VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

AUTOMATICKÉ ČTENÍ SPZ S VYUŽITÍM TECHNIK HLUBOKÉHO UČENÍ
NUMBER PLATE RECOGNITION USING DEEP LEARNING TECHNIQUES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Ladislav Dobrovský

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Radomil Matoušek, Ph.D.

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Ladislav Dobrovský**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Radomil Matoušek, Ph.D.**
Akademický rok:	2017/18

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Automatické čtení SPZ s využitím technik hlubokého učení

Stručná charakteristika problematiky úkolu:

S využitím technik hlubokého učení, navrhnete systém pro automatické rozpoznávání SPZ automobilů. Aplikace bude zahrnovat jak vlastni zpracování dat z kamery umístěné ve vhodném testovacím prostředí, tak vyhodnocení obrazu s automatickou identifikací a rozpoznáním SPZ.

Cíle diplomové práce:

1. Rešerše knihoven pro strojové učení – hluboké učení. Zahrnut musí být framework TensorFlow.
2. Zpracování dat z kamery, klient/server aplikace pro načtení obrazových dat.
3. Implementace (návrh, učení, verifikace) hlubokého učení pro rozpoznávání SPZ.
4. Vyhodnocení efektivnosti, diskuze k vlastnímu řešení a případnému dalšímu rozšíření např. o identifikaci typu automobilu.

Seznam doporučené literatury:

WEIGEL, Van B. Deep learning for a digital age: technology's untapped potential to enrich higher education. San Francisco: Jossey-Bass, c2002. Jossey-Bass higher and adult education series. ISBN 0-7879-5613-9.

VATSA, Mayank, Richa SINGH a Angshul MAJUMDAR. Deep learning in biometrics. Boca Raton: CRC Press, 2018.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2017/18

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Předmětem této práce je rešerše hlubokých konvolučních umělých neuronových sítí a jejich využití na řešení problému automatického rozpoznávání státních poznávacích značek. Po přehledu teorie a současných trendů byl určen další směr vývoje. Práce zkoumá několik typů konvolučních neuronových sítí a komplexní architekturu systému spolupracujících aplikací. V praktické části je popsána implementace a výsledky experimentů se zhodnocením vhodnosti zkoumaných sítí pro praktické využití.

ABSTRACT

Focus of this thesis is research of deep convolutional artificial neural networks and their usage as solution to automatic license plate recognition problem. After summary of theory and current trends the further direction of development has been chosen. Thesis investigates several types of convolutional neural networks and complex architecture of system of collaborating applications. In practical part is described implementation and results of experiments with assessment of networks' suitability for practical use.

KLÍČOVÁ SLOVA

rozpoznávání registračních značek; státní poznávací značky; umělé neuronové sítě; konvoluční neuronové sítě; hluboké učení; detekce objektů

KEYWORDS

license plate recognition; artificial neural networks; convolutional neural networks; deep learning; object detection

BIBLIOGRAFICKÁ CITACE

DOBROVSKÝ, L. *Automatické čtení SPZ s využitím technik hlubokého učení*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2018. 80 s. Vedoucí diplomové práce doc. Ing. Radomil Matoušek, Ph.D..

PODĚKOVÁNÍ

Tímto bych rád poděkoval svému vedoucímu doc. Ing. Radomilu Matouškovi, PhD, rodině a přátelům za extrémní podporu při studiu a vypracování této diplomové práce.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením doc. Ing. Radomila Matouška, PhD a s použitím literatury uvedené v seznamu literatury.

V Brně dne 22. 3. 2018 .....

Ladislav Dobrovský

OBSAH

1 ÚVOD

V dnešním rozvinutém světě se prosazuje stále více praktické nasazení umělé inteligence. Zvyšuje se její vliv od chytrých telefonů až po chytrá města a továrny. Skloňují se pojmy jako strojové učení (*Machine Learning*) a hluboké učení (*Deep Learning*). Jako nejčastější odůvodnění zaznívají hesla: bezpečnost, produktivita, pokrok a zvýšení kvality lidského života. Pokud vůbec připustíme, že lze kvalitu lidského života nějak měřit, pak zde figuruje lékařská péče, pohodlí, jistoty a svoboda. Pro lékařskou péči je umělá inteligence nasazována u vyhodnocení testů, zpracování velkých dat a dolování znalostí z nich. Do kategorie pohodlí lze zařadit vše od aplikace chytrého telefonu, která pomocí umělé inteligence identifikuje interpreta písně hrající z rozhlasu, až po kooperujícího robota v továrně. To se už dotýkáme dalších hesel, tedy produktivity a pokroku. Zbývá bezpečnost a svoboda. Pro bezpečnost se metody umělé inteligence nasazují stále častěji. Nasazení automatického čtení SPZ je motivováno bezpečností a zvýšeným pohodlím řidičů i správců majetku. Tento technický problém je řešený v této práci poměrně novými technikami hlubokého učení (hlubokých neuronových sítí). Výpočetní prostředky umělých neuronových sítí jsou inspirované fungováním biologických mozků, resp. struktury biologické neuronové sítě a neuronů samotných. Možnosti těchto nových přístupů stále rostou spolu s rostoucím množstvím dat, na kterých je lze učit, a rostoucím výpočetním výkonem počítačových čipů. Klesá význam ručně laděných algoritmů počítačového vidění a stále častěji se těmto hlubokým neuronovým sítím předkládají surová data. Tento trend ukazuje cestu k opravdové umělé inteligenci schopné samostatného učení a vnímání světa.

Doufejme, že další směr vývoje povede spíše k poslednímu zmíněnému kriteriu kvality lidského života, tedy ke svobodě - zvyšování možností projevu jedince.

Řešené téma – detekce SPZ – lze využít v rámci parkovacích domů, areálů veřejných institucí a firem pro usnadnění kontroly identity a chytrá města v konceptu *Smart Cities* jak pro zvýšení bezpečnosti, tak pro zvýšení pohodlí. Tohoto tématu se také dotýká zavádění nové legislativy EU GDPR (Obecné nařízení o ochraně osobních údajů).

Další vývoj ve světě praktického nasazení metod umělé inteligence lze jen těžko předvídat. Mnohé možné aplikace jsou zneužitelné pro nevídané autonomní zbraňové systémy. Doufejme, že budoucnost skýtá více možností mírumilovné prosperity než souboj o téměř vyčerpané přírodní zdroje se samočinnými vraždícími stroji ve společnosti velkých bratrů.

2 REGISTRAČNÍ ZNAČKY VOZIDEL V ČR

Státní poznávací značky silničního motorového vozidla a přípojného vozidla (registrační značky – RZ) používané v ČR určuje zákon č. 56/2001 Sb., o podmínkách provozu vozidel na pozemních komunikacích. Dále registrační značky upřesňuje vyhláška č. 343/2014 Sb. Některé starší registrační značky vydané v Československu a ČR zůstávají v platnosti, jejich výskyt se však velmi snižuje.

Nejčastěji jsou na automobilech instalovány jednořádkové RZ o rozměru 520x110 mm v provedení černého textu na bílém pozadí se 7 znaky. Rozpoznáváním právě tohoto formátu se práce zabývá.

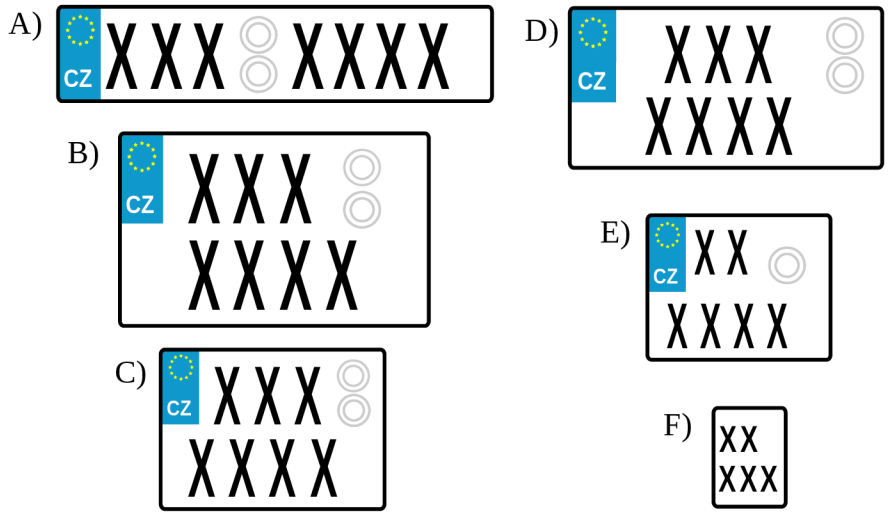
Povolené formáty registračních značek jsou (odpovídá obrázku 1):

A) jednořádková 520x110 mm se 7 znaky
B) dvouřádková 340x200 mm se 7 znaky,
C) dvouřádková 280x200 mm se 7 znaky,
D) dvouřádková 320x160 mm se 7 znaky,
E) dvouřádková 200x160 mm se 6 znaky,
F) dvouřádková 80x110 mm s 5 znaky.

Při kombinaci se pak používají vzory:

- 101: 2x A – osobní automobily
- 102: 1x A – přípojná vozidla
- 103: 1x A, 1x B – nákladní vozidla
- 104: 1x B – přípojná vozidla
- 105: 2x C – osobní automobily
- 106: 1x A, 1x C – osobní automobily
- 115: 2x D – osobní automobily, americký formát
- 116: 1x A, 1x D – osobní automobily, americký formát
- 118: 1x E – motocykly
- 119: 1x F – mopedy

Ochranným prvkům a kontrolním nálepkám přítomným na registračních značkách se tato práce nevěnuje. Na zadní RZ je údaj o platnosti STK, případně měření emisí.



Obr. 1: Registrační značky motorových vozidel v ČR

Na značkách jsou povoleny alfa-numerické znaky s výjimkou písmen **G**, **O**, **Q** a **W**, aby se předešlo záměnám. V minulosti se písmeno G na značkách objevovalo pro okres Gottwaldov, tyto značky však byly vyměněny. Od roku 2016 je možné si za příplatek zažádat o značku s vlastním textem – značky na přání – mají **8** znaků. Musí obsahovat alespoň jednu číslici a nesmí obsahovat nevhodné výrazy (vhodnost schvaluje úřad s pomocí seznamu nevhodných slov a frází). Kromě značek na přání je obvyklý formát takový, že první tři znaky udávají sérii vydávanou krajem a první vyskytující se písmeno udává kraj. Následující 4 znaky jsou obvykle číslice. Značky na přání výrazně zvyšují obtížnost řešení zadané úlohy. Rozšíření o čtení osmého znaku pro značky na přání není předmětem této diplomové práce.

Použitý řez písma (font) není přesně specifikovaný v zákoně, ovšem výrobce používá font shodný s bitmapovým fontem na obrázku 2.

Obr. 2: Použitý bitmapový font ekvivalentní s fontem používaným na SPZ

Registrační značky pro zvláštní vozidla (traktory, pracovní stroje a jejich přípojná vozidla) mají žluté pozadí. Značky vozidel pro vývoz mají červený pruh. Značky diplomatických vozidel jsou rozlišeny modrým písmem a rámem, dříve žluté písmo na modrém pozadí. Registrační značky trvale manipulační mají zelené písmo a rám.

Zkušební vozy mají také zelené písmo a rám, jejich značky mají místo první trojice písmeno F. Zelené písmo i rám platí také pro historická vozidla, které mají místo první trojice písmeno V jako veterán.

2.1 Registrační značky používané v zahraničí

V zahraničí jsou používány různé značky počtem a umístěním znaků, některé státy nesjednocují používaný řez písma. Častý je počet symbolů 5 až 8. Jejich rozpoznávání není předmětem této práce.

Obr. 3: Ukázka zahraničních registračních značek

3 METODY DETEKCE A ČTENÍ SPZ (RZ)

3.1 Klasické metody zpracování obrazu

Počátky vývoje metod počítačového vidění a zpracování obrazu sahají do druhé poloviny šedesátých let dvacátého století. Nejvyšší metou je zde neutuchající snaha o „pochopení“ informací obsažených v obraze přímo strojem, případně s doplněním jiných 3D dat o prostředí. Ručně psané algoritmy pro detekci hran a jednodušších vzorů znamenaly mnohé aplikace ve výzkumu a průmyslu (automatické linky s detekcí vizuálních markerů). Rekonstrukce 3D objektů a automatické polygonální modelování z těchto hran také slavilo určitý úspěch. Další snahy detekovaly vzory textury povrchů, informace zaostření čočky kamery a tedy detekci rozostřených částí obrazu pro určení vzdálenosti od kamery, využití informací o stínech, skládání panoramatických snímků, detekce pohybujících se objektů, segmentace obrazu atd. V této kapitole je popsáno několik metod relevantních pro detekci registrační značky silničních vozidel. V současnosti se pro nové velmi složité problémy upouští od ručně psaných a laděných algoritmů ve prospěch možností strojového a hlubokého učení.

3.1.1 Algoritmus Viola-Jones

Tento algoritmus byl představen Paulem Violou a Michaelem Jonesem v roce 2001. Původně se jednalo o detektor tváří využívající dvourozměrné Haarovy vlnky (Haar wavelet), kterými hledá v obraze typické přechody intenzity pro hledaný objekt. Tyto se nazývají Haar-like features (příznaky) a jejich kombinací v kaskádě vzniká výsledný detektor. Takto lze tedy v obrazu registrační značku lokalizovat a je třeba dále provést rozpoznání znaků OCR s případnou prostorovou transformací. Při trénování klasifikátoru se běžně používá algoritmus AdaBoost, který vybírá podmnožinu příznaků, a celý proces trénování zrychluje.

3.1.2 Local Binary Patterns (LBP)

Metoda lokálních binárních vzorů pracuje s histogramem pro lokální okolí bodu. Obrázek musí být převeden do stupňů šedi a musí být provedeno prahování. Relativní výhodou je, že nalezené příznaky jsou invariantní k rotaci.

3.1.3 Histogram of oriented gradients (HOG)

Přestože byl algoritmus popsán už v roce 1984 v žádosti o patent, dočkal se většího rozšíření až po roce 2005. Vychází z předpokladu, že lze popsat příznak v obrazu distribucí gradientů a natočením hran v buňkách, do kterých je obraz rozdělen. Implementuje se pomocí Support Vector machines (SVM). [21]

3.1.4 Umělé neuronové sítě (ANN)

Klasické dopředné neuronové sítě [1] s nižším počtem plně propojených skrytých vrstev se s úspěchem používají pro mnoho aplikací, včetně čtení registračních značek vozidel. Tyto jednodušší klasifikátory však vyžadují klasické zpracování obrazu, transformace a často se využívají až pro samotné čtení značky OCR znak po znaku z příznaků, které jsou z obrazu extrahovány ručně psaným a laděným algoritmem. Tedy oproti dále diskutovaným metodám hlubokého učení (deep learning) přestavují překonaný článek vývoje, který projevuje v poslední době naplnění svého potenciálu. Ovšem ani hluboké neuronové sítě ještě dostatečně neřeší problémy jako je cílená snaha o jejich zmatení. [9]

3.1.5 OpenALPR

Open Automatic License Plate Recognition je open source C++ knihovna pro rozpoznávání registračních značek vozidel klasickými metodami. Podle dokumentace využívá LBP algoritmus, Savuola binarization method [6] a HOG detektory. [16] Po nalezení registrační značky v obraze je provedena prostorová transformace tak, aby značka byla čtena přímo kolmo. Pak probíhá samotné čtení jednotlivých znaků OCR. Výsledkem je několik možností textu značky ohodnocených pravděpodobnostmi. Detektory lze dodatečně trénovat pro lepší rozpoznání určitého typu značky. Ve výchozím nastavení jsou detektory obecné, aby detekovaly značky používané v mnoha zemích, je tedy žádoucí při použití v ČR provést dodatečný trénink. Tato knihovna má též komerční rozšířenou verzi s lepším škálováním a akcelerací. Také je nabízena na ní založená placená cloudová služba (SaaS).

3.2 Čtení a zpracování obrazu z kamery

Pro prvotní získání obrazových dat z kamery je vhodné použít jednoduché porovnávání změny po sobě následujících snímků a od určité hodnoty míry rozdílnosti je archivovat pro ruční zpracování a vytvoření trénovací množiny. K tomu stačí jednodušší skripty běžící na pracovní stanici. Dále je pro nasazení výsledné aplikace nezbytné vytvořit službu běžící na serveru, která už bude využívat natrénovaný detektor automobilů a značek. Je nutné zajistit dostatečný výkon pro zpracování v reálném čase, případně s použitím více vláken běžících na více procesorových jádrech. Také je třeba uvažovat možnou chybovost a selhání spojení při čekání na snímek a automaticky službu anebo připojení ke kameře restartovat.

Připojení kamery k počítači se zajišťuje buď po Ethernetu tzv. IP kamerou nebo USB. Kvůli možným fyzickým vzdálenostem serveru a kamery je vhodnější volbou právě IP kamera.

Pro potřeby hlubokého učení není obraz zpracován, kromě ořezu důležité části. Pro zvýšení výkonu můžeme uvažovat o snížení rozlišení, pokud bude zachována čitelnost symbolů na registrační značce vozidla.

Pro sběr dat a testovací účely byla IP kamera umístěná za oknem v interiéru.

Obr. 4: Použitá IP kamera Hikivision DS-2CD2642FWD-IZS

4 DEEP LEARNING – HLUBOKÉ UČENÍ

Jako *Deep Learning* (DL) se označuje aplikace umělých neuronových sítí (ANN) s vysokým počtem skrytých vrstev (*Deep Neural Network*). Vstupní vrstva se vyznačuje vysokým počtem neuronů. S úspěchem se využívají kombinace vrstev konvolucí a *subsamplingu* při zpracování vstupů s vysokým rozlišením. Se zvyšováním výkonu paralelního výpočetního HW roste složitost a velikost sítí a úloh, které lze efektivně řešit. Mezi významné milníky vývoje patří konvoluční síť AlexNet, která se v soutěži ImageNet Large Scale Visual Recognition Challenge v roce 2012 umístila první s výrazným náskokem. Dalšími úspěchy nasazení DL jsou sítě AlphaGo a AlphaZero, které jsou v současnosti jedny z nejlepších programů hrajících stolní hry go a šachy. Pojem DL není vzhledem k počtu skrytých vrstev v síti přesně definován. O hlubokých neuronových sítích se začalo hovořit při třech a více plně propojených vrstvách. S nástupem konvolučních sítí se začal počet vrstev rapidně zvyšovat a v současnosti jsou používány i sítě s více než 1000 skrytými vrstvami pro zvládnutí datových množin jako je CIFAR-10.

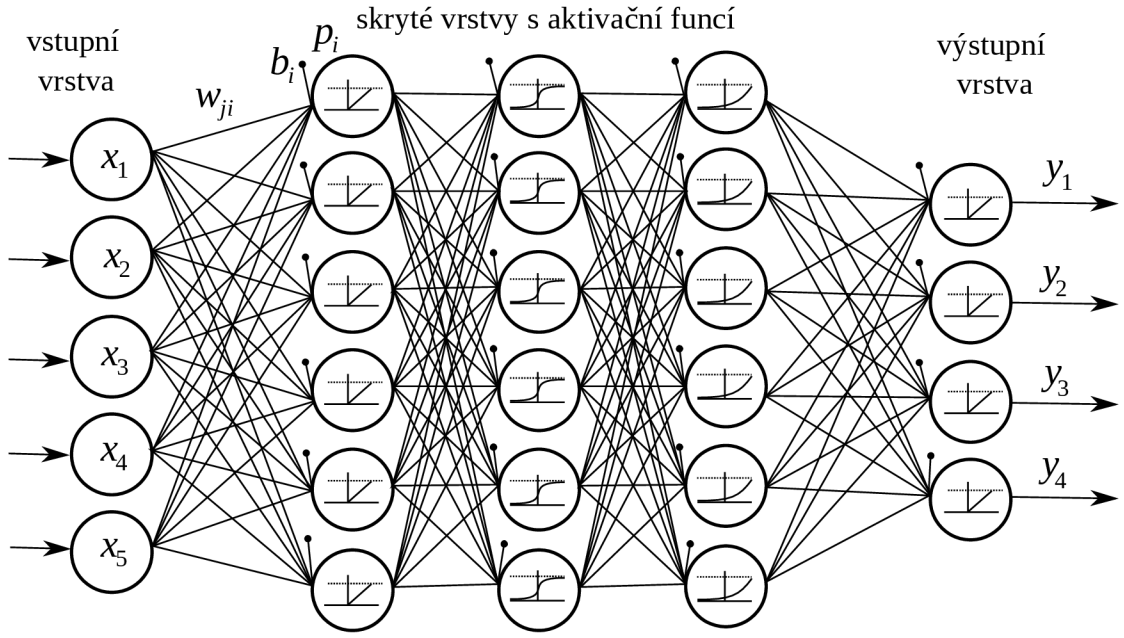
4.1 Dopředné neuronové sítě (feedforward ANN)

V těchto sítích je průběh zpracování výpočtu a aktivace umělých neuronů ve vrstvách, které jdou po sobě. V grafu reprezentace sítě nevznikají žadné smyčky (sítě se smyčkami se označují jako recurent neural network RNN a jde o něco jiného než dále popsané R-CNN). Mezi jednotlivými vrstvami umělých neuronů (perceptronů) s potenciálem p_i a biasem b_i jsou hranami grafu sítě znázorněny spojení, kterým jsou přiřazeny hodnoty vah w_{ji}. Vstupní vrstva neuronů má vstupem určený potenciál x_i. Výstup poslední (výstupní) vrstvy se značí y_i. Nastavením těchto vah se mění účinek výstupu neuronů, které sami mají určitou aktivační funkci $\varphi_i(\mathbf{x})$. V průběhu vývoje se aplikovali pro změnu nastavení vah různé optimalizační přístupy jako genetické algoritmy a stochastický sestup po gradientu. Některé tyto techniky lze kombinovat. Pro výpočet gradientu chyby učení se používá algoritmus zpětného šíření *backpropagation*, který lze využít i pro samotné učení. U reálných aplikací hlubokého učení jsou stochastické přístupy nezbytné pro dosažení dostatečné přesnosti. Práce se zabývá učením s učitelem (supervised learning), které spočívá ve využití množiny známých vstupů a jejich obrazů tak, aby je výsledná funkce neuronové sítě přiřazovala správně. Při dostatečné velikosti a vhodném pokrytí oblasti úlohy je síť schopná správně předpovědět obrazy vstupů, které nebyly v trénovací množině, a tím se úspěšně stává nástrojem pro rozpoznávání vzorů.

Neuronovou síť *N* lze zapsat jako vektorovou funkci:

$$\mathbf{y} = f_N(\mathbf{x}) = f_N(\mathbf{x}, \mathbf{b}, \mathbf{w}, \boldsymbol{\varphi}) \tag{1}$$

kde f_N je funkce celé neuronové sítě *N* s konfigurací matice biasů **b**, matice vah **w**, a matice aktivačních funkcí neuronů **φ** pro vstupní vektor **x**. Matice **b** a **w** jsou parametry při učení a optimalizaci. Volba aktivačních funkcí **φ** může být předmětem optimalizace, většinou je volena expertně.

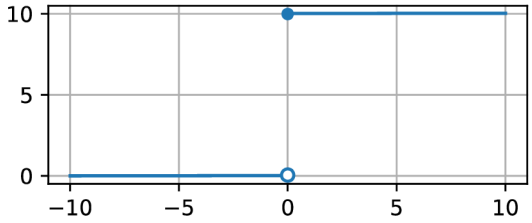


Obr. 5: Příklad dopředné neuronové sítě (Feed Forward Neural Network, FFNN) s pěti vstupními a čtyřmi výstupními neurony a třemi skrytými vrstvami

4.1.1 Aktivační funkce

Požadavky kladené na výběr funkce se různí podle rozsahu vstupních a výstupních hodnot, typu neuronové sítě, resp. modelu neuronu a metod učení (např. požadavky na spojitost a diferencovatelnost). V kontextu biologického vzoru je základním požadavkem monotónnost aktivační funkce. S výjimkou specifických lineárních sítí typu ADELINE, MADELINE se vždy využívají nelineární aktivační funkce. Jednou z dodržovaných praktik je, že aktivační funkce jsou pro celou vrstvu stejné. V určitých případech lze vytvořit dvě paralelní vrstvy a pro každou mít jinou aktivační funkci. Dále jsou popsány vybrané aktivační funkce. Poznamenejme, že pro deep learning a sítě použité v této práci má největší význam funkce označená jako ReLU.

Jednotkový skok byla první prakticky využitá aktivační funkce u tzv. McCullochova a Pitsova modelu neuronu (1943). Pro potřeby vícevrstvých neuronových sítí, vč. hlubokého učení, se však nehodí. Důvodem je existence nespojitosti funkce a využití gradientního algoritmu učení implementovaného v metodě *backpropagation* (zpětné šíření chyby). Jelikož je funkce v bodě $x = 0$ nediferencovatelná a zároveň je na zbytku definičního oboru derivace rovna 0, nemá smysl uvažovat pro učení metody založené na gradientu. Tuto aktivační funkci lze však využít pro aplikace učené pomocí genetického algoritmu a jiných metod, které neřeší gradienty, ale vnímají síť pouze jako problém optimalizace vah pro minimalizaci účelové funkce chyby výstupu.



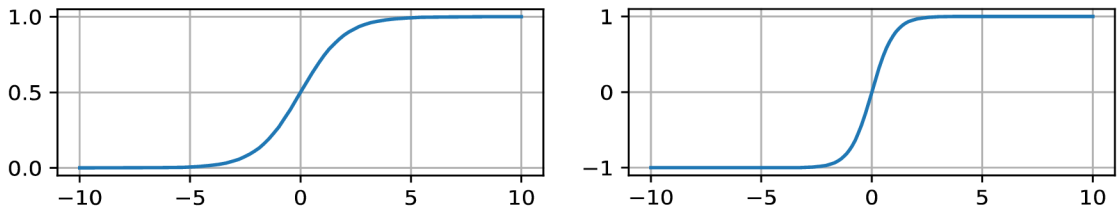
Obr. 6: Aktivační funkce jednotkový skok

$$\varphi(x) = \begin{cases} 0 & \text{pro } x < 0 \\ 1 & \text{pro } x \geq 0 \end{cases} \tag{2}$$

Sigmoida (3) je spojitá diferencovatelná funkce s výstupem v intervalu (0; 1), tím je vhodná pro gradientní přístup optimalizace a metodu *backpropagation*. Její derivace není monotónní a může způsobit uváznutí při trénování. V kontextu neuronových sítí (*perceptron*) se jedná o základní a frekventovaně využívaný model aktivační funkce neuronu, který se přibližuje biologickému modulu a současně umožňuje uvedenou matematickou manipulaci ve smyslu gradientní optimalizace. V případě požadavku na bipolární výstup je uvedený model realizován např. pomocí funkce **hyperbolický tangens** (4) s výstupem v intervalu (-1; 1).

$$\varphi(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{3}$$

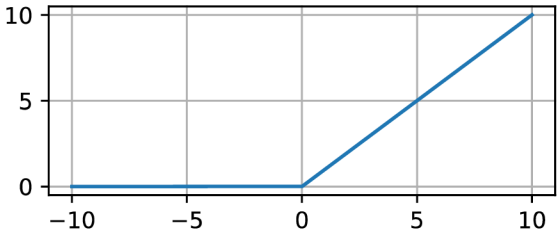
$$\varphi(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \tag{4}$$



Obr. 7: Aktivační funkce sigmoida a hyperbolický tangents

Rectifier (5) - usměrňovač (**ReLU** – Rectifier Linear Unit). je jednoduchá lineární lomená funkce jejíž použití v umělých neuronových sítích je efektivně obhájeno z hlediska biologické inspirace [14]. Později se ukázala významná vhodnost pro hluboké neuronové sítě a počítačové vidění. Neexistence derivace v bodě 0 se kompenzuje přiřazením $f'(0) = 0$, po této kompenzaci je pak možné použít *backpropagation* a realizovat výpočet gradientu. Zároveň síť získá některé vhodné vlastnosti jako je řídkost příznakových map (feature maps sparsity) [13]. S takto definovanou derivací dosahuje její monotónnosti, což je patrně největší výhoda při učení metodami založenými na gradientech. Funkce ReLU má několik variant s dalším specifickým využitím pro neuronové sítě, jsou to Noisy ReLU, Leaky ReLU, a ELU.

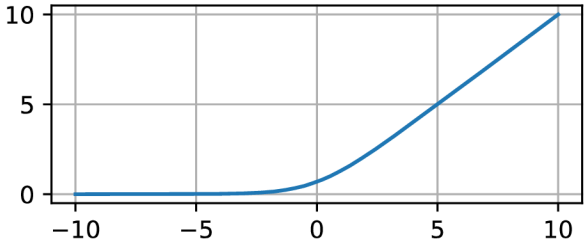
$$\varphi = \begin{cases} 0 & \text{pro } x < 0 \\ x & \text{pro } x \geq 0 \end{cases} \tag{5}$$



Obr. 8: Aktivační funkce ReLU

Analytická funkce (SoftPlus) (6) je hladkou aproximací ReLU. Vzhledem k definici vyžaduje složitější (časově náročnější) výpočet. Její derivací je logistická funkce, resp. sigmoida, viz výše.

$$\varphi(x) = \ln(1 + e^{x}) \tag{6}$$



Obr. 9: Aktivační funkce SoftPlus

4.1.2 Zpětné šíření – backpropagation

Pomocí metody zpětného šíření - *backpropagation* - je počítána chyba výstupu *y* u jednotlivých vzorů v trénovací množině a gradient této chyby s příspěvkem jednotlivých neuronů ve vrstvách aplikací řetízkového pravidla. Základní kvadratická chybová funkce *E* podle vzorce (7), pro *n* vzorků (8) a její parciální derivace (9). Jedna polovina je do vzorce chybové funkce volena proto, aby kompenzovala derivaci druhé mocniny a celý výpočet pro výstupní vrstvu se tak zjednodušil.

$$E(y,y')=\frac{1}{2}\|y(x)-y'(x)\|^2 \tag{7}$$

$$E=\frac{1}{2n}\sum_x \|y(x)-y'(x)\|^2 \tag{8}$$

$$\frac{\partial E}{\partial y'}=y'-y \tag{9}$$

Výstup neuronu *j*, o_j, s aktivační funkcí φ a vstupy o_k s nastavením vah w_{kj} jako suma u_j:

$$o_j=\varphi(u_j)=\varphi\left(\sum_{k=1}^{n} w_{kj}o_k\right) \tag{10}$$

Pak pro parciální derivaci chybové funkce vzhledem k váze w_{kj} platí vztah:

$$\frac{\partial E}{\partial w_{ij}}=\delta_j o_i \tag{11}$$

Kde $_j$ je rekurzivní funkcí parciální derivace chybové funkce vzhledem k výstupu neuronu *j*:

$$\delta_j=\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial u_j}=\begin{cases}\dfrac{\partial E}{\partial u_j}\ \varphi(u_j) & \text{pro } j \text{ výstupní}, o_j=y \\ \displaystyle\sum_{\lambda\in L}\left(\frac{\partial E}{\partial o_\lambda}\frac{\partial o_\lambda}{\partial u_\lambda}w_{\lambda j}\right) & \text{pro } j \text{ skryté vrstvy}\end{cases} \tag{12}$$

Nové hodnoty vah w_{ij} se upravují iteračním algoritmem ve směru parciální derivace přímo nebo stochasticky o nastavenou hodnotu parametru η (learning rate):

$$w_{ij,2}=w_{ij,1}+\eta\frac{\partial E}{\partial w_{ij,1}}=w_{ij,1}+\eta\delta_j o_i \tag{13}$$

Pro stochastickou variantu pak platí rozšíření:

$$w_{ij,2}=w_{ij,1}+\eta\frac{\partial E}{\partial w_{ij,1}}+\xi(t)=w_{ij,1}+\eta\delta_j o_i+\xi(t) \tag{14}$$

kde ξ(t) je stochastický člen (term).

Pro klasifikaci do více tříd se využíva jako chybová (*cost*, *loss*) funkce *cross entropy*:

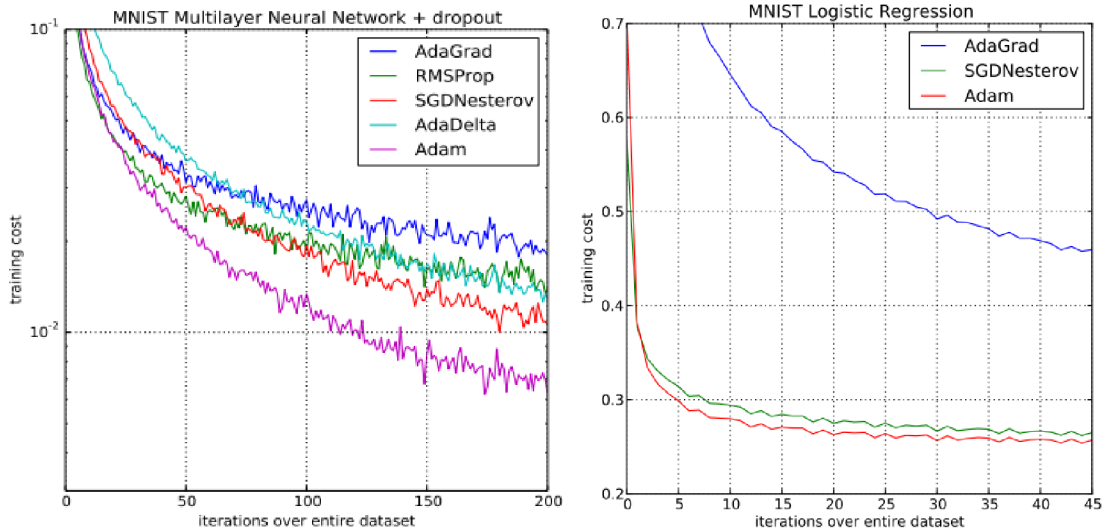
$$E = -\sum_j d_j \log(p_j) \tag{15}$$

kde d_j je pravděpodobnost cíle,
p_j je pravděpodobnost výstupu j po provedení aktivační funkce softmax:

$$p_j = \frac{e^{x_j}}{\sum_k e^{x_k}} \tag{16}$$

4.1.3 Algoritmus Adam

Jedním z nejúspěšnějších obecných (nespecifický pro konkrétní problém) stochastických optimalizačních algoritmů pro trénování hlubokých neuronových sítí je Adam [11]. Je modifikací stochastického sestupu po gradientu a pokouší se o vyhnutí se uváznutí v lokálním minimu. Celkově řeší problémy klasického přímého využití gradientu vypočítaného pomocí *backpropagation*. Adam staví na kombinaci přístupů z algoritmů RMSProp a AdaGrad.
Adaptive Gradient Algorithm (AdaGrad) udržuje *learning rate* odděleně pro jednotlivé parametry a zvládá lépe problémy s takzvanými řídkými gradienty (sparse gradients), kterými se vyznačují mnohé problémy řešené hlubokými sítěmi.
Root Mean Square Propagation (RMSProp) také udržuje *learning rate* odděleně pro jednotlivé parametry a upravuje je adaptabilně na základě průměrování, což zlepšuje výkon u problémů zatížených šumem (noisy). Další rozšíření algoritmu se nazývá AdaMax.



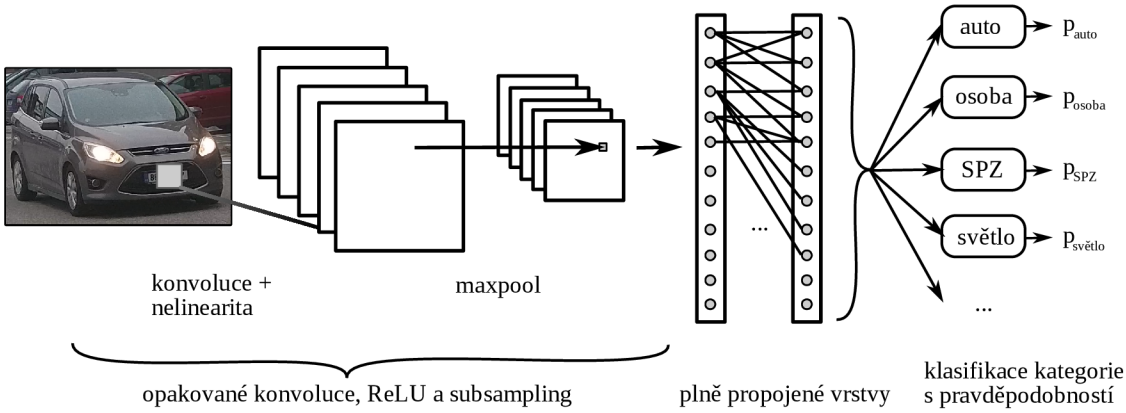
Obr. 10: Adam vítězí v porovnání algoritmů [11]

4.2 Konvoluční neuronové sítě

Jsou právě takové dopředné neuronové sítě, které obsahují vrstvy konvolucí. Ukazuje se, že dobrých výsledků se zatím dosahuje střídáním konvoluce, ReLU a maxpool. V některých případech je vhodné provádět maxpool méně často, než konvoluce a ReLU. Po těchto operacích následuje převod do 1D (flatten) a následují plně propojené vrstvy vhodné velikosti až po výstupní vrstvu, která v případě klasifikátoru obsahuje alespoň tolik neuronů jako je tříd. Při *transfer learningu* často dochází k nevyužití některých výstupních neuronů.

Ukazuje se, že postupné konvoluční vrstvy kódují různé úrovně příznaků (features) a jejich postupné skládání. Tato struktura je inspirovaná pozorovanou biologií vizuálního cortexu mozků zvířat, ve kterých existují skupiny nervových buněk vykazující podráždění určitými hranami ve viděném obraze na určitém místě a pak skupiny buněk, které reagují na vzory vyskytující se kdekoli v obraze. Pro učení sítě se oproti plně propojeným vrstvám výrazně snižuje pomocí konvoluce počet vah (parametrů), které je třeba nastavovat. To činí z učení jednodušší a časově výpočetně lépe zvládnutelný proces.

V některých případech se zařazují *dropout* vrstvy, které mají eliminovat *over-fitting* neuronů a zajistit tak lepší klasifikaci při validaci.

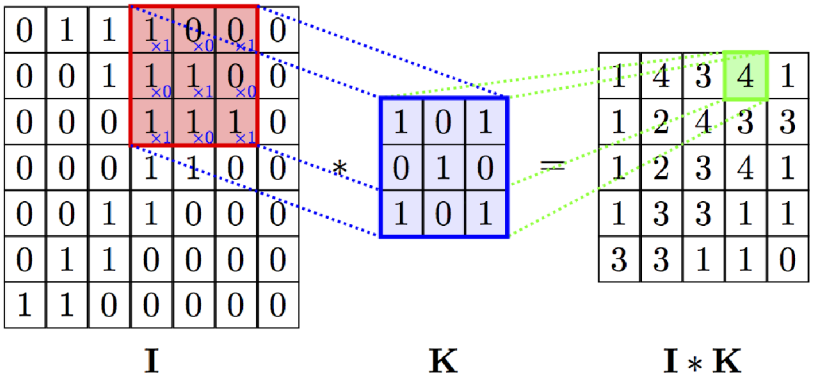


Obr. 11: Architektura konvoluční neuronové sítě pro klasifikaci do více tříd

4.2.1 Konvoluční vrstvy

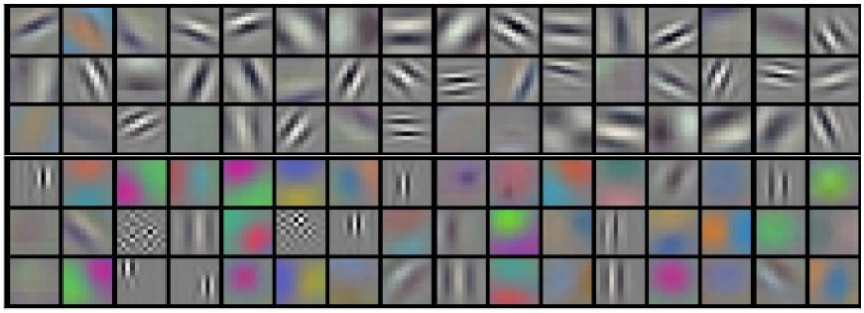
Na obrazu **I** = *f*(*x*, *y*) se provádí dvourozměrná diskrétní konvoluce s jádrem **K** = *h*(*i*, *j*) podle vzorce (17), znázorněná na obrázku 12. Výstup je zmenšený o body na okraji vstupu, podle velikosti jádra. Informace z nich je však použita v krajních bodech výstupu. Vhodným jádrem lze například zvýraznit/detekovat hrany objektů a jiné lokální informace v obraze. Hodnoty jádra jsou předmětem parametrizace a učení. Toto jádro se v některých případech nazývá neuron konvoluční vrstvy a jeho rozměr vnímaným/zorným polem (receptive field).

$$g(x,y) = f(x,y) * h(x,y) = \sum_{i=-S/2}^{S/2} \sum_{j=-R/2}^{R/2} f(x-i, y-j) \cdot h(i,j) \qquad (17)$$



Obr. 12: 2D diskrétní konvoluce s jádrem. [26]

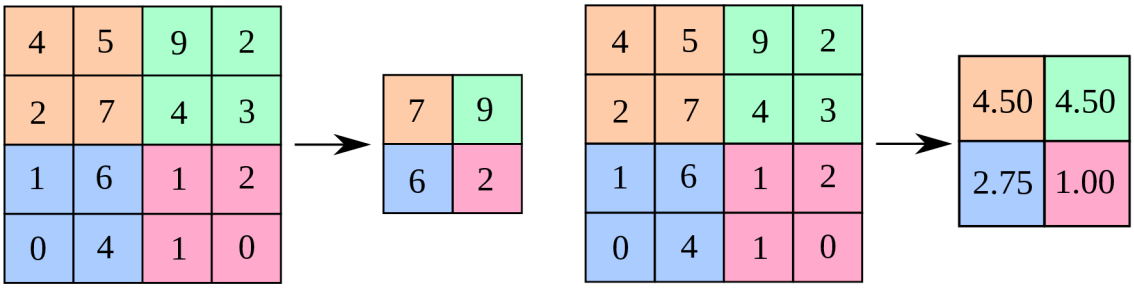
Z pohledu teorie neuronové sítě se tvoří dvou (a více, pro více barevných/informačních kanálů) rozměrná neuronová vrstva, kde není spojení (váha by byla nulová) mezi všemi neurony následující vrstvy. Jádra konvoluce jsou také v literatuře označovány jako filtry.



Obr. 13: 96 filtrů rozměru 11x11x3 jedné konvoluční vrstvy sítě ImageNet [14]

4.2.2 Agregační vrstvy (pooling)

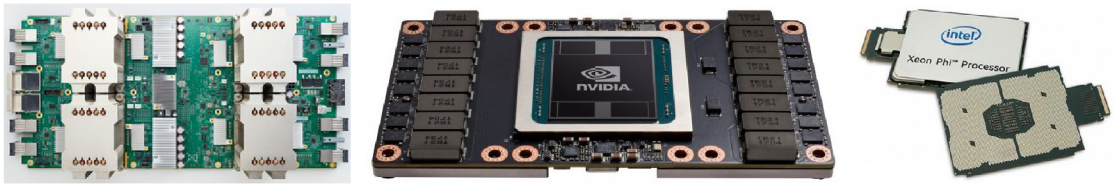
Účelem tohoto typu vrstev neuronové sítě je redukovat rozměr dat po konvoluci (downsampling). Toho se dosahuje spojením výstupu více neuronů, buď výběrem maxima (maxpooling, Obr. 14a), nebo průměrováním hodnot (avgpooling, Obr. 14b). Například 2x2 maxpooling zmenšuje rozlišení v obou osách o polovinu, tedy plochu celkového obrazu o čtvrtinu. Maxpool vrstvy se vhodně střídají s konvolučními. Není nutné, aby byly oblasti výběru čtvercové, s úspěchem se využívá například maxpool 1x2 a podobně. Výběr maxima je výpočetně méně náročný než průměrování.
Vrstvy agregačního charakteru se ukazují jako nevhodné pro generativní sítě jako jsou *variational autoencoders* (VAE) a *generative adversarial networks* (GAN).
Některé sítě jako například dále diskutovaný ResNet používají tyto vrstvy v minimálním množství a problém řeší velkým množstvím konvolučních vrstev, které tak redukují rozměr dat předávaný mezi vrstvami a zároveň mohou rozpoznávat další příznaky.
Relativní výhodou agregačních vrstev je, že nemají žádné parametry, které je třeba nastavovat učením a jsou výpočetně nenáročné.



Obr. 14: (a) Maxpooling 2x2, (b) Avgpooling 2x2

4.3 Akcelerace učení a inference pomocí SIMD, GPU a TPU

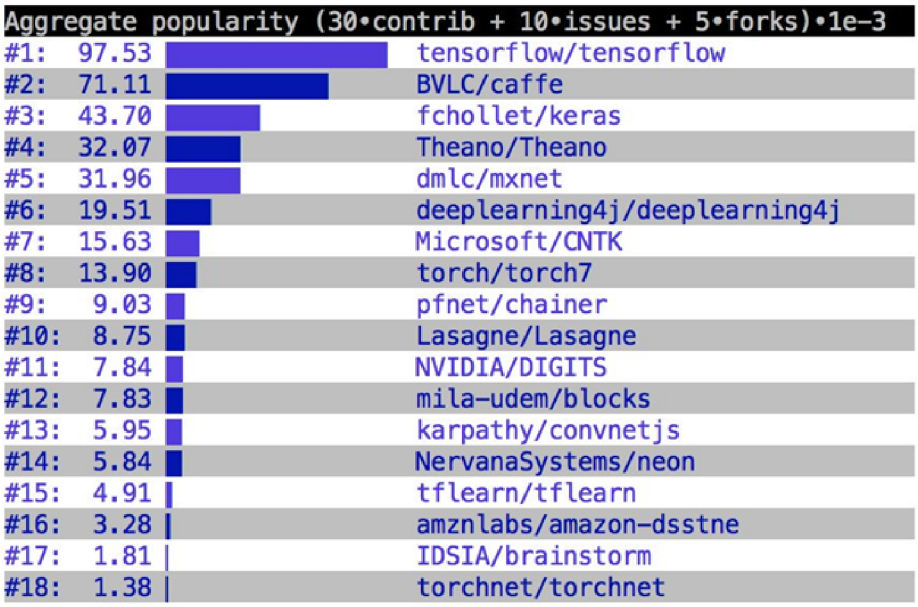
Výpočty pro deep learning jsou vysoce datově paralelní. A to jak při výpočtu jednotlivých vrstev neuronové sítě, tak pro trénování. Při trénování se akceleruje spuštění vícero instancí stejné sítě. Pro tyto operace jsou výhodnými procesory právě ty, architekturou vycházející původně určených pro grafické výpočty, které jsou ve své podstatě také maticového či tensorového charakteru. Většina frameworků podporuje akceleraci na GPU přes rozhraní NVIDIA CUDA nebo SYCL (OpenCL). Pro použití na běžných CPU se využívá vektorových SIMD jednotek, které byly původně také určeny hlavně pro grafické výpočty. SIMD lze využít buď autovektorizací kompilátoru, ruční instrict optimalizací nebo pomocí OpenCL.
Kromě GPU/SIMD existuje TPU (Tensor Processing Unit) od společnosti Google. Lze si je pronajmout v jejich cloudu, ovšem samotný HW zakoupit do vlastní instalace nelze. Jako HW speciálně navržený pro deep learning dosahuje vysokého výkonu. Ze zveřejněné specifikace je hlavní částí obvodu pole 8-bitových násobiček, druhá generace chipu už obsahuje také výpočty v plovoucí desetinné čárce.
Celkově se ukazuje, že pro deep learning je důležitější množství neuronů, vrstev a výpočtů, než jejich přesnost, a proto často 16 nebo i 8 bitů je plně postačujících pro trénink sítě.



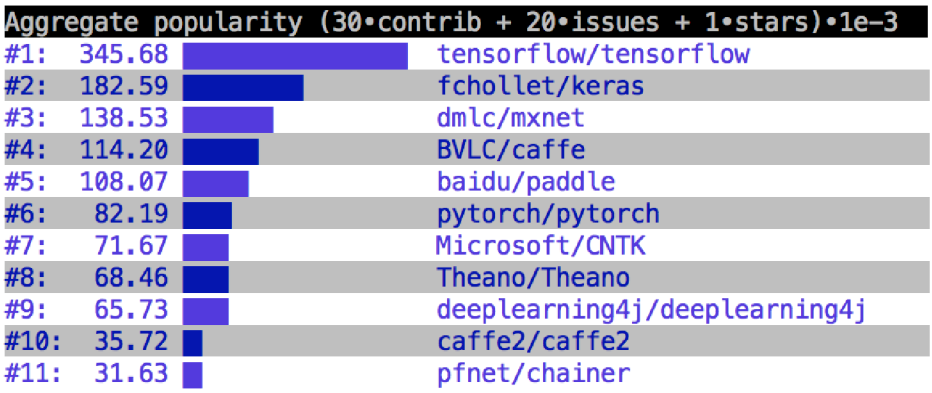
Obr. 15: Google TPU 2 (4x), NVIDIA GV100 GPU a Intel Xeon Phi CPU

4.4 Frameworky pro Deep Learning

V posledních letech se s přibývajícími aplikacemi Deep Learningu objevilo na trhu velké množství knihoven a frameworků usnadňujících učení, nasazení a kooperaci neuronových sítí bez nutnosti popisu sítě pouze pomocí funkcí a maticových výpočtů. [31] Často tyto frameworky umožňují výsledný natrénovaný model exportovat pro vyhodnocení inference v optimalizované formě, například ve formátu protocol bufferů (Google protobuf).



Obr. 16: Statistika GitHub popularity deep learning frameworků pro Q4/2017



Obr. 17: Statistika GitHub popularity deep learning frameworků k 8. 3. 2018

4.4.1 Tensorflow

Vznikl ve společnosti Google, původně z projektu DistBelief týmu Google Brain, který se začal stále více používat napříč aktivitami společností spadajících pod skupinu Alphabet. Umožňuje běh trénování i inference na více CPU anebo GPU (s využitím NVIDIA CUDA a SYCL/OpenCL rozhraní). Je možné jej v Google Compute Engine cloudu provozovat s jejich TPU (Tensor Processing Unit) pro vyšší energetickou efektivitu výpočtu. Tyto speciální TPU procesory nelze koupit, pouze pronajmout v rámci cloudu.
Jako API s vyšší úrovní abstrakce se doporučuje použít Keras, přesto samotný TensorFlow obsahuje poměrně vysokou úroveň abstrakce pro vrstvy a grafy neuronových sítí.
Vizualizace trénování a validace se všemi potřebnými grafy obstarává program TensorBoard, který pracuje jako webová služba spustitelná lokálně.
Oddělený git repositář *tensorflow/research* na serveru *github.com* obsahuje větší množství navazujících projektů a modelů, jako je TensorFlow Object detection API dále využívané v řešení této práce.

4.4.2 Keras

Je framework vyšší úrovně abstrakce přímo určený pro deep learning pro jazyk Python. Sám neimplementuje operace s maticemi/tenzory, ale naopak vyžaduje použití dohromady s jiným podporovaným frameworkem/knihovnou. Na výběr je TensorFlow, CNTK a Theano. Od verze 2 je výchozím doporučeným TensorFlow backend.

4.4.3 Theano

Theano je knihovna pro jazyk Python, která si klade za cíl dále zlepšit a zrychlit matematické výpočty s tensory a grafy. Snaží se o optimalizaci, paralelní zpracování a JIT kompilaci pro CPU a GPU. Deep learning lze řešit pomocí těchto základnějších operací. Po vydání verze 1.0 byl vývoj ukončen s minimální další podporou. Autoři z laboratoře LISA (Université de Montréal) v oznámení o ukončení vývoje zmiňují přítomnost shodné funkcionality v ostatních frameworcích, s tím, že připomínají některá svá prvenství. Není tedy mnoho důvodů uvažovat o použití této knihovny pro nový projekt.

4.4.4 Torch

Tento framework je primárně určen pro jazyk Lua. Neboť Lua nemá ve standardní knihovně ekvivalent pro obecné numerické a vědecké výpočty. Je ambicí projektu Torch tento nedostatek nahradit. Ovšem umožňuje propojení do jazyků C a Python. Práci

s GPU zajišťuje rozhraní NVIDIA CUDA. Torch má velké množství komunitou spravovaných balíčků a rozšíření pro konkrétní aplikace vědeckých výpočtů, počítačového vidění, zpracování přirozené řeči, distribuovaných výpočtů, atd.

4.4.5 CNTK

Microsoft Cognitive Toolkit je framework pro deep learning s podporou jazyků Pythonu, C++, C# a BrainScript. Lze jej provozovat na GPU v cloudu Azure. Obsahuje mnoho užitečných abstrakcí pro práci s vícero druhy sítí.

4.4.6 Caffe

Dříve velmi rozšířený framework pocházející z Californské Univerzity v Berkley. Autoři se chlubí vysokou rychlostí své implementace, hlavně nasazených natrénovaných modelů. Za zmínku stojí, že Caffe je výchozí variantou v prostředí NVIDIA DIGITS.

4.4.7 Caffe2

Je forkem projektu Caffe s mnoha vylepšeními co se týče podporovaného HW a škálování při trénování modelu i nasazení a to i na mobilních platformách. Stojí za ním společnost Facebook. Přímo podporované jazyky jsou C++ a Python.

4.4.8 Darknet

Je knihovna v jazyce C pro neuronové sítě od Josepha Redmona z University of Washington. Zmiňuje se hlavně kvůli implementaci sítě pro detekci objektů YOLO, která byla uvažována v práci pro detekci značky a celého automobilu. Knihovna podporuje urychlení výpočtu přes NVIDIA CUDA. Určitou nevýhodou je nižší úroveň abstrakce v jazyce C.

4.4.9 MXNet

Tento framework je jedním z projektů v inkubátoru nadace Apache. Kromě svého vlastního API obsahuje i implementaci API Gluon. Podporováno je několik programovacích jazyků jako Python, C++, R a Julia.

4.4.10 Gluon

Specifikace API a funkčnosti frameworku Gluon byla vytvořena spoluprací firem Amazon a Microsoft. První dostupná kompatibilní implementace tohoto API je v již zmíněném MXNet. Gluon je navržen pro snadnost aplikace deep learningu v rámci cloudu AWS. Podpora v Microsoft CNTK byla oznámena, zatím nebyla uvolněna.

4.4.11 Chainer

Tento framework je prezentován jako flexibilní a intuitivní. Je podporován hlavně firmami IBM a Microsoft. Přímo je podporován pouze jazyk Python a vyzdvihuje se přímá kompatibilita s knihovnou NumPy. Akcelerace na GPU je zajištěna přes knihovnu CuPy, která zpřístupňuje NumPy kompatibilní rozhraní s NVIDIA CUDA .

4.4.12 Sonnet

Společnost Deepmind, autoři AlphaGo a AlphaZero, uvolnila tento framework založený na Tensorflow, který má usnadňovat návrh neuronových sítí. Poměrně zjednodušuje použití rekurentních sítí RNN. Pro toto rozhraní s vyšší úrovní abstrakce je podporován jazyk Python.

4.4.13 BigDL

Knihovna určená pro Apache Spark pro využití deeplearning s Big Data (Hadoop, Spark) na distribuovaných systémech. Implementace je vhodná pro procesory Intel (zmíněny jsou v dokumentaci serverové modely Xeon) s využitím knihovny MKL. Podporovány jsou jazyky Scala a Python. Knihovna poskytuje API podobné frameworku Keras ve verzi 1.2.2. Ovšem návrh je inspirován frameworkem Torch.

4.4.14 TensorRT

Toto řešení od společnosti NVIDIA neumožňuje trénink sítě, ale je čistě frameworkem pro optimalizaci a nasazení natrénovaných sítí pro inferenci signálů. Přímo podporované jsou sítě z frameworků TensorFlow a Caffe. Pro ostatní frameworky je zveřejněné API pro definici natrénované sítě. Po optimalizaci je možné dosáhnout až 40x vyššího výkonu. TensorRT je možné také použít dohromady s DeepStream API, které umožňuje dekódování a inferenci snímků videa přímo na GPU se snížením zátěže CPU.

4.4.15 cuDNN

Je knihovna od firmy NVIDIA implementující základní operace potřebné pro neuronové sítě pomocí rozhraní NVIDIA CUDA. Mnoho frameworků pro deep learning přidává svoji podporu pro GPU NVIDIA právě použítím této knihovny.

4.4.16 Deep Learning Studio - Desktop

Firma Deep Cognition nabízí produkt, který má usnadňovat používání známých modelů sítí a vytváření nových. Obsahuje pokročilé grafické uživatelské rozhraní a funkci AutoML. Produkt je komerční uzavřený software.

4.5 Rozdíly mezi Deep Learningem a Machine Learningem

Umělou inteligenci lze dělit na klasické a nové metody. Mezi klasické patří například prohledávání stavového prostoru algoritmy jako je A*, alfa-beta prořezávání, atd. Nové metody jsou Soft Computing, Machine Learning (strojové učení, ML) a Deep Learning (hluboké učení, DL). Významy těchto pojmů se často překrývají. Objevuje se mezi experty názor, že si DL zaslouží vlastní kategorii kvůli zcela jinému přístupu při učení a analýze dat s minimálním preprocesingem bez manuálních algoritmů pro detekci příznaků signálu (hrany, fonémy, textury, slovní spojení, atd.). ML zahrnuje vše od jednodušších metod jako lineární regrese a k-means clustering přes rozhodovací stromy (decision trees), náhodné lesy (random forests), analýzu hlavních komponent (PCA) až po support vector machines (SVM) a umělé neuronové sítě (ANN). SoftComputing se prolíná s ML, zařazují se sem například genetické algoritmy, genetické programování, hejnové algoritmy (swarm), gramatická evoluce, SOMA, HC12. SoftComputing přístupy jsou stochastické algoritmy, které jsou často inspirované přírodními procesy a oproti ML většinou nevyžadují trénovací data. Je tedy logické, že by mohl ML zahrnovat i aplikace hlubokých neuronových sítí, které z běžných umělých neuronových sítí vycházejí. [28]

Autor Nikos Paragios také polemizuje nad čistým vědeckým přínosem článků o aplikacích hlubokého učení, pokud nerozvíjejí metodu samotnou, ovšem neupírá význam těchto aplikací. Poukazuje na poměrně velký odklon od klasických metod ML a diskutuje, zda není brzy se přestat věnovat výzkumu v této oblasti přestože DL má velmi pozitivní výsledky v aplikacích. Dále vyjadřuje obavu umělého umrtvení výzkumu ML z možného důvodu, že se studenti ani v doktorském studiu s metodami ML neseznámí a automaticky budou spoléhat na použití DL i ve specifických případech, kde by možná ručně navržený algoritmus ML byl vhodnější. [29]

Michael Copeland naopak předpovídá zářnou budoucnost, kterou DL umožní. Samočinné automobily, inteligentní asistenti, možná i skutečně myslící roboty. Argumentuje, že klasické metody ML se neprosadily ani při řešení některých problémů, na které byly navrženy a DL svými výsledky ukazuje jasný směr s nevídanou přesností. [30]

4.6 Detekce automobilu a registrační značky

Při řešení tohoto problému se nabízí pomocí deep learningu několik technik. Základní je natrénovat jednodušší klasifikátory pro detekci požadovaných objektů a metodou posouvání okna (sliding window) projít postupně různě velké výřezy celého obrazu. Tato metoda je však výpočetně velmi náročná a byly vyvinuty techniky výrazně efektivnější.

4.6.1 Regional based Convolutional Neural Network (R-CNN)

Využívá se algoritmus Selective Search [8] pro navržení oblastí, které jsou dále zpracovány CNN detektorem. Tím se redukuje výrazně počet výpočtů celé CNN oproti metodě posuvného okna. Selective Search hledá v obraze určité podobnosti textur, intenzity a barvy. Navržené oblasti nemají stejný poměr stran (aspect ratio), což se nijak nekompenzuje a na CNN detektor se výřez obrazu transformuje změnou velikosti bez dodržení poměru stran. Dobře natrénovaný CNN detektor zvládá tento nedostatek kompenzovat.
Verze Faster R-CNN používá místo Selective Search další neuronovou síť Regional Proposal Network (RPN). Vývojovým mezičlánkem je síť SPPnet, která zde není dále popsána.

4.6.2 Single Shot Detector (SSD)

Tento typ sítě dosahuje vyváženého poměru mezi rychlostí výpočtu a přesností. Na vstupním obrazu se nejprve provede pouze jedna konvoluční síť, která vypočte mapu příznaků. Až na této mapě pak počítá další konvoluční sítě pro různé velikosti detekovaných objektů. [2]

4.6.3 Výhody využití předem trénované sítě

U hlubokých konvolučních neuronových sítí se ukazuje pozoruhodný jev. Protože vrstvy na začátku sítě rozpoznávají vlastnosti obrazu jako jsou hrany a jiné lokální informace v obrazu a závěrečné vrstvy řeší klasifikaci, je možné zkrátit dobu tréninku sítě použitím konfigurace trénované na jiné datové množině s úplně jinými požadavky na zařazení do tříd objektů. Přestože bude pak výsledná klasifikace v prvních iteracích úplně špatně, stejně jako u náhodného nastavení sítě, v průběhu trénování se bude nastavení prvních vrstev měnit méně, zatímco nastavení vah posledních vrstev výrazně. Tato metoda se běžně označuje jako *transfer learning*, v daném kontextu jde tedy o využití částečně před-trénované sítě. Další výraznou výhodou *transfer learningu* je menší trénovací množina příkladů, než by bylo třeba u trénování sítě od začátku

s náhodnými vahami. Proto se tato metoda s úspěchem používá i tam, kde není možné nebo je nepřiměřeně obtížné více dat pro trénink získat. [6]

4.6.4 Veřejně dostupné datové množiny

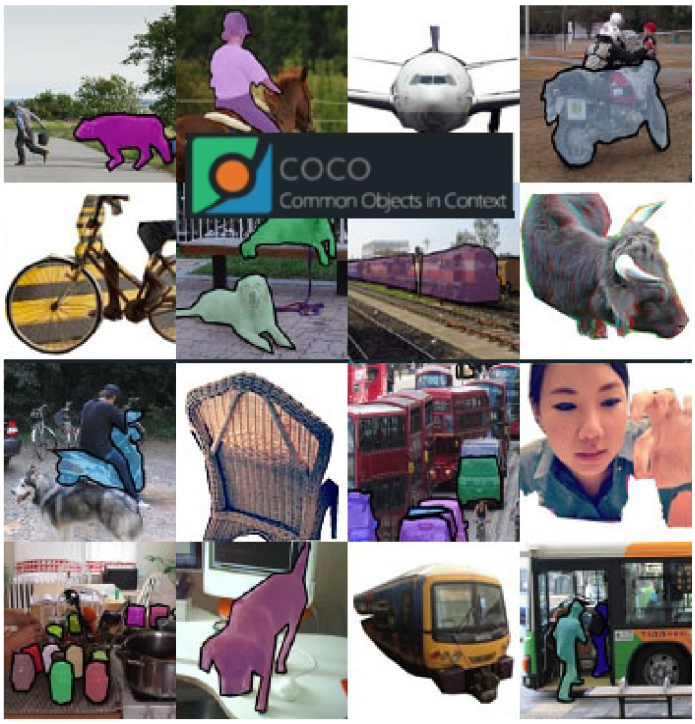
Pro trénink klasifikátorů existuje celá řada "standardních" datových množin připravených obrazů s anotacemi obdélníkových obálek (bounding box) anebo masek pro segmentaci, případně obecně polygonálních obálek. Liší se rozlišením obrázků, jejich počtem, existencí připraveného rozdělení na trénovací a validační množiny, přesností popisu a množstvím tříd.

Open Images Dataset V4

V práci není použita síť předem trénovaná na této datové množině, přestože byla uvažována k použití. Obsahuje přes 9 milionů obrázků se slovním popisem a na 1,74 milionech z nich je 14,6 milionů obdélníkových obálek objektů rozdělených do 600 tříd. Jde o zatím největší datovou množinu pro detekci objektů s obdélníkovými obálkami.

COCO

Zkratka pro Comon Objects in COntext. Tedy běžné objekty v kontextu, jako jsou dopravní prostředky, lidé v různých situacích, sporty, zvířata, rostliny, jídlo, atd. Obsahuje obdélníkové obálky a masky pro segmentaci. Celkově obsahuje 123 287 obrázků a na nich 886 284 instancí objektů k detekci rozdělených do 171 tříd. [12]



Obr. 18: Ukázka COCO datové množiny s maskami pro segmentaci

KITTI

Hlavním důvodem použití je fakt, že obsahuje velké množství automobilů na silnici, protože je vytvořena ze záznamů kamery jedoucího vozidla po silnici, také jsou dostupná data z LIDARu pro trénink rozpoznávání trojrozměrných objektů a jejich kvádrových obálek. [13]

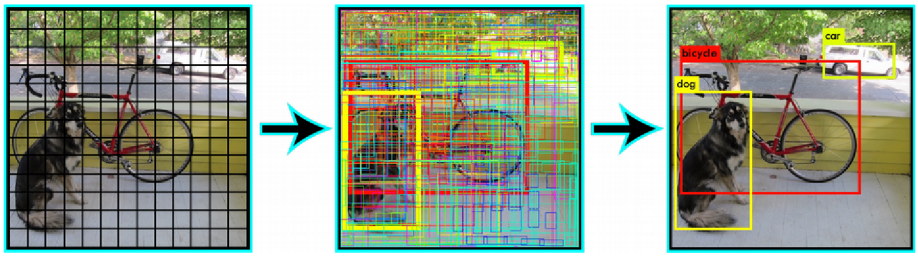
Obr. 19: Ukázka KITTI datové množiny a automobilu se systémem pro její pořízení.

4.6.5 Navrhované a použité sítě

Pro detekci automobilu a značky byly uvažovány 3 následující sítě, z toho 2 byly úspěšně použity s TensorFlow Object Detection API. Byla zvažována architektura, dostupnost modelu a trénované varianty pro transfer learning podle obsahu datové množiny.

YOLO

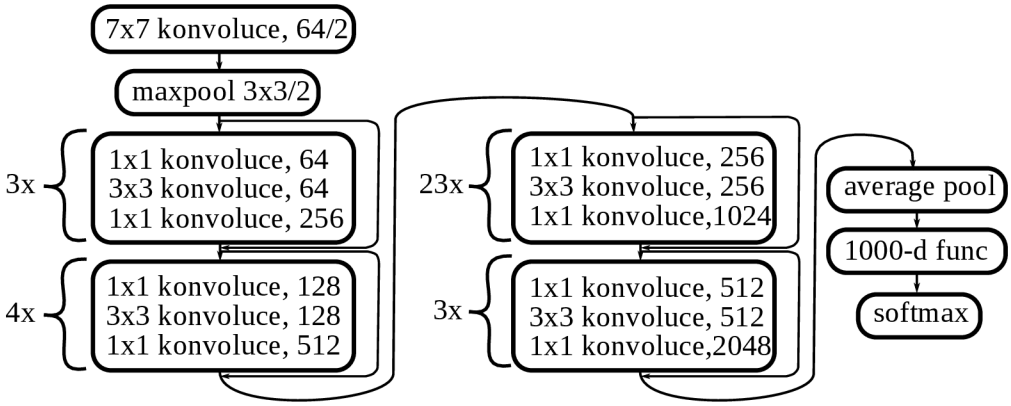
Název této sítě je akronymem slovního spojení You Only Look Once (podíváš se pouze jednou) a vystihuje snahu autora o řešení s jedním průchodem výpočtu v jediné natrénované konfiguraci. Její zpracování je rychlejší než u SSD, ale podle některých testů nedosahuje tak vysoké přesnosti. Ve verzi 3 podle autora dosahuje přesnosti lepší [24]. Také má problémy s větším množstvím menších objektů, kvůli způsobu, jak rozděluje obrazu na buňky a v každé buňce odhaduje pouze jedinou třídu klasifikace. Tato síť byla využita pro detekci norských registračních značek vozidel a zároveň čtení jednotlivých písmen (OCR) v disertační práci [7]. V této práci není nakonec testována, kvůli nedostupnosti modelu kompatibilního s TensorFlow Object Detection API. Nabízí velmi zajímavé řešení hlavně pro mobilní aplikace s uživatelskou interakcí.



Obr. 20: Funkce YOLO sítě: mřížka - pravděpodobnosti buněk - určení obálek [24]

R-CNN ResNet 101 (KITTI)

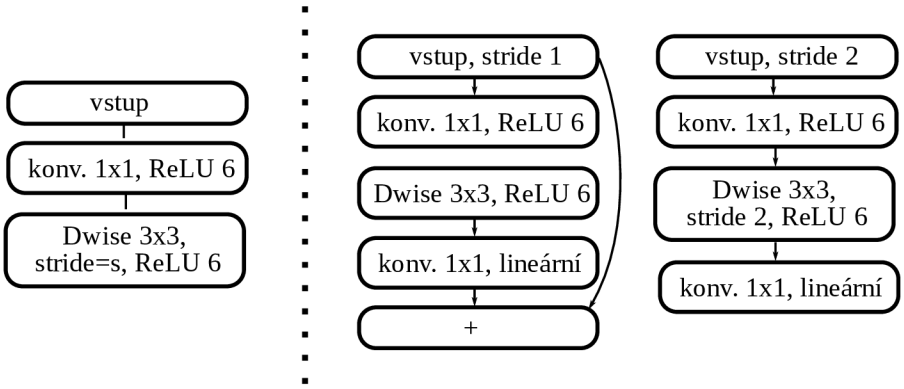
Tato síť, předem trénovaná na datasetu KITTI, dosahuje vysoké přesnosti při nalezení a rozpoznání objektů. Je také dostupná předem trénovaná na Open Images Dataset V2, pro tuto aplikaci však už nebyla tato varianta otestována. Jde o variantu Faster R-CNN sítě vylepšenou o spoje do více následujících vrstev podobné funkčnosti RNN. Vazby zpět jsou ale zakázané, a proto se jedná stále o variantu dopředné neuronové sítě. Varianta značená jako 101 má celkem právě tolik vrstev. Zajímavým poznatkem je maximální využití konvolučních vrstev. Přítomná je pouze jedna maxpool a jedna avgpool vrstva [15].



Obr. 21: Struktura sítě ResNet 101

SSD MobileNet V1 (COCO)

Jde o single shot detector postavený na architektuře sítě MobileNet, která se snaží být efektivní implementací vhodnou i pro mobilní zařízení. Je zde tedy kladen zvláštní důraz na výkon a optimalizaci struktury [4]. Další optimalizace nabízí verze 2 [5], která velmi výrazně snižuje náročnost na použitou paměť při mírném zhoršení přesnosti, při řešení této úlohy je však využita první verze. Předem trénovaná byla tato síť na datové množině COCO. Pro řešený problém je dostačující přesnost detekce této sítě.



Obr. 22: Struktura konvolučních bloků sítí MobileNet V1 a V2

4.6.6 TensorFlow Object Detection API

Tento projekt je snahou o rozšíření frameworku TensorFlow o možnosti detekce objektů v obraze. V době psaní práce není toto API zařazeno v hlavním repositáři zdrojových kódů TensorFlow, ale ve vedlejším repositáři *models* v adresáři *research /object_detection*. Z tohoto důvodu není dostupná plná dokumentace API, ani přesné použití v reálné aplikaci. Předpokládá se zkoumání zdrojového kódu uživatelem s případnými vlastními úpravami. Vlastní nasazení natrénovaného modelu není přímo tímto API podporováno a je třeba výsledný "zamrazený" výpočetní graf načíst přímo v TensorFlow, přiřadit vstupy a vyhodnotit výstup. Moduly z *object_detection* už nejsou potřeba.

API se skládá z několika modulů a skriptů, které se starají o trénování, validaci a integraci s TensorBoard. Motivací autorů k vytvoření tohoto API bylo porovnání architekury a výkonu různých sítí se sjednocením metody implementace [3].

(a) SSD+Mobilenet, lowres

(b) SSD+InceptionV2, lowres

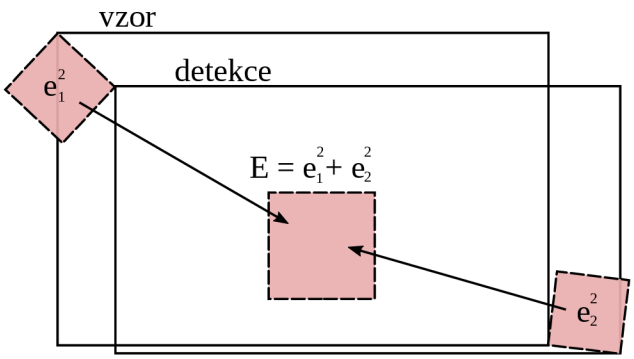
(c) FRCNN+Resnet101, 100 proposals

(d) RFCN+Resnet10, 300 proposals

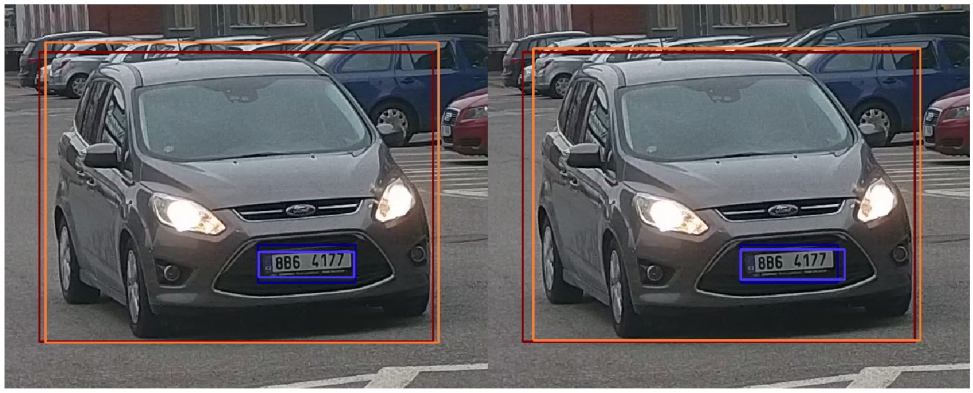
Obr. 23: Porovnání detekčních sítí pomocí TensorFlow Object Detection API [3]

4.6.7 Validace a porovnání sítí

Pro porovnání kvality detekce se využívá součet kvadratické chyby levého horního a pravého dolního rohu obdélníkových obálek, geometrický význam na obrázku 24. Pokud je detekce úplně neúspěšná, do statistiky chyby obálek se nezapočítává. Dále je třeba porovnávat časovou náročnost výpočtu.



Obr. 24: Kvadratická chyba detekce obdélníkových obálek



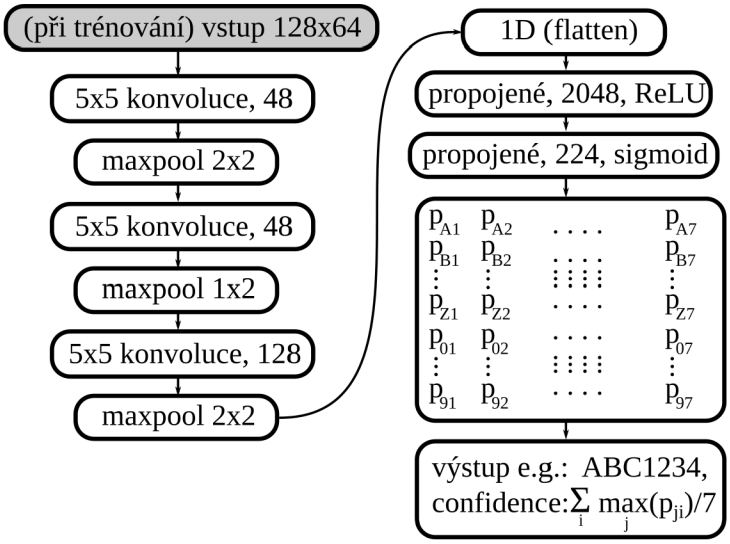
Obr. 25: Vizuální porovnání detekce Resnet 101 (vlevo) a SSDmobilenetV1 (vpravo)

4.7 Čtení nalezené značky

Hlavní inspirací pro řešení se stal článek o čtení britských registračních značek z blogu Mattewa Earla – Matt's ramblings [17]. Jeho řešení vychází z článku o čtení čísel na domech z fotografií Google Streetview [18] a článku o prolamování CAPTCHA textů v obrázcích [19]. Toto řešení je ovšem pouze konceptem, který nelze v praxi přímo nasadit. Využívá jedinou konvoluční síť a metodu posuvného okna. Na výkonném GPU trvá zpracování jediného snímku několik sekund, což je nepoužitelné pro video a detekci v reálném čase.
Obraz je převeden do odstínů šedi, protože je předpoklad černého textu na bílém pozadí. Pro jiná barevná provedení značek by bylo vhodné provést práhování, nebo natrénovat síť pro barevný obraz. To může být předmětem dalšího vývoje projektu.

4.7.1 Struktura sítě

Z větší části je použita stejná struktura sítě jako v příspěvku Mattewa Earla, byla však znovu plně implementována pomocí frameworku Keras. Původní příspěvek je ve starší verzi TensorFlow a je psán velmi nízkoúrovňově.
Síť dodržuje trojnásobné střídání konvolučních a maxpool vrstev. Na výstupu je 224 pravděpodobností, které pak lze zapsat do matice, kde jsou ve sloupcích pravděpodobnosti detekce znaku na pozici. Ideálně rozpoznané znaky tedy mají v každém sloupci pouze jednu jedničku a zbytek nuly. Pro finální aktivaci je třeba použít sigmoidu (nesmí být softmax) a jako chybovou funkci *binární crossentropy*.



Obr. 26: Struktura sítě pro čtení registrační značky se 7 symboly

4.7.2 Generování trénovací množiny za běhu

U tohoto problému se ukazuje jako velmi výhodné trénovat na synteticky generované množině. Jako pozadí pro značku jsou voleny náhodně obrázky s přidaným šumem, aby se předešlo naučení sítě. Poloha, velikost a natočení značky v obrázku je také náhodné. Natočení je voleno tak, aby značka byla čitelná zleva doprava tak jak by se měla nacházet detekovaná v obrazu z kamery. Síť by se tak měla naučit poznávat znaky na jednotlivých pozicích.

Dokumentace Kerasu také uvádí příklady, kdy je vhodné náhodně generované transformace zavést při učení na malé nesyntetické množině příkladů, jedná se tedy o zavedený a ověřený postup při učení konvolučních sítí.

Jsou generovány dvě různé množiny, obtížná varianta je plně náhodná s možnou přítomností všech znaků na všech pozicích. Pro prvotní experimenty byla generována jednodušší varianta, používající pouze písmena A a B v první trojici a zbytek znaků pouze číslice. Lze takto ověřovat správnost návrhu a implementace s nižším časem tréninku sítě.

U obtížné množiny byla provedena v průběhu vývoje změna, byl odstraněn pruh EU a náhodně se mění světlost celé značky. Do obrazu se přidává šum s normálním rozložením pravděpodobnosti.

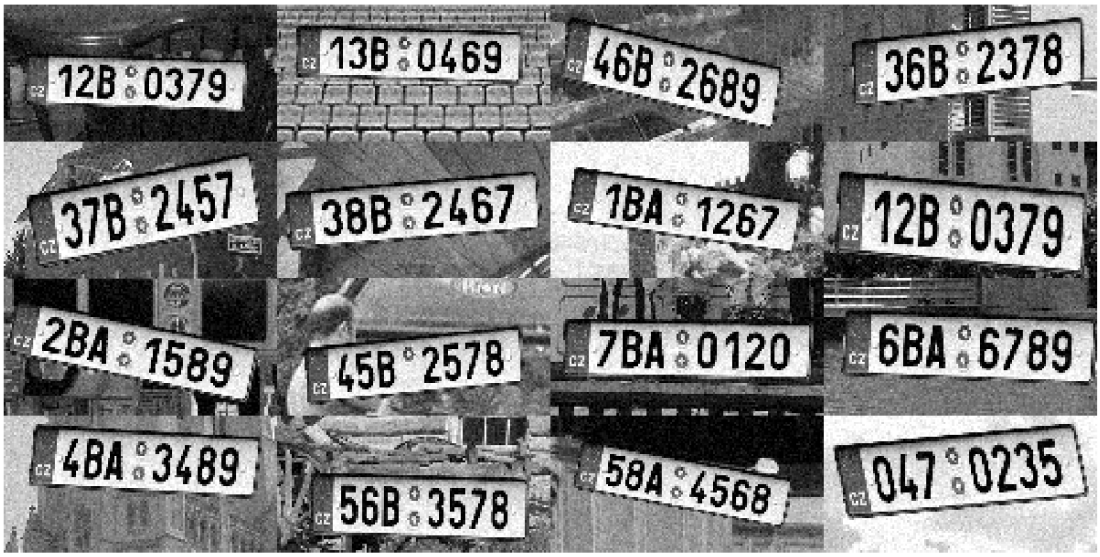
4.7.3 Validace

Na n prvkové validační množině lze ověřovat správnost tréninku sítě statistikami o úspěšnosti detekce jednotlivých znaků. U úspěšných detekcí je třeba uvažovat nad tím, jak výrazné bylo přiřazení správné odpovědi – míra důvěry C (confidence), rovnice (18). Pro neúspěšné detekce lze zkoumat podobnost jednotlivých znaků a expertně ověřit čitelnost.

$$C = \sum_{i=1}^{7} \max_{j} p_{ji} \tag{18}$$

kde p_{ij} odpovídá pravděpodobnosti rozpoznávaného znaku v matici pravděpodobností, obrázek 26 a příloha B.

Obr. 27: Příklad generovaných registračních značek pro trénink, obtížná množina



Obr. 28: Příklad generovaných registračních značek pro trénink, jednoduchá množina

5 IMPLEMENTACE

5.1 Výběr technologií

Při vlastním řešení návrhu a programování spolupracujícího systému aplikací, je nutno vybrat vhodné kompatibilní prostředky programovacích jazyků a dostupných knihoven. Oddělením systému na jednotlivé spolupracující části je možná škálovatelnost a udržitelnost vývoje při propojování částí standardními prostředky, jako jsou IPC služby operačního systému (pipe, socket, soubor mapovaný do paměti, signály), síťové BSD sockety (např. s vyššími protokoly HTTP REST API) nebo jiné služby pro zasílání zpráv (DBus, RabbitMQ, ZeroMQ, ActiveMQ, atd.).

5.1.1 Python

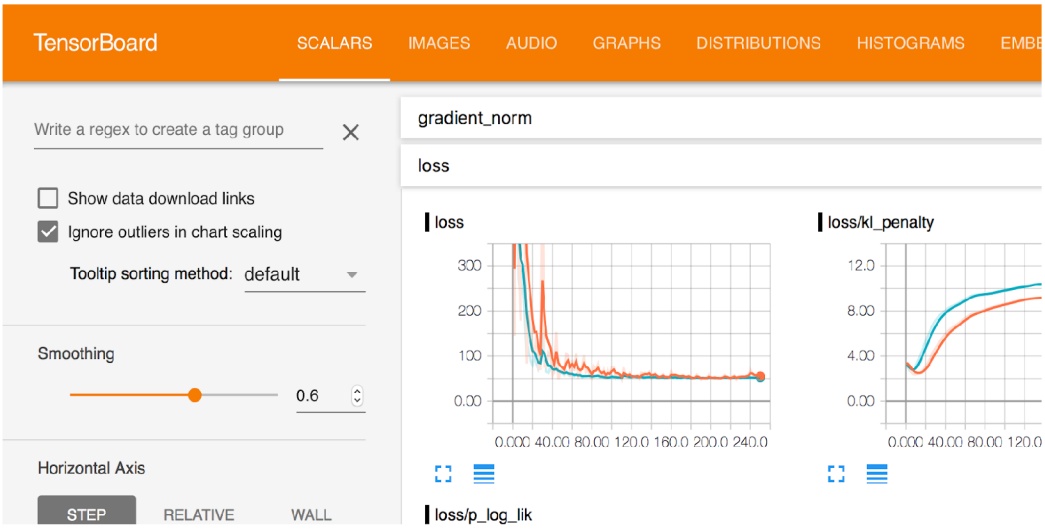
Tento moderní, obecný a objektově orientovaný programovací jazyk se dnes velmi často aplikuje v oblasti datascience a umělé inteligence. V základní použité implementaci CPython jde o interpretovaný jazyk. Vysokého výkonu v aplikacích se dosahuje tím, že v Pythonu samotném nejsou napsány samotné náročné operace, ale stává se pohodlným prostředníkem mezi programátorem a knihovnami pro náročné numerické operace, které jsou psané v jazycích jako C++ a Fortran.

5.1.2 OpenCV

Tato C++ knihovna s wrapperem pro Python umožňuje použití jak lokálně připojených kamer, například přes rozhraní USB, tak přijímat obraz z IP kamery přes protokol RTSP (Real Time Streaming Protocol). OpenCV umí, co se týče zpracování obrazu, mnohem více, hlavně klasické algoritmy, které ale v práci nebyly využity. V novějších verzích obsahuje i přímou podporu integrace s frameworky hlubokých neuronových sítí (Caffe 1, TensorFlow, Torch). Knihovna je původně vyvíjena firmou Intel a je optimalizovaná pro jejich procesory, obsahuje však i moduly pro akceleraci na GPU s rozhraním NVIDIA CUDA.

5.1.3 Keras + Tensorflow

Pro svoji snadnost ovládání a napojení na TensorFlow byl zvolen Keras framework pro implementaci sítě, která čte registrační značku. Tensorflow samotný se s výhodou využil s Object Detection API pro nalezení automobilu a registrační značky v obraze. V druhém případě se využívá TensorBoard pro vizualizaci průběhu trénování a inference na validační množině.



Obr. 29: TensorBoard pro vizualizaci průběhu trénování

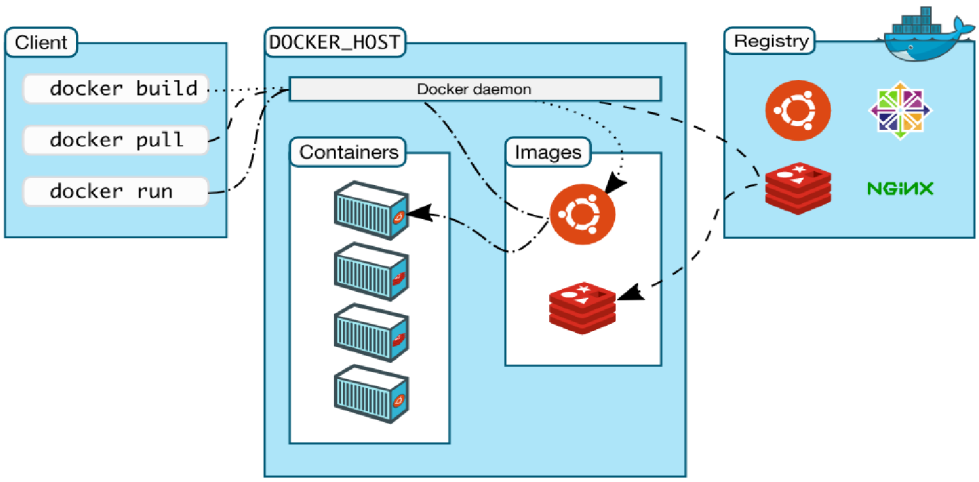
5.1.4 Flask

Tento webový framework založený na knihovně werkzeug pro Python je vhodný, jak pro uživatelské webové rozhraní, tak pro vzájemně propojené microservice s HTTP JSON, případně REST API. To umožňuje vytvářet heterogenní architektury, kde se přísné rozdělení na server/klient ztrácí a vzniká tak hierarchie spolupracujících objektů (služeb).

Jeho volba byla ovlivněna autorovou zkušeností s několika projekty, které úspěšně využívají využívající tento framework pro webové rozhraní informačních systémů a služeb s JSON API.

5.1.5 Docker

Pro konečné nasazení je rozpracován Dockerfile pro vytvoření obrazu virtualizačního kontejneru karna48/base a na něm založených kontejnerech jednotlivých služeb. Obraz staví na obrazu nvidia/cuda ve verzi 9.1, založené na obrazu Ubuntu 16.04. S variantou nvidia-docker lze v kontejneru používat i akceleraci na GPU. Toto virtualizační řešení má velkou výhodu při nasazení, předchází problémům s instalací, závislostmi a změnami v systému. Také usnadňuje případné škálování a verzování aplikace. Je možné aplikaci rozdělit do více kontejnerů, každý pro jinou službu a dalšími kontejnery rozšiřovat funkcionalitu, která může být vyvíjena na novějších verzích knihoven, bez nutnosti udržovat kompatibilitu starších částí aplikace. Ke zvážení je použití nástroje Kubernetes.



Obr. 30: Kontejnerová architektura s Dockerem [dokumentace]

5.1.6 Nastavení

Pro lokální nastavení instance aplikace všechny služby předpokládají existenci souboru `local_config.py`, který je vyňatý z git repositáře, kde se nachází pouze jeho šablona. Zde se nastaví vše potřebné, jako je výřez obrazu kamery, porty jednotlivých služeb, cesty v adresářové struktuře, URL k IP kameře, atd. Při rozdělení služeb do více složek či kontejnerů, lze mít u každé části jiný soubor `local_config.py` s nastaveními, která se týkají pouze dané části. Importuje se jako modul a tak musí být dostupný prostřednictvím proměnné prostředí `PYTHON_PATH` při startu služby. Zde uvedená nastavení nelze měnit bez restartu služby.

Bylo by vhodné umožnit nastavení také pomocí proměnného prostředí, což by dále umožňovalo v kombinaci s kontejnery vyšší škálovatelnost. Bude to předmětem dalšího vývoje.

Příklad konfigurace služeb:

```
import os.path
TF_models_research_dir = '/opt/tensorflow/research'
working_dir = os.path.dirname(__file__)
mysql_host = '127.0.0.1'; mysql_user = 'usr'
mysql_password = 'pass'; mysql_db = 'rzcznet'
site_static_dir = 'flask_site/static'
plate_reader_test = os.path.join(working_dir,
                                 'datasets/reader_validate')
plate_reader_test_glob = os.path.join(plate_reader_test, '*.png')
plate_reader_bgs = os.path.join(working_dir, 'datasets/bgs')
cam_url = 'rtsp://user:pass@00.00.11.11:554/path'
```

5.2 Microservice jako OOP

Koncept objektově orientovaného programování je založen na zprávách, které si objekty posílají. Tyto objekty mohou být v podstatě cokoliv. Často je OOP úzce vnímáno v kontextu jazyků, jako jsou například C++ a Java, kde zasílání zpráv mezi objekty zajišťuje volání metody, tedy funkce pracující se stavem objektu v paměti. V širším kontextu mohou být objekty celé programy běžící jako služby, které si zasílají zprávy. Tyto zprávy mohou měnit stav objektu. Mikroslužby tedy můžeme vnímat jako OOP přístup, který nám umožňuje pro každou službu použít jiný programovací jazyk a může přispět k lepší udržitelnosti, škálování a jednoduchosti nasazení řešení. Také lze v tomto přístupu spatřovat podobnost s filosofií UNIX-ových systémů, kde více menších jednoúčelových programů spolupracuje při řešení problému. Nicméně přístup mikroslužeb, oproti monolitickým řešením, přináší své vlastní výzvy a problémy při návrhu, implementaci a nasazení. Hlavně pokud tvoří služby distribuovaný systém, komunikující po lokální síti či internetu, je třeba i pro nejjednodušší operace počítat s možností nedostupnosti síťového připojení, či služby jako takové. Jednoduchá řešení v podobě opakovaných pokusů o provedení, operace může vést kaskádovitě k zahlcení celého systému s rysy DDoS útoku. Také takovýto systém pro větší nasazení vyžaduje monitoring s pokročilým hlášením chyb (například řešení Sentry.io) a pokud možno automatické restarty padnuvších (chyba segmentace paměti, nepovolený přístup) či uvázlých (deadlock) služeb.

Co se týče propojení služeb, nabízí se lokální unixové sockety, TCP/IP, HTTP, lokální i vzdálené služby zasílání zpráv (DBus, RabbitMQ, ...). V implementaci je použito HTTP GET a POST s daty ve formátu JSON a lokální unix socket pro binární data.

Často jako první nástroj, kterým by bylo možno služby propojit, se nabízí relační databáze. Toto řešení se však, hlavně u problémů typu konzumenti/producenti – fronta, ukazuje jako velmi nešťastné a označuje se jako design anti-pattern. Tabulka relační databáze tak velmi špatně škáluje a při implementaci se objevují problémy s atomicitou sekvence SELECT/UPDATE a následnou serializací transakcí, které probíhají na celé tabulce fronty.

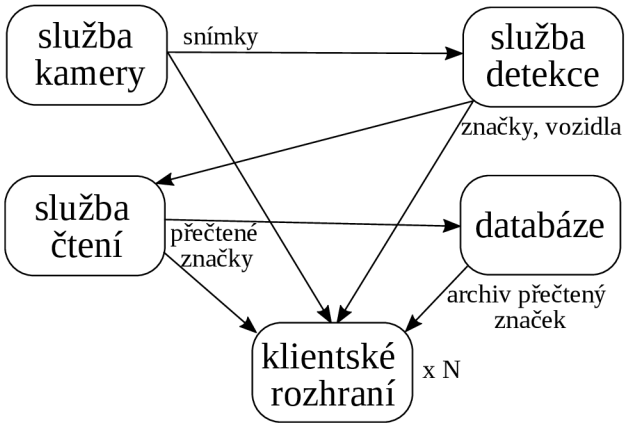
V rámci OOP přístupu se jeví jako nejlepší dodržování principů SOLID, jako třídu objektů budeme uvažovat celý program běžící služby:

- Single responsibility principle: třída/služba by měla zoodpovídat za jednu konkrétní činnost, tedy při změně specifikace jedné konkrétní vlastnosti software, by měla být modifikována pouze jedna třída/služba
- Open/closed principle: software by měl být otevřený rozšíření a uzavřený modifikaci, to znamená, že by se očekávané funkční chování nemělo měnit; toto platí hlavně v souvislosti s rozhraním

- Liskov substitution principle: potomci třídy/objektu by měly pro součásti, které očekávají instanci rodiče beze změny; u služeb jde o udržování staršího rozhraní a chování i v nových verzích
- Interface segregation principle: vícero rozhraní zaměřených na konkrétní klienty a případy užití, je lepší než jedno obecné rozhraní; t.j. více metod, endpointů pro jednotlivé typy chování
- Dependency inversion principle: je třeba záviset na obecných rozhraních a ne na konkrétních nízkoúrovňových implementacích, aby byla možná transparentní výměna součástí; například služba pracující s ukládáním perzistentních dat by obecně měla být nahraditelná napojením do databáze, ukládáním na disk, či přístupem do distribuovaného síťového úložiště. To také souvisí s vhodným předáváním nastavení.

Při povolení heterogenního prostředí, co se týče běhu více verzí jedné služby, je zde jistá podobnost s prototypovou dědičností objektů, na rozdíl od třídní dědičnosti, kterou lze vnímat jako určité pokřivení OOP paradigma. Zároveň je třídní dědičnost ve staticky typovaném jazyce paměťovou optimalizací, což platí i pro mikroslužby, pokud běží na stejném stroji ve stejném kontejneru; jinak je tato vlastnost nepodstatná.

Také je vhodné poznamenat, že přístup Microservices se liší od Service Oriented Architekture, která je předchůdcem a má více vlastností monolitické aplikace s rozdělenými rolemi klienta a serveru.



Obr. 31: Flexibilní architektura mikroslužeb

Z obrázku je vidět, že podle dostupného HW a potřeb lze v takovéto architektuře škálovat jednotlivé služby. Například navýšit pro více kamer počet služeb je obsluhující, ale služeb pro detekci může být méně. Stejně tak jedna služba pro čtení značek může být spojena s více detekčními službami.

5.3 Detekce automobilu a registrační značky

Pro využití TensorFlow Object Detection API je nutné vytvořit trénovací a validační množiny obrázků s obdélníkovými obálkami ve formátu `TFRecords`, který je založený na protocol buffers. Nástroj s uživatelským rozhraním k tomu určený je popsán dále. Je nutné vytvořit konfigurační soubory:

- `label_map.pbtxt` pro mapování textových popisů na číslo třídy
- `pipeline.config` pro nastavení parametrů tréninku, validace, umístění souborů datových množin a nastavení vah pro *transfer learning.*

Také byly vytvořeny skripty pro usnadnění opakovaného spuštění trénování, validace a propojení s TensorBoard.
Předem natrénované sítě byly získány z TensorFlow Object Detection API Model ZOO. Program pro natrénovaný detektor s použitím „zamrazeného“ grafu sítě je v příloze A.

Ukázka některých důležitých nastavení v `pipeline.config`:

```
train_input_reader {
  label_map_path: "data/rzcz_label_map.pbtxt"
  tf_record_input_reader {
    input_path: "datasets/kamera1_train_245imgs.tfrecords"
  }
}
eval_config {
  visualize_groundtruth_boxes: true
  num_examples: 44
  num_visualizations: 44
}
eval_input_reader {
  label_map_path: "data/rzcz_label_map.pbtxt"
  tf_record_input_reader {
    input_path: "datasets/kamera1_val_44imgs.tfrecords"
  }
}
```

5.4 Čtení registrační značky

Již popsaný způsob trénování sítě pro čtení 7 znakové značky je implementován ve dvou variantách. Pro prvotní testování je generována pouze podmnožina značek Jihomoravského kraje, v implementaci označovány jako „brněnské“ obsahující v první trojici písmena A, B a dále pouze čísla. Trénovat lze z dynamicky generované množiny, která je implementovaná pomocí generátorů – funkcí v Pythonu využívající klíčové slovo *yield*. Také je pro testování správnosti a opakovatelnosti možné trénovat z předem vytvořených souborů obrázků.

Generátor „brněnských“ značek:

```
def brno_gen_in():
   prefix = itertools.combinations('0123456789BA', 3)
   suffix = itertools.combinations('0123456789', 4)
   for p, s in itertools.zip_longest(prefix,
                                     suffix,
                                     fillvalue=('0', '1', '2')):
       if len(s) < 4:
           s = s + ('0',)
       yield ''.join(p)+''.join(s)
```

Výsledná struktura sítě je deklarovaná v Kerasu následovně:

```
allowed_chars = '0123456789ABCDEFHIJKLMNPRSTUVXYZ' # 32 znaků
n_binary_categories = 7 * len(allowed_chars) # 224 pravděpodobností
M = Sequential()
M.add(Conv2D(48, (5, 5), activation='relu', input_shape=(128, 64, 1)))
M.add(MaxPooling2D(pool_size=(2, 2)))
M.add(Conv2D(48, (5, 5), activation='relu'))
M.add(MaxPooling2D(pool_size=(1, 2)))
M.add(Conv2D(128, (5, 5), activation='relu'))
M.add(MaxPooling2D(pool_size=(2, 2)))
M.add(Flatten())
M.add(Dense(2048, activation='relu'))
M.add(Dense(n_binary_categories, activation='sigmoid'))
M.compile(loss='binary_crossentropy', optimizer='adam')
```

Po tréninku je model uložen s hodnotami vah ve formátu HDF5 a je možné pokračovat v tréninku dodatečně z uložené konfigurace. U tohoto trénování není využito TensorBoard, ale Keras pravidelně vypisuje do konzole informace o průběhu tréninku (loss, ETA, epoch).
Pro nasazení a inferenci je model načten z HDF5 a přímo použit s voláním metody `predict`.

Značku je nutné z vektoru kategorických pravděpodobností dekódovat, z pozic se pomocí metody `reshape` stane matice 7 sloupců a index řádku, na kterém je maximum ve sloupci, se pomocí `numpy.argmax` dostane do vektoru `idx`, z informace o pozici se pak rekonstruuje příslušný znak a spojí se do řetězce, který funkce vrací:

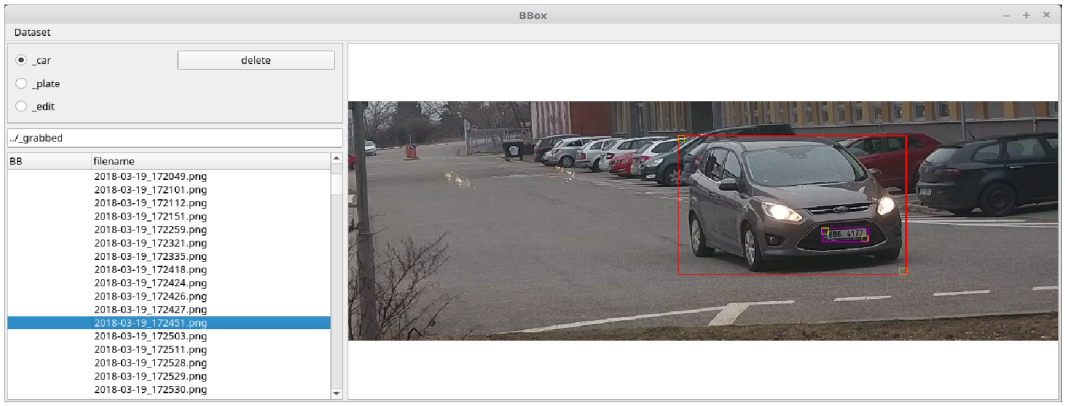
```
def vec_to_code(vec):
    idx = numpy.argmax(vec.reshape((-1, len(allowed_chars))), 1)
    return "".join(allowed_chars[i] for i in idx)
```

Naopak při vytváření trénovací a validační množiny příkladů je využita z Kerasu pomocná funkce `to_categorical`:

```
def code_to_vec(code):
    c_vec = []
    for c in code:
        for i in keras.utils.to_categorical(allowed_chars.index(c), 7):
            c_vec.append(i)
    return numpy.array(c_vec).reshape((224,))
```

5.4.1 Nástroj na tvorbu trénovací množiny `bbox_dataset_maker`

Pro vytváření trénovací a validační množiny příkladů na detekci automobilu a značky, byl vytvořen podpůrný program `bbox_dataset_maker` s využitím knihovny PyQt. Uživatel pomocí něj může k fotografiím přiřazovat obdélníkové obálky (bounding box) tříd car (auto) a plate (registrační značka). Množina přiřazených obdélníků se ukládá v SQLite databázi, je důležitá relativní cesta obrázků k pracovnímu adresáři aplikace. Fotografie s obálkami pak pro trénování a validaci uživatel exportuje ve formátu `TFRecord` jako množinu objektů třídy `train.Example` s vlastnostmi požadovanými TensorFlow Object Detection API. Lze exportovat najednou pouze jednu složku s fotografiemi. Pro potřeby aplikace bylo toto řešení naprosto dostačující.



Obr. 32: Nástroj `bbox_dataset_maker`

Poznámka pro lepší ladění programu: Protože v novějších verzích PyQt5 se přestalo předávat do vyšší úrovně vyhození výjimky, pokud nastane uvnitř reakce na událost (timer, akce uživatele) v hlavní smyčce, byl implementován dekorátor `@qmessagebox_exception`, aby byla výjimka vypsána na standardní výstup a do zvláštního okna:

```
def qmessagebox_exception(f):
  def decorated(*args, **kwargs):
    try:
      return f(*args, **kwargs)
    except Exception as e:
      msg = '\n'.join((f.__name__, type(e).__name__, str(e),
                       'args: '+repr(args), 'kwargs: '+repr(kwargs)))
            print(msg)
            QMessageBox.critical(None, 'Error', msg)
    return decorated
```

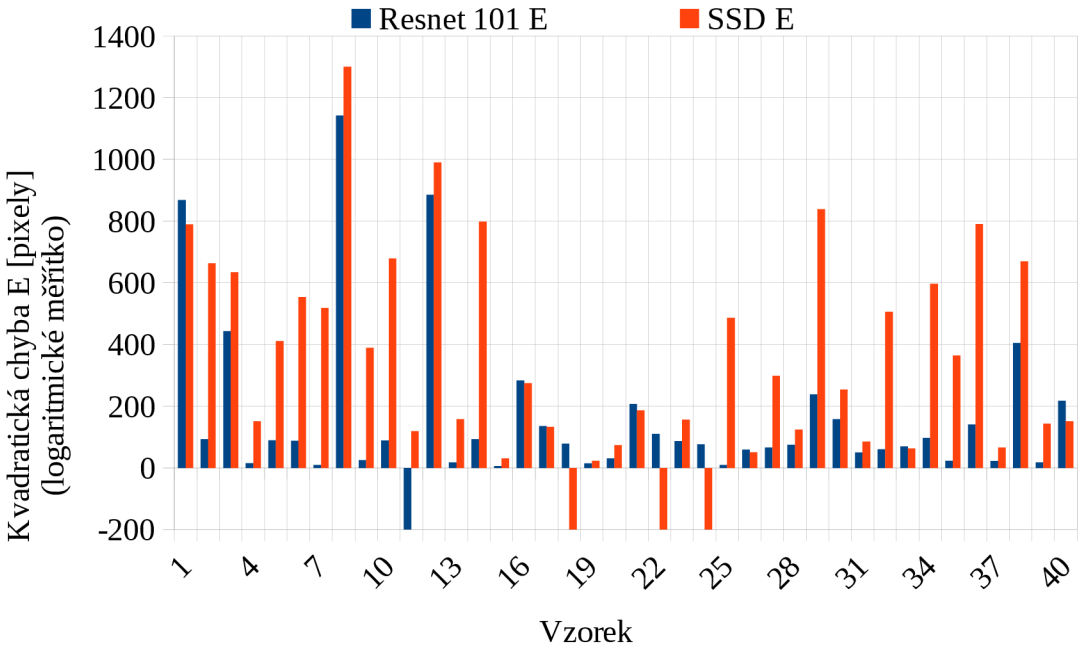
6 VÝSLEDKY

6.1 Detekce automobilu a značky

Pro sítě ResNet101 a SSDmobilenetV1 byla zjišťována na validační množině úspěšnost a přesnost detekce. Jako metrika přesnosti byla použita kvadratická odchylka levého horního a pravého dolního rohu obdélníkové obálky, která měla být detekována (sekce 4.6.7, obrázek 24). Validační množina obsahuje 51 vzorků obdélníkových obálek k detekci ve 25 obrázcích pořízených kamerou.

Tabulka 1: Porovnání výkonu ResNet 101 a SSDmobilenetV1 na validační množině

Parametr	**Úspěšné detekce [-]**	**Průměrný čas výpočtu t [ms]**	**Průměrná chyba E [pixel²]**	**Maximální chyba E [pixel²]**	**Minimální chyba E [pixel²]**
ResNet	49 / 51	142	153	1142	6
SSD	47 / 51	17	344	1300	2



Obr. 33: Porovnání kvality detekce sítí Resnet 101 a SSDmobilenetV1, 40 vzorků (neúspěšné detekce nabývají zápornou hodnotu)

6.2 Validace čtení registrační značky

6.2.1 Jednoduchá množina

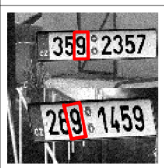
Pro jednoduchou množinu (podmnožina běžných značek JMK) byly po počátečním trénování ($t = 0$) a pak průběžně zjišťovány na validační množině $n = 20300$ počty správně detekovaných číslic. Tabulka 1 znázorňuje celkové zlepšování modelu s mírnými regresemi. Trénovací množina byla generována dynamicky.

Výsledná míra důvěry (confidence) je průměrně 0,9974, pro úspěšné detekce je průměrná hodnota 0,9981, a pro detekce s chybou 0,8755. Po dalších experimentech bude tedy možné expertně nastavit hranici, při které bude detekce vyhodnocena jako neúspěšná.

Tabulka 2: Průběh učení čtení registračních značek na jednoduché množině

t [min]	Počet správně přečtených symbolů							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	19470 95,91 %	645 3,18 %	135 0,67 %	34 0,17 %	13 0,06 %	3 0,15 %	0 0,00 %	0 0,00 %
23	↑ 20038 98,70 %	↓ 209 1,03 %	↓ 34 0,17 %	↓ 14 0,07 %	↓ 3 0,01 %	↓ 0 0,00 %	~ 0 0,00 %	~ 0 0,00 %
38	↑ 20134 99,18 %	↓ 146 0,72 %	↓ 14 0,07 %	↓ 5 0,02 %	↓ 1 0,005 %	~ 0 0,00 %	~ 0 0,00 %	~ 0 0,00 %
61	↑ 20193 99,47 %	↓ 86 0,42 %	↑ 16 0,08 %	↓ 2 0,01 %	↑ 2 0,01 %	~ 0 0,00 %	↑ 1 0,005 %	~ 0 0,00 %
108	↑ 20235 99,68 %	↓ 54 0,27 %	↓ 8 0,04 %	~ 2 0,01 %	↓ 0 0,00 %	↑ 1 0,005 %	↓ 0 0,00 %	~ 0 0,00 %

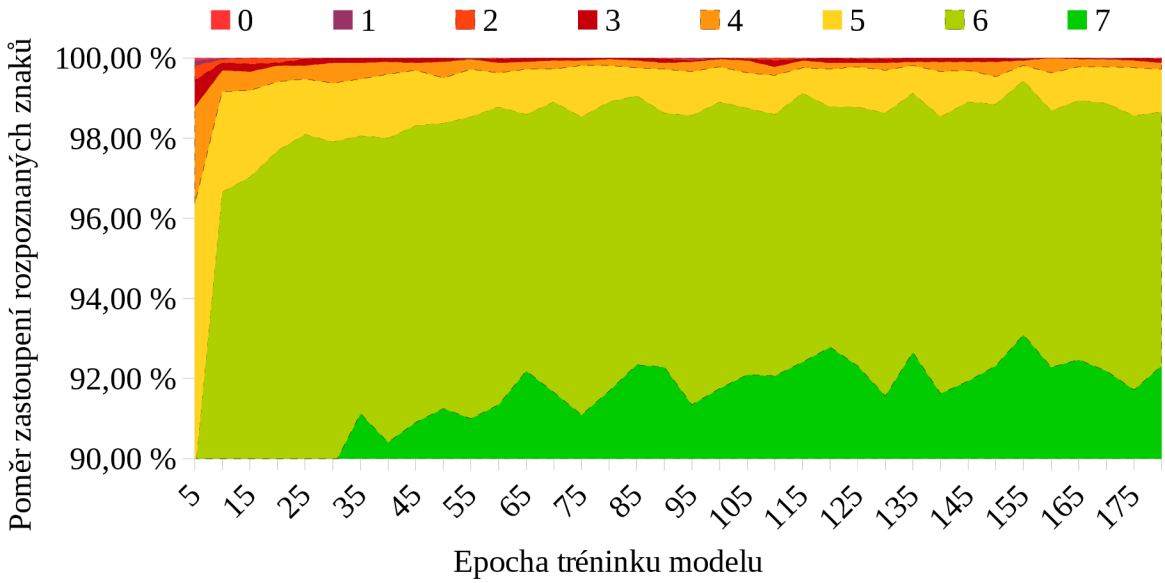
Tabulka 3: Chybné detekce pro model na jednoduché množině (t = 108 min)

Záměna	7 (6)	8 (9)	5 (4)	6 (5)	B (8)	9 (A)
četnost	5	4	4	4	4	4
příklady						

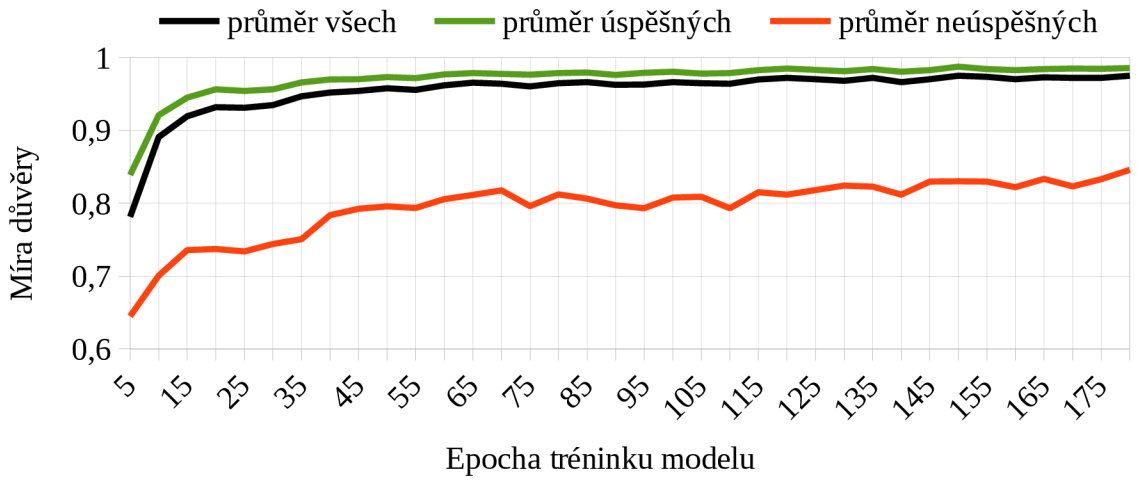
6.2.2 Obtížná množina

Validace během tréninku – V_T

Během tréninku byla sledován, která obsahuje $n = 3\,200$ vzorků. Značky jsou generovány v sekvenci ze 32 povolených znaků: *AAAAAAA*, *BBBBBBB*, až *9999999*. To je 32 variant, kde lze ověřit detekci znaku na každé pozici.



Obr. 34: Učení sítě pro čtení značky, obtížná množina, validace V_T



Obr. 35: Růst míry důvěry (confidence), obtížná množina, validace V_T

Míra důvěry (confidence) je průměrně 0,9749, pro úspěšné detekce je průměrná hodnota 0,9856, a pro detekce s chybou 0,8459.

Tabulka 4: Počty detekcí na obtížné množině, validace V_T

Počet správně přečtených symbolů V_T							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
2954	203	34	5	3	1	0	0
92,31 %	6,34 %	1,06 %	0,15 %	0,09 %	0,01 %	0,00 %	0,00 %

Validace výsledná – V_V

Pro konečnou validaci byla náhodně generována množina značek n =200 000 vzorků. Míra důvěry (confidence) je průměrně 0,9786, pro úspěšné detekce je průměrná hodnota 0,9911, a pro detekce s chybou 0,8434. Tedy potvrzuje se výsledek, že je možné stanovit hodnotu hranice pro vyhodnocení detekce.

Nejčastější záměny jsou: 360x přečteno K namísto X; 251x přečteno 8 namísto B; 226x přečteno B namísto 8; 214x přečteno I namísto Y; 204x přečteno C namísto 0.

Tabulka 5: Počty detekcí na obtížné množině, validace V_V

Počet správně přečtených symbolů V_V							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
183 084	13 724	2 209	591	239	95	44	14
91,54 %	6,86 %	1,10 %	0,30 %	0,12 %	0,05 %	0,02 %	0,01 %

Tabulka 6: Příklady čtení značek, validace V_V

Načteno	**U05NYTJ**	**U82V010**	**SEK7TTJ**	**K019S89**	**V23BJ33**	**EAYAAIR**
Obraz						
Počet s. Míra d.	0 0,4516	0 0,2026	1 0,5613	2 0,4860	3 0,5078	4 0,6488
Načteno	**M84S3UC**	**6KJL22J**	**AB4IPND**	**E0K69ZH**	**DH54X6U**	**HBHNX18**
Obraz						
Počet s. Míra d.	5 0,7182	5 0,7585	6 0,8483	6 0,7640	7 0,9711	7 1,0000

6.3 Čtení reálné značky z kamery

Z experimentů vyplývá, že při použité metodě konvolučních filtrů 5x5 musí být v dobrém poměru velikost obrazu registrační značky vůči velikosti artefaktů ztrátové komprese H.264/MPEG-4 AVC, kterou kamera používá. Problémové obrazy jsou často na hranici čitelnosti i pro člověka. Lze uvažovat o ztrátové kompresi trénovací množiny a síť naučit rozpoznávat značky z obrazu s nepříznivým poměrem velikosti artefaktů vůči velikosti čtených znaků.

Tabulka 7: Příklady čtení reálné značky z kamery

č. vz.	Načteno	Obraz	Rozpoznané znaky	Míra důvěry	Poznámka
1	**D8641TI**		3	0,2556	nízké rozlišení, artefakty komprese
2	**8B64177**		7	0,9783	vyhlazení artefaktů komprese
3	**8B64177**		7	0,7168	práhování
4	**SEIIII7**		0	0,0097	nízké rozlišení a kontrast, artefakty komprese
5	**CZI5JJD**		1	0,35687	změna kontrastu, artefakty přetrvávají

7 ZÁVĚR

Současný stav řešení problému automatického rozpoznávání registračních značek, knihoven pro deep learning a architektuře klient/server aplikací byl popsán a zohledněn při implementaci výsledného řešení.

Využití *transfer learningu* u sítí pro detekci objektů, tedy automobilu a registrační značky, se ukázalo jako velmi vhodná metoda dosahující vysoké přesnosti a použitelnosti při poměrně malé trénovací množině. Jako ideální varianta pro řešení problému, vzhledem k poměru přesnosti a rychlosti zpracování výpočtu, se jeví síť SSDmobilenetV1.

Čtení znaků na značce se ukazuje jako funkční, ovšem problematické se zvolenou IP kamerou a jejím umístění. Obraz byl často kvůli vzdálenosti a artefaktům ztrátové komprese na hranici čitelnosti i pro člověka. Proces učení se ukazuje jako vyhovující.

Další vývoj bude směřovat k zvýšení vstupního rozlišení a komplexnosti sítě (více konvolučních vrstev) pro čtení textu registračních značek s možností čtení značek na přání, dvouřádkových, zahraničních a speciálních. Lze uvažovat o možnostech zpracování přímo barevného obrazu, nebo naopak provádět předzpracování obrazu klasickými metodami, kterým se autor snažil v práci vyhnout. Také je vhodné do trénovací množiny zahrnout reálná data z provozu s manuální korekcí.

8 SEZNAM POUŽITÉ LITERATURY

[1] ŠÍMA, Jiří a NERUDA Roman. *Teoretické otázky neuronových sítí*. Vyd. 1. Praha: Matfyzpress, 1996. 390 s. ISBN 80-85863-18-9.

[2] LIU, Wei, et al. Ssd: Single shot multibox detector. In: *European conference on computer vision*. Springer, Cham, 2016. p. 21-37.

[3] HUANG J, RATHOD V, SUN C, ZHU M, KORATTIKARA A, FATHI A, FISCHER I, WOJNA Z, SONG Y, GUADARRAMA S, MURPHY K: *Speed/accuracy trade-offs for modern convolutional object detectors*. CVPR 2017

[4] G. HOWARD, Andrew; ZHU, Menglong; CHEN, Bo; KALENICHENKO, Dmitry; WANG, Weijun; WEYLAND, Tobias; Andreetto, Marco; Adam, Hartwig: *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. (2017).

[5] SANDLER, Mark, et al. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. arXiv preprint arXiv:1801.04381, 2018.

[6] OQUAB, Maxime, et al. Learning and transferring mid-level image representations using convolutional neural networks. In: *Computer Vision and Pattern Recognition* (CVPR), 2014 IEEE Conference on. IEEE, 2014. p. 1717-1724.

[7] JØRGENSEN, Hogne. Automatic License Plate Recognition using Deep Learning Techniques. 2017. Master's Thesis. NTNU.

[8] UIJLINGS, Jasper RR, et al. Selective search for object recognition. International journal of computer vision, 2013, 104.2: 154-171.

[9] NGUYEN, Anh; YOSINSKI, Jason; CLUNE, Jeff. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015. p. 427-436.

[10] G. Howard, Andrew & Zhu, Menglong & Chen, Bo & Kalenichenko, Dmitry & Wang, Weijun & Weyand, Tobias & Andreetto, Marco & Adam, Hartwig. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.

[11] KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] CHEN, Xinlei, et al. Microsoft COCO captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.

[13] GEIGER, Andreas, et al. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 2013, 32.11: 1231-1237.

[14] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. 2012. p. 1097-1105.

[15] HE, Kaiming, et al. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016. p. 770-778.

[16] *Dokumentace projektu OpenALPR*, dostupné z https://github.com/openalpr /openalpr/wiki/OpenALPR-Design

[17] EARL, Matthew. Number plate recognition with Tensorflow, *Matt's ramblings*, dostupné z *https://matthewearl.github.io/2016/05/06/cnn-anpr/* [poslední přístup 2018-05-09]

[18] GOODFELLOW, Ian J., et al. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[19] Stark, Fabian, et al.: *CAPTCHA recognition with active deep learning.*, Workshop New Challenges in Neural Computation 2015. Citeseer, 2015. Dostupné z: https://vision.in.tum.de/_media/spezial/bib/stark-gcpr15.pdf

[20] SAUVOLA, Jaakko; PIETIKÄINEN, Matti. Adaptive document image binarization. *Pattern recognition*, 2000, 33.2: 225-236.

[21] TSAI, Grace. *Histogram of oriented gradients*. University of Michigan, 2010, 1.1: 1-17.

[22] WILLIAMS, ANTHONY, 2017. *Deep Learning with Keras: Introduction to Deep Learning with Keras*. 2nd ed. B.m.: Createspace.

[23] FANDANGO, ARMANDO and NICK MCCLURE, 2018. *Mastering TensorFlow 1.x*. Birmingham: Packt Publishing.

[24] REDMON, Joseph; FARHADI, Ali. YOLOv3: An Incremental Improvement. *arXiv preprint arXiv*:1804.02767, 2018.

[25] HAHNLOSER, Richard HR, et al. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 2000, 405.6789: 947.

[26] PetarV. 2D Convolution, *github*, dostupné z https://github.com/PetarV-/TikZ /tree/master/2D%20Convolution

[27] KAWAHARA, Jeremy, 2017. *What is the derivative of ReLU?*, dostupné online z http://kawahara.ca/what-is-the-derivative-of-relu/

[28] PEREZ, Carlos E., W*hy Deep Learning is Radically Different from Machine Learning*, *medium.com* dostupné z https://medium.com/intuitionmachine/why-deep-learning-is-radically-different-from-machine-learning-945a4a65da4d

[29] PARAGIOS, Nikos., *Computer Vision Research: The deep "depression"*, dostupné z https://www.linkedin.com/pulse/computer-vision-research-my-deep-depression-nikos-paragios/

[30] COPELAND, Michael, 2016. *What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?*, dostupné z https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

[31] BAKKER, Indra den, 2017. *Battle of the Deep Learning frameworks—Part I: 2017, even more frameworks and interfaces*, dostupné z https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

9 SEZNAM OBRÁZKŮ A TABULEK

Seznam obrázků

Seznam tabulek

10 SEZNAM PŘÍLOH

CD s textem práce.
Příloha A – Detektor vozidel a značek
Příloha B – Matice pravděpodobností přečtení znaků

PŘÍLOHA A – DETEKTOR VOZIDEL A ZNAČEK

```
import tensorflow as tf
import collections
import os

Detection = collections.namedtuple('Detection', ['class_id',
                                                 'label',
                                                 'score',
                                                 'bbox'])
BBox = collections.namedtuple('BBox', ['x1', 'y1', 'x2', 'y2'])

class Detector:
  IG_PATH_DEFAULT = os.path.join(os.path.dirname(__file__),
                                 'exported_model',
                                 'ssd_mobilenet_v1_coco',
                                 'frozen_inference_graph.pb')

  def __init__(self,
               tf_session,
               score_threshold,
               inference_graph_path=IG_PATH_DEFAULT):
    self.score_threshold = score_threshold
    self.label_map = {1: 'car',
                      2: 'plate'}
    self.tf_session = tf_session

    # load the frozen graph
    with tf.gfile.FastGFile(inference_graph_path, 'rb') as f:
      self.graph_def = tf.GraphDef()
      self.graph_def.ParseFromString(f.read())

    # use the current graph in session for computation
    self.tf_session.graph.as_default()
    tf.import_graph_def(self.graph_def, name='')
```

```
    # get the output tensors from graph
    self.num_detections_tensor = \
      tf_session.graph.get_tensor_by_name('num_detections:0')
    self.detection_scores_tensor = \
      tf_session.graph.get_tensor_by_name('detection_scores:0')
    self.detection_boxes_tensor = \
      tf_session.graph.get_tensor_by_name('detection_boxes:0')
    self.detection_classes_tensor = \
      tf_session.graph.get_tensor_by_name('detection_classes:0')
    self.tensor_list = [self.num_detections_tensor,
                        self.detection_scores_tensor,
                        self.detection_boxes_tensor,
                        self.detection_classes_tensor]

  def infere(self, img_rgb):
      rows = img_rgb.shape[0]
      cols = img_rgb.shape[1]
      # use image data as input for the graph
      feed_dict = {'image_tensor:0':
                   img_rgb.reshape(1, rows, cols, 3)}
      # compute all tensors in list with image input
      out = self.tf_session.run(self.tensor_list,
                                feed_dict=feed_dict)
      detections = []
      num_detections = int(out[0][0])
      for i in range(num_detections):
          score = float(out[1][0][i])
          if score > self.score_threshold:
              class_id = int(out[3][0][i])
              bbox = tuple(float(v) for v in out[2][0][i])
              x1 = int(bbox[1] * cols); y1 = int(bbox[0] * rows)
              x2 = int(bbox[3] * cols); y2 = int(bbox[2] * rows)
              inf = Detection(class_id=class_id,
                              label=self.label_map[class_id],
                              score=score,
                              bbox=BBox(x1, y1, x2, y2))
              detections.append(inf)
      return detections
```

PŘÍLOHA B – MATICE PRAVDĚPODOBNOSTÍ PŘEČTENÍ ZNAKŮ

$$\mathbf{P} = \begin{pmatrix}
p_{0,1} & p_{0,2} & p_{0,3} & p_{0,4} & p_{0,5} & p_{0,6} & p_{0,7} \\
p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} & p_{1,5} & p_{1,6} & p_{1,7} \\
p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} & p_{2,5} & p_{2,6} & p_{2,7} \\
p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} & p_{3,5} & p_{3,6} & p_{3,7} \\
p_{4,1} & p_{4,2} & p_{4,3} & p_{4,4} & p_{4,5} & p_{4,6} & p_{4,7} \\
p_{5,1} & p_{5,2} & p_{5,3} & p_{5,4} & p_{5,5} & p_{5,6} & p_{5,7} \\
p_{6,1} & p_{6,2} & p_{6,3} & p_{6,4} & p_{6,5} & p_{6,6} & p_{6,7} \\
p_{7,1} & p_{7,2} & p_{7,3} & p_{7,4} & p_{7,5} & p_{7,6} & p_{7,7} \\
p_{8,1} & p_{8,2} & p_{8,3} & p_{8,4} & p_{8,5} & p_{8,6} & p_{8,7} \\
p_{9,1} & p_{9,2} & p_{9,3} & p_{9,4} & p_{9,5} & p_{9,6} & p_{9,7} \\
p_{\mathrm{A},1} & p_{\mathrm{A},2} & p_{\mathrm{A},3} & p_{\mathrm{A},4} & p_{\mathrm{A},5} & p_{\mathrm{A},6} & p_{\mathrm{A},7} \\
p_{\mathrm{B},1} & p_{\mathrm{B},2} & p_{\mathrm{B},3} & p_{\mathrm{B},4} & p_{\mathrm{B},5} & p_{\mathrm{B},6} & p_{\mathrm{B},7} \\
p_{\mathrm{C},1} & p_{\mathrm{C},2} & p_{\mathrm{C},3} & p_{\mathrm{C},4} & p_{\mathrm{C},5} & p_{\mathrm{C},6} & p_{\mathrm{C},7} \\
p_{\mathrm{D},1} & p_{\mathrm{D},2} & p_{\mathrm{D},3} & p_{\mathrm{D},4} & p_{\mathrm{D},5} & p_{\mathrm{D},6} & p_{\mathrm{D},7} \\
p_{\mathrm{E},1} & p_{\mathrm{E},2} & p_{\mathrm{E},3} & p_{\mathrm{E},4} & p_{\mathrm{E},5} & p_{\mathrm{E},6} & p_{\mathrm{E},7} \\
p_{\mathrm{F},1} & p_{\mathrm{F},2} & p_{\mathrm{F},3} & p_{\mathrm{F},4} & p_{\mathrm{F},5} & p_{\mathrm{F},6} & p_{\mathrm{F},7} \\
p_{\mathrm{H},1} & p_{\mathrm{H},2} & p_{\mathrm{H},3} & p_{\mathrm{H},4} & p_{\mathrm{H},5} & p_{\mathrm{H},6} & p_{\mathrm{H},7} \\
p_{\mathrm{I},1} & p_{\mathrm{I},2} & p_{\mathrm{I},3} & p_{\mathrm{I},4} & p_{\mathrm{I},5} & p_{\mathrm{I},6} & p_{\mathrm{I},7} \\
p_{\mathrm{J},1} & p_{\mathrm{J},2} & p_{\mathrm{J},3} & p_{\mathrm{J},4} & p_{\mathrm{J},5} & p_{\mathrm{J},6} & p_{\mathrm{J},7} \\
p_{\mathrm{K},1} & p_{\mathrm{K},2} & p_{\mathrm{K},3} & p_{\mathrm{K},4} & p_{\mathrm{K},5} & p_{\mathrm{K},6} & p_{\mathrm{K},7} \\
p_{\mathrm{L},1} & p_{\mathrm{L},2} & p_{\mathrm{L},3} & p_{\mathrm{L},4} & p_{\mathrm{L},5} & p_{\mathrm{L},6} & p_{\mathrm{L},7} \\
p_{\mathrm{M},1} & p_{\mathrm{M},2} & p_{\mathrm{M},3} & p_{\mathrm{M},4} & p_{\mathrm{M},5} & p_{\mathrm{M},6} & p_{\mathrm{M},7} \\
p_{\mathrm{N},1} & p_{\mathrm{N},2} & p_{\mathrm{N},3} & p_{\mathrm{N},4} & p_{\mathrm{N},5} & p_{\mathrm{N},6} & p_{\mathrm{N},7} \\
p_{\mathrm{P},1} & p_{\mathrm{P},2} & p_{\mathrm{P},3} & p_{\mathrm{P},4} & p_{\mathrm{P},5} & p_{\mathrm{P},6} & p_{\mathrm{P},7} \\
p_{\mathrm{R},1} & p_{\mathrm{R},2} & p_{\mathrm{R},3} & p_{\mathrm{R},4} & p_{\mathrm{R},5} & p_{\mathrm{R},6} & p_{\mathrm{R},7} \\
p_{\mathrm{S},1} & p_{\mathrm{S},2} & p_{\mathrm{S},3} & p_{\mathrm{S},4} & p_{\mathrm{S},5} & p_{\mathrm{S},6} & p_{\mathrm{S},7} \\
p_{\mathrm{T},1} & p_{\mathrm{T},2} & p_{\mathrm{T},3} & p_{\mathrm{T},4} & p_{\mathrm{T},5} & p_{\mathrm{T},6} & p_{\mathrm{T},7} \\
p_{\mathrm{U},1} & p_{\mathrm{U},2} & p_{\mathrm{U},3} & p_{\mathrm{U},4} & p_{\mathrm{U},5} & p_{\mathrm{U},6} & p_{\mathrm{U},7} \\
p_{\mathrm{V},1} & p_{\mathrm{V},2} & p_{\mathrm{V},3} & p_{\mathrm{V},4} & p_{\mathrm{V},5} & p_{\mathrm{V},6} & p_{\mathrm{V},7} \\
p_{\mathrm{X},1} & p_{\mathrm{X},2} & p_{\mathrm{X},3} & p_{\mathrm{X},4} & p_{\mathrm{X},5} & p_{\mathrm{X},6} & p_{\mathrm{X},7} \\
p_{\mathrm{Y},1} & p_{\mathrm{Y},2} & p_{\mathrm{Y},3} & p_{\mathrm{Y},4} & p_{\mathrm{Y},5} & p_{\mathrm{Y},6} & p_{\mathrm{Y},7} \\
p_{\mathrm{Z},1} & p_{\mathrm{Z},2} & p_{\mathrm{Z},3} & p_{\mathrm{Z},4} & p_{\mathrm{Z},5} & p_{\mathrm{Z},6} & p_{\mathrm{Z},7}
\end{pmatrix}$$