.0001, pokles hmotnosti = 0
AdamW: krok učení =0.0001, pokles hmotnosti = 0.01
RMSprop: krok učení = 0.0001, pokles hmotnosti = 0, hybnost = 0.9
SGD: krok učení = 0.0001, pokles hmotnosti = 0, hybnost = 0.9

Tabulka 11: Srovnání kvality klasifikace v závislosti na optimalizační funkci

		TP [-]	FP -]	TN [-]	FN [-]	Přesnost [%]	Citlivost [%]	Specifičnost [%]	skore F1 [%]
Optimalizační funkce = Adam	GoogleNet	306	3	287	4	98,8	99,0	99,0	98,9
	AlexNet	289	13	285	13	95,7	95,7	95,6	95,7
	DenseNet169	299	3	297	1	99,3	99,0	99,0	99,3
	SqueezeNet	301	2	293	4	99,0	99,3	99,3	99,0
	DenseNet121	287	3	306	4	98,8	99,0	99,0	98,8
	MobilNet_v2	287	5	308	0	99,2	98,3	98,4	99,1
	ResNet50	288	10	293	9	96,8	96,6	96,7	96,8
	ShuffleNet_v2	289	5	300	6	98,2	98,3	98,4	98,1
	model na bázi CNN	211	77	229	83	73,3	73,3	74,8	72,5
	model na bázi LaNet	216	82	195	107	68,5	72,5	70,4	69,6
	model na bázi AlexNet	154	143	142	161	49,3	51,9	49,8	50,3
	model na bázi ResNet50	273	8	306	13	96,5	97,2	97,5	96,3
Optimalizační funkce = AdamW	GoogleNet	300	14	269	17	94,8	95,5	95,1	95,1
	AlexNet	270	23	291	16	93,5	92,2	92,7	93,3
	DenseNet169	298	0	302	0	100,0	100,0	100,0	100,0
	SqueezeNet	287	23	273	17	93,3	92,6	92,2	93,5
	DenseNet121	301	2	293	4	99,0	99,3	99,3	99,0
	MobilNet_v2	292	4	297	7	98,2	98,6	98,7	98,2
	ResNet50	294	2	300	4	99,0	99,3	99,3	99,0
	ShuffleNet_v2	298	6	289	7	97,8	98,0	98,0	97,9
	model na bázi CNN	150	141	161	148	51,8	51,5	53,3	50,9
	model na bázi LaNet	171	125	181	123	58,7	57,8	59,2	58,0
	model na bázi AlexNet	134	153	152	161	47,7	46,7	49,8	46,0
	model na bázi ResNet50	231	80	230	59	76,8	74,3	74,2	76,9
Optimalizační funkce = RMSprop	GoogleNet	286	6	307	1	98,8	97,9	98,1	98,8
	AlexNet	282	16	272	30	92,3	94,6	94,4	92,5
	DenseNet169	288	4	303	5	98,5	98,6	98,7	98,5
	SqueezeNet	281	26	263	30	90,7	91,5	91,0	90,9
	DenseNet121	211	0	288	1	99,8	100,0	100,0	99,8
	MobilNet_v2	292	5	298	5	98,3	98,3	98,3	98,3
	ResNet50	273	21	294	12	94,5	92,9	93,3	94,3
	ShuffleNet_v2	265	32	263	40	88,0	89,2	89,2	88,0
	model na bázi CNN	254	54	232	60	81,0	82,5	81,1	81,7
	model na bázi LaNet	206	97	216	81	70,3	68,0	69,0	69,8
	model na bázi AlexNet	155	143	147	155	50,3	52,0	50,7	51,0
	model na bázi ResNet50	178	133	172	117	58,3	57,2	56,4	58,7
Optimalizační funkce = SGD	GoogleNet	284	17	284	15	94,7	94,4	94,4	94,7
	AlexNet	281	23	279	17	93,3	92,4	92,4	93,4
	DenseNet169	313	3	278	6	98,5	99,1	98,9	98,6
	SqueezeNet	283	15	281	21	94,0	95,0	94,9	94,0
	DenseNet121	273	25	280	22	92,2	91,6	91,8	92,1
	MobilNet_v2	279	10	294	17	95,5	96,5	96,7	95,4
	ResNet50	275	18	288	19	93,8	93,9	94,1	93,7
	ShuffleNet_v2	253	51	224	72	79,5	83,2	81,5	80,4
	model na bázi CNN	143	154	153	150	49,3	48,1	49,8	48,5
	model na bázi LaNet	152	149	148	151	50,0	50,5	49,8	50,3
	model na bázi AlexNet	152	150	146	152	49,7	50,3	49,3	50,2
	model na bázi ResNet50	198	97	222	83	70,0	67,1	69,6	68,8



Obrázek 62: Grafické srovnání přesnosti klasifikace pro jednotlivé optimalizační funkce

Z teoretických předpokladů zmíněných v kapitole 3.2.5, lze předpokládat dosažení nejlepších výsledků použitím optimalizační funkce Adam. Optimalizační funkce Adam je velmi závislá na správné volbě kroku učení (ideálně volit lr=0.0001). Ze srovnání přesnosti klasifikace optimalizačních funkcí (obrázek 62), lze stanovit následující závěry.

Optimalizační funkce Adam dosahuje přesnosti klasifikace přes 95 % u modelů z knihovny PyTorch. U vlastních modelů založených na bázi CNN architektur dosahuje optimalizační funkce Adam nejlepších výsledků pro model na bázi ResNet 50. Optimalizační funkce AdamW dosahuje srovnatelných výsledků s optimalizační funkcí Adam u modelů z knihovny PyTorch. Pro vlastní modely na bázi CNN architektur dosahuje optimalizační funkce horších výsledků v porovnání s funkcí Adam řádově o 15 %. Optimalizační funkce RMSprop dosahuje srovnatelných výsledků s optimalizační funkcí Adam u modelů z knihovny Pytorch. Pro vlastní modely na bázi CNN a LaNet architektur dosahuje optimalizační funkce RMSprop lepších výsledků přesnosti klasifikace než optimalizační funkce Adam. Naopak u vlastního modelu na bázi ResNet 50 dosahuje funkce RMSprop mnohem horších výsledků než funkce Adam. Optimalizační funkce SGD dosahuje nejhorších výsledků v přesnosti klasifikace. Oproti všem ostatním optimalizačním funkcím dosahuje nižší přesnosti klasifikace řádově o 2-3 %. U vlastních modelů na základě CNN architektur je výsledná přesnost klasifikace optimalizační funkce SGD horší o 20 % v porovnání s optimalizační funkcí Adam.

Z výsledků této kapitoly lze optimalizační funkci Adam prohlásit za nejvhodnější optimalizační funkci pro unární klasifikaci daného průmyslového výrobku.

6. ZÁVĚR

Tato práce se zaměřuje na klasifikaci průmyslového výrobku, pomocí technik strojového učení, konkrétně pomocí neuronových sítí. Použití neuronových sítí v této oblasti klasifikace je hojně rozšířené. Tento typ klasifikace je snadno modifikovatelný pro různé průmyslové výrobky, tato práce se konkrétně zaměřuje na klasifikaci rezistorů. Pro samotný trénink a ověření funkčnosti CNN sítě obsahuje práce jednoúčelovou aplikaci v prostředí Python. Cílem práce je kromě vytvoření aplikace otestování vlivu jednotlivých hyperparametrů na kvalitu klasifikace.

První a druhá kapitola práce se zaměřuje na definování jednotlivých typů a metod klasifikace. Cílem těchto kapitol je poskytnout ucelený pohled na problematiku klasifikace. V této části práce je definován rozdíl mezi unární, binární a více třídní klasifikací. Dále jsou zde definovány jednotlivé klasifikační metody, které jsou rozděleny na metody hustoty, hraniční metody a rekonstrukční metody.

Třetí kapitola poskytuje teoretický základ k návrhu CNN architektur. Tato kapitola definuje jednotlivé vrstvy architektur, jejich propojení a aktivační funkce. Součástí kapitoly je definice jednotlivých hyperparametrů, spolu s jejich ideálním nastavením. Teoretické poznatky z této kapitoly práce jsou použity v praktické části pro návrh vlastních modelů na bázi CNN architektur.

Poslední kapitola práce se věnuje otestování funkčnosti aplikace a ověření kvality klasifikace jednotlivých CNN architektur pro různé nastavení hyperparametrů. Tato kapitola práce se zaměřuje na otestování hyperparametrů, které mají podle teoretických předpokladů největší vliv na výslednou kvalitu klasifikace. Těmito hyperparametry jsou velikost dávky, krok učení, počet cyklů, rozložení datasetu a optimalizační funkce. Z experimentů provedených v kapitole 5.2 lze vyvodit následující závěry pro nastavení hyper parametrů. Velikost dávky a krok učení jsou hyper parametry s největším dopadem na kvalitu klasifikace. Z experimentů v kapitole 5.2.2 jasně vyplývá nutnost volby nižšího kroku učení a menší velikosti dávky, za optimální hodnotu těchto hyperparametrů je zvolen krok učení 0,0001 a velikost dávky 16. Tyto hodnoty hyperparametrů se shodují z teoretickými předpoklady z kapitoly 3.2. Dalším z testovaných hyperparametrů je počet tréninkových cyklů. Z experimentů provedených v kapitole 5.2.3 je stanoven potřebný počet tréninkových cyklů na 10. Tato hodnota byla zvolena z důvodu dobrého poměru kvality klasifikace vůči době tréninku. Předpřipravené architektury z knihovny PyTorch dosahují po 10 tréninkových cyklech přesnosti klasifikace okolo 97 %. Dále v kapitole 5.2.4 byl ověřován teoretický předpoklad rozložení datasetu, kdy podle tohoto předpokladu je nutno využít minimálně 70 % datasetu pro trénink. Tento předpoklad se nepotvrdil, pro tento konkrétní příklad unární klasifikace stačí využít 50 % datasetu pro trénink. Modely, které pro trénink využily pouze 50 %, dosahovaly stejných výsledku jako modely, které využily 70 % datasetu pro trénink. Posledním ověřovaným hyperparametrem byla optimalizační funkce. V dnešní době CNN architektury využívají celou řadu optimalizačních funkcí. Každá z těchto funkcí přistupuje k aktualizaci vah sítě trochu jiným způsobem. Z teoretických předpokladů v kapitole 3.2.5 se optimalizační funkce Adam

jeví jako nejoptimálnější. Provedené experimenty v kapitolo 5.2.5 tento předpoklad potvrzují. Srovnáním CNN architektur z knihovny Pytorch a vlastních modelů na bázi CNN lze dojít k závěru, že předpřipravené architektury dosahují výrazně vyšší kvality klasifikace. Tento rozdíl je pravděpodobně způsoben optimalizačními funkcemi, kterými jsou architektury z knihovny PyTorch vybaveny.

Cílem praktická části práce bylo vytvoření aplikace pro trénink a vyhodnocení funkčnosti klasifikátorů. Vytvořená aplikace umožňuje efektivně trénovat klasifikační algoritmy na bázi CNN architektur. Aplikace využívá předpřipravené sítě z knihovny PyTorch, zároveň ale obsahuje i mé vlastní modely na bázi CNN architektur. Aplikace dovoluje efektivně trénovat jednotlivé CNN architektury s různým nastavením hyperparametrů. Součástí odevzdání je anotovaný obrázkový dataset čítající 1200 snímků rezistorů. Tento dataset je v praktické části práce použit pro trénink jednotlivých CNN architektur. Dataset je rozdělen na poloviny, kdy snímky 1-600 obsahují pozitivní část datasetu a snímky 601-1200 obsahují negativní část datasetu.

Aplikace trénuje CNN sítě na základě anotovaného obrázkového datasetu. Dalším krokem může být úprava této aplikace pro klasifikování průmyslových výrobků v reálném čase na základě vstupů z kamery, případně doplnění o automatické vyhledávání požadovaného výrobku na snímku.

LITERATURA

[1] TAX, David. *One-class classification* [online]. Mekelweg, 2001 [cit. 2020-10-24]. Dostupné z: http://homepage.tudelft.nl/n9d04/thesis.pdf . Doktorandská práce. TU Delft.

[2] NGUYEN, Long D., Dongyun LIN, Zhiping LIN a Jiuwen CAO. Deep CNNs for microscopic image classification by exploiting transfer learning and feature concatenation. *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* [online]. IEEE, 2018, 2018, , 1-5 [cit. 2021-05-04]. ISBN 978-1-5386-4881-0. Dostupné z: https://www.researchgate.net/publication/324961229_Deep_CNNs_for_microscopic_image_classification_by_exploiting_transfer_learning_and_feature_concatenation

[3] AMUEL, Arthur. *Some Studies in Machine Learning Using the Game of Checkers* [online]. 1959 [cit. 2021-7-4]. Dostupné z: https://www.cs.virginia.edu/~evans/greatworks/samuel1959.pdf

[4] JOY, Abin. *Logistic Regression Using Python* [online]. 2019 [cit. 2021-6-12]. Dostupné z: https://medium.com/@abinmj656/logistic-regression-using-python-28a269152f53

[5] Ing. Petr Honzík, Ph. D. *-Základy statistiky používané ve strojovém učení* 2014-5 [online] [cit.2020-11-27]. Dostupné z: http://vision.uamt.feec.vutbr.cz/STU/lectures/01_Strojove_uceni_zakladni_prehled_a_terminologie.pdf

[6] MINOGUE, Paul. *Anomaly detection – can it help in contact centre management?* [online]. 2020 [cit. 2021-6-24]. Dostupné z: https://www.edgetier.com/blog/anomaly-detection-can-it-help-in-contact-centre-management/

[7] Amazon ML-*Amazon Machine Learning: Developer Guide , 2016-08-02* [online][[cit.2020-11-27]. Dostupné z: https://docs.aws.amazon.com/machine-learning/latest/dg/binary-classification.html

[8] Mohammed Terry-Jack - *Tips and Tricks for Multi-Class Classification 2019-09-28* [online] [cit.2020- 11-27]. Dostupné z: https://medium.com/@b.terryjack/tips-and-tricks-for-multi-class-classification-c184ae1c8ffc

[9] S. KHAN, Shehroz a Michael G. MADDEN. *One-class classification: taxonomy of study and review of techniques* [online]. Oxford, 2014 [cit. 2020-10-24]. Dostupné z: https://www.cambridge.org/core/journals/knowledge-engineering-review/article/oneclass-classification-taxonomy-of-study-and-review-of-techniques/EEDBC97CFD54B2B65BEB5FCD71593782

[10] Ing. Petr Honzík, Ph. D. *-Lineární modely, diskriminační analýza a podpůrné vektory* 2014-5 [online] [cit.2020-11-27]. Dostupné z: http://vision.uamt.feec.vutbr.cz/STU/lectures/Linearni_modely_diskriminacni_analyza_a_SVM.pdf

[11] SRAMRMA P., Ashwim. *Gaussian Distribution* [online]. 2020 [cit. 2021-6-25]. Dostupné z: https://dev.to/ashwinsharmap/gaussian-distribution-797

[12] Oscar Contreras Carrasco - *Gaussian Mixture Models Explained 2019-6-3*[online] [cit. 2020- 10-29]. Dostupné z: https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95

[13] Peter M. - Parzen density estimation – 2019-09-20 [online][cit. 2020- 10-29]. Dostupné z: https://math.stackexchange.com/questions/3363136/parzen-density-estimation

[14] BRERETON, Richard. *Support Vector Machines for classification and regression* [online]. 2010 [cit. 2021-6-25]. Dostupné z: https://www.researchgate.net/figure/Some-definitions-for-SVDD_fig30_41124138

[15] Davida Mount'ca -*Approximation Algorithms Load Balancing k-center selection* [online] [cit. 2020- 11-6]. Dostupné z: https://present5.com/approximation-algorithms-load-balancing-k-center-selection/

[16] BAND, Amey. *How to find the optimal value of K in KNN?* [online]. 2020 [cit. 2021-6-29]. Dostupné z: https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb

[17] ELMOGY, Ahmed Mohamad, Hamada RIZK a Amany M. SARHANA. *OFCOD: On the Fly Clustering Based Outlier Detection Framework* [online]. 2020 [cit. 2021-7-1]. Dostupné z: https://www.researchgate.net/publication/348112285_OFCOD_On_the_Fly_Clustering_Based_Outlier_Detection_Framework

[18] CHEN, Yu-Zhong. *Sparse dynamical Boltzmann machine for reconstructing complex networks with binary dynamics* [online]. 2018 [cit. 2021-6-27]. Dostupné z: https://www.researchgate.net/figure/A-schematic-illustration-of-the-K-means-algorithm-for-two-dimensional-data-clustering_fig2_324073652

[19] KUMAR, Gustav, Sandeep SHARMA a Hasmat MALIK. *Learning Vector Quantization Neural Network Based External Fault Diagnosis Model for Three Phase Induction Motor Using Current Signature Analysis* [online]. 2016 [cit. 2021-7-4]. Dostupné z: https://www.researchgate.net/publication/306072258_Learning_Vector_Quantization_Neural_Network_Based_External_Fault_Diagnosis_Model_for_Three_Phase_Induction_Motor_Using_Current_Signature_Analysis

[20] TRE, Ben. *Principal Component Analysis (PCA)* [online]. 2021 [cit. 2021-7-12]. Dostupné z: https://dinhanhthi.com/principal-component-analysis/

[21] KOEHSEN, Will. Random Forest Simple Explanation. *Https://williamkoehrsen.medium.com/* [online]. online: Will Koehrsen, 2017 [cit. 2020-11-4]. Dostupné z: https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d

[22] Ariel Goldberger*-Training Neural Networks for binary classification,* 2019-03-8 [online][[cit.2020-11-27]. Dostupné z: https://medium.com/duke-ai-society-blog/training-neural-networks-for-binary-classification-identifying-types-of-breast-cancer-keras-in-r-b38fb26a500c

[23] FUERTES, T. *Outliers detection with autoencoder, a neural network* [online]. 2019 [cit. 2021-6-28]. Dostupné z: https://quantdare.com/outliers-detection-with-autoencoder-neural-network/

[24] JAY, Prakash. *Image Classification Architectures review* [online]. [online] [cit.2020-12-28]. Dostupné z: https://medium.com/@14prakash/image-classification-architectures-review-d8b95075998f

[25] KARIM, Raimi. *Illustrated: 10 CNN Architectures* [online]. 2019 [cit. 2021-4-4]. Dostupné z: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

[26] VIDHYA, Analytics. *What is the Convolutional Neural Network Architecture?* [online]. 2020 [cit. 2021-5-1]. Dostupné z: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/

[27] SAKTHI, Raj. *Talented Mr. 1X1: Comprehensive look at 1X1 Convolution in Deep Learning* [online]. 2020 [cit. 2021-7-10]. Dostupné z: https://medium.com/analytics-vidhya/talented-mr-1x1-comprehensive-look-at-1x1-convolution-in-deep-learning-f6b355825578

[28] KHOSLA, Savya. *CNN | Introduction to Pooling Layer* [online]. 2019 [cit. 2021-5-1]. Dostupné z: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

[29] TORCH CONTRIBUTORS, *TORCH.ADD* [online]. 2019 [cit. 2021-7-13]. Dostupné z: https://pytorch.org/docs/stable/generated/torch.add.html

[30] TORCH CONTRIBUTORS, *TORCH.CAT* [online]. 2019 [cit. 2021-7-13]. Dostupné z: https://pytorch.org/docs/stable/generated/torch.cat.html

[31] SHARMA, Sagar. *Activation Functions in Neural Networks* [online]. 2017 [cit. 2021-4-9]. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[32] Activation function. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2021 [cit. 2021-5-1]. Dostupné z: https://en.wikipedia.org/wiki/Activation_function

[33] RADEČIČ, Dario. *Softmax Activation Function Explained* [online]. 2020 [cit. 2021-05-04]. Dostupné z: https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60

[34] NORY, Mustafa, Mustafa ALJUMAILI a Nezar ISMAT. *Fire Detection Using Convolutional Deep Learning Algorithms* [online]. 2019 [cit. 2021-4-15]. Dostupné z: https://www.researchgate.net/figure/CNN-Neuron-Architecture-The-neuron-process-the-incoming-signals-WiXi-by-aggregating_fig3_332670534

[35] JORDAN, Jeremy. *Setting the learning rate of your neural network* [online]. 2018 [cit. 2021-6-15]. Dostupné z: https://www.jeremyjordan.me/nn-learning-rate/

[36] ZULKIFI, Hafidz. *Understanding Learning Rates and How It Improves Performance in Deep Learning* [online]. 2018 [cit. 2021-7-14]. Dostupné z: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10

[37] BUSHEAV, Vitaly. *Stochastic Gradient Descent with momentum* [online]. 2017 [cit. 2021-6-17]. Dostupné z: https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

[38] VASANI, Dipan. *This thing called Weight Decay* [online]. 2019 [cit. 2021-6-25]. Dostupné z: https://towardsdatascience.com/this-thing-called-weight-decay-a7cd4bcfccab

[39] BROWNLEE, Jason. *Loss and Loss Functions for Training Deep Learning Neural Networks* [online]. 2017 [cit. 2021-4-9]. Dostupné z: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

[40] BROWNLEE, Jason. *How to Choose Loss Functions When Training Deep Learning Neural Networks* [online]. 2020 [cit. 2021-6-20]. Dostupné z: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

[41] SMOLYAKOV, Vadim. *Neural Network Optimization Algorithmss* [online]. 2018 [cit. 2021-6-21]. Dostupné z: https://towardsdatascience.com/neural-network-optimization-algorithms-1a44c282f61d

[42] A TORCH CONTRIBUTORS. *ADAM W.* [online]. 2019 [cit. 2021-6-28]. Dostupné z: https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html#torch.optim.AdamW

[43] BROWNLEE, Jason. *Impact of Dataset Size on Deep Learning Model Skill And Performance Estimates* [online]. 2020 [cit. 2021-6-19]. Dostupné z: https://machinelearningmastery.com/impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/

[44] BROWNLEE, Jason. *How to Control the Stability of Training Neural Networks With the Batch Size* [online]. 2020 [cit. 2021-6-21]. Dostupné z: https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

[45] SHEN, Kevin. *Effect of batch size on training dynamics* [online]. 2018 [cit. 2021-7-4]. Dostupné z: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

[46] MOHAJON, Joydwid. *Confusion Matrix for Your Multi-Class Machine Learning Model* [online]. 2020 [cit. 2021-7-2]. Dostupné z: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

[47] JAN, Bilan, Haleem FERMAN a Murad KHAN. *Deep learning in big data Analytics: A comparative study* [online]. 2017 [cit. 2021-5-1]. Dostupné z: https://www.researchgate.net/publication/321787151_Deep_learning_in_big_data_Analytics_A_comparative_study

[48] ZHANG, Aston, Zack C. LIPTON, Mu LI a Alex J. SMOLA. *Convolutional Neural Networks (LeNet)* [online]. 2019 [cit. 2021-04-12]. Dostupné z: http://d2l.ai/chapter_convolutional-neural-networks/lenet.html

[49] SAXENA, Shipra. *The Architecture of Lenet-5,* Analytics Vidhya [online]. 2021 [cit. 2021-04-05]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

[50] WEI, Jerry. *AlexNet: The Architecture that Challenged CNNs* [online]. 2019 [cit. 2021-04-10]. Dostupné z: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951

[51] SAXENA, Shipra. *Introduction to The Architecture of Alexnet*. Analytics Vidhya [online]. 2021 [cit. 2021-04-05]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/

[52] BRESSEM, Keno K., Lisa C. ADAMS, Christoph ERXLEBEN, Bernd HAMM, Stefan M. NIEHUES a Janis L. VAHLDIEK. *Comparing different deep learning architectures for classification of chest radiographs* [online]. 2020 [cit. 2021-4-28]. Dostupné z: https://www.nature.com/articles/s41598-020-70479-z

[53] ARSHAY, Paras. *VGGNet-16 Architecture: A Complete Guide* [online]. 2020 [cit. 2021-04-08]. Dostupné z: https://www.kaggle.com/blurredmachine/vggnet-16-architecture-a-complete-guide

[54] RAJ, Bharath. *A Simple Guide to the Versions of the Inception Network* [online]. 2018 [cit. 2021-04-14]. Dostupné z: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

[55] SHAIKH, Faizan. *Deep Learning in the Trenches: Understanding Inception Network from Scratch* [online]. 2018 [cit. 2021-04-10]. Dostupné z: https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/

[56] PYTORCH, team. *INCEPTION_V3* [online]. 2015 [cit. 2021-04-10]. Dostupné z: https://pytorch.org/hub/pytorch_vision_inception_v3/

[57] DWIVEDI, Priya. *Understanding and Coding a ResNet in Keras* [online]. 2019 [cit. 2021-04-14]. Dostupné z: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

[58] FULTON, Lawrence V., Diane DOLEZEL, Jordan HARROP, Yan YAN a Christopher P. FULTON. *Classification of Alzheimer's Disease with and without Imagery Using Gradient Boosted Machines and ResNet-50* [online]. 2019 [cit. 2021-05-01]. Dostupné z: https://www.mdpi.com/2076-3425/9/9/212

[59] LEE, Jeong Hoon, Eun Ju HA, DaYoung KIM, Young Jun JUNG, Subin HEO, Yong-ho JANG, Sung Hyum AN a Kyungmim LEE. *Application of deep learning to the diagnosis of cervical lymph node metastasis from thyroid cancer with CT: external validation and clinical utility for resident training* [online]. 2019 [cit. 2021-05-01]. Dostupné z: https://link.springer.com/content/pdf/10.1007/s00330-019-06652-4.pdf

[60] EMARA, Taha, Heba M. AFITY, Fatma Helmy ISMAIL a Aboul Ella HASSANIEN. *A Modified Inception-v4 for Imbalanced Skin Cancer Classification Dataset* [online]. 2019 [cit. 2021-5-1]. Dostupné z: https://ieeexplore.ieee.org/abstract/document/9068110

[61] NGUYEN, Long D., Dongyun LIN, Zhiping LIN a Jiuwen CAO. *Deep CNNs for microscopic image classification by exploiting transfer learning and feature concatenation. 2018 IEEE International Symposium on Circuits and Systems (ISCAS)* [online]. IEEE, 2018, 2018, , 1-5 [cit. 2021-05-04]. ISBN 978-1-5386-4881-0. Dostupné z: doi:10.1109/ISCAS.2018.8351550

[62] *GoogLeNet series* [online]. 2021 [cit. 2021-04-14]. Dostupné z: https://www.programmersought.com/article/25475139715/

[63] ARDAKANI, Ali Abbasian, Alireza Rajabzadeh KANAFI, U. Rejendra ACHARYA, Nazanin KHEDEM a Afshin MAAMMADI. *Application of deep learning technique to manage COVID-19 in routine clinical practice using CT images: Results of 10 convolutional neural networks* [online]. 2020 [cit. 2021-04-20]. Dostupné z: https://www.sciencedirect.com/science/article/abs/pii/S0010482520301645?casa_token=s253Vcb3jTIAAAAA:UYv4e7OUFNQLDKOdcWV4JS6v8d056gPHs4Qqrx8gTEF-Wo0bl2GrD8QToSgRpSGnlMQ4_zKmoA

[64] KOSS, Frank V. *Gesture-Recognition-for-Robotic-Control-Using-Deep-Learning* [online]. 2017 [cit. 2021-4-15]. Dostupné z: https://www.researchgate.net/figure/Comparison-of-popular-CNN-architectures-The-vertical-axis-shows-top-1-accuracy-on_fig2_320084139

[65] DOSHI, Sanket. *Various Optimization Algorithms For Training Neural Network* [online]. 2019 [cit. 2021-7-22]. Dostupné z: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

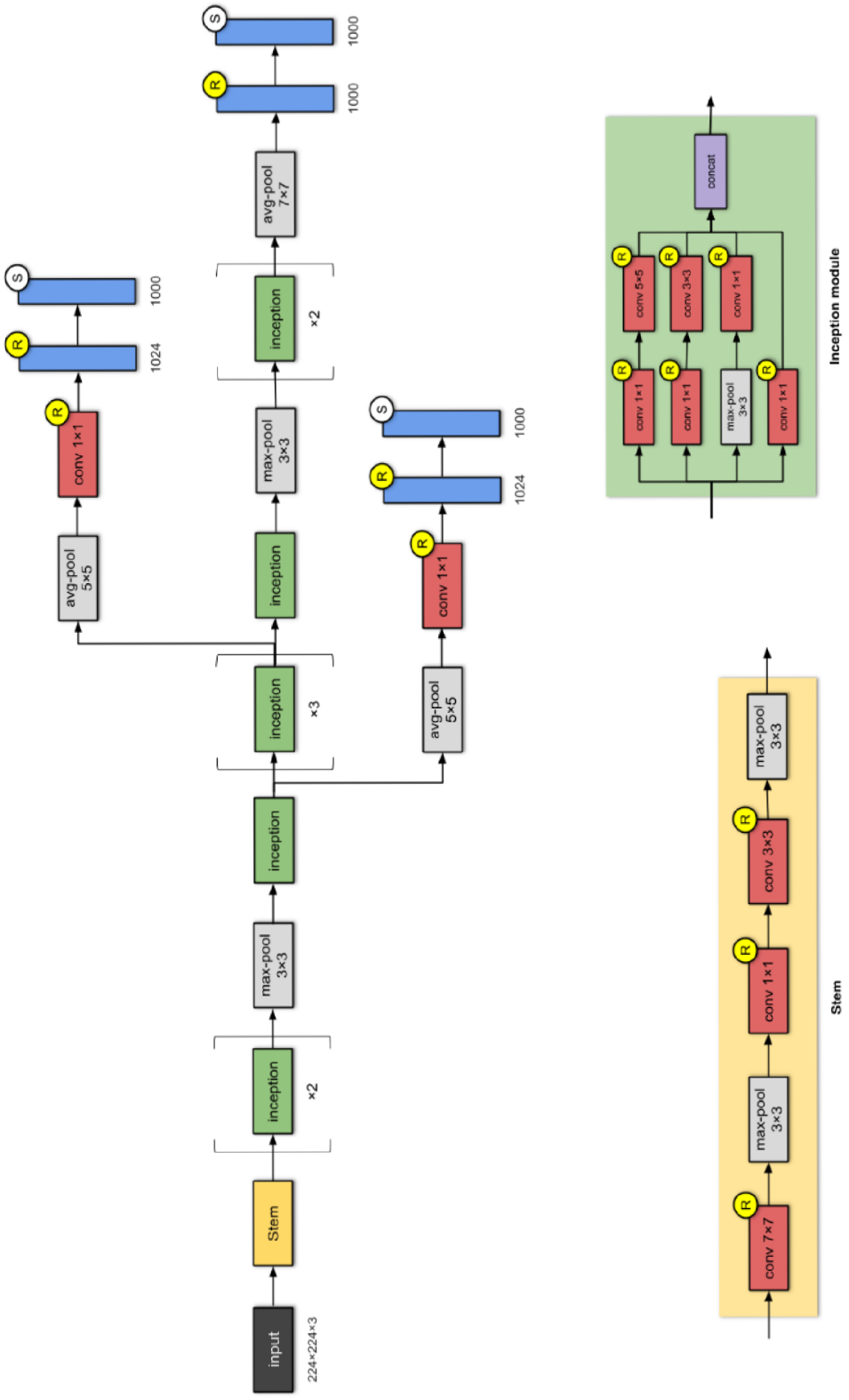
SEZNAM SYMBOLŮ A ZKRATEK

LF	- chybová funkce
ERR	- celková chyba
N	- počet tréninkových objektů
$f(x_{i,}b)$	- predikovaná výstupní veličina
y_i	- skutečná výstupní veličina
b	- předpojatost modelu
F	- míra
P	- skutečně pozitivní objekty v realitě
$\hat{P}$	- skutečně pozitivní objekty v modelu
Err	- chyba kvalitativní chybové funkce
TP	- skutečně pozitivní klasifikace
FN	- falešně negativní klasifikace
TN	- skutečně negativní klasifikace
$\varepsilon_{0\text{-}1}(f(x_{i,}w), y_i)$	- binární chyba modelu
$f(x_{i,}w)$	- výstup modelu pro daný vstupní objekt x
$\varepsilon_{MSE}(f(x_{i,}w), y_i)$	- střední kvadratická chyba modelu
$\varepsilon_{ce}(f(x_{i,}w), y_i)$	- chyba křížové entropie pro model
$f(x_{i,}w),y_i$	- chyba modelu pro daný vstupní objekt x
$f_{Bayes}(x)$	- funkce Bayesovy chyby pro objekt x
$p(\omega\|x)$	- posterior pravděpodobnosti dán třídou ω pro objekt x
$\varepsilon_{emp}(f,w, X^{tr})$	- chyba tréninkových dat
$p(\omega_T\|x)$	- hustota pravděpodobnosti cílové třídy pro objekt x
$p(\omega_T)$	- hustota pravděpodobnosti cílové třídy
p(x)	- hustota pravděpodobnosti pro objekt x
$p(\omega_0)$	- hustota pravděpodobnosti odlehlé hodnoty
$p(X\|\ \omega_0)$	- hustota pravděpodobnosti odlehlé hodnoty pro objekt x
μ	- vektor průměrů data setu
z	- sloupcový vektor testovaných objektů
Σ	- plná kovarianční matice
d	- velikost vstupního vzorku
p_N (x; μ, Σ)	- Gausova distribuce
n_{freeN}	- počet volných parametrů v modelu N
N_{Mog}	- počet Gaussianů
α_j	- směšovací koeficienty
p_N (x; μj, Σj)	- Gaussova distribuce
p_{MoG} (x)	- směs Gaussianů reprezentovaná lineární kombinací normálních distribucí
N_{Mog}	- počet Gaussianů
$n_{freeMOG}$	- počet volných parametrů u směsi Gaussianů
p_N (x; x_i, hI)	- Odhadovaná hustota nejčastějších Gaussovských jader

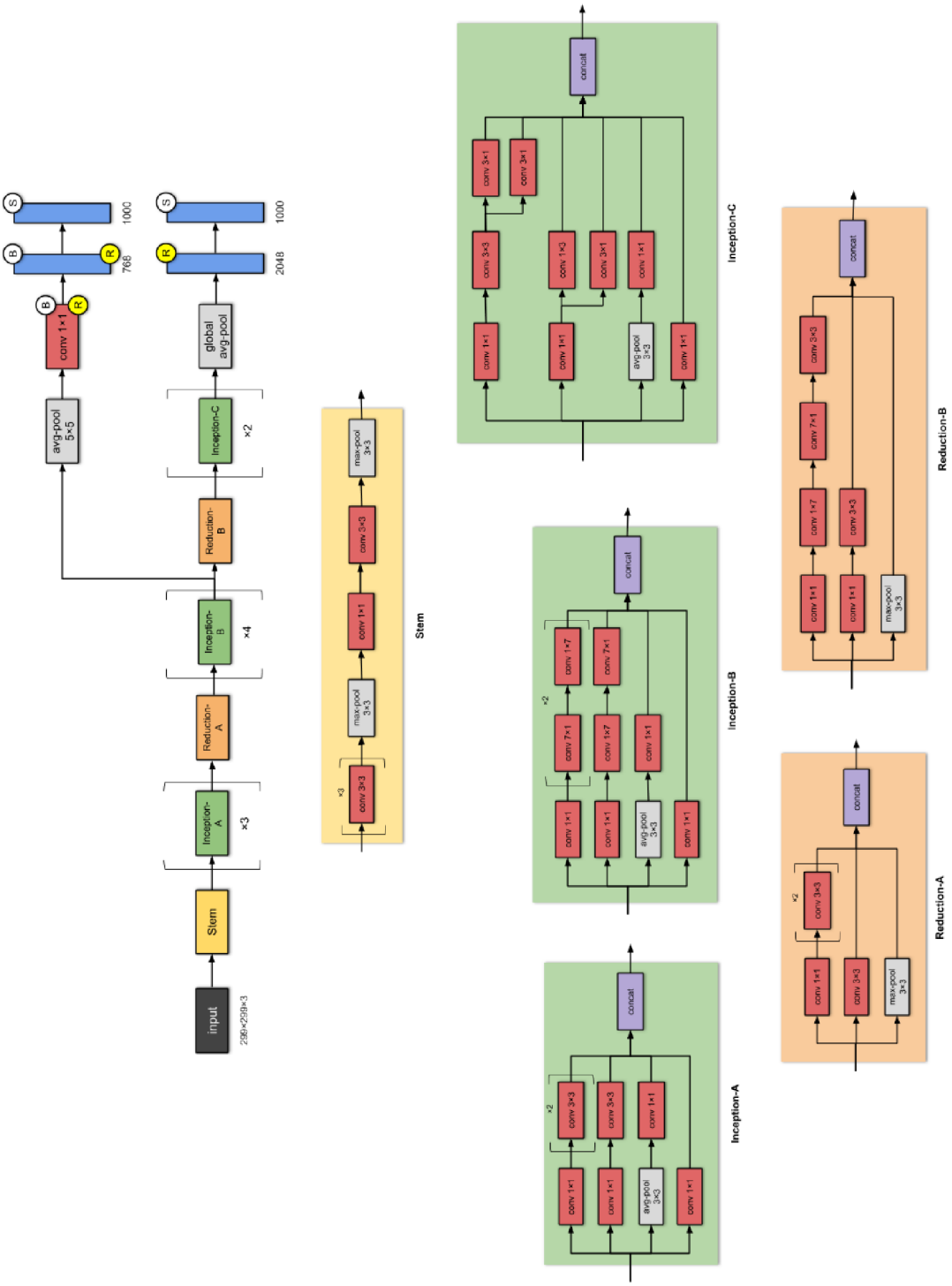
h	- šířka odhadu Parzenu
I	- matice identity
$p_p(x)$	- odhad hustoty Parzen pro objekt x
n_{freep}	- počet volných parametrů u odhadu hustoty parzen
R	- poloměr hypersféry u SVDD metody
a	- střed hypersféry u SVDD metody
$\varepsilon_{strust}(R,a)$	- odhad chyby pro SVDD metodu s poloměrem R a středem a
x_i	- testovaná objekt
μ_k	- pozice středu sféry k
$\varepsilon_{k\text{-}centr}$	- minimalizovaná chyba k-centres
z	- pozice testovaného objektu
$d_{k-centr}$	- minimalizovaná vzdálenost středů testovaného objektu a sféry
k	- počet sfér
$n_{freek-c}$	- počet volných parametrů
$NN^{tr}_k(z)$	- k-tý nejbližší soused v testovací sadě z
V	- objem sféry obsahující objekt
$f_{NN}{}^{tr}(z)$	- odhad lokální hustoty
$\varepsilon_{k\text{-}m}$	- chyba metody K-means
$d_{k\text{-}m}$	- minimalizovaná vzdálenost testovaného objektu a středu sféry
$\eta(x_i - \mu_k)$	- Rychlost učení
x_i	- vektor tréninkový objekt
μ_k	- vektor prototyp
k	- počet klastrů
$n_{freek-m}$	- počet volných parametrů
M	- počet obrazců v databázi
$n_{freekPCA}$	- počet volných parametrů
$f_{diab}(z;w)$	- výstup klasifikátoru s volnými parametry w a vstup z
$d_{diab}(z)$	- vzdálenost mezi původním objektem a rekonstruovaný objekt
h_{diab}	- velikost úzké vrstvy
$n_{freekdiabo}$	- počet volných parametrů
h_{auto}	- počet skrytých jednotek v automatickém kodéru
$n_{freekauto}$	- počet volných parametru
Θ	- váhy modelu
e_k	- krok učení
L	- gradient
m	- velikost dávky
g	- gradient přechodu
s	- čtverec gradientů přechodů

SEZNAM PŘÍLOH

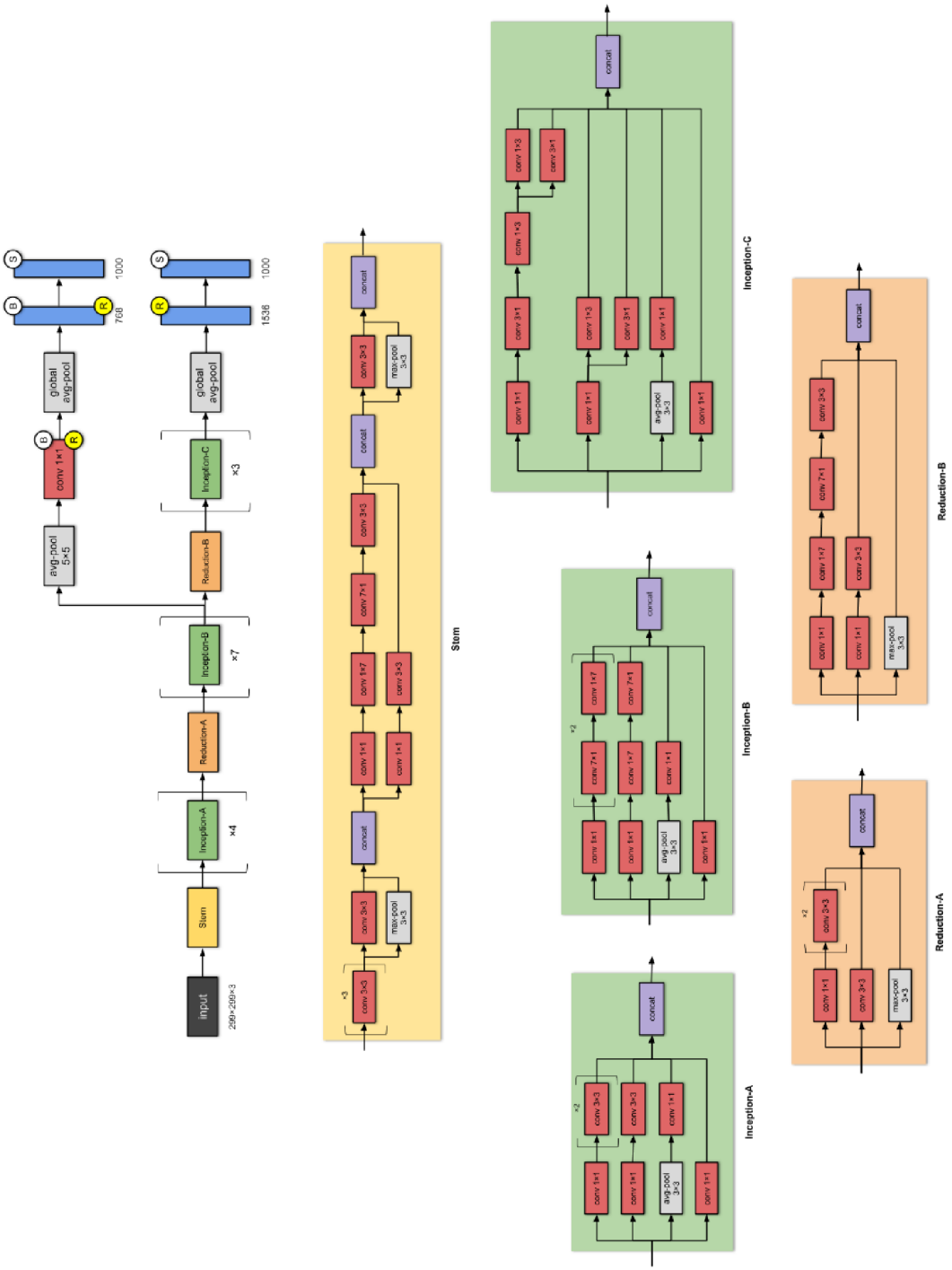
Příloha A - Inception v1 architektura [25]



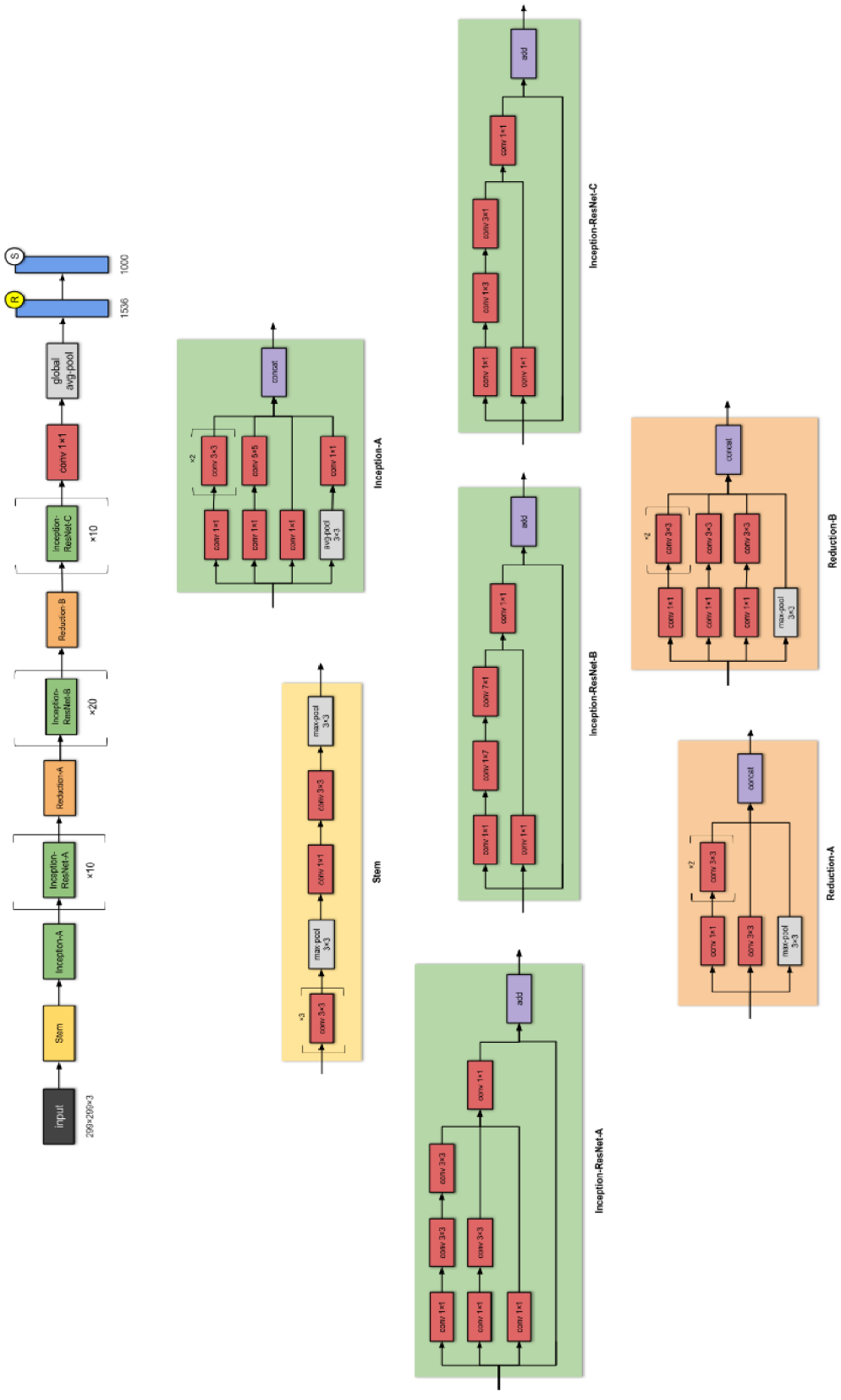
Příloha B - Inception v3 architektura [25]



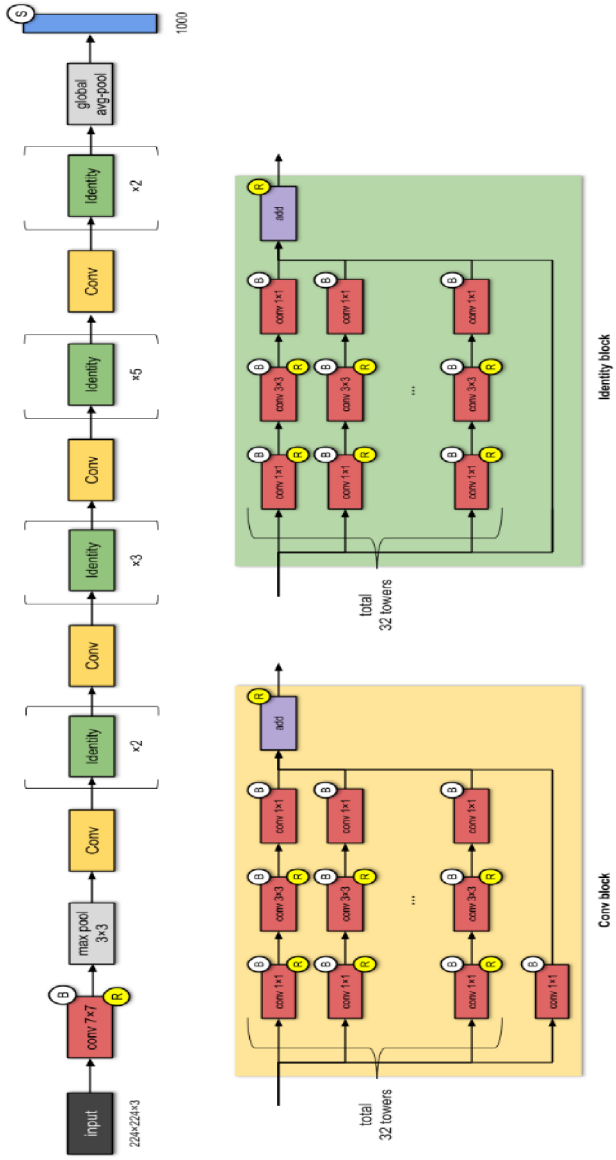
Příloha C - Inception v4 architektura [25]



Příloha D - Inception ResNets architektura [25]



Příloha E - ResNeXt-50 architektura [25]



Příloha F - GoogleNet architektura [63]

Příloha G - Seznam přiložených programy a dat

Přílohou práce je obrazový dataset čítající 1200 snímků, dataset je součástí elektronicky odevzdávané práce

Přílohou je Matlabovský script s názvem „snímek.m“ pro ovládání kamery, přiložený script je součástí elektronicky odevzdávané práce

Přílohou je Matlabovský script s názvem „zmenseni.m“ pro úpravu snímků z kamery, přiložený script je součástí elektronicky odevzdávané práce

Přílohou je Python script s názvem „Gui_app.py“ script obsahuje grafickou aplikaci, přiložený script je součástí elektronicky odevzdávané práce

Přílohou je Python script s názvem „Klasifikator_main.py“ script obsahuje hlavní program, přiložený script je součástí elektronicky odevzdávané práce

Přílohou je Python script s názvem „Vlastní_AlexNet.py“ script obsahuje vlastní AlexNet architekturu, přiložený script je součástí elektronicky odevzdávané práce

Přílohou je Python script s názvem „Vlastní_CNN.py“ script obsahuje vlastní CNN architekturu, přiložený script je součástí elektronicky odevzdávané práce

Přílohou je Python script s názvem „Vlastní_LaNEt5.py“ script obsahuje vlastní LaNet5 architekturu, přiložený script je součástí elektronicky odevzdávané práce

Přílohou je Python script s názvem „Vlastní_ResNet.py“ script obsahuje vlastní ResNet architekturu, přiložený script je součástí elektronicky odevzdávané práce

Přílohou je Python script s názvem „Vlastní_VGG11.py“ script obsahuje vlastní VGG11 architekturu, přiložený script je součástí elektronicky odevzdávané práce

Přílohou je Python script s názvem „Vlastní_VGG16.py“ script obsahuje vlastní VGG16 architekturu, přiložený script je součástí elektronicky odevzdávané práce

Přílohou je Python script s názvem „Vlastní_VGG19.py“ script obsahuje vlastní VGG19 architekturu, přiložený script je součástí elektronicky odevzdávané práce