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:*** 17.5.2021

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 hyper parametrů na kvalitu unární klasifikace pro jednotlivé architektury. Součástí odevzdání práce je potom praktický příklad reimplementace unárního klasifikátoru.

Klíčová slova

unární klasifikátor, obrazová data, klasifikace, binární klasifikátory, multi-class klasifikátory, reimplementace unárního klasifikátoru

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 hyper parameters on the quality of unary classification for individual architectures. Part of the submission is a practical example of reimplementation of the unary classifier.

Keywords

unary classifier, image data, classification, binary classifiers, multi-class classifiers, unimplementation of unary classifier

Bibliografická citace

Citace tištěné práce:

BENEŠ, Jiří. *Unární klasifikátor obrazových dat.* Brno, 2021. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/134347. 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. 2021-05-03]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/134347. 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: 12. května 2021 ----------

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Karlu Hrákovi, 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: 12. května 2021 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Na úvod této práce je důležité definovat si pojem strojové učení, jeho oblasti použití a vývoj. Citace: „Strojové učení lze definovat jako nauku o nástrojích které umožňují učení umělých objektů.“ [2]. Tuto definici lze chápat tak že strojové učení popisuje matematické nástroje pro vytvoření modelu dat vzhledem na základě získaných zkušenostem s cílem co nejvíce zlepšit výkonost daného modelu.

Strojové učení bylo vytvořeno za účelem z dokonalení zpracovávání dat, a to co do rychlosti tak i do přesnosti. Se strojovým učením se v dnešní době nejčastěji setkáme v oblastech syntézy či komprimace obrazu, rozpoznávání řeči, u hlasové syntézy nebo u síťového provozu. Další oblastí, kde se můžeme setkat se strojovým učením jsou počítačové hry. Zde má strojové učení za cíl konkurovat člověku. Pro tuto oblast bylo strojové učení vyvinuto, v roce 1959 Arthur Samuel ze společnosti IBM poprvé aplikoval strojové učení. V následujících letech došlo k rozšíření strojového učení do spousty odvětví. Strojové učení je nejpoužívanější pro zpracování obrazu či videa, 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.

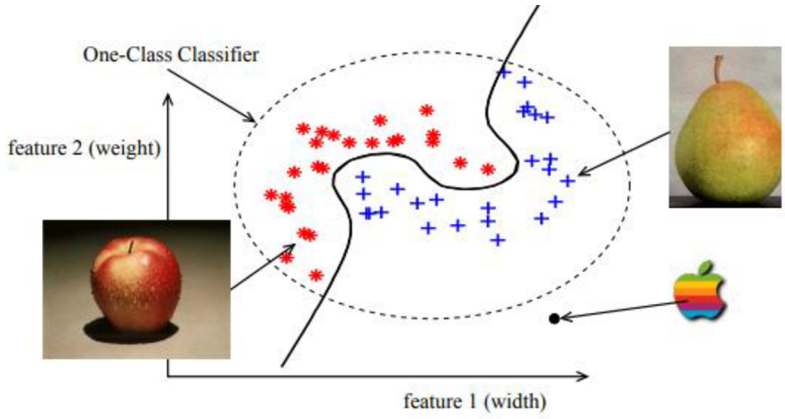
Tato práce se dále zabývá klasifikačními algoritmy. Klasifikační algoritmy jsou takové algoritmy, které jsou schopny na základě své vnitřní funkce roztřídit vstupní data, a to ať už numerická, obrazová nebo data jiného typu. Před samotným rozdělením klasifikačních algoritmů je nutné si nejprve definovat jednotlivé pojmy v klasifikaci. V této části jsou definované pojmy zabývající se návrhem a požadavky na klasifikační funkci. V první kapitole se snažím objasnit postup učení klasifikátoru a obecným definováním klasifikačního problému. Následně v druhé kapitole se snažím představit jednotlivé postupy pro vyhodnocení kvality klasifikace, nejčastěji se setkáme s vyhodnocením za pomoci chybové funkce. Následující kapitola je potom zaměřena na vyhodnocení klasifikační funkce a to několika metodami, kdy každá metoda má za cíl minimalizovat chybovost klasifikační funkce. Poslední kapitola této části se potom zabývá návrhem klasifikační funkce tak aby byla schopna klasifikovat i data která nebyla obsažena v tréninkové sadě. Tato metoda se nazývá generalizace a velice důležitá pro vytvoření optimální klasifikační funkce. Cílem této kapitoly je tedy seznámit čtenáře s požadavky a postupy, které jsou používaný pro návrh klasifikační funkce.

Následující kapitola této práce se zabývá rozdělením klasifikátorů. Obecně lze rozdělit klasifikátory do několika tříd, kdy každá třída přistupuje ke klasifikaci odlišně a odlišné jsou i výsledky jednotlivých metod. V základu lze klasifikační algoritmy rozdělit na unární, binární a multi-class klasifikátory. Každý z těchto klasifikátorů lze následně rozdělit na základě typu a objemu vstupních dat, vnitřní klasifikační funkci a nebo na rozdělení výstupních dat. Tato práce se potom zaměřuje na unární klasifikaci, principy unární klasifikace a rozdělením unárních klasifikátorů. Poslední kapitola této práce má za cíl představit jednotlivé unární klasifikační architektury. Cílem bylo vybrat nejpoužívanější metody a srovnat jejich možnosti klasifikace, složitost návrhu, přesnost klasifikace a další parametry. Výsledkem je potom ucelený přehled jednotlivých metod a jejich srovnání v poslední kapitole této části práce. Cílem poslední kapitoly je praktická ukázka klasifikace na reálných datech. Kapitola se zabývá postupným definováním jednotlivých hyperparametrů a jejich vlivu na kvalitu klasifikace. Cílem kapitoly je nalezení optimálního rozložení hyperparametrů pro konkrétní CNN sítě a následně srovnat výkonost jednotlivých sítí.

1. KLASIFIKÁTOR

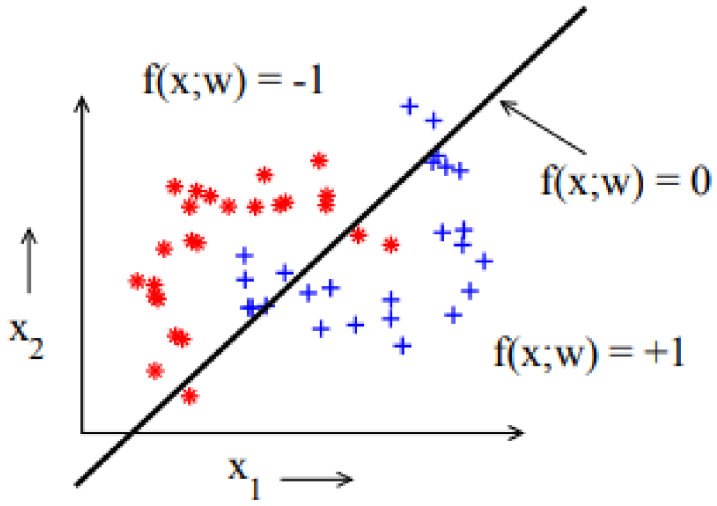
1.1 Definice klasifikátoru

Jednotlivé klasifikační metody lze rozdělit několika způsoby nejčastěji se však setkáme s rozdělením na unární, binární a multi-class klasifikátory. Tímto způsobem lze rozdělit klasifikátory na základě množství výstupních klasifikovaných množin. Před samotným popisem jednotlivých klasifikátoru je nutné klasifikátor definovat. Mezi klasicky uváděné příklady můžeme uvést příklad klasifikace jablek v ostatním ovoci. V tomto příkladu tvoří trénovací sadu obrázky jablek, už nás nezajímá, jaké další ovoce budeme klasifikovat.



Obrázek 1: Příklad klasifikace [2]

Jak vidíme na obrázku 1 tak klasifikační funkce rozděluje klasifikovaný prostor na dvě části. V první části se nachází pozitivně klasifikovaná jablka, zatímco v druhé části se nachází ostatní objekty. V praxi se většinou setkáme s jednodušším rozdělením prostoru za pomocí klasifikační funkce. Nicméně platí že pomocí klasifikační funkci můžeme libovolně rozdělit klasifikovaný prostor.[2]



Obrázek 2: Rozdělení klasifikovaného prostoru pomocí klasifikační funkce [2]

Jak vidíme na obrázku 2 opět došlo k rozdělení klasifikovaného prostoru pomocí klasifikační funkce f(x, w), v tomto případě se již nejedná o naprosto ideální rozdělení. V praxi je funkce f(x, w) volena dopředu a závisí na parametrech x a vektoru w které jsou nastaveny podle konkrétní úlohy klasifikace. Příkladem těchto funkcí jsou lineární klasifikátory, Gaussiány, neuronové sítě nebo vektorové klasifikátory [2]. Na obrázku 2 lze vidět že většina dat byla klasifikována správně ale došlo k malé chybě. Tato chyba vznikla na základě špatně zvolené klasifikační funkce, konkrétně zde byla zvolena lineární klasifikační funkce.

1.2 Definování chybové funkce

Chybová funkce určuje rozdíl mezi skutečnou výstupní veličinou y a predikovanou výstupní veličinou, tato predikovaná výstupní veličina je dána klasifikační funkcí na základě vstupní veličiny x. Klasifikační funkce lze rozdělit na dvě hlavní skupiny. Kvantitativní chybové funkce, do kterých patří metody nejmenší čtverec vzdálenosti (MNČ), váhová MNČ, absolutní chyba a polynomická chyba MNČ [2]:

Čtverec vzdálenosti (MNČ):

$$LF = [y_i - f(x_i, \boldsymbol{b})]^2, \quad Err = \frac{1}{N}\sum_{1}^{N}[y_i - f(x_i, \boldsymbol{b})]^2 \qquad [2]\ (1.0)$$

Váhová MNČ:

$$LF = [y_i - f(x_i, \boldsymbol{b})]^2 \cdot f(x_i), \quad Err = \frac{1}{N}\sum_{i=1}^{N}[y_i - f(x_i, \boldsymbol{b})]^2 \cdot f(x_i) \qquad [2]\ (1.1)$$

Absolutní chyba:

$$LF = |y_i - f(x_i, \boldsymbol{b})|, \quad Err = \frac{1}{N}\sum_{1}^{N}|y_i - f(x_i, \boldsymbol{b}) \qquad [2]\ (1.2)$$

Polynomická chyba:

$$LF = \frac{1}{d}|y_i - f(x_i, \boldsymbol{b})|^d \qquad [2]\ (1.3)$$

Druhou skupinou chybových funkcí jsou kvalitativní chybové funkce. Kvalitativní chybové funkce počítají chybovost jako poměr mezi chybně a správně klasifikovanými objekty [2]. Výpočet skutečně pozitivních(P) a správně klasifikovaných objektů (T):

$$P = TP + FN$$
$$T = TP + TN \qquad [2]\ (1.4)$$

Chyba kvalitativní chybové funkce (Err) :

$$F = \frac{2 \cdot PPV \cdot Sens}{PPV + Sens} = \frac{2 \cdot TP}{\hat{P} + P} \quad [2] (1.5)$$

$$Err = F / (T+F)$$

1.3 Vyhodnocení kvality klasifikace

Samotné vyhodnocení přesnosti klasifikace lze provést několika metodami, nicméně 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 volíme tak aby výsledná chyba klasifikace byla co nejmenší. [2]

1) První možností je diskrétní klasifikování:

$$\varepsilon_{0-1}(\mathrm{f}(x_i, \mathrm{w}), y_i) = \begin{cases} 0, & pokud\ f(x_i, \mathrm{w}) = y_i \\ 1, & jinak \end{cases} \quad [2] (1.6)$$

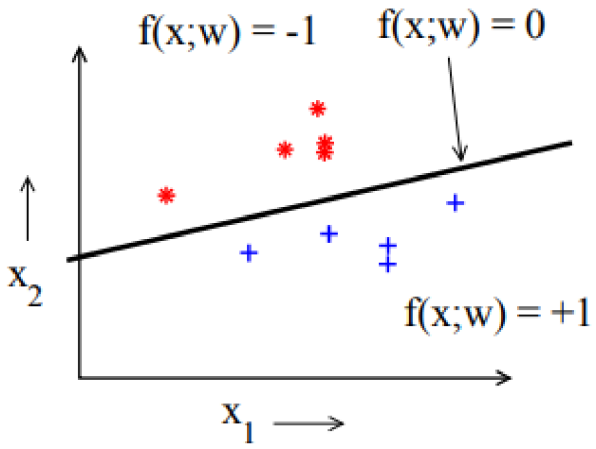
2) Nejběžnější chybou při klasifikaci je střední kvadratická chyba (MSE):

$$\varepsilon_{MSE}(\mathrm{f}(x_i, \mathrm{w}), y_i) = (\mathrm{f}(x_i, \mathrm{w}), y_i)^2 \quad [2] (1.7)$$

3)Třetím typem je křížová entropie (porovnává se rozdíl mezi modelem a skutečným rozložením, výsledky jsou změněny na kladné hodnoty $y_i = \{0, 1\}$):

$$\varepsilon_{\mathrm{ce}}(f(\mathbf{x}_i; \mathbf{w}), y_i) = f(\mathbf{x}_i; \mathbf{w})^{y_i} \left(1 - f(\mathbf{x}_i; \mathbf{w})\right)^{1-y_i} \quad [2] (1.8)$$

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 nebo neobvykle. Například může obsahovat jenom velice malý počet dat nebo velké odchylky, popřípadě data mohou být silně zašuměná, a to až do takové míry že nelze stanovit přesný model. Čím méně dat je obsaženo v tréninkové sadě tím výraznější je tento problém. Tento problém je ukázán na obrázku 3 kde z původní tréninkové sady která obsahovala 25 objektů zůstalo pouze 5 příkladů. Tímto dochází k odchýlení navrženého lineárního modelu od optimálního řešení pro danou tříd. V praxi potom stojíme před problémem, jestli takto malý vzorek dat vůbec odpovídá realitě. [2][11]



Obrázek 3: Návrh modelu s omezenými daty [2]

V praxi se ovšem můžeme setkat s daty které nebyla součástí tréninkové sady. Cílem návrhu modelu tedy není vytvořit model který si dokáže nejlépe poradit s daty obsaženými v trénovací sadě, ale model který si dokáže poradit s novými objekty které chceme klasifikovat. Hlavním cílem tedy je najít model který vykazuje dobré zobecnění („generalization“). Takovýto model dokáže správně klasifikovat i data neobsažená v tréninkové sadě. Chceme-li odhadnou, jak dobře model zobecňuje je nutné ho testovat novou sadou dat která nebyla použita pro tréninku. Tím že použijeme rozdílná data také zjistíme skutečnou hodnotu výkonu modelu. Sada dat, která je používaná pro odhad chyby klasifikátoru se nazývá testovací sada. [2][11]

1.4 Zobecňování

Jak jsem uvedl v předchozí kapitole zobecňování je vlastnost modelů, které popisuje jejich schopnost klasifikovat data která nebyla obsažena v trénovací sadě. Pro vyhodnocení kvality zobecňování je použito tak zvané Bayesovo rozhodovací pravidlo. Toto pravidlo přiřadí každému objektu x v trénovacích datech hodnotu na základě pravděpodobnosti p(ω|x). Kde p(ω|x) je pravděpodobnost že objekt x patří do třídy ω. Pro Bayesovu funkci potom platí [2]:

$$f_{Bayes}(\mathbf{x}) = \begin{cases} +1 & \text{if} \quad p(\omega_1|\mathbf{x}) \geq p(\omega_2|\mathbf{x}), \\ -1 & \text{if} \quad p(\omega_1|\mathbf{x}) < p(\omega_2|\mathbf{x}). \end{cases}$$

[2] (1.9)

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 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).

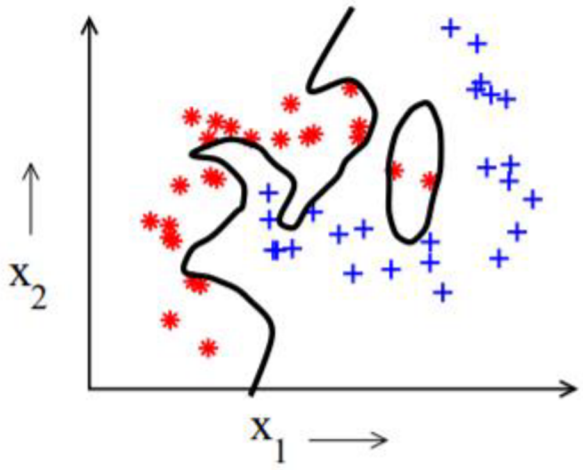
S výjimkou použití uměle vygenerovaných dat, u kterých známe distribuci a rozdělení tříd, nelze Bayesovo rozdělení v praxi použít.

V praxi je skutečná chyba nahrazena aproximací empirickou chybou tréninkových dat [2]:

$$\mathcal{E}_{\text{emp}}(f, \mathbf{w}, \mathcal{X}^{tr}) = \frac{1}{N} \sum_{i} \varepsilon(f(\mathbf{x}_i; \mathbf{w}), y_i) \qquad \text{[2] (1.10)}$$

Pro tuto aproximaci je nutné použít dostatečně velký vzorek tréninkových dat která mají shodnou distribuci se skutečnými daty. Optimalizací empirické chyby ε_{emp} není zaručena optimalizace skutečné chyby ε_{true}[2].

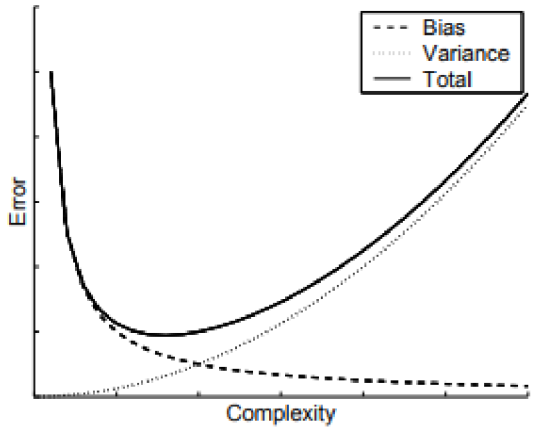
Zaměřme se na případ, kdy se nám podařilo navrhnout model až příliš „těsně“. Tento případ může nastat pokud jsou některé objekty příliš vzdálená od cílové množiny, ale mají být vyhodnocena jako cílové objekty. Tento stav se označuje jako přetrénovaný model takovémuto výsledku se snažíme zabránit.



Obrázek 4: Přetrénovaný model [2]

Na obrázku 4 je potom vidět takováto přetrénovaní. Protože je tento model pro tato data příliš flexibilní, najde v datech strukturu, která tam opravdu není. Přetrénovaný model naznačuje že levá třída se ve skutečnosti skládá ze dvou samostatných shluků objektů. Objekty umístěné mezi těmito dvěma shluky budou klasifikovány jako odlehlé hodnoty! Testování tohoto modelu s nezávislými testovacími daty odhalí, že výkon zobecnění není příliš vysoký. Protože funkce modelu f (x; w) by měla být definována pro všechna možné objekty x, objem dat která by měla být popsána, se exponenciálně zvyšuje s velikostí modelu. Tomu se říká kletba dimenzionality. Snížením počtu funkcí na objekt klesá počet stupňů volnosti ve modelu f (x; w) a zvyšuje se výkon zobecňování. To znamená, že zvýšení počtu funkcí na objekt může poškodit výkon klasifikace (na druhou stranu ale

můžeme klasifikovat s větší přesností data nacházející se poblíž hrany klasifikační funkce)! Jedním z řešení kletby dimenzionality a nadměrného klasifikování je použití redukce nebo výběru prvků a zachování pouze několika málo nejlepších vlastností dat v množině. Je také možné vypočítat malý počet nových funkcí ze sady starých funkcí (například lineární kombinace starých funkcí), tato metoda se nazývá extrakce funkcí. Ale když se použije složitá funkce f (x; w) (s velkým počtem volných parametrů w), tyto metody stále nebudou dostatečné k zajištění dobrého zobecnění. Důležitou veličinou je počet objektů, velikost vzorku N, s ohledem na rozměrnost prostoru funkcí d.
Malou velikostí vzorku je myšleno, že počet ukázkových objektů není dostatečný pro odhad všech volných parametrů w ve funkci f (x; w) s dostatečnou přesností. Vyskytuje se také opačný problém nedostatečné flexibility. Zde funkce f (x; w) není dostatečně flexibilní, aby sledoval všechny charakteristiky v datech. Model pak ukazuje nenulovou odchylku. Tyto systematické chyby nedokáže napravit ani nekonečné množství tréninkových dat. Rozdíl mezi výsledky modelu a tím, co ukazují datové štítky y (získané za pomocí např. Bayesovou funkcí), v průměru nezmizí. Tento klasifikátor má nízkou složitost. To je ukázáno na obrázku 2. Ačkoli bylo k dispozici přiměřené množství tréninkových dat pro problém klasifikace, vybraný model nebyl dostatečně flexibilní, aby zachytil charakteristiky v datech. V tomto příkladu měl být použit jiný klasifikátor, flexibilnější než lineární. [2]

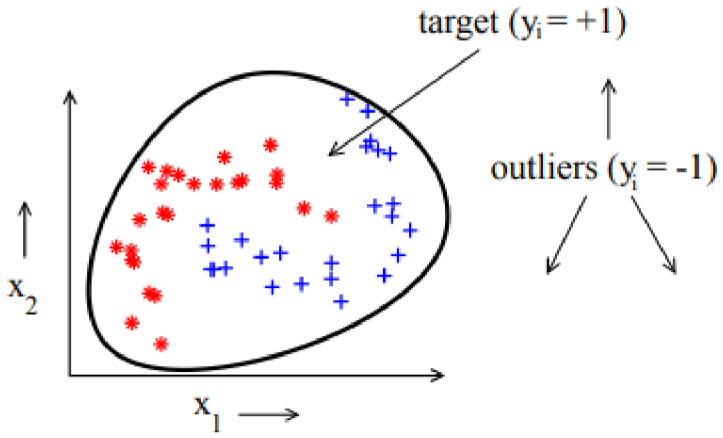


Obrázek 5: Nárust složitosti pro minimalizaci chyby [2]

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.

1.5 Unární klasifikátory

Prvním zde definovaným klasifikátorem je unární klasifikátor. Tento typ klasifikátoru rozlišuje pouze jednu třídu objektů tzv. cílové objekty a všechny ostatní objekty jsou označena za odlehlé objekty.[2]



Obrázek 6: unární klasifikace [2]

Na obrázku 6 je možné vidět použití unárního klasifikátoru na malém vzorku dat. Cílové objekty jsou zde označeny y = 1 a odlehlé objekty jsou zde reprezentovány y = -1.
Na obrázku nejsou žádné odlehlé objekty. Plná čára zde reprezentuje možný model, který rozlišuje mezi cílovými a odlehlými objekty. V literatuře můžeme najít několik různých názvů (outlier detection, novelty detection nebo concept learning), svůj název odvozují podle oblasti použití. Všechny ale pracují na stejném principu. Snaží na detekovat neobvyklé objekty v klasifikovaných datech. Tyto neobvyklá data mohou být následkem chyby v měření nebo následkem špatně provedeného 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 vzdálená data může použít například neuronové sítě které jsou schopny odhadnou pravděpodobnost a jsou tedy schopny poskytnout vysoce spolehlivé výstupy i pro objekty, které jsou vzdáleny od trénovací sady. V praxi potom může dojít ke stavu, kdy v jedné třídě objektů je dostatečný počet dat, zatímco v druhé třídě je jich nedostatek. V praxi se může 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 vzdálený

objekt dojde k vyhodnocení chybového stavu a už není potřeba zjišťovat jaký přesně stav nastal.[2]

	objekt z cílové třídy	objekt z odlehlé třídy
klasifikován jako	true positive TP	false positive FP
klasifikován jako	true negative TN	false negative FN

Tabulka 1: Matice záměny [1]

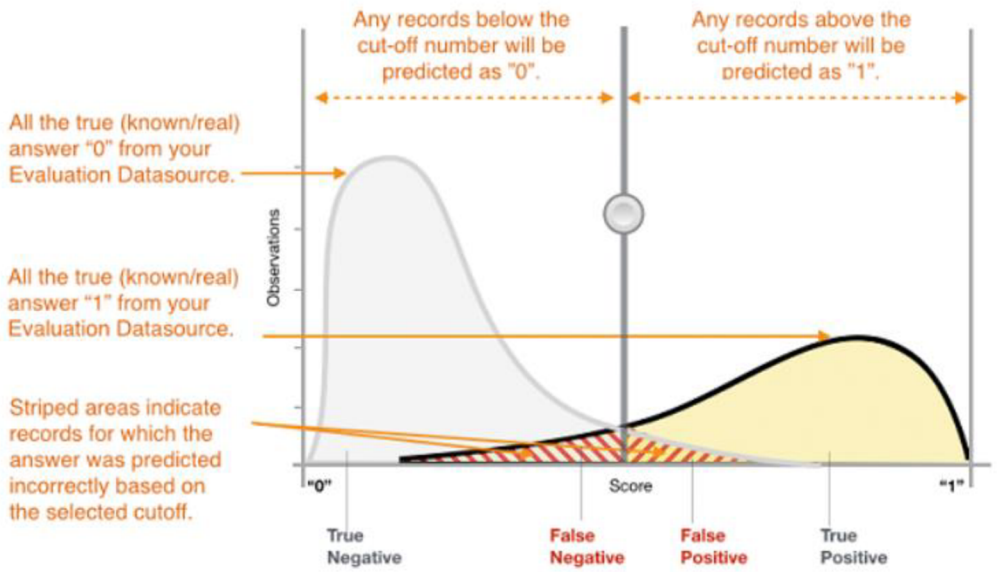
Pro stanovení kvality klasifikace unárních klasifikátorů můžeme použít tak zvanou matici záměny (confusion matrix). K výpočtu skutečné chyby klasifikace bychom potřebovali znát úplnou hustotu pravděpodobnosti obou tříd (cílové třídy i odlehlé třídy). V případě unárního klasifikátoru známe pouze hustotu pravděpodobnosti cílové třídy. To znamená, že lze minimalizovat pouze počet pozitivních objektů třídy, které klasifikátor jedné třídy nepřijímá (tj. Falešné negativy, FN). Při absenci příkladů a distribuce vzorků z objektů odlehlé třídy není možné odhadnout počet odlehlých objektů, které budou přijaty klasifikátorem jedné třídy (falešné poplachy, FP). Proto je pro odhad výkonu a zobecnění přesnosti klasifikace klasifikátoru jedné třídy nutné omezené množství dat z odlehlé třídy. Pokud však během testování není odlehlá třída prezentována v rozumném poměru, dalo by se manipulovat se skutečnými hodnotami přesnosti unárního klasifikátoru. V takových nevyvážených souborech dat může být použito několik dalších metrik výkonu, například f-skóre, geometrický průměr atd. Abychom předešli triviálním řešením jako je přijímání všech dat je nutné stanovit předpoklad o odlehlé hodnotě. Předpokládejme že žádný příklad odlehlé hodnoty není k dispozici, potom tedy lze předpokládat že odlehlé hodnoty jsou rovnoměrně rozloženy kolem cílových dat. Použitím Baeysova pravidle lze získat pravděpodobnost pro cílovou třídu [2]:

$$p(\omega_T|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_T)p(\omega_T)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\omega_T)p(\omega_T)}{p(\mathbf{x}|\omega_T)p(\omega_T) + p(\mathbf{x}|\omega_O)p(\omega_O)} \quad [2](1.11)$$

S unární klasifikací je spojeno i několik specifických problémů. Bez příkladu odlehlých hodnot tedy lze empirickou chybu definovat pouze na cílových datech. Dalším problémem spojeným s unárními klasifikátory je vynucení hladkosti (model neobsahuje skokové změny a je všude spojitě definován) a uzavření modelu kolem cílových dat. Obecně tato zvláštní omeze ztěžují návrh modelu a přináší další problémy jako je kletba dimenzionality a to velmi navyšuje požadavky na počet tréninkových objektů.

1.6 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). Toho můžeme využít při definování modelu a při jejím vyhodnocení.

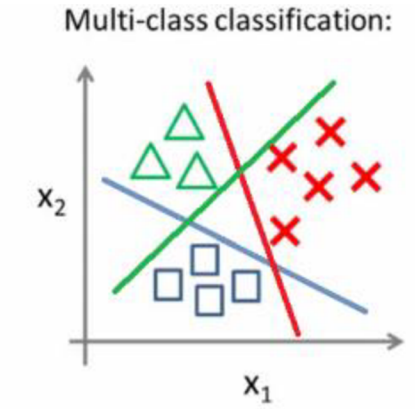


Obrázek 7: Binární klasifikátory – rozložení výsledků [8]

Na obrázku 7 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 a falešné pozitivy tak jak je patrné z obrázku 7. Tyto chyby zde vychází z metody binární klasifikace, protože nás zajímá, kolik cílových objektů nebylo za cílové objekty označeno tzv. falešně negativní a také naopak kolik vzdálených 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. Jak je tedy patrné 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 objektu v tréninkové sadě. Sice dokážeme s jistotou prohlásit, že se jedná o odlehlé hodnoty, ale už nedokážeme stanovit jestli se jedná o typickou ukázku nebo náhodnou odchylku. Tato neznalost potom může způsobit, že ve výsledku navržená klasifikační funkce nebude dosahovat požadované přesnosti. Vliv těchto chyb lze částečně minimalizovat zvětšením trénovací sady.[8]

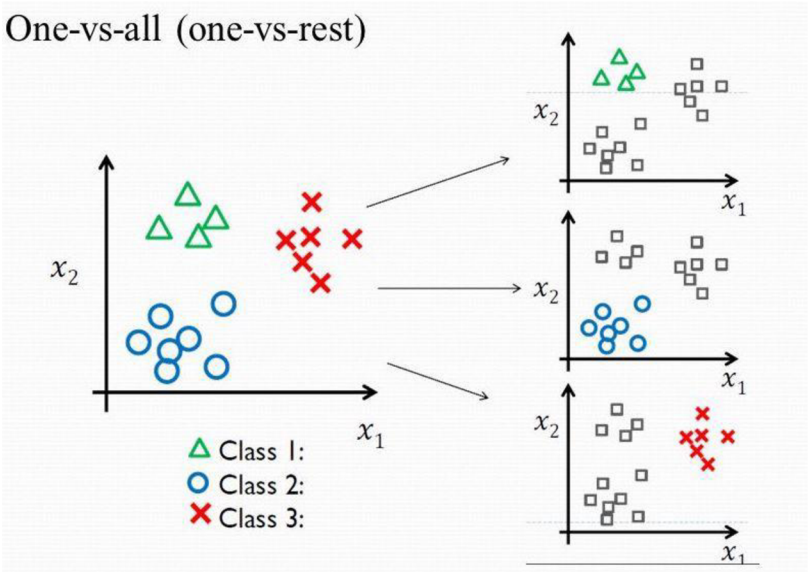
1.7 multi-class klasifikátory

Třetím a posledním typem klasifikátorů je multi-class klasifikátor. Jak jeho název napovídá je určen pro několik vstupních množin dat. Jediným rozdílem oproti binárnímu klasifikátoru je zde přítomnost n-množin namísto dvou.[7]



Obrázek 8: multi-class klasifikace [7]

Zajímavou modifikací je potom tzv. one-vs-all klasifikátor. Úprava spočívá v rozdělení vstupních množin na cílovou množinu a odlehlé objekty. Výhodou je že pokud objekt není součástí cílové skupiny může s jistotou určit že se jedná o odlehlou hodnotu.[7]

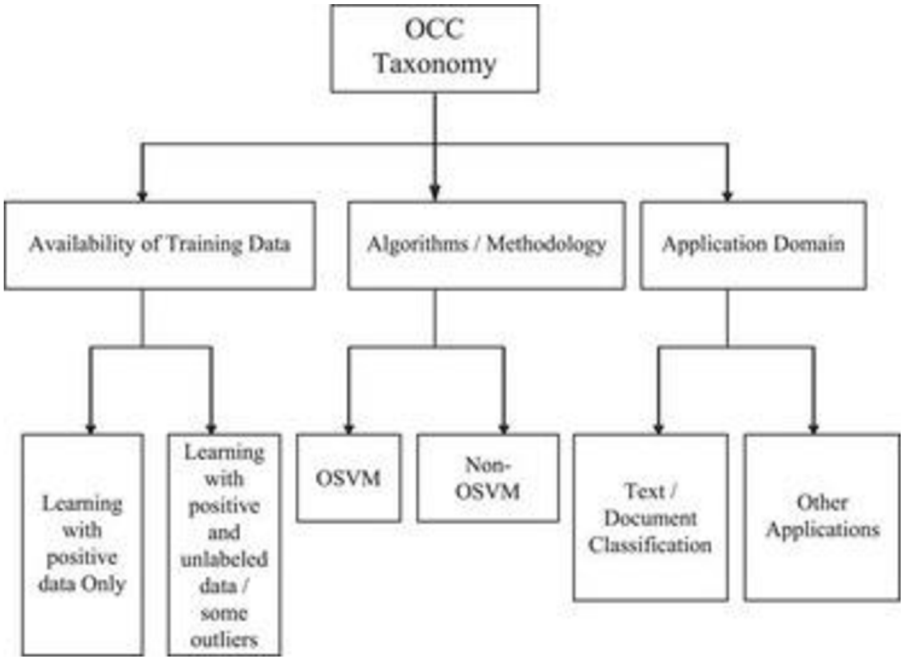


Obrázek 9: modifikace multi-class klasifikace [7]

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 byli typickými zástupci dané třídy objektů.[7]

2. KLASIFIKAČNÍ METODY

2.1 Rozdělení klasifikačních metod



Obrázek 10: Rozdělení unárních klasifikačních metod [1]

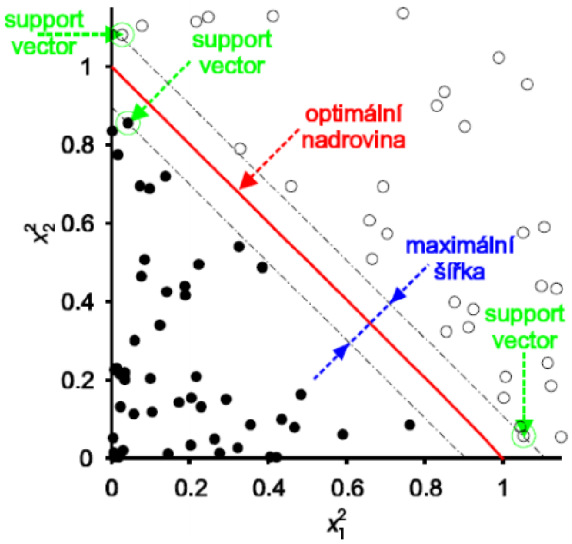
Klasifikační metody můžeme rozdělit na základě znalosti jejich metod, jejich tréninkových dat a v neposlední řadě podle místa aplikace.

Kategorie 1: Dostupnost dat při učení
Dostupnost tréninkových dat hraje klíčovou roli v jakémkoli algoritmu unárních klasifikátorů. Byli stanoveny tři hlavní kategorie:
1) Učení pouze s pozitivními příklady.
2) Učení s pozitivními příklady a určitým množstvím špatně vybraných negativních příkladů nebo uměle vytvořených odlehlých hodnot.
3) Učení s pozitivními a neoznačenými daty.

Kategorie 2: Použité algoritmy
Většinu hlavních vývojových algoritmů unárních klasifikátorů lze klasifikovat do dvou širokých kategorií, jak bylo provedeno buď pomocí
1)OSVM
2)NON-OSVM (včetně různých neuronových sítí, rozhodovacích stromů, nejbližších sousedů a dalších).

Kategorie 3: Použitá doména aplikace
1)Klasifikace textu / dokumentu
2)Další aplikační domény

Typy unární klasifikace lze rozdělit podle dostupnost dat při učení. První podskupinu tvoří klasifikátory učené pouze z pozitivní sady tréninkových dat. Pokud by byl tento způsob učení použit na problém klasifikace ovoce, trénování by probíhalo pouze na tréninkovém datasetu obsahující pouze 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 problému, typický problémem bývá neznalost kompletního rozložení odlehlých hodnot takže nemůže z jistotou stanovit že uvedené příklady jsou typickými zástupci. [1] Základním rozdělením klasifikátoru podle použitých algoritmů je na OSVM a NON-OSVM. Metody se dělí na základě využívání/nevyužívání podpůrného vektoru dat. Základem metody OSVM je lineární klasifikátor do dvou tříd. 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, že hodnota minima vzdáleností bodů od roviny je co největší v literatuře je tato hodnota označována jako maximální šířka. Na popis roviny stačí pouze body ležící na okraji tohoto pásma a těch je málo – tyto body se nazývají podpůrné vektory (v angličtině support vectors) a odtud tedy název metody. [1]



Obrázek 11: ukázka OSVM [10]

Posledním z kritérii, 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 klasifikace textu (email, dokumenty, textové zprávy …) a druhou skupinou jsou klasifikátory pro obrazovou a další klasifikaci. [1]

2.2 Charakteristika unárních klasifikačních metod

Základní srovnání unárních metod můžeme provést za pomocí parametrů z matice záměny viz. Tab. 1. Tyto parametry definují, s jakou přesností pracuje daný klasifikátor, nicméně jednotlivé metody lze srovnávat i podle následujících charakteristik:

Robustnost vůči odlehlým hodnotám: Předpokládá se, že tréninková sada dat obsahuje cílové objekty. Může se ale stát, že v tréninkových datech jsou obsaženy vzdálené hodnoty. Cílem je natrénovat model na cílové objekty a tyto vzdálené objekty vyloučit. U metod které jsou optimalizovány na základě podobnosti nebo vzdálenosti vůči tréninkové sadě jsou objekty poblíž prahu klasifikační funkce kandidát-my na odlehlé hodnoty. Zatím co u metod které jsou optimalizovány pouze na danou prahovou hodnotu je nutné prověřit odlehlé hodnoty v tréninkové sadě jestli se opravdu jedná o odlehlé hodnoty a popřípadě je s tréninkové sady vyloučit. [1]

Začlenění známých odlehlých hodnot: Jsou-li k dispozici některé odlehlé objekty (u kterých víme že se jedná o odlehlé hodnoty s naprostou jistotou), mohou být použity k dalšímu zpřísnění popisu. Aby mohly být tyto informace začleněny do metody, musí být model flexibilní. Například když se jako model cílové množiny použije jedna Gaussova distribuce, model a tréninkový postup nejsou dostatečně flexibilní, aby odmítly jediný odlehlý stav. Jiné metody na druhou stranu, například hustota Parzen, mohou tento odlehlý bod začlenit do odhadu pravděpodobnosti a zpřesnit si tak svoji přesnost odhadu. V neposlední řadě by měla být možné přidat parametr do unárního klasifikátoru, který reguluje kompromis mezi chybou cíle a odlehlé hodnoty. [1]

Parametry a snadná konfigurace: Jedním z nejdůležitějších aspektů pro snadné ovládání metody uživatelem je počet volných parametrů, který je třeba předem zvolit, a také jejich počáteční hodnoty. Pokud je zahrnuto velké množství volných parametrů, může být nalezení vhodného modelu velmi obtížné. Tento problém se stává ještě výraznějším, když příslušné parametry nejsou intuitivní veličiny, které nelze tedy apriori předpokládat, odvodit nebo odhadnout. Při správném nastavení těchto parametrů bude dosaženo dobrých výkonů, ale při nesprávném nastavení může metoda zcela selhat. Tyto parametry se často nazývají „magic parameters“, protože mají často velký vliv na konečný výkon a nejsou stanovena žádná jasná pravidla, jak je nastavit. Například při tréninku neuronových sítí musí uživatel zvolit počet skrytých vrstev a počet neuronů na vrstvu. Tato čísla nelze předem intuitivně určit a pouze pokusem a omylem lze zjistí vhodnou velikost sítě. Jiné metody, například SVDD, předem vyžadují hodnotu pro f_T. Toto číslo přímo souvisí s problémem, který se uživatel pokouší vyřešit, a je tak snazší ji nastavit. [1]

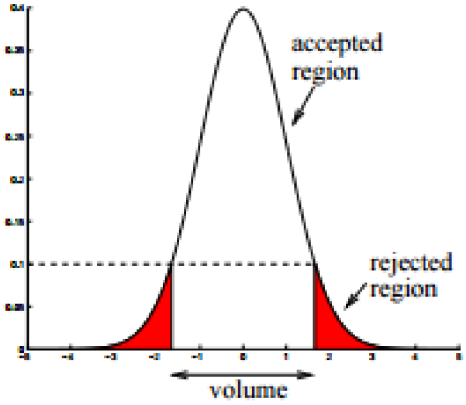
Rychlost výpočtu a požadavky na velikost paměti: Některé z metod pro svoje fungování vyžadují relativně dlouhý výpočetní proces, který zabraňuje jejich použití v určitých aplikacích. Vzhledem k tomu že školení jednotlivých klasifikátorů většinou probíhá off-line náročnost na paměť zde často odpadá. [1]

2.3 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 Parzenovu hustotu. Když je velikost tréninkových dat dostatečně vysoká 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 vykazují tyto metody velice dobré výsledky. [2]

2.3.1 Gaussův model

Nejjednodušším z modelů je normální nebo Gaussovo rozložení hustoty. Z tohoto předpokladu lze vycházet za použití centrálního limitního teorému a lze stanovit že objekty z jedné třídy dat mají podobnou hustotu rozložení a zároveň obsahují malý počet nezávislých chybových dat. Cílem metody je nalezení prahové hodnoty.[1]



Obrázek 12: Gaussův model [2]

Pravděpodobnost distribuce pro d-dimenzionální objekt x je dána vztahem:

[2](2.1)

$$p_{\mathcal{N}}(\mathbf{z};\boldsymbol{\mu},\Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{z}-\boldsymbol{\mu})\right\}$$

Počet volných parametrů v Gaussově modelu je tedy:

$$n_{\text{free}\mathcal{N}} = d + \frac{1}{2}d(d-1)$$ [2](2.2)

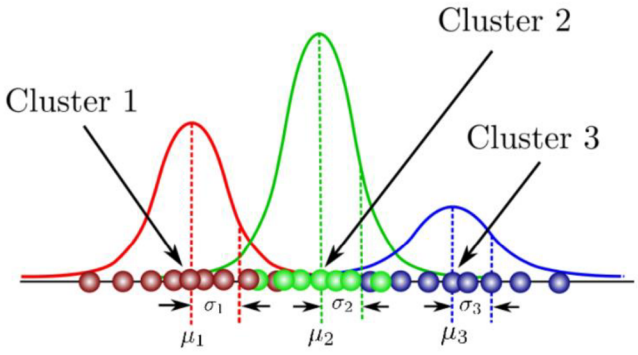
2.3.2 Směs Gaussianů

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. Kdy směs Gaussianů je lineární kombinací normálních distribucí [2]:

$$p_{MoG}(\mathbf{x}) = \frac{1}{N_{MoG}} \sum_j \alpha_j p_{\mathcal{N}}(\mathbf{x};\boldsymbol{\mu}_j,\Sigma_j)$$ [2] (2.3)

Celkový počet volných parametrů ve směsi Gaussianů je:

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



Obrázek 13: Mixture of Gausian [5]

Na obrázku 13 lze vidět že pro jednotlivé tréninkové sady jsou určeny odpovídající funkce hustoty na základě rozložení jejich dat, ve výsledku je potom celková hustota určena vhodnou kombinací jednotlivých Gaussianů podle směšovacích koeficientů. [2]

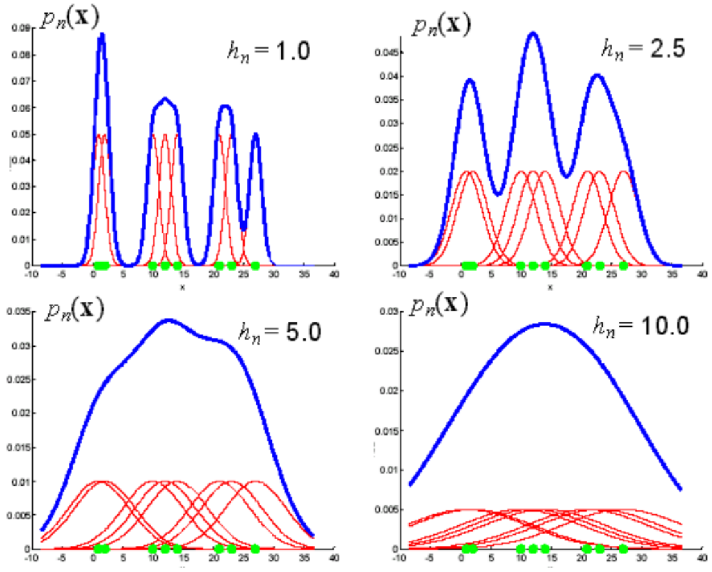
2.3.3 *Odhad hustoty Parzen*

Třetí metodou je odhad hustoty Parzen, který je opět rozšířením předchozí metody. Odhadovaná hustota je směsí nejčastějších Gaussovských jader soustředěných na jednotlivé tréninkové objekty s (často) diagonálními kovariantními maticemi Σi = hI. [2]

$$p_p(\mathbf{x}) = \frac{1}{N}\sum_i p_{\mathcal{N}}(\mathbf{x}; \mathbf{x}_i, h\mathcal{I}) \qquad \text{[2] (2.5)}$$

Stejná šířka h v každém směru prvku znamená, že Parzenův odhad hustoty předpokládá stejně vážené objekty v tréninkové sadě, a proto bude citlivý na změnu velikosti hodnot objektů trénovací sady, zejména pro menší velikosti vzorků. Trénink metody Parzen se skládá z určení jediného parametru, optimální šířky jádra h. Tím pádem je počet volných parametrů roven 1. [2]

$$n_{freep} = 1 \qquad \text{[2] (2.6)}$$



Obrázek 14: vliv parametru h na Parzenův odhad [6]

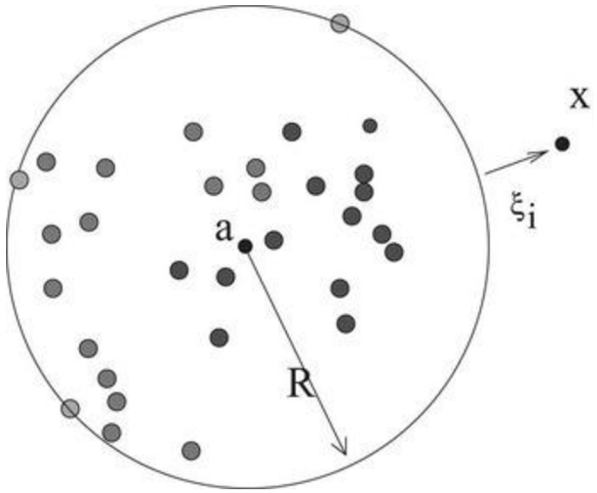
Jak je vidět na obrázku 14 je výsledná přesnost odhadu přímo závislá na přesném stanovení parametru h. Tím že přesnost modelu závisí pouze na jednom parametrů je velice snadné tento parametr nastavit. [3]

2.4 Hraniční metody

Tyto hraniční metody se zaměřují na případy kdy je k dispozici pouze omezený počet dat v tréninkové sadě. Cílem tedy není odhadnou celou hustotu rozložení ale pouze hranici oddělující cílové a odlehlé objekty. Tato práce je zaměřena na metody SVDD, k-mean a NN-d hraniční metody. Všechny tyto metody pracují na podobné principu tedy na určení vzdálenosti mezi jednotlivými objekty v tréninkové sadě. Tento postup je značně citlivý na změnu velikosti ale na druhou stranu vyžaduje mnohem méně dat v tréninkovém datasetu než metody hustoty. Výstupem těchto metod už není pravděpodobnost. Výstupem je v tomto případě funkce, která dokáže zachytit většinu z cílových objektů. [2]

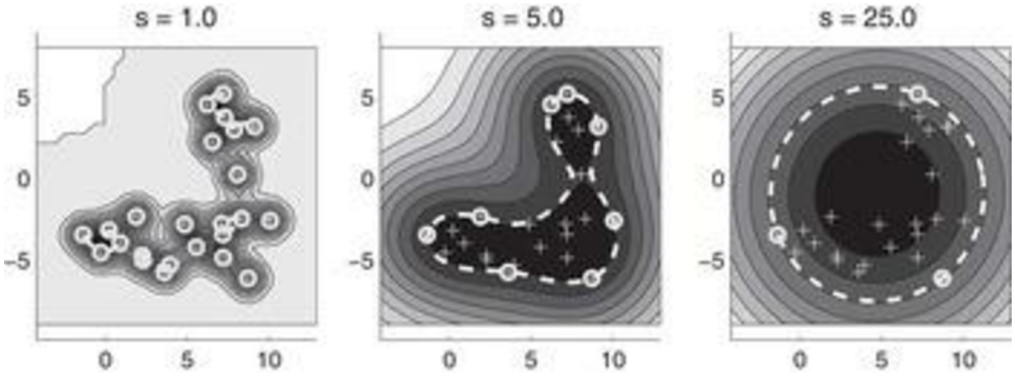
2.4.1 SVDD „Support Vector Data Description“ klasifikátory

Klasifikátor SVDD odmítne daný testovací bod jako odlehlý, pokud spadne mimo hyper-sféru. SVDD však může odmítnout určitou část pozitivně označených dat, když se objem hyper-sféry sníží. Hyper-sférický model SVDD může být flexibilnější zavedením funkcí jádra. Zvažujeme-li polynomiální a Gaussovo jádro a zjistíme, že Gaussovo jádro funguje lépe pro většinu uvažovaných datových souborů. Lze také používat různé hodnoty šířky jádra. Čím větší je šířka jádra, tím méně podpůrných vektorů je vybráno a popis se stává sférickějším. Použití Gaussovského jádra místo polynomiálního vede k přísnějším popisům, ale vyžaduje více dat, aby podporovalo pružnější hranice. Metoda nefunguje dobře, když existují velké rozdíly v hustotě mezi objekty pozitivní třídy; v takovém případě začne odmítat cílové body s nízkou hustotou jako odlehlé hodnoty. [1]



Obrázek 15: SVDD metoda [1]

Na obrázku vidíme SVDD klasifikátor s poloměrem R se středem v bodě a. Objekt X je zde definován jako odlehlá hodnota. Tyto odlehlé hodnoty potom tvoří chyby při klasifikaci pomocí SVDD. Takových to hodnot je možné se zbavit pokud je zahrneme do klasifikované sféry za pomocí změny poloměru nebo za pomocí vytvoření druhé klasifikační sféry. [2]



Obrázek 16: Změna poloměru SVDD metody [2]

Na obrázku 16 jsou data rozložena do oblého tvaru to zabraňuje jejich snadné klasifikaci. Na obrázku 16 v jednotlivých krocích 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, jestli stojíme o rychlejší klasifikaci za pomocí jedné sféry nebo o kvalitnější klasifikaci za pomocí několika sfér. Takováto pomocná sférická data se nazývají Gaussovo jádro. Zvětšováním poloměru těchto jader se stává sféričtější a je potřeba méně těchto jader. Pro vyhodnocení těchto klasifikátorů se používají binární funkce, které vrací 1 uvnitř sfér a -1 pro body mimo které nejsou do sfér zahrnuty. [2]

Pro výpočet chyby SVDD metod potom lze použít podmínku:

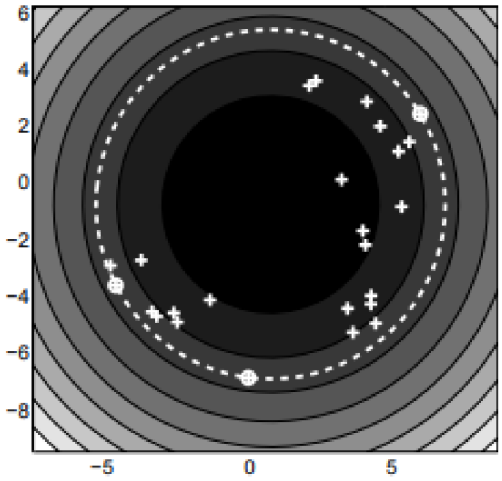
$$\mathcal{E}_{\text{struct}}(R, \mathbf{a}) = R^2 \qquad \text{[2] (2.7)}$$

která musí být minimalizována omezeními:

$$\|\mathbf{x}_i - \mathbf{a}\|^2 \leq R^2, \qquad \text{[2] (2.8)}$$

2.4.2 Trénování SVDD metod na negativních datech

Pokud jsou k dispozici negativní příklady (objekty, které by měly být odmítnuty odlehlé hodnoty), lze je použít během tréninku na vylepšení popisu (tj. získat přísnější hranici kolem dat v oblastech, kde jsou odlehlé objekty). Předpokládejme, že trénujeme SVDD na příkladech dvou tříd. Jedna z tříd je považována za cílovou třídu (a tedy druhá třída je odlehlá třída). Je možné trénovat na těchto datech jak SVDD, tak tradiční (dvoutřídní nebo multi-class) klasifikátor. SVDD se liší od konvenčního klasifikátoru, tím že SVDD splňuje omezení, že vždy získá uzavřenou hranici kolem jedné z tříd (klasifikační funkce je sféra kolem cílových dat). Kromě toho nevyžaduje přísný reprezentativní vzorek cílových dat. To je výslovně uvedeno v definici chyby SVDD. [2] Konvenční klasifikátor na druhé straně rozlišuje mezi dvěma (nebo více) třídami bez zvláštního zaměření na některou ze tříd. Klasifikátor minimalizuje pravděpodobnost chyby klasifikace. Očekává se, že konvenční klasifikátor bude fungovat velmi špatně, když je k dispozici jen několik odlehlých příkladů a odlehlá třída je extrémně pod vzorkovaná. Pokud je k dispozici velký vzorek z cílové třídy, tak velké množství odlehlých hodnot v takovém případě bude konvenční klasifikátor fungovat lépe než SVDD. SVDD je pak omezeno omezením uzavřené hranice kolem dat. Volba mezi SVDD a běžným klasifikátorem je proto ovlivněna množstvím odlehlých objektů a jejich dostupnosti pro trénink. Tím že při trénování použijeme odlehlá data dojde k výraznému zvýšení kvality klasifikace. [2]

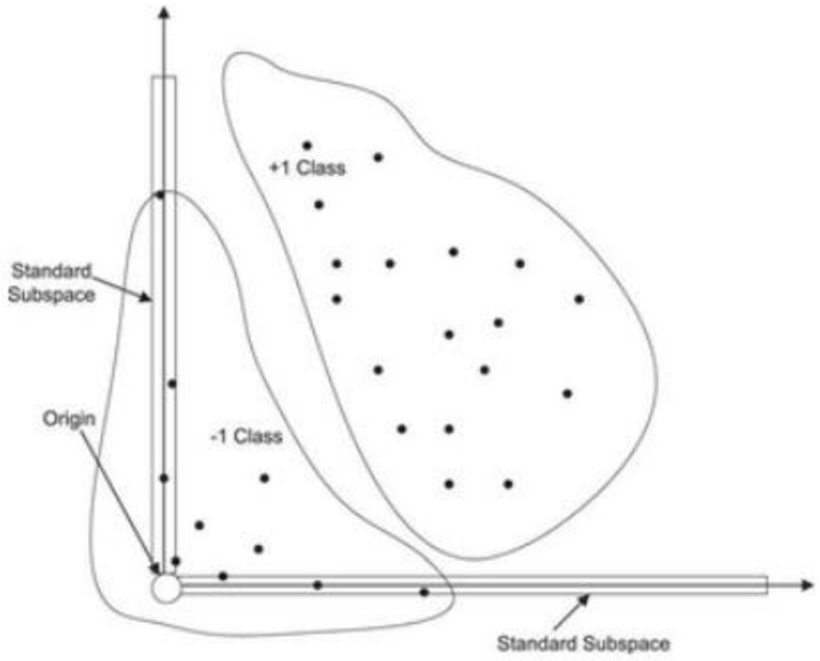


Obrázek 17: Příklad SVDD metody s negativními daty [2]

Na obrázku 17 jsou označeny cílová data (křížky) a oblehla data (bílé body). V tomto případu je model sestrojena na základě odlehlých hodnot a to tak aby poloměr sféry cílové oblasti byl co největší a zároveň byl ohraničen odlehlými daty.

2.4.3 Další varianty SVDD metod

Další s SVDD alternativních metod je Schölkopf metoda konstruující hyper-rovinu na místo hyper sféry kolem dat a to tak že hyper-rovina je maximálně vzdálená od počátku a může oddělit oblasti, které neobsahují žádná data. Tato funkce používá binární funkci, která vrací +1 v „malé“ oblasti obsahující data a –1 jinde. Zavádějí proměnnou, která řídí účinek odlehlých hodnot, tj. tvrdost nebo měkkost hranice kolem dat. Schölkopf navrhuje použití různých jader v této metodě, což odpovídá řadě nelineárních odhadů. V praktických implementacích je metoda Schölkopf a metoda SVDD srovnatelná a fungují stejně, a obě fungují nejlépe, když se použije Gaussovo jádro. Další z možných modifikací SVDD je Manevitz a Yousef, tato metoda zkoumá použití OSVM pro získávání informací. Tato metoda navrhuje jinou verzi než kterou navrhl Schölkopf. Metoda Manevitze a Yousefa je založena na identifikaci odlehlých dat, která jsou reprezentativní pro druhou třídu. Myšlenkou jejich metodiky je pracovat nejprve v prostoru funkcí a předpokládat, že nejen počátek je členem třídy odlehlých hodnot, ale také všechny datové body blízké počátku jsou považovány za šum nebo odlehlé hodnoty. Geometricky řečeno vektory které leží na podprostoru malé dimenze, tj. osách, plochách atd., jsou považovány za odlehlé hodnoty. Pokud má tedy vektor několik nenulových položek, znamená to, že datový objekt sdílí s vybranou podmnožinou databáze velmi málo položek a bude s ním zacházeno jako s odlehlou hodnotou.[1]



Obrázek 18: upravená OSVM (Metoda Manevitze a Yousefa) [1]

2.4.4 K-centers

Tato metoda pokrývá trénovací dataset malými sférickými oblastmi se stejným průměrem. Tato metoda se používá pro popis složitosti trénovací sady. Středy jednotlivých sfér jsou umisťovány tak aby maximum všech minim vzdáleností mezi trénovacími objekty a středy jednotlivých sfér byla co nejmenší, výsledná chyba je potom minimální [2]:

$$\mathcal{E}_{\text{k-centr}} = \max_i \left(\min_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 \right) \qquad \text{[2] (2.9)}$$

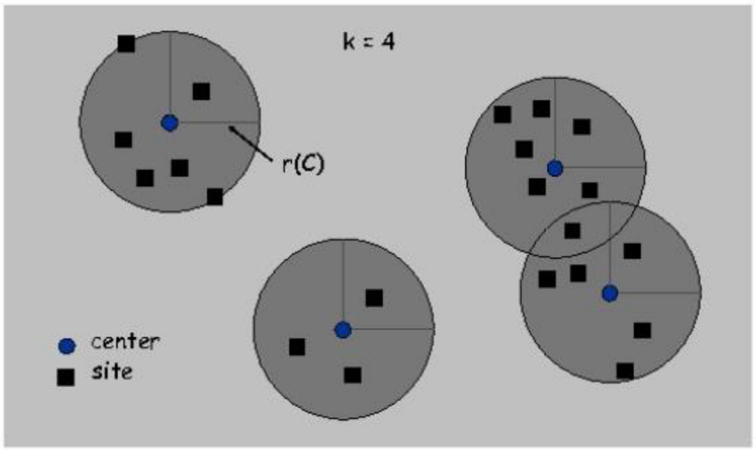
Metoda k-centers používá strategii dopředného vyhledávání, která začíná náhodnou inicializací. Poloměr je určen maximální vzdáleností k objektům, které by měla příslušná sféra zachytit. Potom je možné určit vzdálenost testovaného objektu Z k objektu z trénovací sady, tato vzdálenost je definována jako [3]:

$$d_{k-centr}(\mathbf{z}) = \min_k \|\mathbf{z} - \boldsymbol{\mu}_k\|^2 \qquad \text{[2] (2.10)}$$

Abychom se během tréninku vyhnuli neoptimálnímu řešení, vyzkouší se několik náhodných inicializací a použije se nejlepší řešení (to s nejmenší chybou $E_{k\text{-centr}}$). Počet volných parametrů je tedy [3]:

$$n_{free k-c} = k \qquad \text{[2] (2.11)}$$

Uživatel musí zadat počet sfér k i maximální počet opakování pro nalezení optimálního minima.

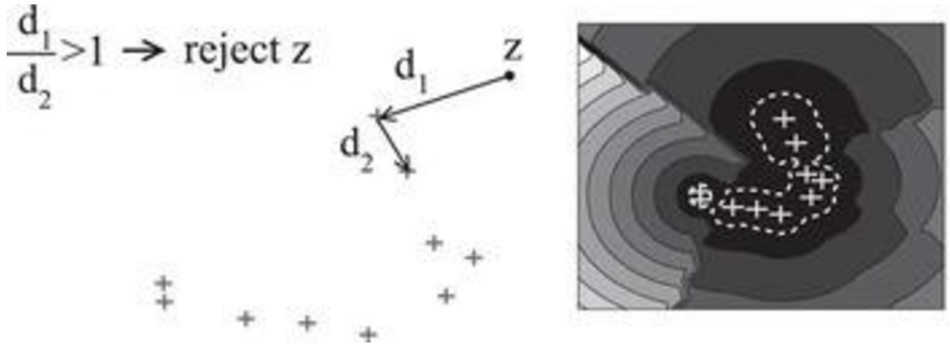


Obrázek 19: k-centers [4]

Na obrázku 19 je potom vidět výsledek této metody pro čtyři zvolené sféry. Je patrné že středy jednotlivých sfér jsou umístěny tak aby vzdálenost ke všem příslušným objektům byla co nejmenší.

2.4.5 Metoda nejbližšího souseda

Další z metod je metoda nejbližšího souseda, NN-d. Může být odvozen 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 hypersféra v dimenzích d, soustředěna kolem testovaného objektu Z. Objem této buňky narůstá, dokud nezachytí k objektů z tréninkové sady. Místní hustota se pak odhaduje podle [3]:



Obrázek 20: Metoda nejbližšího souseda [2]

$$p_{\mathrm{NN}}(\mathbf{z}) = \frac{k/N}{V_k(\|\mathbf{z} - \mathrm{NN}_k^{tr}(\mathbf{z})\|)}$$

[2] (2.12)

U unární klasifikace je testovaný objekt Z přijat pokud je lokální hustota větší nebo rovna hustotě nejbližšímu sousedovy z trénovací sady. Pro lokální hustotu k=1 potom tedy [2]:

[2] (2.13)

$$f_{\mathrm{NN}^{tr}}(\mathbf{z}) = I\left(\frac{V(\|\mathbf{z} - \mathrm{NN}^{tr}(\mathbf{z})\|)}{V(\|\mathrm{NN}^{tr}(\mathbf{z}) - \mathrm{NN}^{tr}(\mathrm{NN}^{tr}(\mathbf{z}))\|)} \leq 1\right)$$

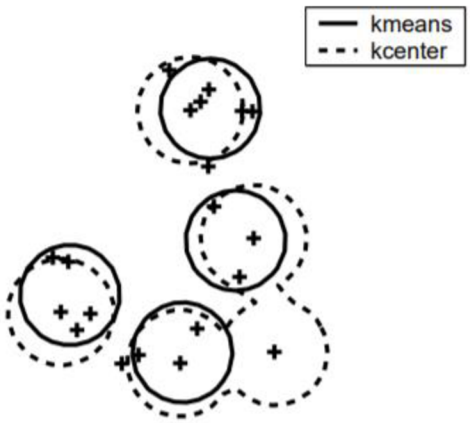
Jednoduchou úpravou této metody potom dostáváme metodu Local Outlier Factor (LOF), která je zkonstruována k hledání odlehlých hodnot ve velkých databázích. U této metody jsou všechny vzdálenosti k nejbližšímu sousedovy zprůměrovány a dále vzdálenost objektu Z k jeho k nejbližším sousedům je nahrazena robustnější definicí vzdálenosti. Když jsou objekty velmi blízko cílových dat, použije se vzdálenost k-tého nejbližšího souseda místo vzdálenost prvního nejbližšího souseda. 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 K, kteří jsou používání a to v rozsahu od 10 do 50. Ačkoli je metoda LOF 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 odmítnut). [2]

2.5 Rekonstrukční metody

Rekonstrukční metody nejsou primárně tvořeny pro unární klasifikaci ale spíše pro modelování dat. Využitím znalostí o datech a jejich generování lze vybrat model a následně tento model těmto datům přizpůsobit. Nyní lze popsat data vzhledem k modifikovanému modelu. Cílem tohoto postupu je snaha o dosažení kompaktnějšího popisu dat a snížení šumu a chyb v datech. 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. [2]

2.5.1 K-means

Metoda shlukování k-means se podobá metodě K-center, ale důležitým rozdílem je chyba, která je minimalizována. Metoda K-center 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-means jsou vzdálenosti k všech objektů zprůměrovány, a proto je metoda odolnější vůči vzdáleným odlehlým hodnotám. Navíc v metodě K-center jsou středy umístěny podle definice na některé z tréninkových objektů, zatímco v metodě k-means jsou všechny středové pozice volné. [2]



Obrázek 21: srovnání k-means a K-center [2]

Umístění center sfér oběma metodami je velmi podobné. Přesné polohy jsou určeny extrémními objekty pro K-center a pomocí různých datových klastrů pro k-means. Největší rozdíl spočívá v tom, že metoda K-center umístí na objekt v pravé dolní části datové sady sféru, zatímco metoda k-means ji považuje za odlehlou hodnotu. [2]

Chyba metody K-means je potom stanovena jako:

$$\mathcal{E}_{\text{k-m}} = \sum_i \left(\min_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 \right)$$

[2] (2.14)

Vzdálenost $d_{k\text{-}m}$ objektu z cílové sady je poté definována jako čtvercová vzdálenost daného objektu k nejbližšímu středu:

$$d_{k-m}(\mathbf{z}) = \min_k \|\mathbf{z} - \boldsymbol{\mu}_k\|^2$$

[2] (2.15)

2.5.2 LVQ

Algoritmus LVQ je super-vizovaná verze klastrování k-means a používá se hlavně pro klasifikační úkoly. Pro každý z tréninkových objektů xi je k dispozici štítek yi označující, ke kterému klastru by měl patřit. LVQ je trénován jako konvenční neuron síť, s výjimkou, že každá skrytá jednotka je prototyp, kde pro každý prototyp μk je definován štítek třídy yk. Cvičný algoritmus aktualizuje pouze klastr nejblíž cvičnému objektu xi:

$$\Delta\boldsymbol{\mu}_k = \begin{cases} +\eta\,(\mathbf{x}_i - \boldsymbol{\mu}_k) & \text{if } y_i = y_k \\ -\eta\,(\mathbf{x}_i - \boldsymbol{\mu}_k) & \text{otherwise} \end{cases}$$

[2] (2.16)

Toto pravidlo aktualizace je iterováno přes všechny tréninkové objekty, dokud není dosaženo konvergence. Je-li k dispozici pouze jedna třída objektů, jedná se o přímou derivaci chybové funkce. [2]

V klasifikování k-means i v LVQ musí být počet potřebných sfér odhadnut. Proto se počet volných parametrů změní na:

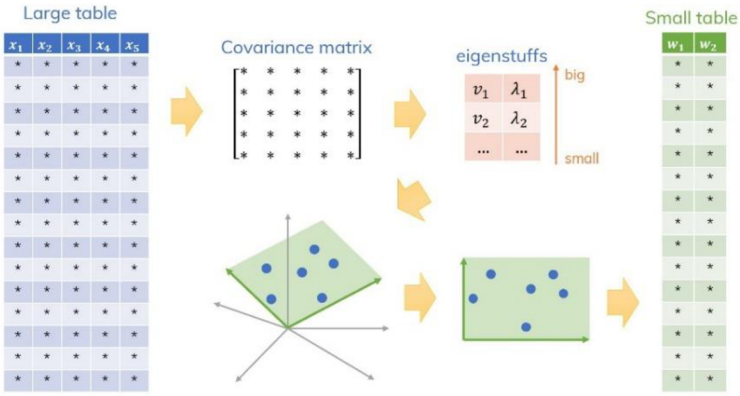
$$n_{\text{free}k-m} = n_{\text{free}LVQ} = kd$$

[2] (2.17)

V metodě LVQ i v metodě k-means musí uživatel zadat počet sfér K a v metoda LVQ dále vyžaduje rychlost učení η.

2.5.3 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. [2]



Obrázek 22: PCA algoritmus [3]

Nejjednodušší postup optimalizace používá k výpočtu vlastních vektorů cílové kovariantní matice a rozklad vlastních čísel. 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. Počet bazických vektorů M je optimalizován tak, aby vysvětlil určitý, uživatelem definované, zlomek rozptylu v datech (uživatel si zde musí stanovit jaké procento dat má být vysvětleno, pokud daný vektor dokáže toto splnit lze vstupní data zmenšit a zanedbat tak data která se nepodařilo vysvětlit). Základní vektory W je tvořen maticí d × M. Vzhledem k tomu, že z ortonormálního základu se počet volných parametrů v PCA stává [2]:

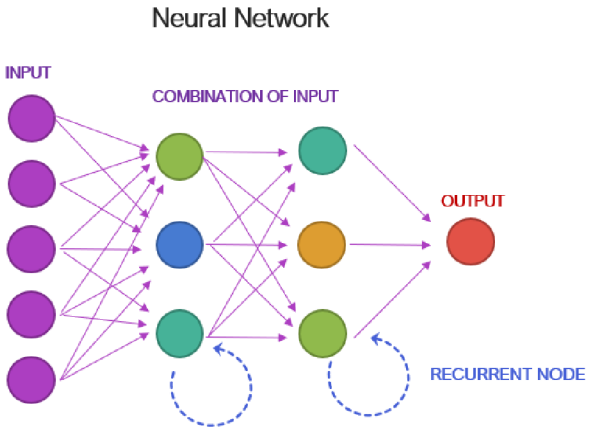
$$n_{\text{free}\,PCA} = \binom{d-1}{M} \qquad \text{[2] (2.18)}$$

PCA funguje dobře pokud je k dispozici jasně definovaný lineární prostor v kterém jsou obsažena data. Jeho výhodou 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 vytvoří průměrný podprostor, který může velmi špatně představovat data v každém z podprostoru. PCA je relativně citlivý na změnu velikosti funkcí, 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 není PCA schopno zahrnout do tréninku negativní příklady. [2]

2.6 Neuronové sítě

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. [9]

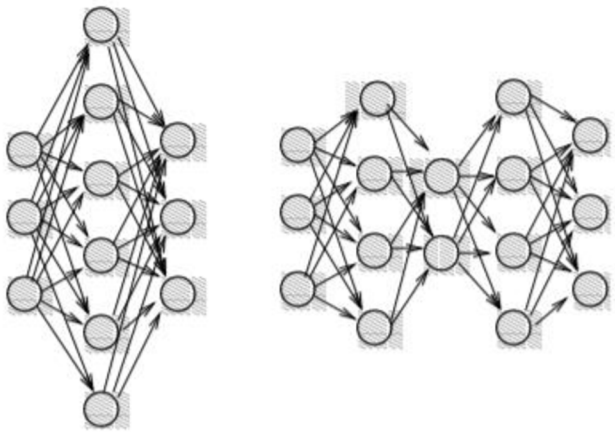


Obrázek 23: klasifikace za pomocí neuronové sítě [9]

Neuronové sítě jsou používány pro klasifikaci z důvodů zlepšení klasifikačních výsledků po přidání skryté vrstvy neuronové sítě. Většinou je volena tří úrovňová neuronová síť s M výstupními neurony a s K skrytými neurony, kde K<M. Neuronová síť je trénována pomocí standardního algoritmu zpětného šíření aby se naučila klasifikační funkci na pozitivních příkladech. Takto navržená neuronová síť se může naučit klasifikovat data která jsou podobná datům která byla použita pro učení. Neuronová síť potom označí tato data jako cílová zatím co všechna ostatní jsou označena za odlehlá. Varianta dopředné neuronové sítě se může učit jak na pozitivních tak negativních datech. V konvenčním dopředném binárním klasifikátoru 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 obvykle používanou k rozhodnutí do které třídy patří neznámý vzorek. V tomto případě, protože neoznačená data mohou obsahovat některé neoznačené pozitivní příklady, může být výstup trénované neuronové sítě menší nebo roven skutečné pravděpodobnosti, že příklad patří do pozitivní třídy. Pokud se předpokládá, že označené pozitivní příklady adekvátně představují pozitivní koncept, lze předpokládat, že neurální síť bude schopna nakreslit hranici třídy mezi negativními a pozitivními příklady. [9]

Auto-Encoders a Diabolo networks

Jedná se o variant klasických klasifikátoru s neuronovou sítí. Obě metody jsou trénovány pro reprodukci vstupních vzorů na jejich výstupní vrstvě (takže by měly provádět operaci klasifikace). Budeme rozlišovat mezi autoencoderem a diabolo sítěmi podle počtu skrytých vrstev a velikostí jednotlivých vrstev. V architektuře autoencoder je přítomna pouze jedna skrytá vrstva s velkým počtem skrytých jednotek. V síti diabolo se používají tři skryté vrstvy s nelineárními sigmoidními přenosovou funkcí a druhá vrstva obsahuje velmi nízký počet skrytých jednotek. Proto se tomu říká úzká vrstva. Další dvě vrstvy mohou mít libovolný počet skrytých jednotek (větší než ve vrstvě úzkého hrdla). [3]



Obrázek 24: Auto encoders a Diabolo networks [2]

Doufáme, že cílové objekty budou rekonstruovány s menší chybou než odlehlé objekty. Vzdálenost mezi původním objektem a rekonstruovaným objektem je pak měřítkem vzdálenosti objektu od tréninkové sady [3]:

[2] (2.19)

$$d_{diab}(\mathbf{z}) \quad = \quad \|f_{diab}(\mathbf{z};\mathbf{w}) - \mathbf{z}\|^2$$

Když se v síti autoencoderu použije jen jedna skrytá vrstva, je nalezen (lineární) typ řešení hlavní komponenty. To znamená, že síť autoencoderu 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 na cílové sadě 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. Když se velikost tohoto podprostoru shoduje s podprostorem v původních datech, může síť diabolo dokonale odmítnout objekty, které nejsou v cílovém datovém podprostoru. Když je podprostor stejně velký jako původní prostor funkcí, nelze očekávat žádný rozdíl mezi cílovými a odlehlými daty.

V aplikaci autoencoderu a sítí diabolo jsou stejné problémy jako při konvenční aplikaci neuronových sítí na klasifikační problémy. Metody jsou velmi flexibilní, ale vyžadují předem definovaný počet vrstev a neuronů, rychlosti učení a kritérium zastavení od uživatele. Výhoda těchto modelů je, že mohou být optimalizovány na daný problém, a které tedy mohou získat velmi dobré výsledky. Počet volných parametrů může být velmi vysoký jak pro autoencodér, tak pro síť diabolo. Počet vstupních a výstupních neuronů je dán rozměrností d dat. Nejprve definujte h_{auto} jako počet skrytých jednotek v automatickém kodéru. Celkový počet volných parametrů v automatickém encodéru (včetně zkreslení) je [2]:

$$n_{\text{free}auto} = d \times h_{auto} + h_{auto} + h_{auto} \times d + d = (2d + 1)h_{auto} + d \quad \text{[2] (2.20)}$$

Pro síť diabolo je počet neuronů v zúžené vrstvě h_{diab} a počet neuronů v dalších skrytých vrstvách, které jsme určeny jako $2*h_{auto,}$ počet volných parametrů je:

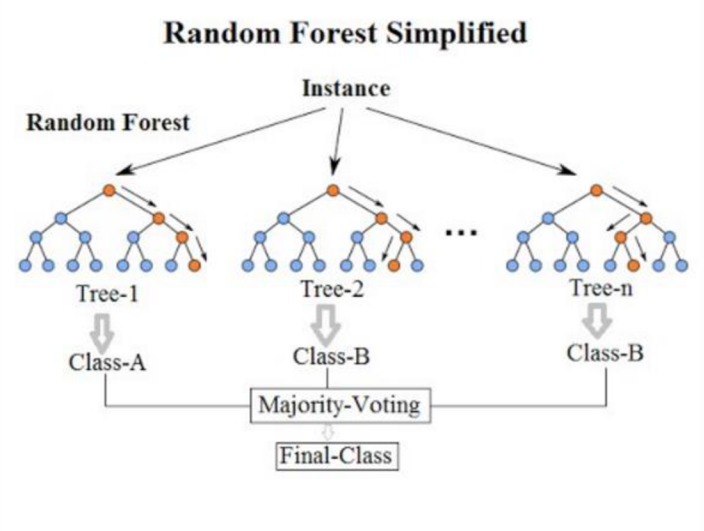
[2] (2.21)

$$n_{\text{free}diab} = d \times 2h_{diab} + 2h_{diab}(h_{diab} + 1) + h_{diab}(2h_{diab} + 1) + 2h_{diab}(d + 1) + d$$

$$n_{\text{free}diab} = h_{diab}(4d + 4h_{diab} + 5) + d$$

2.7 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 [3].



Obrázek 25: Ukázka rozhodovacího stromu [3]

Princip tvorby rozhodovacího stromu

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

1. Získání informací o uzlu
2. Rozhodni o uzlu, zda bude dál dělen nebo z něj udělej list a rozhodni o jeho výstupní hodnotě
3. Vyber nejlepší atribut 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 [3]

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 [3]

2.8 Srovnání metod unární klasifikace

V této části je uvedeno srovnání jednotlivých metod v závislosti na požadovaném počtu parametrů, volných parametrů a na citlivost jednotlivých metod.

method	scaling sensitivity	number of free parameters	number of user defined parameters
Density approaches			
normal density	–	$d + d(d+1)/2$	regularization λ
Mixture of Gaussians	+	$(d+2)N_{MoG}$	N_{MoG}, # iterations
Parzen density	+/–	1	0
Boundary approaches			
k-centers	+	k	k and # iterations
NN-d	+	0	0
SVDD	+	N	f_{SV}^{out}
SVDD-neg	+	N	f_{SV}^{out}
Reconstruction approaches			
k-means	+	kd	k, # iterations
PCA	–	$\binom{d-1}{M}$	fraction of preserved var.
mixture of PCAs	–	$\left(\binom{d-1}{M} + d\right) N_{MPCA} + 1$	dim. and # subspaces
SOM	+	$k^{d_{SOM}} d$	dim. and subspace size
auto-encoder	–	$(2d+1)h_{auto} + d$	# hidden units
diabolo network	–	$h_{diab}(4d + 4h_{diab} + 5) + d$	# hidden units, dim. subspace

Tabulka 2: Srovnání metod podle počtu parametrů [2]

Citlivost metod na změnu měřítka naznačuje, jak se metoda může přizpůsobit změně velikosti jedné z funkcí. Některé metody silně závisí na správně definované vzdálenosti mezi objekty, jako je NN-d, ale také další hraniční metody. Zatímco Gaussův model je možné přizpůsobit za použití kompletní kovariantní matice. První sloupec tabulky 2 udává jestli je daná metoda závislá na měřítko (+) nebo je nezávislá a lze ji optimalizovat (-). Počet volných parametrů (sloupec 2) potom udává flexibilitu metody a citlivost na přetrénování. Ve třetím sloupci tabulky 2 jsou parametry které musí uživatel zadat. Pro příklad si uveďme například metody NN-d a Perzen tyto metody nepotřebují žádný parametr který by bylo potřeba volit a závisí tak pouze na datech v trénovací sadě. Zatímco například metoda SVDD vyžaduje parametr fsv které udává jak velký počet hodnot v tréninkové sadě má být odmítnuto. Ve výsledku je potom možné z této tabulky stanovit které metody unární klasifikace závisí na tréninkové sadě a které metody si dokáži vytvořit nezávislý model. Ze sloupce 3 potom můžeme vyvodit které metody závisí na správném definování parametrů od uživatele. [2]

2.9 Srovnání robustnosti metod unárních klasifikátorů

V předchozích částech je uvedeno velké množství unárních klasifikátorů. Každý z nich má odlišné charakteristiky týkající se počtu parametrů, odolnosti vůči odlehlým hodnotám atd. V následujících dvou částech je uvedeno krátké zhodnocení jednotlivých metod, aby bylo možné získali lepší přehled toho, co lze očekávat od různých metod unární klasifikace. Důležitou charakteristikou klasifikátorů jedné třídy je jejich robustnost vůči několika odlehlým hodnotám v tréninkových datech.

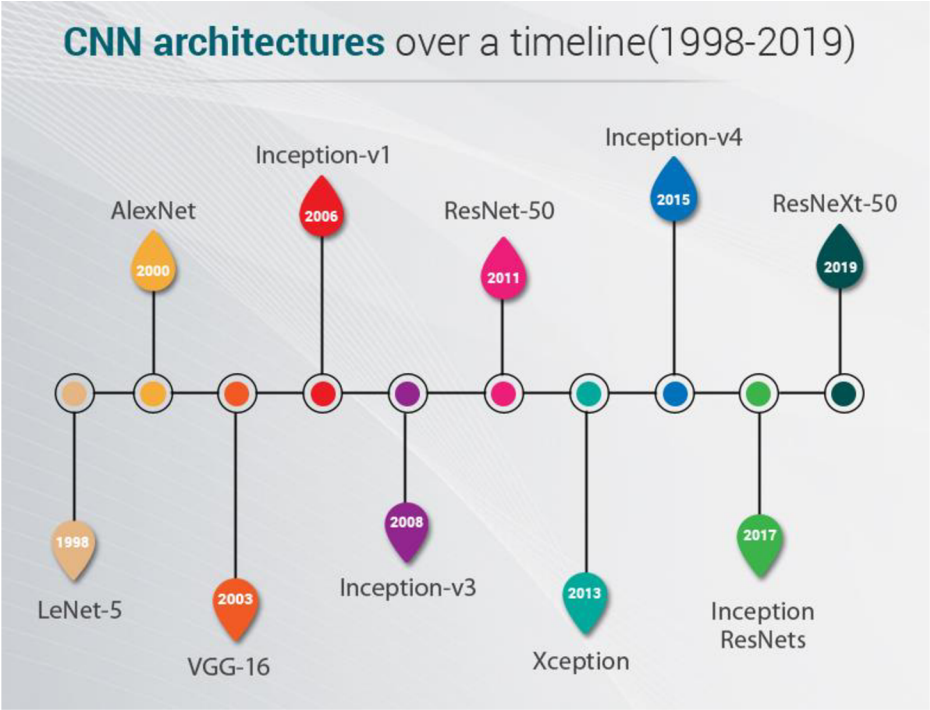
method	Unlabeled outliers in training		Labeled outliers in training	
Gaussian model	–	estimation of covariance matrix may suffer, when regularization is used, very large outliers can be rejected	–	outlier density should be modeled using few examples
mixture of Gaussians	–	estimation covariance matrix suffers	–	outlier density should be modeled using few examples
Parzen	+	density estimation only influenced locally	+	outlier density should be modeled.
k-centers	–	ball centers sensitive	–	outlier can obtain a ball, but overlapping balls give problems
NN-distance	–	distance quotient extremely sensitive	–	use a Nearest Neighbor Classifier
SVDD	+	a user defined fraction of the data can be rejected	+	outliers can be forced outside the description
LVQ	–	prototypes will be placed on or near outliers	+	in essence a classifier, it can repel from outliers
k-means	–	prototypes will be placed on or near outliers	–	in essence a clustering, cannot repel from outliers
PCA	–	estimation covariance matrix suffers from outliers	–	outlier density should be modeled using few examples
SOM	–	prototypes will be placed near outliers	+	repel from outliers
auto-encoder	–/+	depends on regularization strength	–/+	a bad representation should be forced
diabolo	–/+	depends on regularization strength	–/+	a bad representation should be forced

Tabulka 3: Srovnání robustnosti jednotlivých metod [2]

V tabulce 3 potom můžeme najít srovnání robustnosti jednotlivých metod unární klasifikace. Kdy v prvním sloupci je uveden typ metody, ve druhém a třetím sloupci je potom uvedeno, jestli si daná metoda dokáže poradit s odlehlým objektem v tréninkových datech, pokud není označen jako odlehlí objekt. Ve čtvrtém a pátém sloupci je potom zobrazeno, jestli si daná metoda dokáže poradit s odlehlým objektem v trénovacích datech, pokud je označen jako odlehlí objekt. [2]

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 systémy, klasifikace obrazu, segmentace obrazu, analýza lékařského obrazu, zpracování jazyka, řešení rozhraní mozek-počítač a analýza 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.[12][13]



Obrázek 26: vývoj CNN architektur [15]

Pro popis následujících architektur byly použity schémata viz. kapitoly níže. Pro popis je nutné definovat funkce jednotlivých funkční bloků. Obecná funkce jednotlivých bloků zůstává stejné ve všech uvedených případech.

3.1 Definování jednotlivých funkčních vrstev CNN architektur:

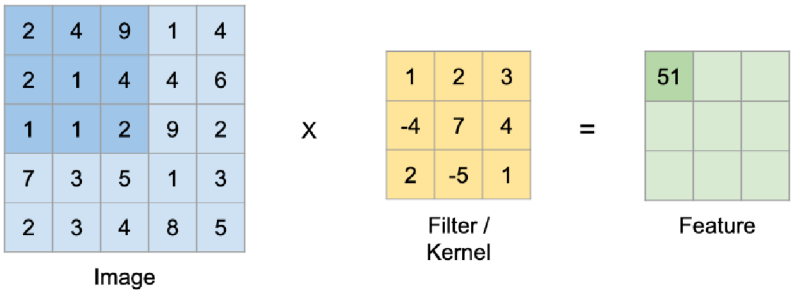
Každá CNN architektura se skládá z jednotlivých vrstev, tyto vrstvy mohou mít v různých architekturách rozdílné parametry nicméně základní funkce je ve všech případech stejná. Mezi nejtypičtější používané vrstvy je potom možné zařadit konvoluční vrstvu, sdružovací vrstvu a aktivační funkci.

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

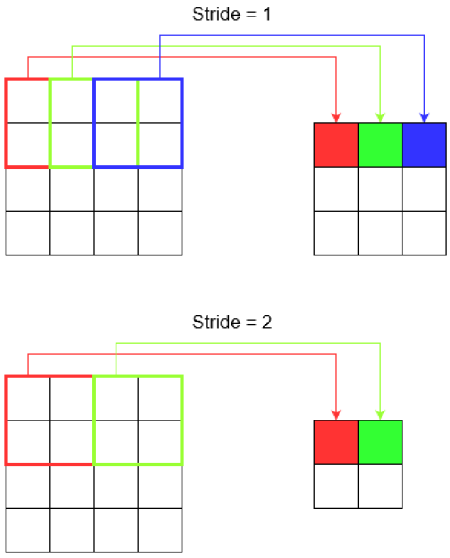
Tento parametr udává velikost konvolučního filtru, pro CNN architektury je zadáván ve čtvercové velikosti NxN, nejčastěji se používa ve velikostech 1x1, 3x3 a 5x5. Jak vidíme na obrázku 27, konvoluce se provádí postupně aplikací filtru na jednotlivé části vstupní mapy, následně je výsledek zapsán do 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.[15]



Obrázek 27: ukázka Kernel funkce [15]

Stride

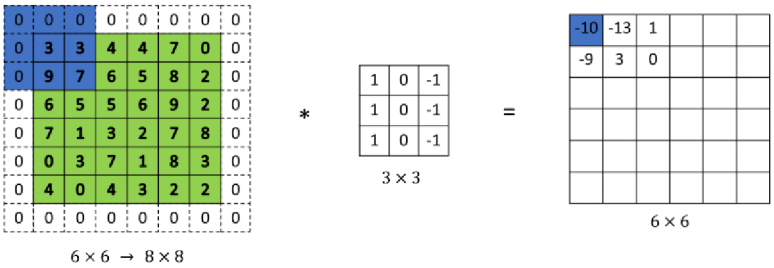
Stride parametr nebo také česky označovaný jako krok. Je parametr udávající velikost posunu konvolučního filtru mezi jednotlivými operacemi. Tato hodnota je primárně nastavena na jedna nicméně ne pro všechny operace je tato hodnota vhodná. Celkově dokáže tento parametr ovlivnit výstupní velikost mapy. Tento parametr je vhodné využít u hlubokých architektur kde za pomocí hlubší znalosti můžeme vyloučit případnou ztrátu dat způsobený kompresí výstupu.[15]



Obrázek 28: ukázka Stride funkce [15]

Padding

Česky označován jako polstrování je parametr umožňující manuální zvětšení vstupní mapy přidáním nulových hodnot na kraje mapy. Tento parametr je vhodné nastavovat u velmi hlubokých sítí kde hrozí velmi velké zmenšení mapy funkcí.[15]

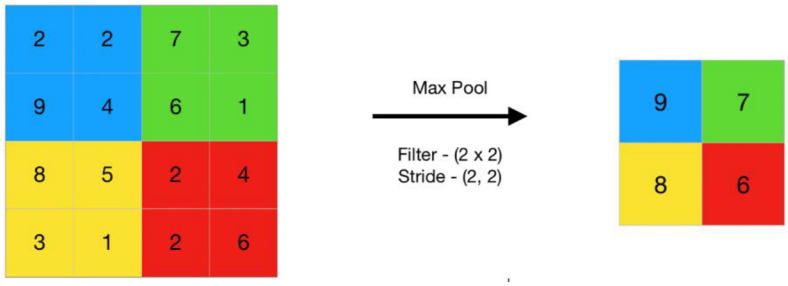


Obrázek 29: ukázka Padding funkce [15]

3.1.2 Pooling vrstva

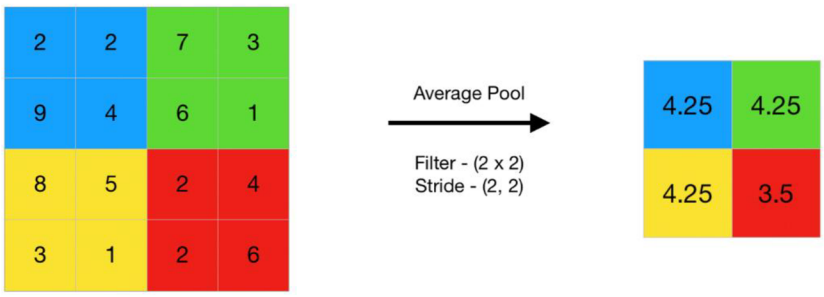
Česky označovaná jako sdružovací vrstva je v konvolučních sítích obecně používána pro zmenšení objemu dat a zrychlení výpočtu. Ke zmenšení dat dojde na základě filtrace vstupních dat filtrem dané velikosti, existují dva typy sdružování.

Max Pooling
Tato operace sdružování, vybírá maximální prvek z oblasti mapy prvků pokryté filtrem. Výstupem po vrstvě max-poolingu by tedy byla mapa funkcí obsahující nejvýznamnější prvky předchozí mapy funkcí. [16]



Obrázek 30: Max pooling [16]

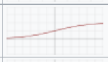
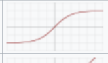
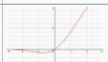
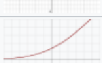
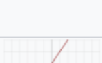
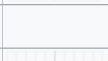
Average Pooling
Tato operce vypočítá průměr prvků přítomných v oblasti mapy prvků pokryté filtrem. Zatímco tedy maximální sdružování poskytuje nejvýznamnější prvek v konkrétní aktualizaci mapy funkcí, průměrné sdružování poskytuje průměr prvků přítomných v dané oblasti mapy prvků. [16]



Obrázek 31: average pooling [16]

3.1.3 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.[37]

Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$	Range	Order of continuity
Identity		x	1	$(-\infty, \infty)$	C^∞
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$ [1]	$f(x)(1-f(x))$	$(0, 1)$	C^∞
tanh		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$	$(-1, 1)$	C^∞
Rectified linear unit (ReLU)[7]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$	C^0
Gaussian Error Linear Unit (GELU)[4]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$	$(-0.17\ldots, \infty)$	C^∞
Softplus[8]		$\ln(1 + e^x)$	$\frac{1}{1+e^{-x}}$	$(0, \infty)$	C^∞
Exponential linear unit (ELU)[9]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C^1 & \text{if } \alpha = 1 \\ C^0 & \text{otherwise} \end{cases}$
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 $\alpha = 1.67326$	$\lambda\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\lambda\alpha, \infty)$	C^0
Leaky rectified linear unit (Leaky ReLU)[11]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
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	$\begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$[2]	C^0
Sigmoid linear unit (SiLU,[5] Sigmoid shrinkage,[13] SiL,[14] or Swish-1[15])		$\frac{x}{1+e^{-x}}$	$\frac{1 + e^{-x} + xe^{-x}}{(1+e^{-x})^2}$	$[-0.278\ldots, \infty)$	C^∞
Mish [16]		$x\tanh(\ln(1+e^x))$	$\frac{(e^x(4e^{2x} + e^{3x} + 4(1+x) + e^x(6+4x)))}{(2+2e^x+e^{2x})^2}$	$[-0.308\ldots, \infty)$	C^∞
Gaussian		e^{-x^2}	$-2xe^{-x^2}$	$(0, 1]$	C^∞

Obrázek 32: aktivační funkce [38]

Mezi nejvíce používané aktivační funkce v dnešních neuronových sítích lze zařadit sigoid, tanh, ReLu a leky ReLu.

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.[37]

Tanh

Funkce velice podobná funkci Sigmoid nicméně výstup je v rozsahu od -1 do 1, toto rozmezí má tu výhodu že záporné vstup jsou mapovány záporně zatímco nulové vstupy jsou mapovány nulou. Funkce Tanh je používaná hlavně při klasifikaci dvou tříd.[37]

ReLu

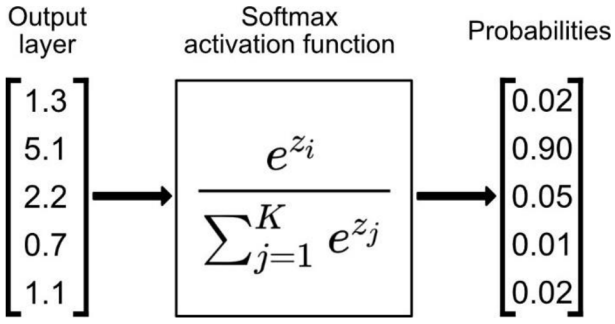
V současné době pravděpodobně nejpoužívanější aktivační funkce. Výstup funkce je potom od 0 po nekonečno s tím že všechny záporné vstupy jsou mapovány nulovou hodnotou.[37]

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.[37]

Softmax

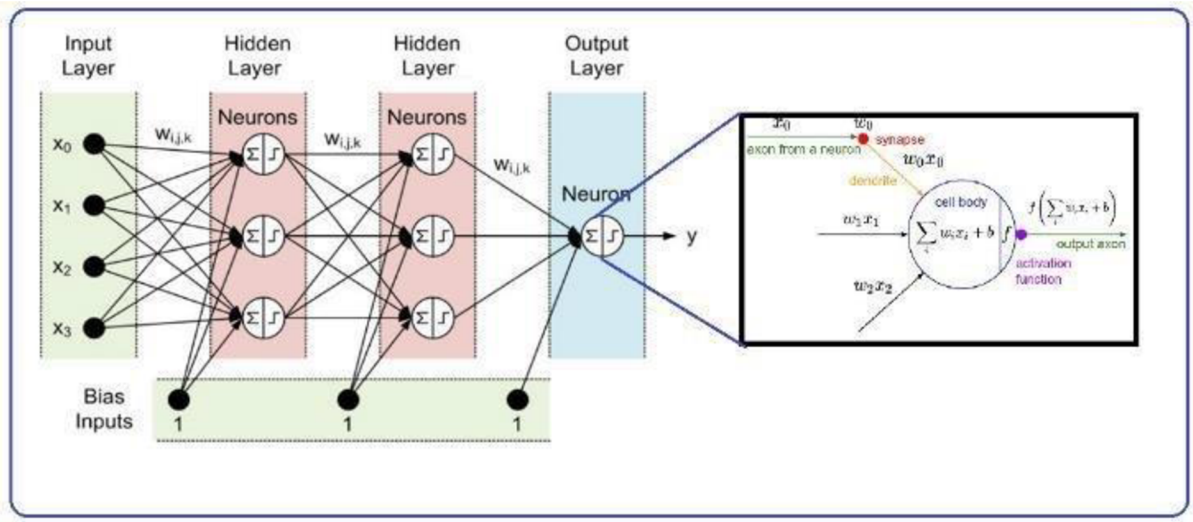
Poslední vrstva v CNN sítích je většinou tvořena funkcí softmax. Tato funkce má na starosti normalizaci výstupu. Tedy před aplikováním funkce softmax mohou některé vektorové komponenty nabývat hodnot větších než jedna nebo záporných hodnot, funkce softmax zajistí na výstupu hodnoty v rozmezí 0-1, takže je lze interpretovat jako hodnoty pravděpodobnosti. Tato funkce také zajistí že celkový součet složek výstupního vektoru je roven 1.[38]



Obrázek 33: Softmax aktivační funkce [35]

3.1.4 Plně připojená vrstva

Tento typ vrstvy slouží pro finální klasifikaci. Každá takováto vrstva se zakládá z určeného počtu neuronů. Neurony v plně propojené vrstvě mají spojení se všemi prvky v předchozí vrstvě. Tyto vrstvy jsou potom obecně zodpovědné za klasifikaci vstupních dat na základě vstupů z předchozí vrstvy a váhovacích funkcí.



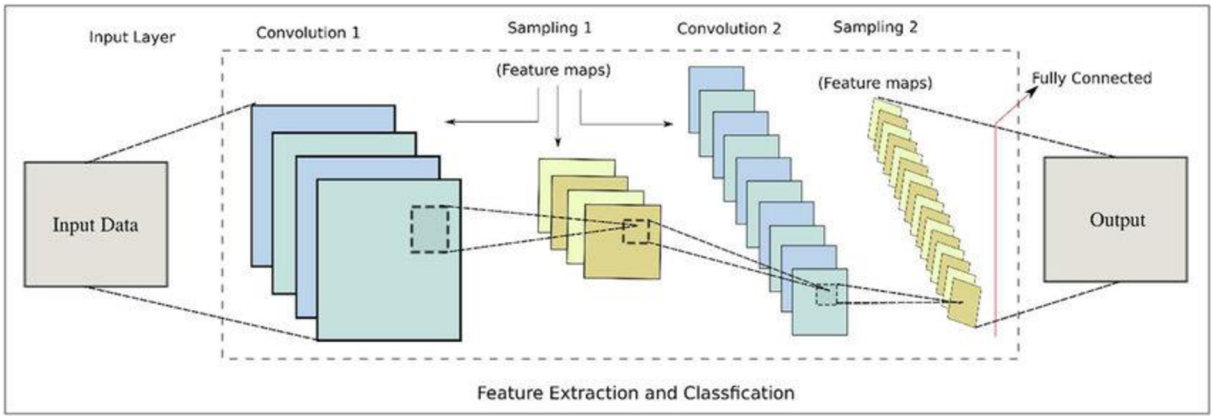
Obrázek 34: Lineární vrstvy v CNN architektuře[40]

3.1.5 Ztrátová funkce

Neuronové sítě jsou trénovány pomocí stochastického gradientního klesání a vyžadují, aby při navrhování a konfiguraci modelu byla zvolena funkce ztráty. Existuje mnoho typů ztrátových funkcí, ze kterých si lze vybrat a každá je vhodná pro jiný model dat nebo aktivační funkci obecně platí že vybrat správnou ztrátovou funkci je velice obtížné. Typicky je model neuronové sítě trénován pomocí algoritmu optimalizace sestupu stochastického gradientu a váhy jsou aktualizovány pomocí zpětného šíření chybového algoritmu. Algoritmus sestupu gradientu se snaží změnit váhy tak, aby další vyhodnocení snížilo chybu. Obecně tak lze definovat tři typy klasifikačních úloh pro které lze vybírat ztrátovou funkci, prvním typem je regrese u této klasifikace se očekává výstup v reálné hodnotě a výstupní vrstva je tvořena jedním uzlem s lineární aktivační funkcí. Ideální funkcí ztrát je funkce ztrát s kvadratickou chybou. Dalším typem je binární klasifikaci u tohoto typu klasifikace je výstupní vrstva tvořena jedním uzlem s sigmoid aktivační funkcí. U této klasifikace je jedné třídě přiřazena hodnota 1 zatímco druhé je přiřazena hodnota 0. Ideální funkce ztrát pro tento typ klasifikace je křížová entropie. A posledním typem klasifikace je klasifikace více tříd, zde je jednotlivým třídám přiřazeno odpovídající číselné označení. Výstupní vrstva je potom tvořena jedním uzlem s aktivační funkcí soft max. Ideální ztrátovou funkcí je potom křížová entropie. [39]

3.2 základní CNN architektura

Všechny CNN architektury jsou postaveny na stejném principu. CNN architektura se skládá s několika druhů vrstev, prvním druhem je konvoluční vrstva v této vrstvě probíhá operace konvoluce mezi vstupními daty (obrázkem) 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 vrstva 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 zároveň přidává do sítě nelinearitu, tímto umožňuje vytvářet aproximace mezi jednotlivými složitými vztahy mezi jednotlivými proměnnými v síti. [36]



Obrázek 35: CNN architektura [36]

3.3 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, každému funkci v architektuře je přiřazeno barevné označení a jsou u ni uvedeny poznámky pro konstrukci dané vrstvy.[12][13]

červě - 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:

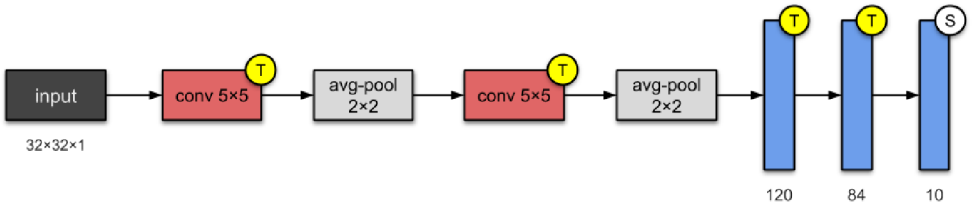
B - Batch normalizace

S - Softmax

x2 - pokud jsou některé funkce opakovaně řazeny za sebe je toto opakování uvedeno pod daným blokem

3.3.1 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.[23]



Obrázek 36: LaNet-5 architektura [13]

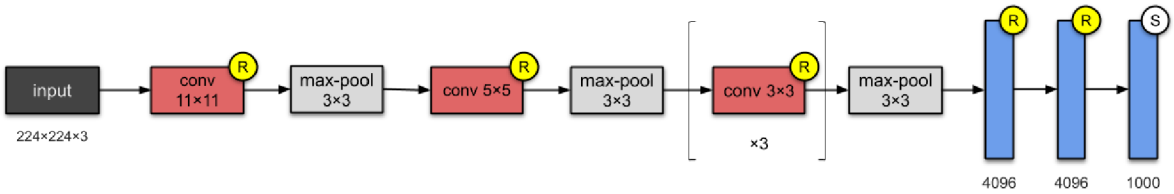
Vstupem do tohoto modelu je obraz ve stupních šedi o velikosti 32x32 proto je počet vstupních kanálů nastaven na jedna. Následně je použita první konvoluce o velikosti filtru 5x5, výsledkem tohoto kroku potom je mapa o velikosti 28x28x6. Následně je na výslednou mapu aplikována pooling vrstva kde dojde ke zmenšení mapy funkcí na poloviční velikost tedy na 14x14x6. Následuje opět konvoluční vrstva s filtrem o velikosti 5x5. Tato vrstva má na výstupu mapu funkcí o velikosti 10x10x16, následně je opět aplikována pooling vrstva pro zmenšení mapy o polovinu tedy na 5x5x16. Výše zmíněné operace mají za cíl vytvořit mapu prvků obsahující významné prvky klasifikovaných objektů jako jsou hrany nebo rohy. Následně je aplikována finální konvoluční vrstva se 120 filtry, výsledkem je potom mapa o velikosti 1x1x120. Následně je výstup přiveden na plně spojenou vrstvu s 84 neurony, a nakonec je připojena výstupní vrstva s 1000 neurony.

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

Tabulka 4: LaNet5 parametry [18]

3.3.2 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 obrázkovou sadou může být například ImageNet datová sada obsahující 14 milionů obrázků spolu s anotacemi. Architekturu AlexNet lze použít například pro detekci v počítačovém vidění s umělou inteligencí. [22]



Obrázek 37: AlexNet architektura [13]

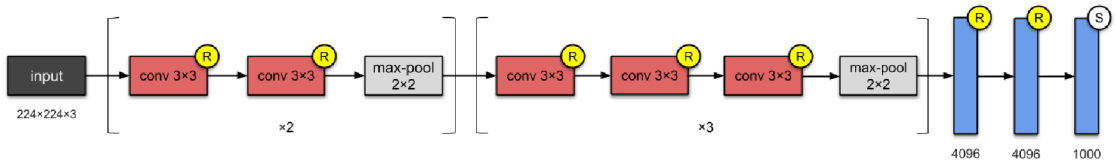
Vstupem modelu jsou RGB obrázky o velikosti 227x227x3. Následně je na vstup aplikována konvoluční funkce s filtrem o velikosti 11x11 s krokem 4. Následně dojde k aplikaci první funkce Maxpooling o velikosti 3x3 s krokem 2 tím dojde ke zmenšení mapy prvků na velikost 27x27x96. Potom je aplikována druhá konvoluce s filtrem velikosti 5x5, u tohoto filtru je stride=1 a padding=2 výsledkem je mapa prvků o velikosti 13x13x256. V tomto kroku jsou aplikovány tři konvoluční funkce které na výstupu vytvoří mapu prvků opět o velikosti 13x13x256. Oproti architektuře LaNet-5 došlo k prohloubení sítě a požití Relu aktivačních funkcí což má za následek zvýšení přesnoti klasifikace. V poslední funkci maxpoolig dojde ke zmenšení mapy prvků na 6x6x256. A v posledním kroku je mapa prvků pomocí tří plně spojených vrstev převedena na výstup architektury, kde výstupní vrstva je tvořena vrstvou s 1000 neurony.[13]

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	-

Tabulka 5: AlexNet parametry [17]

3.3.3 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 pro architekturu určenou pro klasifikaci velmi objemných datasetů. Hlavním rozdílem oproti AlexNet je změna velých vstupních filtrů řadou po sobě jdoucích filtrů o velikosti 3x3. Další nezanedbatelné výhody patří možnost zpracovávat i obrázky ve vysokém rozlišení a relativně snadná implementace. Mezi největší nevýhodu patří velmi dlouhá doba trénování. [24]



Obrázek 38: VGG-16 architektura [13]

Vstupem architektury je potom RGB obrázek o rozměrech 224x224 pixelů. Vstupní data následně prochází řadou malých konvolučních filtrů o rozměrech 3x3. Následně je pomocí funkce max-pool zmenšen objem mapy prvků na polovinu. Druhá část VGG-16 architektury je potom srovnatelná s architektůrou AlexNet ze které vychází nicméně v případě VGG-16 jsou použity tři po sobě jdou konvoluce o rozměrech filtru 3x3. Na výstupu architektury jsou potom aplikovány tři plně spojených vrstev. První dvě vrstvy o velikosti 4096 neuronu pomocí aktivační Relu funkce přidávají do architektury nelinearitu. Poslední vrstva o velikosti 1000 neuronů potom zajišťuje výstup architektury pomocí své aktivační funkce soft-max, v novějších aplikacích potom tuto aktivační funkci nahrazuje Relu.[13]

	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

Tabulka 6: VGG-16 parametry [19]

3.3.4 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 architektuře Inception-v1, základ tvoří paralelní filtry s různými velikostmi. Grafické znázornění dané architektury je potom uvedeno v příloze A.[25]

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								

Tabulka 7: Inception v-1 parametry [20]

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 na dodatku A. Další úpravou je potom zavedení pomocné klasifikace. Jelikož architektura Inception-v1 je již hodně hluboká architektura muže se u ní projevit efekt mizejícího gradientu, aby bylo tomuto efektu zamezeno je celková ztráta počítaná jak z reálné ztráty tak za pomocí těchto dvou pomocných ztrát.[25]

```
# Celková ztráta použitá počáteční sítí během tréninku.
total_loss = real_loss + 0,3 * aux_loss_1 + 0,3 * aux_loss_2
```

Obrázek 39: výpočet skutečných ztrát [33]

3.3.5 Inception-v3

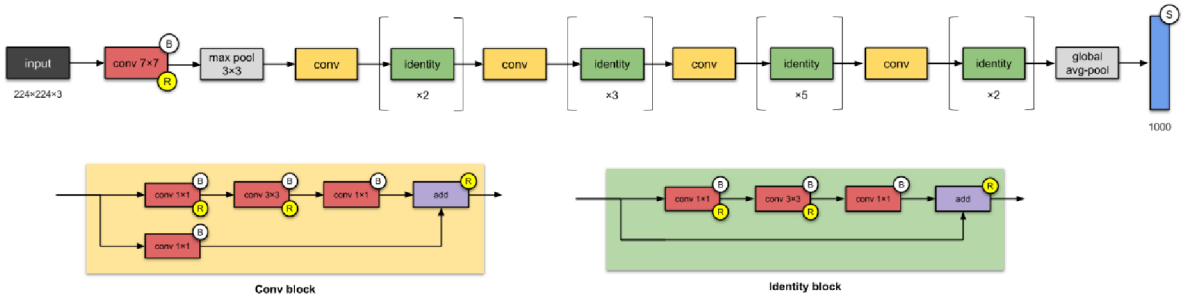
Jedná se o přímé vylepšení předchozí architektury Inception-v1. Jak je možné vidět na modelu architektury v příloze B. Cílem je opět zvýšit výpočetní rychlost architektury a omezit ztrátu informace během výpočtu 5x5 konvolucí. Z matematického hlediska je konvoluce 5x5 2,78krát dražší než konvoluce 3x3, tedy pokud je konvoluce 5x5 nahrazena dvěma konvolucemi 3x3 dostáváme stejný výsledek v kratším čase. Dále architektura pracuje s ekvivalentním výpočtem konvoluce 3x3, tedy dojde k postupnému výpočtu konvoluce 1x3 a následně konvoluce 3x1 tato metoda opět ušetří třetinu času potřebného pro výpočet. Tyto metody jsou použity k sestavení tří klíčových modulů ze kterých je následně architektura Inception-v3 tvořena.[29]

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Tabulka 8:Inception v-3 parametry [21]

3.3.6 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 vrstvými. 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 5 stupňů, každý s konvolucí a blokem 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.[26]

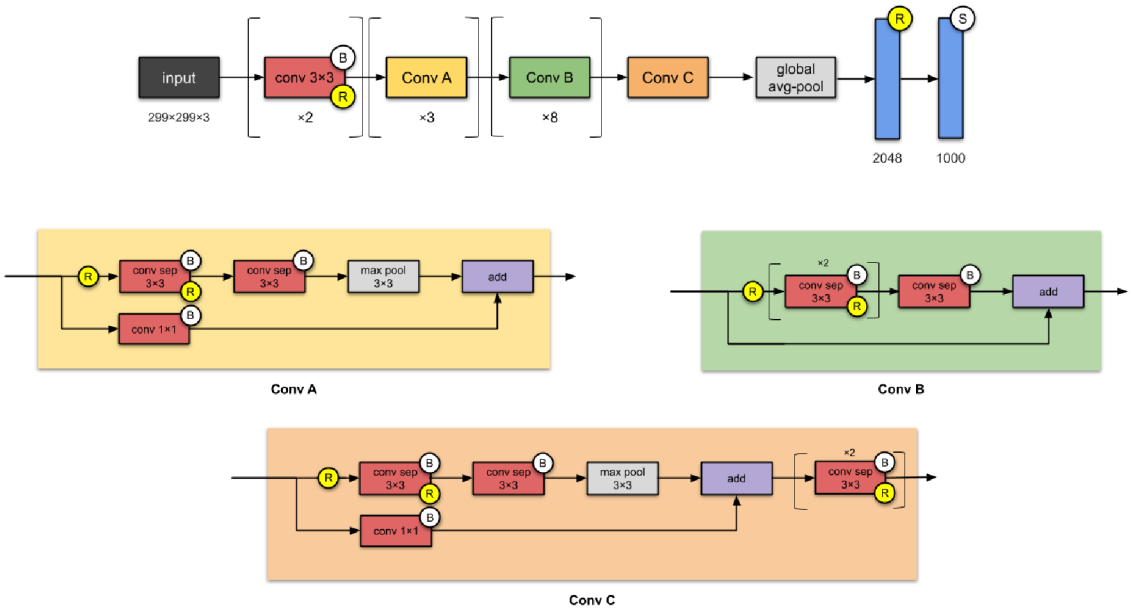


Obrázek 40: ResNet-50 architektura [13]

Zajímavou aplikací pro architekturu ResNet-50 je použití této sítě pro diagnostiku Alzheimerovy chorob. Cílem práce nylo natrénování sítě pro rozeznání typických poškození mozku.[30]

3.3.7 Xception

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



Obrázek 41: Xception architektura [13]

Typickou ukázkou použití Xception architektury muže být opět vyhodnocování lékařských dat. V praxi potom můžeme vyhodnocovat anomálie v rentgenových snímcích pacientů například vyhledávat metastází na krční tepně. [40]

3.3.8 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[13]. Mezi typické příklady použití opět můžeme zařadit vyhodnocování lékařských dat, například studie vytvořená na základě architektury Inception-v4 na rozpoznávání rakoviny kůže[33].

3.3.9 Inception-ResNets

Inception-Resnet-v2 je definován na základě kombinace struktury Inception a zbytkového připojení, v bloku Inception jsou konvoluční filtry různých velikostí kombinovány zbytkovými připojeními. Použití zbytkových spojení nejen předchází problémům s degradací způsobeným hlubokými strukturami, ale také zkracuje dobu tréninku. [13]. 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.[34]

3.3.10 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 použitou 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 a spojuje tak výhody jednotlivých architektur. Architektura je potom zobrazena v příloze D.

3.3.11 GoogleNet

Architektura postavená na základě Inception modulu který přebírá z Inception-v1 architektury. Zároveň tento blok doplňuje řadou metod pro snížení množství parametrů architektury a zároveň dochází k prohloubení architektury. Využitím konvoluce 1x1 dochází k výraznému snížení operací potřebných pro výpočet, například pokud chceme provést konvoluci 5×5 se 48 filtry bez použití konvoluce 1×1 celkový počet operací 112,9 milionů zatímco pokud použijeme mezi krok s konvolucí 1x1 bude celkový počet operací roven 5,3 milionů, tato redukce umožňuje výrazně snížit čas pro trénink sítě. U takto hluboké sítě je opět použit princip pomocných výstupů pro trénování sítě. Celková ztráta je opět počítaná jak z reálného výstupu tak s pomocí těchto pomocných ztrát.[27]

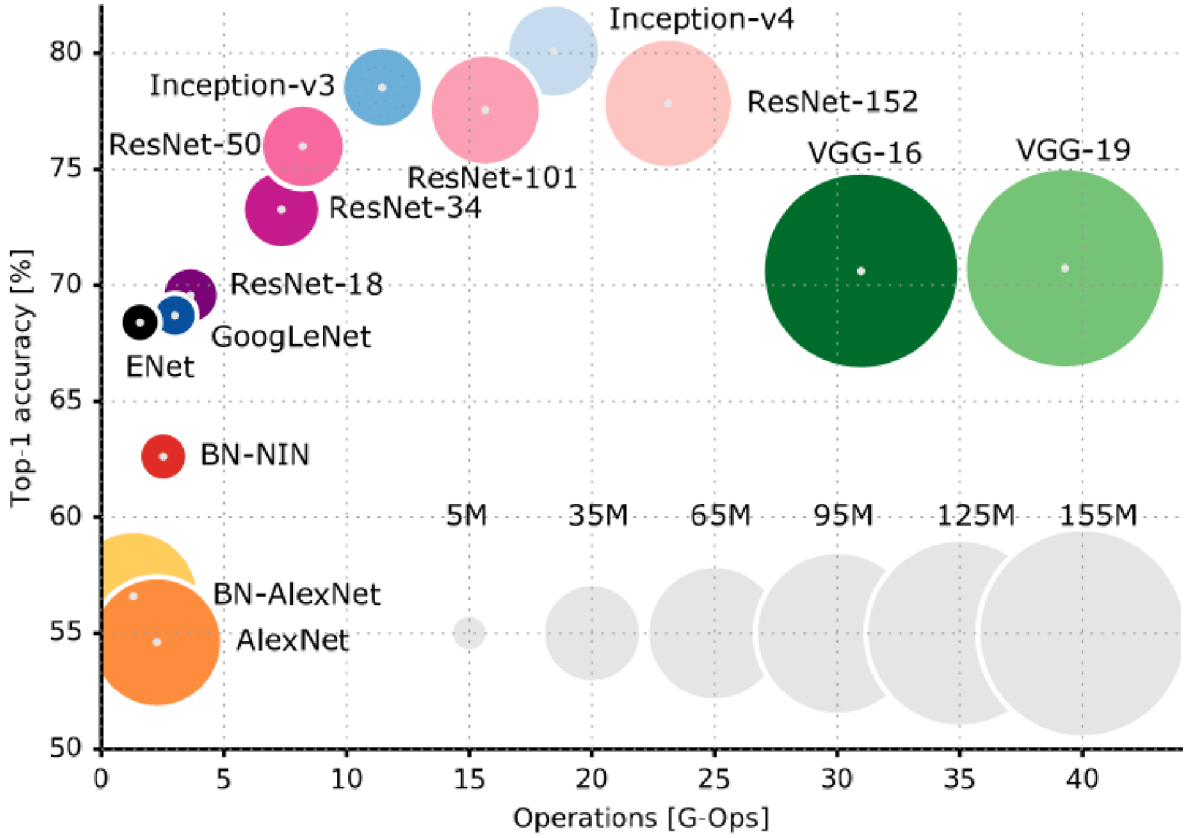
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	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								

Tabulka 9: GoogleNet parametry [27]

Architektura byla navržena tak, aby pamatovala na výpočetní účinnost. Myšlenka, že architekturu lze spustit na jednotlivých zařízeních i při nízkých výpočetních zdrojích. Jednou ze zajímavých aplikací je možnost vyhodnocení stavu pacientů s onemocněním COVID-19 na základě rentgenových snímků[29].

3.4 Srovnání architektur

Srovnání jednotlivých architektur lze provádět několika způsoby, nejběžnějšími jsou srovnání na základě přesnosti dané architektury a to buďto jako top-1 nebo top-5. Dalším způsobem je srovnání na základě počtu cyklů potřebných k natrénování sítě dané architektury. Zároveň lze jednotlivé architektury srovnávat na základě jejich velikosti, počtu parametrů.

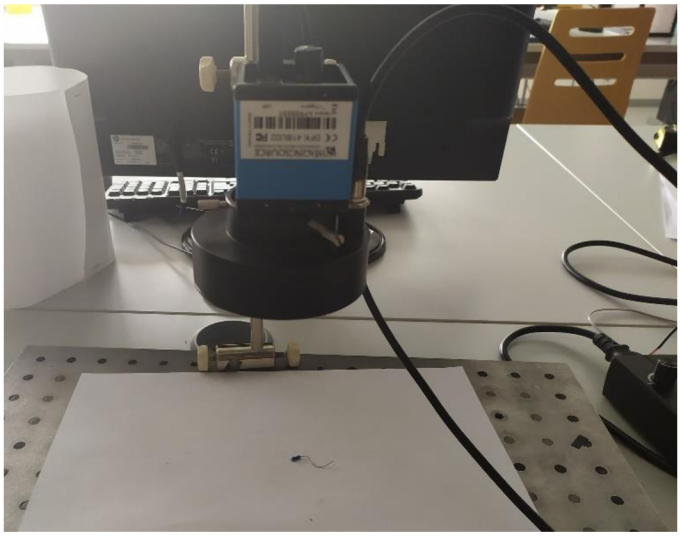


Obrázek 42: Srovnání CNN architektur [42]

Využitím tohoto srovnání lze vytvořit několik základních předpokladů které lze využít při návrhu sítí v praktické části práce. Prvním z těchto předpokladů je výkonost jednotlivých architektur, ze srovnání je zřejmé že výkon jednotlivých architektur se pohybuje mezi 50-80 %. Druhým předpokladem je počet cyklů potřebných pro trénování jednotlivých sítí, jak je vidět 20 cyklů je 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.

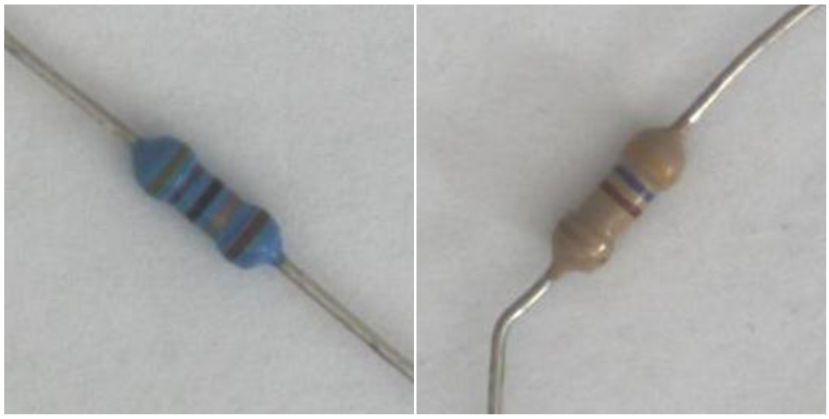
4. ANOTOVANÝ DATASET

Pro získání datasetu byla použita kamera DFK 41BU02 se sériovým číslem 47900031 od společnosti ImagingSourse. Pro ovládání kamery potom byl použit Matlabovský script viz. přílohy.



Obrázek 43: 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 je potom csv anotace daného datasetu obsahující seznam snímků a jejich kategorii (1 pro pozitivní příklady a 0 pro negativní). Jednotlivé snímky byly před použitím pro trénink CNN architektur upraveny na velikost 256x256.



Obrázek 44: typická ukázka datasetu (poz. snímek, neg. snímek)

5. NÁVRH A VYHODNOCENÍ REIMPLEMENTACE

Poslední část práce se zaměřuje na provedení reimplementace v prostření Python. Cílem je vytvoření jednoduché aplikace pro testování jednotlivých architektur vhodných pro unární klasifikaci spolu s možností nastavování hyperpaametrů architektury. Pro samotnou reimplementaci je používána knihovna PyTorch a rozšíření Cuda.

Cuda
Jedná se o rozšíření pro Python podporující výpočet na grafické kartě. Díky tomuto rozšíření je Python schopen efektivně provádět složité výpočty při tréningu CNN sítí a 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 případě malý nebo jednoduchých operací kdy samotné nahrávání do paměti GPU zabere řádově mnohem více času než samotný výpočet na CPU. Program je vytvořen tak že možné jej spustit i bez naistalovány Cuda, kdy celý výpočet bude počítán na CPU a efektivita bude limitován velikostí pamětí RAM. Během praktického testování práce byla vždy použita grafická karta pro výpočet výsledků.

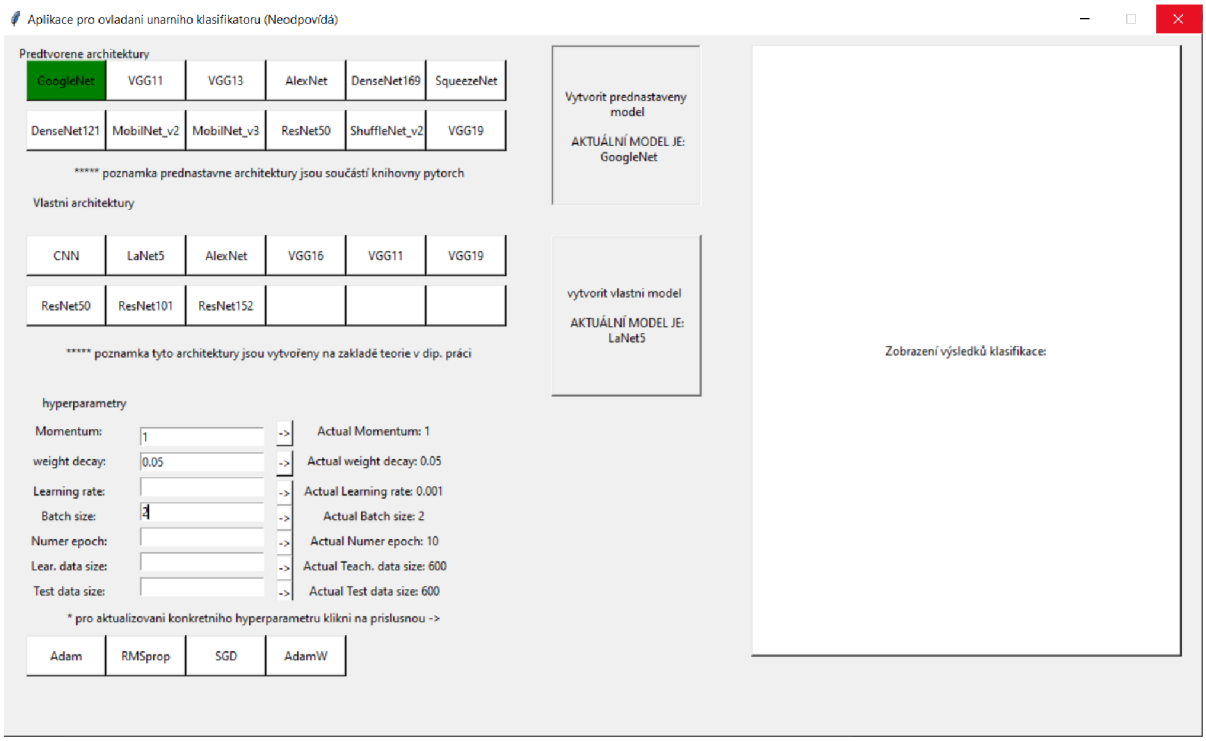
PyTorch
PyTorch je optimalizovaná knihovna tenzorů pro hluboké učení pomocí GPU nebo CPU. Samotná knihovna je volně stažitelná a osahuje všechny funkce pro tvorbu, optimalizaci a vyhodnocení CNN sítí. Tato knihovna je velice dobře optimalizována pro prostření Python a zároveň je velice jednoduchá z uživatelského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

Vlastní reimplementační část se skládá z několika po sobě jdoucích logických programových bloků. Pro lepší přehlednost jsou potom jednotlivé části rozděleny do samostatných souborů.

5.1 Grafická aplikace

V zadání práce je vyžadováno provedení řady měření se změnou jednotlivých hyperparamtrů, pro zjednodušení zadávání je součástí práce grafická aplikace sloužící jako vstup programu i jako výstup pro zobrazení výsledků. Samotné grafické okno lze rozdělit na několik částí, první část tvoří volba požadované architektury. Tato část je následně rozdělena na dvě poloviny kdy v horní části jsou přetvořené architektury které jsou součástí knihovny Pytorch ve spodní části jsou na výběr vlastní architektury. Druhá část vstupů se zabývá definováním hyperparametrů, zadávání není omezeno a je tedy možné zadat libovolnou hodnotu. Ideální hodnoty hyperparametrů jsou uvedeny v kapitole 5.4. Druhá část je potom tvořena tlačítky pro spouštění výpočtu sítě, u každého z tlačítek je uvedeno pro jako architekturu se spustí výpočet, tlačítka jsou rozlišena pro tvorbu sítě na základě vlastní architektury nebo na základě přetvořené architektury. Poslední část potom tvoří pole pro zobrazení výsledků. V této části jsou vypsány hyperparametry sítě, typ sítě a výsledky jak ve formě procentuální přesnosti tak i pomocí matice záměny.

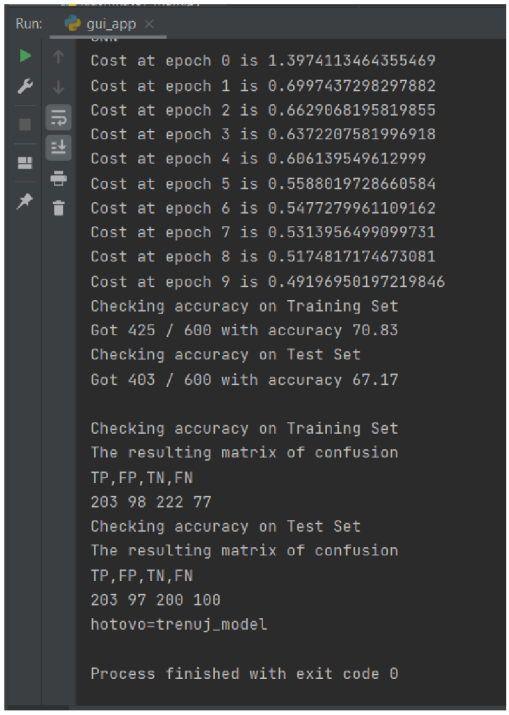
Grafické rozhraní pro Gui_app.py



Obrázek 45: Grafická aplikace pro ovládání programu

Pro správné fungování programu je nezbytné aby byl spouštěna právě přes tuto grafickou aplikaci která zajistí správná zadání hyperparametrů, tvorbu sítě i její otestování.

V samotné aplikaci pro klasifikaci je implementována funkce pro zobrazení výsledků v reálnám čase do konzole programu, tedy pro každý cyklus je zde možnost kontroly výkonu dané sítě zatímco v samotné grafické aplikaci jsou výsledky zobrazeny až po dokončení všech výpočetních funkcí.



Obrázek 46: ukázka výpisu výsledků v reálnem čase pomocí konzole

5.1.1 Definování hyperparametrů

Jak bylo řečeno v kapitole 5.1 o zadání jednotlivých hyper parametrů se stará grafická aplikace ve které může operátor měnit jejich velikosti. Za hyperparametr lze považovat jakýkoli parametr CNN architektur který lze nastavovat, nejčastěji se ale uvádí rychlost učení, velikost dávky, optimalizační a ztrátová funkce popřípadě typ a velikost vstupního datasetu.

Rychlost učení

Jedná se hyperparametr který udává o kolik se má daný model změnit v závislosti na odhadnuté chybě. Obecně je rychlost učení označována jako nejdůležitější hyperparametr. Správná volba velikosti je zásadní pro natrénování kvalitního modelu, příliš nízká rychlost učení má za následek zpomalení procesu učení zatímco příliš vysoká hodnota může způsobit nestabilitu modelu.[43]

Velikost dávky
Velikost dávky definuje počet vzorků, které se budou šířit po síti. Velká velikost dávky umožňuje zrychlit trénink modelu nicméně může způsobit numerickou nestability v případě že je zvolena nesprávná velikost rychlosti učení. Velikost dávky se obvykle volí jako 32,64,128 nebo 256.[43]

Rozložení datasetu
Rozložení datasetu ovlivňuje kolik vstupních dat bude použito pro trénink a kolik pro testování. Příliš málo vzorků při tréninku může způsobit vytvoření modelu který nebude schopen správně klasifikovat všechny objekty obsažené v tréninkové sadě, protože tyto data nebyla dostatečně zastoupena při tréninku.

Optimalizační funkce
Optimalizační funkce zajišťuje správnou úpravu jednotlivých váhovacích funkcí při tréninku. Obecně je optimalizační funkce Adam považována za nejefektivnější a nejčastěji používanou funkci.[43] V dnešní době se začínají objevovat studie prokazující že použití optimalizační funkce Adam na obrazovou klasifikaci nemusí přinést ty nejoptimálnější výsledky. Zatímco optimalizační funkce SGD dosahuje stabilnějších výsledků za cenu delší doby učení.[44]

Ztrátová funkce
Ztrátová funkce slouží pro určení výkonu daného modelu a určuje se jako rozdíl mezi skutečným výstupem modelu a předpokládaným výstupem daném štítkem. Ztrátová funkce je přímo závislé na typu výstupu daného modelu, proto je nezbytné při návrhu dané architektury již dopředu uvažovat na tím jaký typ ztrátové funkce bude použit a příslušně upravit výstupní vrstvu. Možnost úpravy ztrátové funkce není ve finální verzi aplikována a pro všechny CNN architektury je používána CrossEntropi funkce.[43]

Momentům
Tento hyperparametr dodává do modelu určitou hybnost. Takto trénované neuronové sítě tedy překročí menší lokální minima a zastaví se pouze v hlubším globálním minimu. Natavuje se pro optimalizační funkci RMSprop a SGD.[43]

Weight decay
Pokles hmotnosti se používá pro regulaci velikosti aktualizací váhových funkcí, aby se zabránilo nadměrnému přizpůsobení.[43]

5.2 Aplikace pro trénink a vyhodnocení

Pro vytvoření a natrénování sítě práce obsahuje samostatný script Klasifikator_main.py, tento script přejímá hyperparametry z grafické aplikace a na jejich základu vytvoří a natrénuje požadovanou síť. Celý script je rozdělen do několika funkčních bloků, kdy každý z těchto bloků je určen pro provedení jedné konkrétní činnosti. Prvním krokem je načtení hyperparametrů a jejich tisk do konzole pro kontrolu. Následně je přiložený obrazový dataset rozdělen na část pro trénink a pro testování. Pro načtení datasetu je použita funkce MujDataset, tato funkce se stará o správné načítání jednotlivých snímku a jejich anotací z přiloženého CSV souboru. Následně dojde k nastavení dané architektury a jejího nahrání do modelu, zároveň dojde k nastavení optimalizační a ztrátové funkce. Následuje cyklická funkce pro trénování sítě, její optimalizaci a výpočet celkového výkonu dané sítě v daném kroku. Posledním krokem je kontrola výkonu natrénované sítě. Pro kontrolu script obsahuje dvě nezávislé funkce, tyto funkce přistupují ke kontrole různými metodami aby byla zajištěna správnost výsledků. První funkce check_accuary kontroluje výstupní tenzor sítě srovnáním s tenzorem obsahujícím hodnoty skutečného výstupu získané z CSV souboru. Druhou funkcí je funkce check_tab_accuracy, tato funkce se stará o výpočet matice záměny na základě znalostí výstupů sítě a rozložení datasetu.

5.3 Vlastní CNN architektury

Součástí odevzdání jsou scripty obsahující vlastní CNN architektury. Tyto architektury vychází z architektur popsaných v kapitole 3.3. Cílem je vytvořit CNN architektury které jsou výkoností srovnatelné s přetvořenými architekturami v knihovně Pytorch, nicméně zároveň umožňují kompletně upravovat jednotlivé vrstvy nebo aktivační funkce. Každá takováto architektura 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 CNN architekturu. Mezi největší výhody takto navržených sítí patří možnost upravení výstupní vrstvy tak aby bylo možné použít i jinou než základní ztrátovou funkci (CrossEntropy).

Seznam vlastních CNN architektur:
Vlastní_AlexNet.py
Vlastní_CNN.py
Vlastní_LaNEt5.py
Vlastní_ResNet.py
Vlastní_VGG11.py
Vlastní_VGG16.py
Vlastní_VGG19.py

5.4 Ověření kvality klasifikace

Vyhodnocení pomocí matice záměn

Pro každou zkoumanou síť byla stanovena odpovídající matice záměn na základě výkonu klasifikace dané sítě. Matice je u každé sítě je matice záměny ve tvaru [TP,FP/TN,FN], kde TP je počet pozitivních dat klasifikovaných jako pozitivní, FP je počet pozitivních dat klasifikovaných jako negativní, TN je počet negativních dat klasifikovaných jako negativní a FN je počet negativních dat klasifikovaných jako pozitivní.

5.4.1 Ověření maximální velikosti Batch

Velikost dávky ovlivňuje množství paralelně zpracovávaných informací. Větší velikost dávky tedy umožňuje rychlejší trénink sítě. Celková velikost dávky je ovlivněna velikostí 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 můžeme provést například průběžným odstraňováním nepotřebných proměnných nebo optimalizací výpočtu lineárních vrstev v CNN sítích, pro příklad lineární vrstva s velikostí 1000 je spojena z další vrstvou s velikostí 1000, v takovém to případě je v paměti tržena matice o velikosti 1000x1000 prvků.

Architektura	GoogleNet	VGG11	VGG13	AlexNet	DenseNet169	SqueezeNet
Velikost Batch	64	8	8	512	16	64
Architektura	DenseNet121	MobilNet v2	MobilNet v3	ResNet50	ShuffleNet v2	VGG19
Velikost Batch	16	32	16	16	128	8

Tabulka 10: Tabulka velikostí dávek pro přetvořené architektury

Vlastní Architektura	CNN	LaNet5	AlexNet	VGG16	VGG11	VGG19
Velikost Batch	256	64	512	16	32	4
Vlastní Architektura	ResNet50	ResNet101	ResNet152			
Velikost Batch	16	4	4			

Tabulka 11: Tabulka velikostí dávek pro vlastní architektury

U sítí s malou velikostí dávky je doba tréninku poměrně dlouhá, řádově se jedná o minuty zatímco sítě s velikosti dávky 64 a větší jsou schopny natrénovat síť během desítek sekund.

Pro maximální využití paměti grafické karty lze po každém natrénování sítě ukončit program, tímto krokem dojde k odstranění veškerých zbylých proměnných z paměti a celá paměť je volná pro další síť.

5.4.2 Ověření velikosti dávky na kvalitu klasifikace

Z teoretických poznatků o velikosti dávky je zřejmé že přesnost sítě je úzce spojena s velikostí dávky a rychlostí učení. Pro ověření tohoto předpokladu lze uvažovat dvě hodnoty rychlosti učení, rychlost učení 0,001 je ve většině případů brána jako referenční. Cílem kapitoly je ověření teoretického předpokladu že pro nižší rychlost učení je výhodnější použít větší velikost dávky zatímco pro větší rychlosti učení je naopak výhodnější použít menší velikosti dávky.[41]

Rychlost učení = 0,001

	GoogleNet	AlexNet	SqueezeNet	ShuffleNet v2	vlastníCNN	VlastníLaNet5	vlastniAlexNet
Batch = 16	299,3/297,1	267,32/261,40	170,131/158,141	285,5/307,3	209,78/238,75	210,90/198,102	151,140/158,151
Batch = 32	297,2/299,1	278,21/274,27	150,154/146,150	302,8/282,3	208,85/202,105	254,52/228,66	154,155/154,137
Batch = 64	289,8/294,9	141,165/156,138	173,129/149,149	287,10/301,2	224,67/228,81	215,91/205,89	170,150/150,130
Batch = 128	*	139,148/148,165	*	262,46/245,47	216,78/223,83	*	127,154/153,166
Batch = 256	*	153,19/159,129	*	*	213,77/248,62	*	150,142/159,149
Batch = 512	*	157,152/135,156	*	*	*	*	150,159/141,150

Tabulka 12: ověření velikosti dávky pro rychlost učení 0.001

Rychlost učení = 0,0001

	GoogleNet	AlexNet	SqueezeNet	Shuffleet v2	vlastní CNN	vlastní LaNet5	vlastní AlexNet
Batch = 16	308,2/289,1	303,2/290,5	300,1/296,3	299,2/294,5	213,86/212,89	199,105/198,98	140,167/154,139
Batch = 32	291,8/285,16	295,12/286,7	276,18/290.16	284,14/291,11	153,150/147,150	212,92/200,96	187,130/167,116
Batch = 64	293,10/291,6	297,6/294,3	298,12/283,7	294,8/287,11	183,98/231,88	177,125/192,106	145,158/158,139
Batch = 128	*	283,8/300,9	*	168,132/182,118	169,131/172,128	*	147,147/146,160
Batch = 256	*	278,19,285,18	*	*	144,158/154,144	*	138,145/145,172
Batch = 512	*	287,28/271,14	*	*	*	*	141,154/154,151

Tabulka 13: ověření velikosti dávky pro rychlost učení 0.0001

Nastavení hyperparametrů:
Počet cyklů = 10
Rozložení datasetu = 600/600
Ztrátová funkce = CroosEtropy
Optimalizér = Adam

Vyhodnocení klasifikace pomocí matice záměn:
Každá sít je vyhodnocena maticí záměny ve tvaru [TP,FP/TN,FN]. Pokud daná architektura nedovoluje vytvoření sítě s požadovanou velikostí dávky je v tabulce označena *.

5.4.3 Ověření vlivu rychlosti učení

Rychlost učení je jedním z nejdůležitějších hyperparametrů. Volba ideální rychlosti učení je náročná, protože příliš malá hodnota může mít za následek dlouhý tréninkový proces, který by se mohl zaseknout, zatímco příliš velká hodnota může mít za následek příliš rychlé učení neoptimální sady závaží nebo nestabilní tréninkový proces.

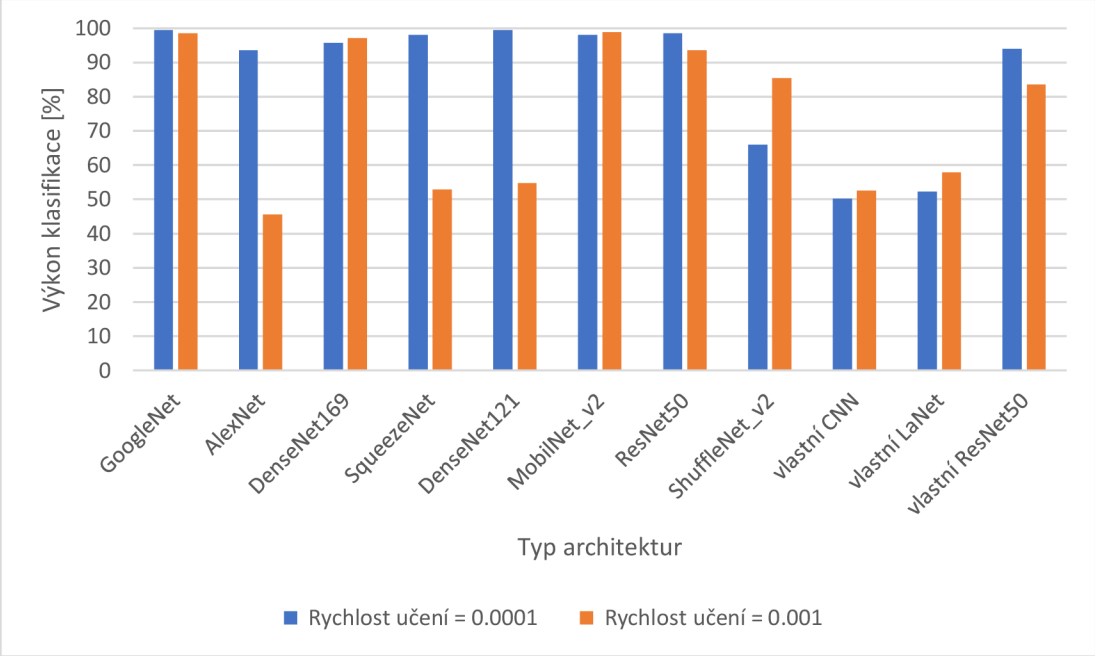
	Velikost dávky = maximum		Velikost dávky = 16	
Rychlost učení =	0.0001	0.001	0.0001	0.001
GoogleNet	290,1/307,2	305,1/287,7	294,5/298,3	286,5/306,3
VGG11	145,150/153,152	145,151/152,152	*	*
VGG13	151,143/156,150	151,156/141,152	*	*
AlexNet	275,19/286,20	151,155/122,172	264,33/268,35	154,148/148,150
DenseNet169	285,12/290,13	293,7/290,10	298,0/301,1	293,7/290,10
SqueezeNet	303,6/285,6	154,129/163,154	272,22/275,31	259,31/294,16
DenseNet121	315,1/282,2	168,138/161,133	269,39/249,49	168,138/161,133
MobilNet_v2	290,4/298,8	301,4/292,3	303,3/287,7	299,4/294,3
MobilNet_v3	128,160/124,288	155,158/141,146	128,160/124,288	155,158/141,146
ResNet50	300,4/291,5	272,17/288,23	300,4/291,5	272,17/288,23
ShuffleNet_v2	195,87/201,117	248,47/265,40	288,4/299,9	286,7/302,5
VGG19	280,224/262,234	196,204/196,104	*	*
vlastní CNN	155,159/146,140	163,143/152,152	150,156/145,149	168,127/171,134
vlastní LaNet	157,150/156,137	173,128/174,125	263,46/256,39	254,46/252,48
vlastní AlexNet	154,149/18,149	148,147/157,148	157,144/143,156	144,151/151,154
vlastní VGG16	150,148/150,152	148,150/155,147	151,150/148,151	138,154/153,155
vlastní VGG11	144,151/151,154	143,153/152,152	198,95/212,95	148,143/143,166
vlastní VGG19	162,138/159,141	150,157/143,150	*	*
vlastní ResNet50	286,20/278,14	256,49/246,49	299,8/279,14	235,53/242,70
vlastní ResNet101	292,8/291,9	288,12/281,19	*	*
vlastní ResNet152	294,7/299,2	292,9/293,6	*	*

Tabulka 14: ověření vlivu rychlosti učení

Nastavení hyperparametrů:
Počet cyklů = 10
Rozložení datasetu = 600/600
Ztrátová funkce = CroosEtropy
Optimalizér = Adam
Velikost dávky = maximální velikost podle kapitoly 5.4.1

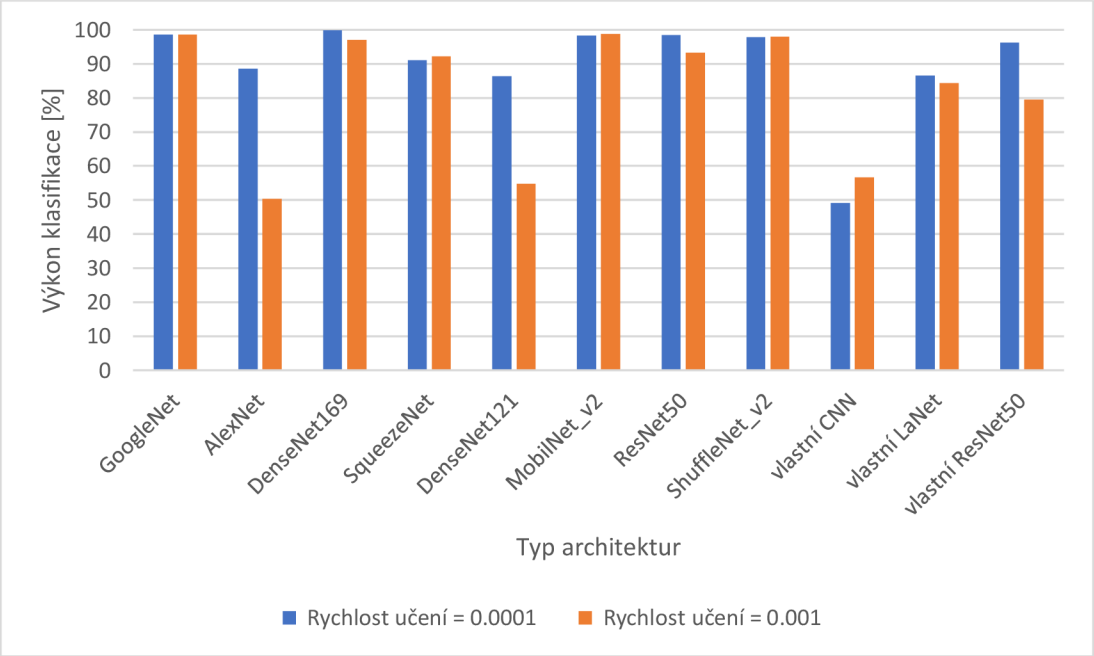
Vyhodnocení klasifikace pomocí matice záměn:
Každá sít je vyhodnocena maticí záměny ve tvaru [TP,FP/TN,FN]. Pokud daná architektura nedovoluje vytvoření sítě s požadovanou velikostí je v tabulce označena *.

Grafické srovnání kvality klasifikace v závislosti na rychlosti učení:
Velikost dávky = maximum podle kapitoly 5.4.1



Obrázek 47: grafické srovnání kvality klasifikace pro rychlost učení, velikost dávky je max

Velikost dávky = 16



Obrázek 48: grafické srovnání kvality klasifikace pro rychlost učení, velikost dávky je 16

5.4.4 Ověření vlivů počtu cyklu na kvalitu klasifikace

Počet cyklů přímo ovlivňuje celkovou přesnost klasifikace a volby správného počtu cyklů je jedním z nejdůležitějších hyperparametrů. Jak je zmíněno v kapitole 3.4 pro většinu architektur je 20 cyklů dostatečných pro vytvoření sítě s dostatečnou přesností. Zároveň platí předpoklad že složitější architektury vyžadují více cyklů pro vytvoření výkonné sítě.

	počet cyklů = 5	počet cyklů = 10	počet cyklů = 15	počet cyklů = 20
GoogleNet	273,38/253,36	284,6/305,5	301,5/287,7	303,3/286,8
VGG11	143,153/150,154	145,150/153,152	152,142/154,152	155,143/158,150
VGG13	145,152/153,154	151,143/156,150	152,149/149,150	156,145/162,137
AlexNet	250,41/258,51	279,18/282,21	282,15/279,24	284,14/277,25
DenseNet169	284,17/290,9	299,2/296,3	299,1/295,5	298,4/295,3
SqueezeNet	288,12/291,9	294,6/294,6	299,3/293,5	302,2/292,4
DenseNet121	293,3/300,4	304,7/282,7	298,2/294,6	301,0/293,6
MobilNet_v2	290,10/295,5	282,6/306,6	297,5/295,3	299,1/295,5
MobliNet_v3	104,196/118,182	142,144/150,164	132,169/116,183	110,192/150,48
ResNet50	295,5/293,7	301,4/293,2	298,2/295,5	299,2/295,4
ShuffleNet_v2	151,147/170,132	185,108/188,119	235,58/239,68	273,13/306,8
VGG19	140/162/149,159	138,151/151,160	144,146/157,143	152,149/169/130
vlastní CNN	150,148/153,149	142,166/151,141	146,173/136,145	204,79/228,89
vlastní LaNet	151,149/156,144	199,110/192,95	119,91/212,74	225,80/227,68
vlastní AlexNet	148,161/144,147	144,148/164,144	155,148/142,155	161,127/151,161
vlastní VGG16	136,158/157,149	148,150/155,147	157,146/140,157	159,148/148,145
vlastní VGG11	142,144/152,162	143,153/152,152	154,151/151,144	158,142/162,138
vlastní VGG19	148,153/159,140	150,157/143,150	150,155/146,149	155,141/149,155
vlastní ResNet50	251,51/248,50	254,48/261,37	271,28/277,24	282,17/286,5
vlastní ResNet101	256,54/243,47	255,47/263,39	281,28/280,21	284,11/285,10
vlastní ResNet152	252,48/247,53	259,53/259,39	275,28/277,24	286,13/287,4

Tabulka 15: ověření počtu cyklů na kvalitu klasifikace, velikost dávky max

Nastavení hyperparametrů:
Rozložení datasetu = 600/600
Ztrátová funkce = CroosEtropy
Optimalizér = Adam
Rychlost učení = 0.0001
Velikost dávky = maximální velikost podle kapitoly 5.4.1

Vyhodnocení klasifikace pomocí matice záměn:
Každá sít je vyhodnocena maticí záměny ve tvaru [TP,FP/TN,FN].

Ověření výsledků pro různý počet cyklů pro velikost dávky =16, tato velikost byla zvolena na základě optimální poměru mezi dálkou výpočtu a přesností výpočtu oproti velikosti dávky = 256.

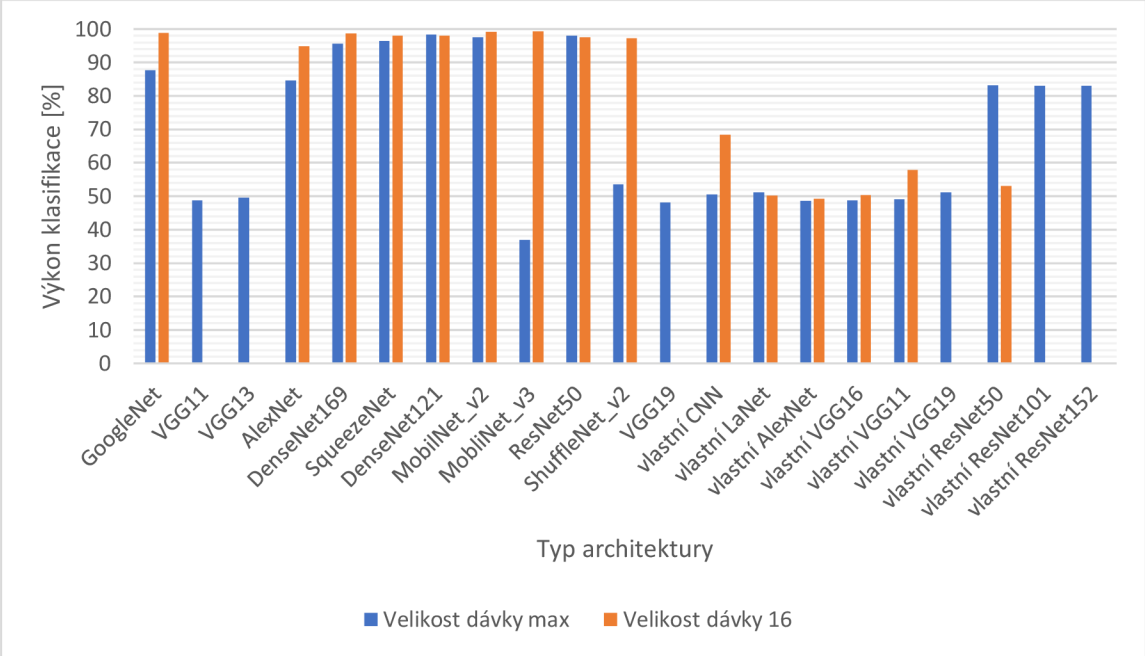
	počet cyklů = 5	počet cyklů = 10	počet cyklů = 15	počet cyklů = 20
GoogleNet	299,5/294,2	298,4/295,3	297,1/298,4	318,5/274,3
VGG11	*	*	*	*
VGG13	*	*	*	*
AlexNet	274,19/295,12	289,8/292,11	292,5/299,4	289,3/307,1
DenseNet169	290,4/301,5	299,2/296,3	299,1/297,3	302,0/298,0
SqueezeNet	306,2/284,8	294,6/294,6	299,3/293,5	294,3/300,3
DenseNet121	306,5/282,7	304,3/2290,3	291,1/295,5	294,2/303,1
MobilNet_v2	302,4/293,1	282,6/306,6	297,4/295,4	297,3/296,4
MobliNet_v3	254,53/242,51	192,74/200,114	242,59/246,53	246,54/247,53
ResNet50	290,7/296,7	301,4/293,2	298,2/295,5	299,2/295,4
ShuffleNet_v2	291,7/292,10	185,108/188,119	235,58/239,68	293,4/301,2
VGG19	*	*	*	*
vlastní CNN	197,101/213,89	222,78/204,96	263,55/245,49	273,45/239,43
vlastní LaNet	143,156/158,143	142,144/150,164	132,169/116,183	198,114/172,116
vlastní AlexNet	143,153/153,151	148,150/155,147	157,146/140,157	155,152/139,154
vlastní VGG16	149,149/153,149	148,152/155,145	143,155/202,100	155,143/161,141
vlastní VGG11	172,123/175,130	200,100/188,112	233,39/230,38	263,39/263,35
vlastní VGG19	*	*	*	*
vlastní ResNet50	155,147/163,135	200,109/202,90	249,69/221,71	289,19/272,20
vlastní ResNet101	*	*	*	*
vlastní ResNet152	*	*	*	*

Tabulka 16: ověření počtu cyklů na kvalitu klasifikace, velikost dávky 16

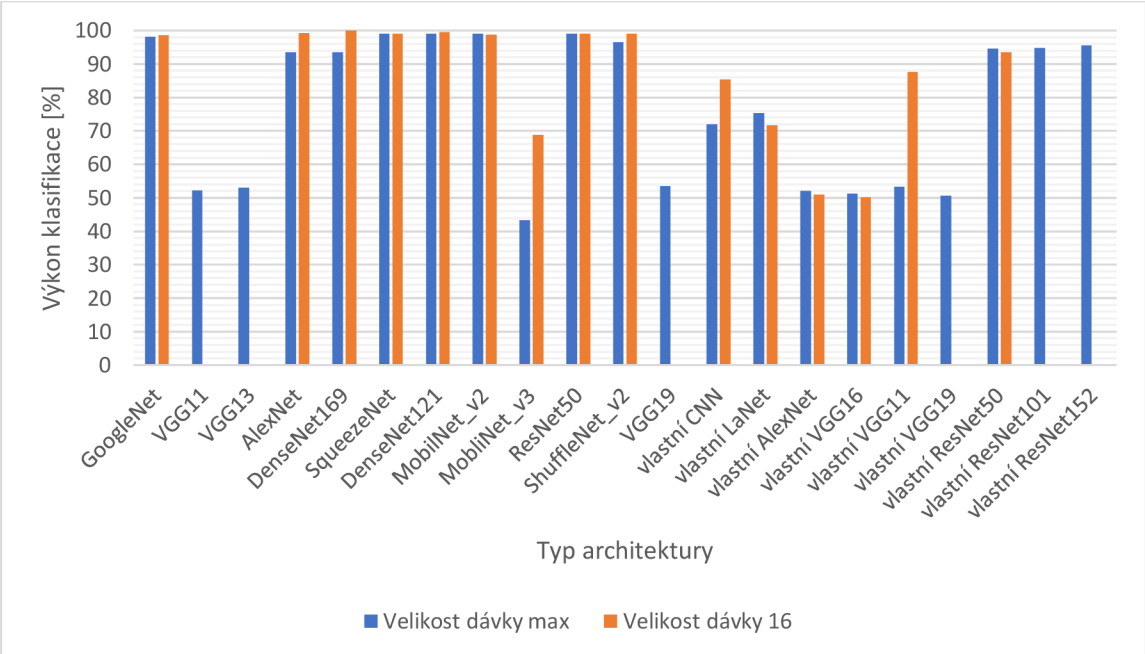
Nastavení hyperparametrů:
Rozložení datasetu = 600/600
Ztrátová funkce = CroosEtropy
Optimalizér = Adam
Rychlost učení = 0.0001
Velikost dávky = 16

Vyhodnocení klasifikace pomocí matice záměn:
Každá sít je vyhodnocena maticí záměny ve tvaru [TP,FP/TN,FN]. Pokud daná architektura nedovoluje vytvoření sítě s požadovanou velikostí je v tabulce označena *.

Grafické srovnání kvality klasifikace v závislosti na počtu cyklů



Obrázek 49: Grafické srovnání výkonu CNN sítí pro různé velikosti dávky, 5 cyklů



Obrázek 50: Grafické srovnání výkonu CNN sítí pro různé velikosti dávky, 20 cyklů

5.4.5 Ověření rozložení datasetu na kvalitu klasifikace

Rozložením datasetu lze ovlivnit kolik snímků bude použito pro vytvoření sítí a kolik pro její otestování. Cílem tohoto měření je určení optimální hranice pro vytvoření dostatečně přesné sítě. O samotné rozdělení datasetu se stará funkce data,random_split, z definice vyplívá že pro jakékoli rozložení datasetu bude daný dataset obsahovat přibližně 50% pozitivních a 50% negativních snímků. Dataset je rozdělen na dvě části a v tabulce je zapsán ve formátu [počet tréninkových snímků, počet testovacích snímků]

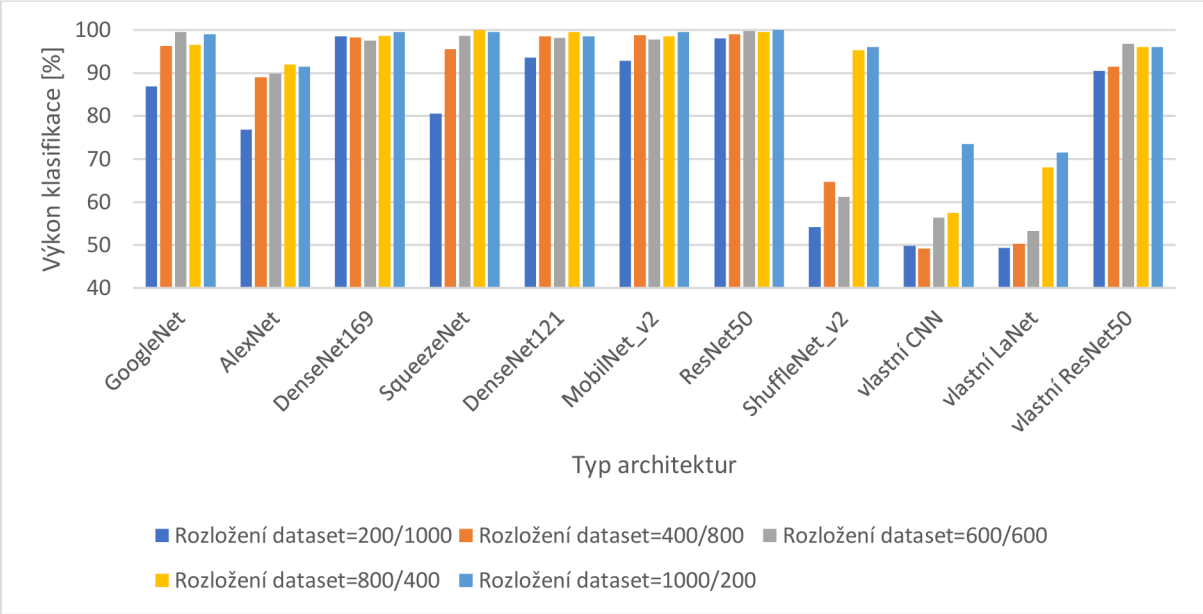
Rozložení dat =	200/1000	400/800	600/600	800/400	1000/200
GoogleNet	475,26/473,26	410,3/381,6	291,1/306,2	188,6/198,8	89,1/109,1
VGG11	251,255/245,249	197,215/198,190	297,300/300,303	99,90/112,99	53,42/55,50
VGG13	180,224/266,230	190,210/195,205	164,140/159,141	99,101/118,92	50,43/52,55
AlexNet	362,133/401,104	354,36/358,52	267,32/272,29	191,13/177,19	92,9/91,8
DenseNet169	489,7/496,8	392,7/395,6	289,1/296,4	196,3/199,2	96,0/103,1
SqueezeNet	412,87/393,108	394,11/372,23	295,6/297,2	214,0/186,0	99,1/100,0
DenseNet121	456,31/479,34	391,9/395,5	294,4/295,7	194,1/204,1	99,1/98,2
MobilNet_v2	471,36/458,38	392,8/395,5	293/294,4	196,5/198,1	104,0/95,1
MobilNet_v3	179,120/401,100	211,189/205,195	147,165/125,163	194,7/186,13	87,2/105,6
ResNet50	498,13/482,7	394,6/398,2	298,0/301,1	199,2/199,0	99,0/101,0
ShuffleNet_v2	280,224/261,235	265,132/253,150	192,105/175,128	187,13/194,6	99,5/93,3
VGG19	280,224/262,234	196,204/196,104	162,138/159,141	100,101/112,85	52,41/62,45
vlastní CNN	252,247/246,255	198,196/195,211	236,73/222,69	108,73/122,97	65,28/82,25
vlastní LaNet	251,244/242,263	203,197/199,201	150,149/152,149	147,53/152,48	64,30/79,27
vlastní AlexNet	251,249/250,250	203,197/198,202	170,144/144,142	198,202/203,197	50,53/47,50
vlastní VGG16	253,245/245,257	217,183/195,205	162,138/159,141	99,101/112,88	56,43/43,58
vlastní VGG11	252,254/243,251	195,217/194,194	297,301/299,303	99,90/112,99	51,44/54,51
vlastní VGG19	279,225/262,234	200,200/195,105	162,138/159,141	99,101/112,88	50,43/52,55
vlastní ResNet50	454,46/451,49	379,34/353,34	289,11/292,8	179,9/205,7	97,5/95,3
vla. ResNet101	458,42/455,45	380,20/378,22	292,8/291,9	190,12/195,3	98,2/98,2
vla. ResNet152	455,45/461,39	382,21/417,80	294,7/199,2	199,1/196,4	99,1/96,4

Tabulka 17: vliv rozložení datasetu

Nastavení hyperparametrů:
Počet cyklů = 10
Ztrátová funkce = CroosEtropy
Optimalizér = Adam
Rychlost učení = 0.0001
Velikost dávky = maximální velikost podle kapitoly 5.4.1

Vyhodnocení klasifikace pomocí matice záměn:
Každá sít je vyhodnocena maticí záměny ve tvaru [TP,FP/TN,FN].

Grafické srovnání kvality klasifikace v závislosti na rozložení datasetu:



Obrázek 51: srovnání rozložení datasetu

5.4.6 Ověření vlivu optimalizační funkce na kvalitu klasifikace

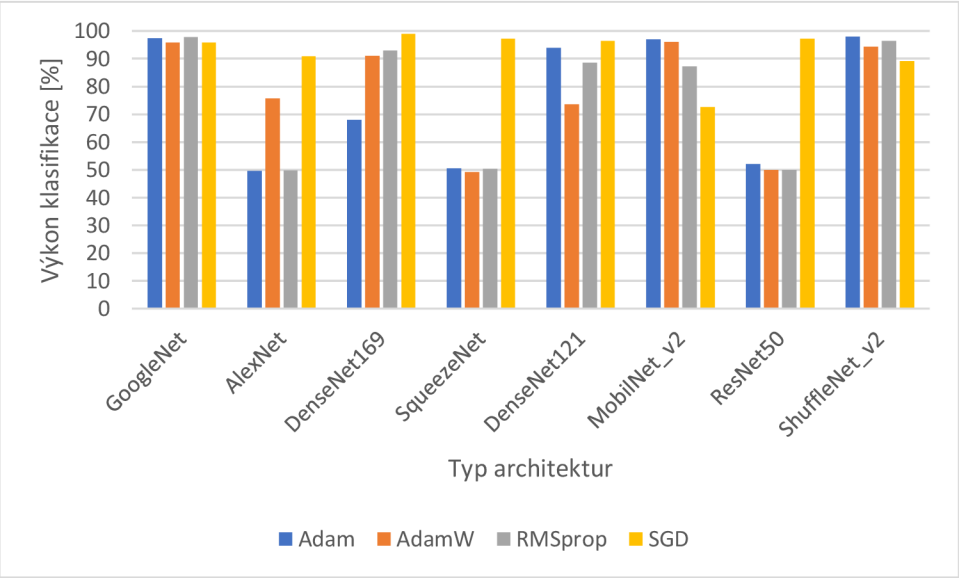
U CNN architektur se používá optimalizační metoda klesajícího gradientu. Volba optimalizační metody ovlivňuje jak rychlost konvergence, tak i to, zda k ní dojde. V posledních několika letech bylo vyvinuto několik alternativ ke klasickým algoritmům klesajícího gradientu. Z teorie je známe že nejlepší optimalizační funkce je Adam. Pro následnou klasifikaci byly použity defaultní nastavení hyperparametrů jednotlivých metod.[44]

Velikost dávky = 16 , rozložení datasetu = 600/600 , cykly = 10

Adam : rychlost učení =0.001, pokles hmotnosti = 0
AdamW : rychlost učení =0.001, pokles hmotnosti = 0.01
RMSprop : rychlost učení = 0.001, pokles hmotnosti = 0, momentům =0
SGD : rychlost učení = 0.001, pokles hmotnosti = 0, momentům =0

	Adam	AdamW	RMSprop	SGD
GoogleNet	299,8/285,8	302,12/273,13	287,6/300,7	287,15/288,10
AlexNet	150,154/147,149	188,30/266,16	157,145/142,156	275,27/270,28
DenseNet169	211,104/197,88	274,32/272,22	282,23/276,19	317,3/277,3
SqueezeNet	141,157/162,140	135,161/160,144	147,152/155,146	279,10/304,7
DenseNet121	290,12/274,24	233,81/209,77	268,25/263,44	290,10/289,11
MobilNet_v2	305,9/277,9	290,11/286,13	263,39/261,37	209,80/227,84
ResNet50	149,150/163,138	157,134/176,133	143,157/157,143	282,12/301,5
ShuffleNet_v2	279,7/309,5	284,16/282,18	293,5/286,16	281,24/254,41

Tabulka 18: srovnání výkonů optimalizačních metod



Obrázek 52: srovnání výkonů optimalizačních funkcí

6. ZÁVĚR

Diplomová práce se zabývá definování klasifikace, klasifikačních metod a jejich využití pro klasifikaci anotovaného obrazového datasetu. V dnešní době se jedná o velice řešené téma s širokou možností využití od průmyslové výroby přes zdravotnictví až po oblasti autonomního řízení.

První dvě části této práce teoreticky rozebírají jednotlivé metody pro klasifikaci dat. Tato část práce detailně rozebírá jednotlivé metody jejich výhody a nevýhody, parametry jednotlivých metod. Tato část se následně zaměřuje na využití těchto metod pro unární klasifikaci obrazových dat. Třetí část práce teoreticky rozebírá jednotlivé typy CNN architektur a jejich parametry. Teoretické poznatky z této části práce jsou následně použity pro tvorbu vlastních CNN architektur v praktické části práce. Součástí této části práce je detailní popis jednotlivých vrstev konvolučních sítí a jejich parametrů. Čtvrtá kapitola práce se zabývá tvorbou notovaného datasetu reálného průmyslového výrobku, kvalita obrazového datasetu má vliv na jak na celkovou kvalitu klasifikace tak na maximální velikost dávky jednotlivých architektur. Poslední část práce se zaměřuje na otestování jednotlivých architektur pro různé nastavení hyperparametrů. Z výsledků v této kapitole práce lze vyvodit následující závěry. Velikost dávky je přímo spjata s velikostí dané architektury a hardwarem na kterém probíhá výpočet, zároveň má velikost dávky vliv na rychlost tréninku, pro velké velikosti dávky dochází k výraznému zrychlení tréninku dané architektury, na druhou stranu při použití větší velikosti dávky dochází ke snížení přesnosti klasifikace. S této části práce je parné že je mnohem lepší volit nižší rychlost učení a dávky pro dosažení výší přesnosti klasifikace a to i za cenu zpomalení tréninku dané architektury. Další část této kapitoly se potom zaměřuje na vliv rozložení datasetu na kvalitu klasifikace. Jak je vidět z výsledků této kapitoly použitý tréninkový dataset musí mýt dostatečnou velikost, aby byl schopen pokrýt všechny typy vstupních dat, jako ideální rozložení se jeví rozložení datasetu na poloviny. Použitím funkce pro náhodné rozdělení datasetu je zajištěno že jak tréninkový tak testovaný dataset bude mít dostatečný počet dat každého typu. Poslední část se potom zaměřuje na vliv jednotlivých optimalizačních metod na kvalitu klasifikace. Jak bylo teoretickým předpokládáno optimalizační funkce Adam je sice schopna pro určité architektury dosáhnou lepší výsledků, na druhou stranu funkce SGD dosahuje dobrých výsledků pro všechny typy architektur, tento výsledek odpovídá teoretickým předpokladům.

Výsledkem práce je tedy anotovaná dataset snímků reálného průmyslového výrobku a program umožňující efektivní návrh jednotlivých CNN architektur s různým nastavením hyperparametrů. Součástí práce je řada měření definující vliv jednotlivých hyperparametrů na kvalitu klasifikace jednotlivých architektur. Práce může být v budoucnu doplněna o možnost úpravy ztrátové funkce a s ní spojené úpravy výstupních vrstev jednotlivých architektur.

LITERATURA

[1] 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.

[2] 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.

[3] 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

[4] 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/

[5] 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

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

[7] 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

[8] 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

[9] 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

[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] 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

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

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

[14] 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

[15] 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/

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

[17] 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/

[18] 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/

[19] 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

[20] 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/

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

[22] 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

[23] 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

[24] HASSAN, Muneeb ul. *VGG16 – Convolutional Network for Classification and Detection* [online]. 2020 [cit. 2021-04-12]. Dostupné z: https://neurohive.io/en/popular-networks/vgg16/

[25] 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

[26] 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

[27] KURAMA, Vihar. *A Review of Popular Deep Learning Architectures: AlexNet, VGG16, and GoogleNet* [online]. 2020. [cit. 2021-04-14]. Dostupné z: https://blog.paperspace.com/popular-deep-learning-architectures-alexnet-vgg-googlenet/

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

[29] 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

[30] 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

[31] KAWATSU, Chris, Aaron ZHAO, Frank V. KOSS a Jacob CROSSMAN. *Gesture Recognition for Robotic Control Using Deep Learning* [online]. 2017 [cit. 2021-05-01]. Dostupné z: https://www.researchgate.net/figure/Comparison-of-popular-CNN-architectures-The-vertical-axis-shows-top-1-accuracy-on_fig2_320084139

[32] 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

[33] 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

[34] 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

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

[36] 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

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

[38] 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

[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] 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

[41] KANDEL, Ibrahem. *The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset* [online]. 2020 [cit. 2021-4-9]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2405959519303455

[42] 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

[43] A KATANFOROOSH, Kian, Daniel KUNIN a Jiaju MA. *Parameter optimization in neural networks* [online]. 2021 [cit. 2021-5-1]. Dostupné z: https://www.deeplearning.ai/ai-notes/optimization/

[44] BUSHEAEV, Vitaly. *Adam — latest trends in deep learning optimization.* [online]. 2018 [cit. 2021-5-1]. Dostupné z: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

SEZNAM SYMBOLŮ A ZKRATEK

Zkratky:

FEKT	Fakulta elektrotechniky a komunikačních technologií
VUT	Vysoké učení technické v Brně
TP	Skutečně pozitivní
FP	Falešně pozitivní
TN	Skutečně negativní
FN	Falešně negativní

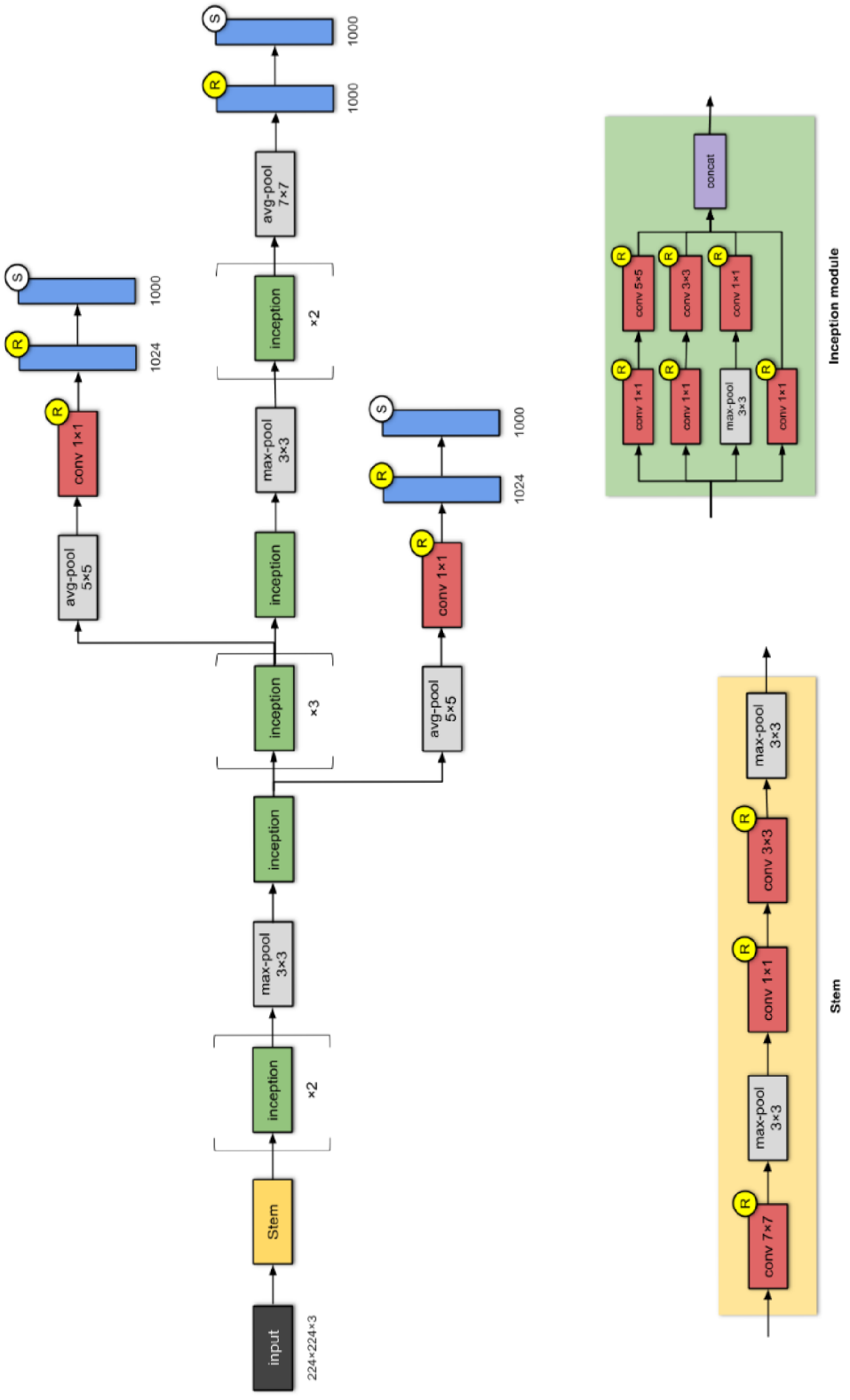
Symboly:

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

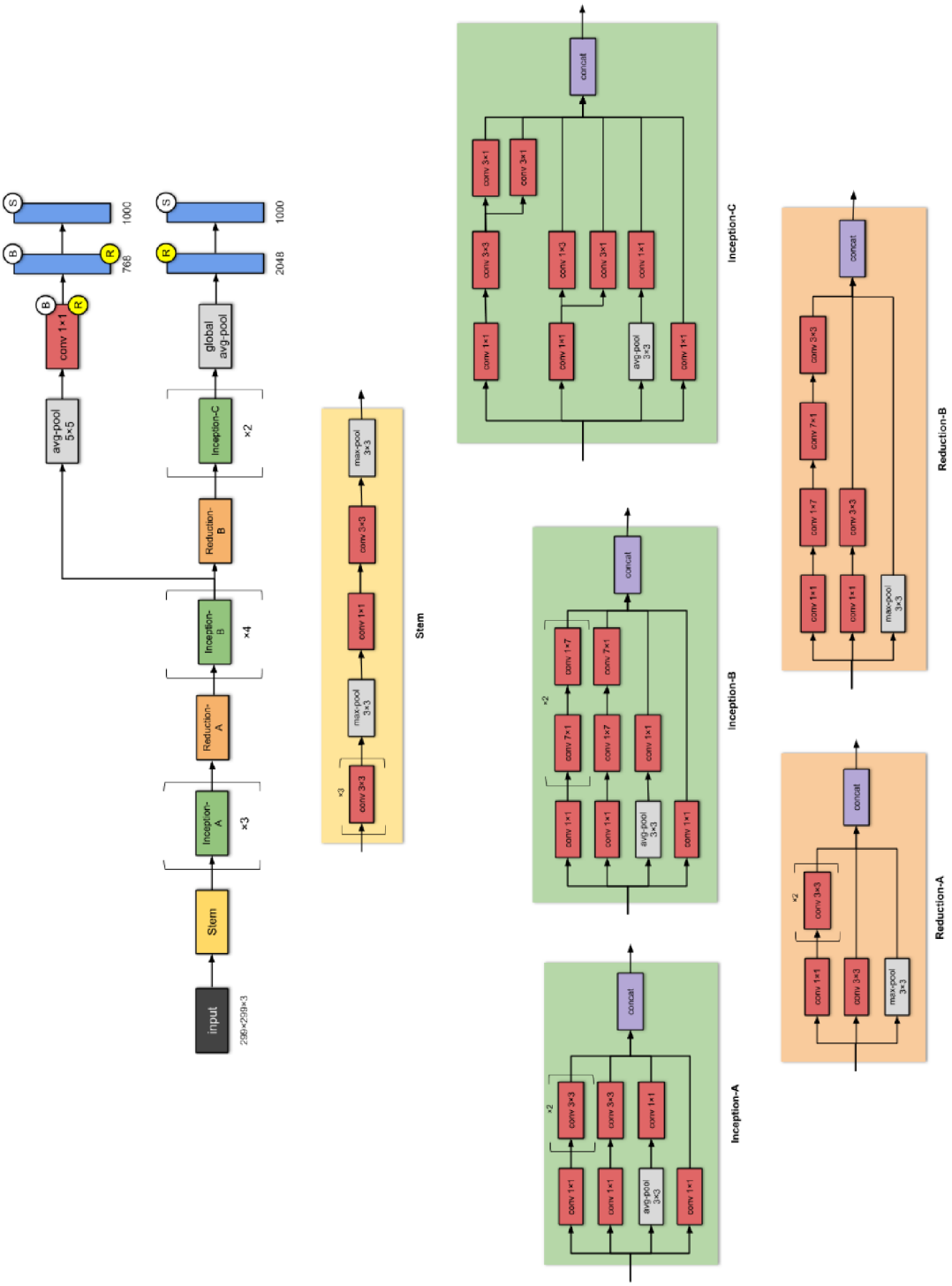
$p_N(x; \mu, \Sigma)$	- Gausova distribuce
n_{freeN}	- počet volných parametrů v modelu N
N_{Mog}	- počet Gaussianů
α_j	- směšovací koeficienty
$p_N(x; \mu j, \Sigma j)$	- Gaussova distribuce
$p_{MoG}(x)$	- směs Gaussianů reprezentovaná lineární kombinací normálních distribucí
N_{Mog}	- počet Gaussianů
$n_{freeMOG}$	- 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

SEZNAM PŘÍLOH

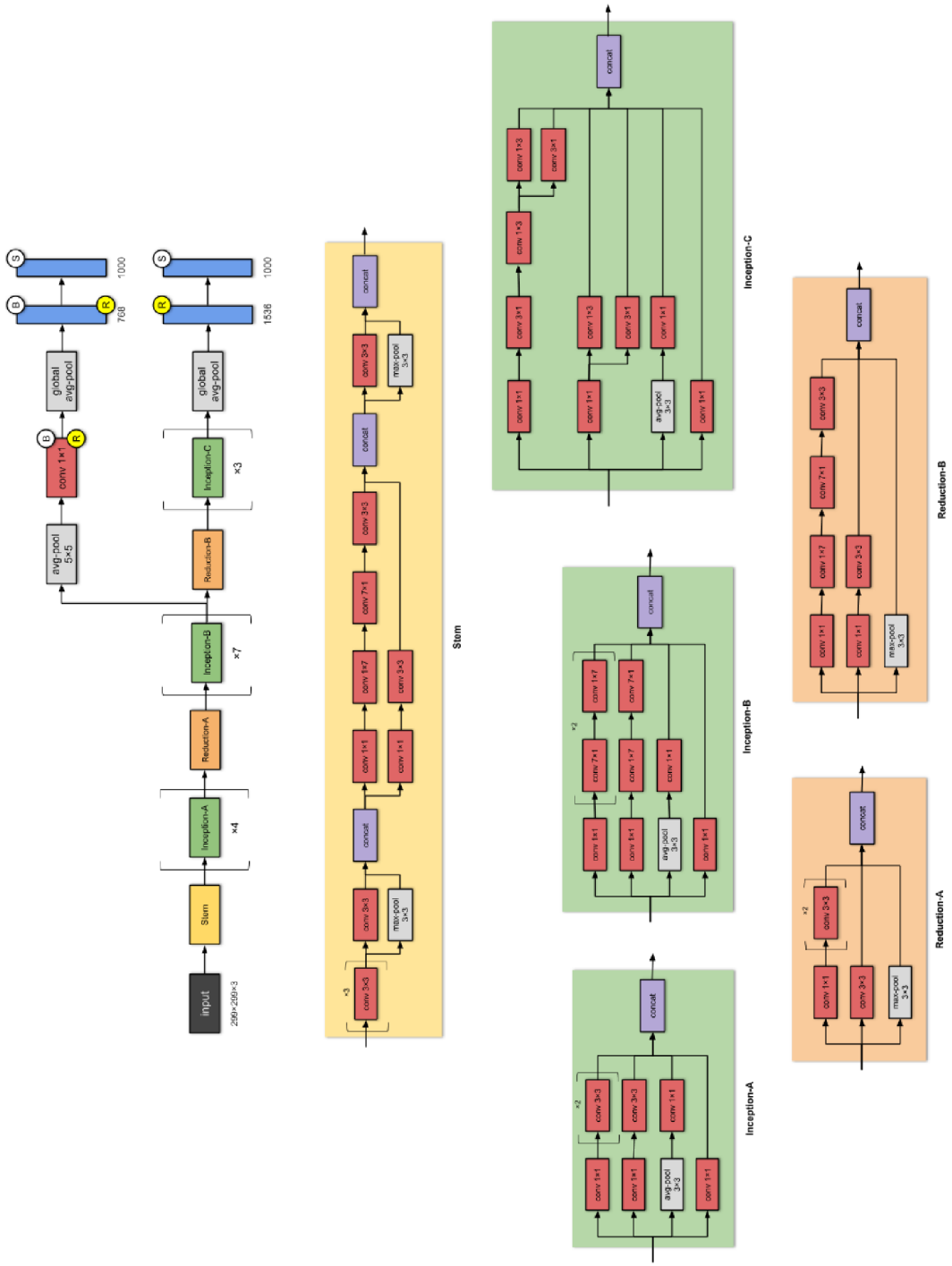
Příloha A - Inception v1 architektura [13]



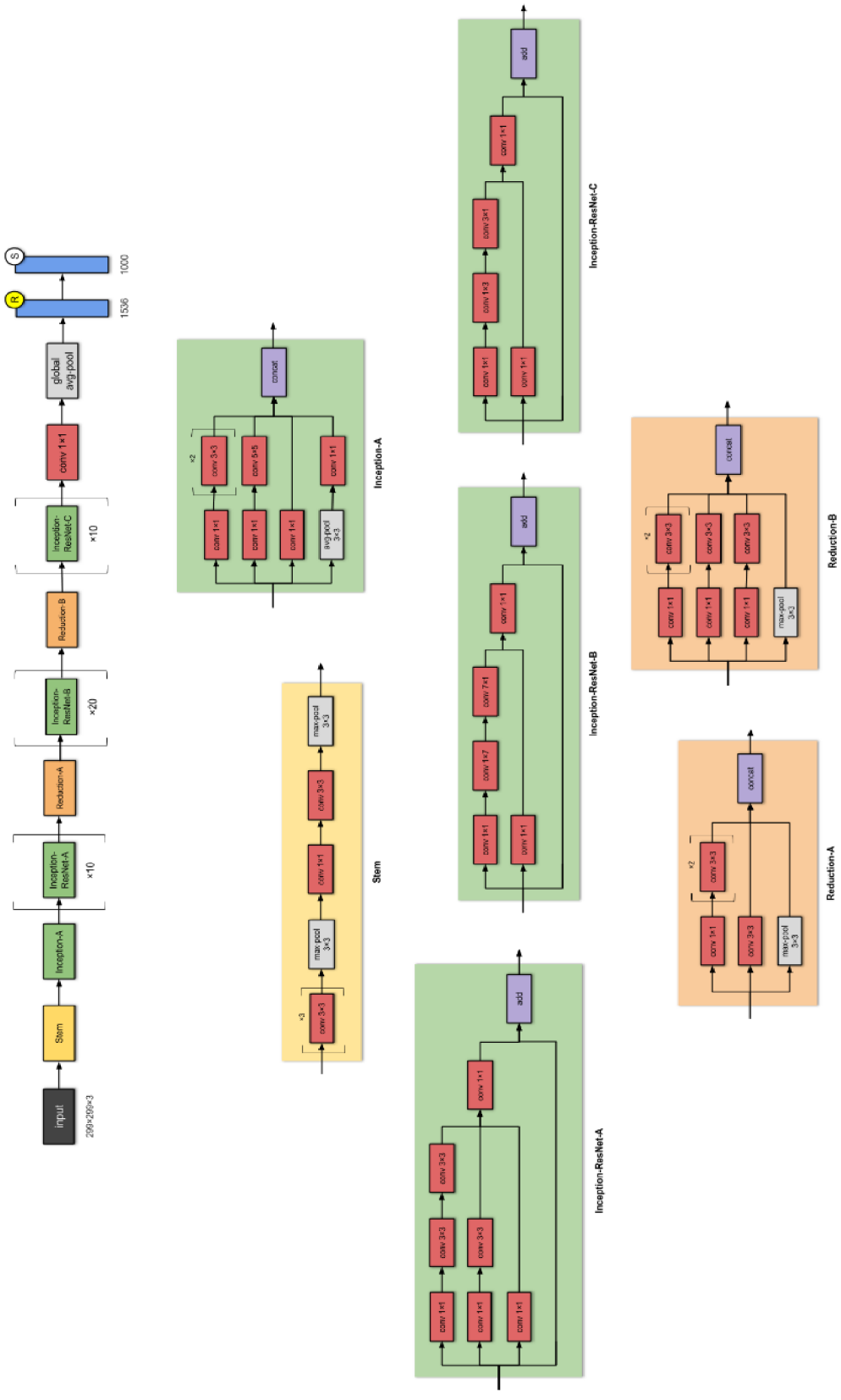
Příloha B - Inception v3 architektura [13]



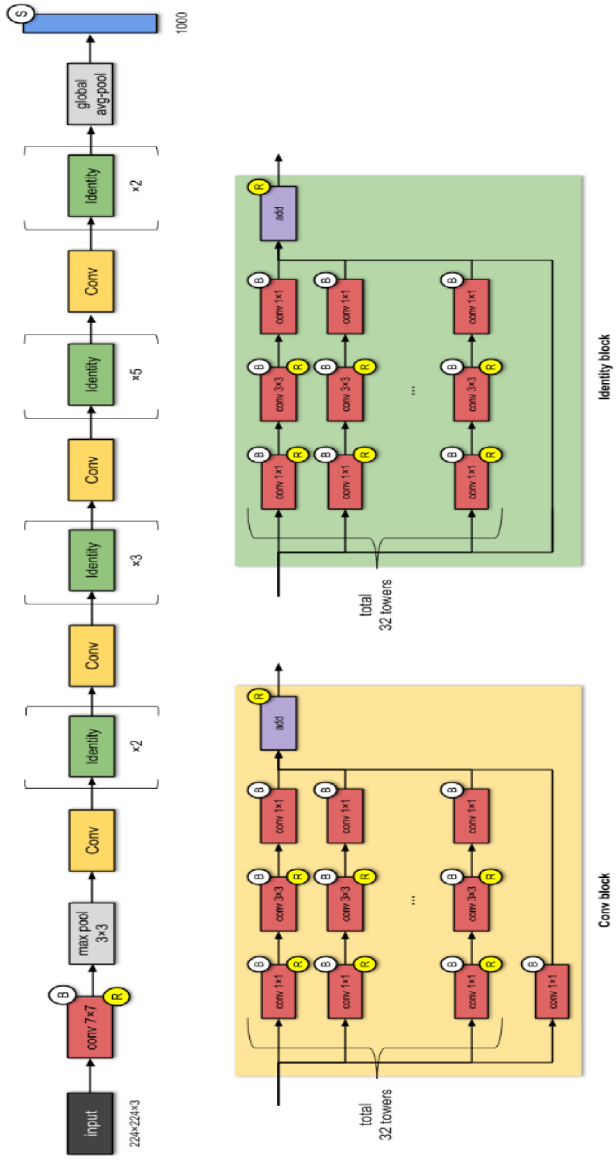
Příloha C - Inception v4 architektura [13]



Příloha D - Inception ResNets architektura [13]



Příloha E - ResNeXt-50 architektura [13]



Příloha F - GoogleNet architektura [28]

Příloha G - Odevzdané programy

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