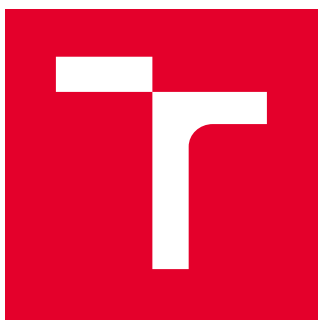


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky  
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE



# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

## ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

## POKROČILÁ ANALÝZA POHYBUJÍCÍCH SE OBJEKTŮ V OBRAZE

ADVANCED ANALYSIS OF MOVING OBJECTS IN THE IMAGE

### BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

### AUTOR PRÁCE

AUTHOR

Ivan Medynskyi

### VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Martin Kiac

BRNO 2021

# Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

**Student:** Ivan Medynskyi

**ID:** 195390

**Ročník:** 3

**Akademický rok:** 2020/21

**NÁZEV TÉMATU:**

## **Pokročilá analýza pohybujících se objektů v obraze**

### **POKYNY PRO VYPRACOVÁNÍ:**

Cílem práce je navrhnout a realizovat aplikaci pro analýzu pohybujících se objektů v obraze. Aplikace by měla využívat na práci s obrazem klasické metody zpracování obrazu a pro detekci pohybujících se objektů ve scéně obrazu vhodnou neuronovou síť. Zvolená neuronová síť by měla být schopna rozeznávat a detekovat různé typy objektů vyskytujících se v silničním provozu jako jsou například osoby, motorové vozidla a podobně. Za tímto účelem je třeba vytvořit vlastní trénovací množinu, pomocí které bude zvolena neuronová síť následně trénovaná. Výsledná aplikace by měla demonstrovat problematiku detekce objektů v obraze využitím neuronových sítí.

### **DOPORUČENÁ LITERATURA:**

[1] Bradski, G., Kaehler, A. Learning OpenCV 3. 1. vydání. O'Reilly Media, 2016. 1024 s. ISBN 978-1-491-93-99-0.

[2] Krizhevsky, A., Sutskever, I., Hinton G. E., ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems 25, 2012. p. 1097-1105.

**Termín zadání:** 1.2.2021

**Termín odevzdání:** 31.5.2021

**Vedoucí práce:** Ing. Martin Kiac

**prof. Ing. Jiří Mišurec, CSc.**  
předseda rady studijního programu

### **UPOZORNĚNÍ:**

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

## ABSTRAKT

Tato práce je zaměřena na zpracování obrazu pomocí knihovny OpenCV a detekci pohybujících se objektů ve videu s použitím konvolučních neuronových sítí. Výsledná aplikace umí detekovat pohybující se objekty z videa nebo v reálném čase a obsahuje uživatelské grafické rozhraní, pomocí kterého uživatel může jednoduše aplikaci ovládat. Součástí této aplikace je konvoluční model YOLO, který je určen k detekci pohybujících se objektů.

## KLÍČOVÁ SLOVA

Detekce objektů, datová sada, konvoluční neuronové sítě, OpenCV, YOLO

## ABSTRACT

This work focuses on image processing using the OpenCV library and detecting moving objects in video using convolutional neural networks. The resulting application can detect moving objects in video or in real time and includes a user interface that allows the user to easily control the application. Part of this application is the YOLO convolutional model, which is designed to detect moving objects.

## KEYWORDS

Convolutional neural networks, dataset, object detection, OpenCV, YOLO

MEDYNSKYI, Ivan. *Pokročilá analýza pohybujících se objektů v obraze*. Brno, 2021, 49 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Martin Kiac,

## PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Pokročilá analýza pohybujících se objektů v obraze“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno .....

.....

podpis autora

## PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Martinu Kiacovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

# Obsah

Úvod	9
<b>1 Teoretická část studentské práce</b>	<b>10</b>
1.1 Visual Studio	10
1.2 Qt	10
1.3 OpenCV	10
1.3.1 Historie OpenCV	10
1.3.2 Proč OpenCV	11
1.3.3 Struktura OpenCV	11
1.4 Teorie zpracování obrazu	12
1.4.1 Snímání a digitalizace obrazu	12
1.4.2 Předzpracování obrazu	13
1.4.3 Segmentace obrazu	17
1.4.4 Popis a klasifikace objektu	19
1.5 Neuronová síť	20
1.5.1 Neuron	20
1.5.2 Fungování neuronové sítě	21
1.5.3 Synapse	22
1.5.4 Funkce aktivace	22
1.5.5 Učení neuronových sítí	23
1.5.6 Výpočet chyby NS	24
1.6 Konvoluční neuronová síť	25
1.6.1 Faster R-CNN	27
1.6.2 SSD: Single Shot MultiBox Detector	27
1.6.3 You Only Look Once — YOLO	28
1.6.4 Popis jak pracuje YOLO	28
<b>2 Návrh a realizace řešení práce</b>	<b>31</b>
2.1 Příprava softwaru	31
2.2 Datová sada	32
2.2.1 Co v sobě obsahuje dataset	32
2.2.2 Příklady datových sad	33
2.2.3 Jak přečíst datovou sadu COCO	34
2.2.4 Yolo_mark	36
2.2.5 Automatická anotace programem CVAT	36
2.3 Trénování	37
2.3.1 Příprava souboru pro trénování modelu detektoru YOLOv4-tiny	38

2.3.2	Trénování sítě YOLOv4-tiny . . . . .	39
2.3.3	Výsledky a porovnání natrénovaného modelu yolov4-tiny a yolov4 . . . . .	40
2.3.4	Chyby v aplikaci a jak zlepšit výsledky . . . . .	42
2.3.5	Grafické uživatelské rozhraní aplikace . . . . .	43
	<b>Závěr</b>	<b>45</b>
	<b>Literatura</b>	<b>46</b>
	<b>Seznam symbolů, veličin a zkratk</b>	<b>48</b>
	<b>A Obsah přiloženého CD</b>	<b>49</b>

# Seznam obrázků

1.1	Struktura OpenCV . . . . .	12
1.2	Čtvercová a hexagonální obrazová matice . . . . .	13
1.3	Výsledek aplikace Gaussova rozostření na vstupní obraz . . . . .	14
1.4	Typy jasové transformace . . . . .	14
1.5	Translace bodu a rotace bodu kolem počátku . . . . .	15
1.6	Příklad změny měřítka ve směru x . . . . .	16
1.7	Příklad zkosení ve směru osy x . . . . .	17
1.8	Histogram s viditelným prahem . . . . .	18
1.9	Sobelovy gradientní operátory . . . . .	19
1.10	Ukázka detekce hranic . . . . .	19
1.11	Struktura neuronové sítě . . . . .	20
1.12	Struktura neuronu . . . . .	21
1.13	Ukázka, jak pracuje synapse . . . . .	22
1.14	Graf Sigmoid . . . . .	23
1.15	Graf Hyperbolicky tangens . . . . .	23
1.16	Učení neuronové sítě s učitelem . . . . .	24
1.17	Struktura CNN . . . . .	26
1.18	Rozpoznávání obrazu člověkem a počítačem . . . . .	26
1.19	Konvoluce vrstvy . . . . .	27
1.20	Ukázka, jak pracuje YOLO . . . . .	28
1.21	Ukázka, jak vypadá a co v sobě obsahuje tensor $7 \times 7 \times 30$ . . . . .	29
1.22	Struktura sítě YOLOv4 . . . . .	30
2.1	Příklad anotovaného snímku . . . . .	33
2.2	Příklad nevhodné datové sady pro síť YOLO . . . . .	34
2.3	Příklad vhodné datové sady pro síť YOLO . . . . .	34
2.4	Ukázka zpracovaného obrazu v GoogleColab . . . . .	36
2.5	Ukázka aplikaci yolo mark . . . . .	37
2.6	Priebeh mAP počas procesu trénování dvou modelu detektoru YOLO . . . . .	41
2.7	Priebeh strátovosti počas procesu trénování dvou modelu detektoru YOLO . . . . .	41
2.8	Výpočet metrik CNN . . . . .	42
2.9	Chyby detekcí . . . . .	43
2.10	Ukázka aplikace . . . . .	44

# Úvod

Člověk dostává očima hodně informací o vnějším světě a poté umí velmi efektivně zpracovávat přijaté informace. V současnosti se počítače též naučily analyzovat obraz přijatý od kamery a získalo to název *počítačové vidění*. Postupem času si počítačové vidění začalo získávat velkou slávu a dnes se používá v různých oblastech, jako je automatizace, medicína, ochrana, ale také ve výrobě pro zlepšení a kontrolu kvality výrobku apod.

Bakalářská práce je věnována analýze pohybujících se objektů v obraze. Cílem této práce je implementace GUI aplikací, která bude schopná detekovat objekty s využitím konvolučních neuronových sítí. Aplikace bude podporovat instalace videa a pomocí dalších nastavení bude možné rozpoznat tyto objekty: auta, lidi, motorky, autobusy apod. K tomu bude potřeba vytvořit vlastní trénovací množinu, pomocí které bude zvolena neuronová síť následně trénovaná.

Tato práce je rozdělena do dvou částí: teoretická a praktická, ty se následně skládají z několika kapitol. Na začátku teoretické části je popsán software vhodný pro řešení úkolu bakalářské práce. V další části je možné se seznámit s problematikou analýzy a zpracování obrazů, této části je třeba věnovat pozornost před řešením hlavního úkolu práce. Pak v bakalářské práci se popisuje co je neuronová síť, s čeho se skládá, jak se učí atd. Poslední úsek teoretické části vysvětluje, co je konvoluční neuronová síť, jaké typy konvolučních modelů existují a jak jsou tyto modely schopné detekovat různé objekty.

Úvodní kapitola praktické části je věnována přípravě softwaru vhodného pro zpracování obrazu a trénování modelu detektoru. V další části je možnost se seznámit s datovými sady, jsou uvedené příklady vhodných a naopak nevhodných datasetů pro trénování modelu detektoru *YOLO*. Poté jsou popsány metody přípravy vlastní datové sady. Následující kapitola praktické části popisuje postup přípravy souborů potřebných pro trénování dvou konvolučních modelů *YOLO* (*yolov4* a *yolov4-tiny*), spouštění trénování a srovnání výsledků obou modelů po trénování. Na konci práce jsou zmíněny chyby aplikace a jsou navržena řešení pro zlepšení detekce.

V závěru této práce byla vytvořena aplikace, která umí detekovat celkem sedm objektů: auta, motorky, jízdní kola, autobusy, kamiony, dodávky a lidi. Detekce je možná provést na trénovacím modelu *yolov4* nebo *yolov4-tiny*.

# 1 Teoretická část studentské práce

## 1.1 Visual Studio

Na řešení problematiky této práce je použito vývojové prostředí *Visual Studio* tzv. IDE (Integrated Development Environment), které vytvořila společnost Microsoft. Samotné prostředí *Visual Studio* obsahuje široké množství nástrojů, podporuje různé technologie a umožňuje vytvářet software pro rozličné platformy. Využitím prostředí Visual Studio je možné pracovat například v programovacím jazyce *C++*, používat framework *Qt* nebo knihovnu *OpenCV*. Všechny tyto nástroje a technologie jsou použity k řešení problematiky této práce.

## 1.2 Qt

Framework *Qt* patří mezi často používané nástroje na vytváření aplikačního softwaru s grafickým uživatelským rozhraním. Tento framework je podporovaný na běžných operačních systémech jako například *Linux*, *Windows*, *Android*, *iOS*, *BlackBerry* a podobně. Tento framework je napsán v jazyce *C++*, může tak být kompilován jakýmkoliv známým kompilátorem pro *C++* jako jsou například *GCC* nebo *MinGW* [1]. Samotný nástroj *Qt* obsahuje intuitivní vývojové prostředí *Qt Creator*, jež umožňuje jednoduchý vývoj uživatelského rozhraní pro aplikace. Výhodou je, že samotné vývojové prostředí Visual Studio podporuje práci s *Qt* frameworkem. Využívat framework *Qt* ve vývojovém prostředí Visual Studio je možné pomocí nástroje *Qt Visual Studio Tools*, který lze do prostředí Visual Studia jednoduše nainstalovat.

## 1.3 OpenCV

Knihovna *OpenCV* patří mezi nejpoužívanější nástroje používané v oblasti počítačového vidění a zpracování obrazu. Knihovna je proto vhodným řešením pro programátory zabývající se počítačovým viděním.

### 1.3.1 Historie OpenCV

Samotná knihovna *OpenCV* byla původně vytvořena společností Intel, která knihovnu implementovala v programovacím jazyce *C/C++*. S knihovnou je možné pracovat v různých programovacích jazycích jako *Python*, *Java*, *Matlab* a používat ji na nejrozličnějších platformách jako jsou *Windows*, *Linux*, *MacOS*. Knihovna je velmi rozsáhlá, obsahuje tisíce optimalizovaných algoritmů. Protože je knihovna distribuována podle podmínek licence BSD, je možné ji využít k nekomerčnímu, ale

i komerčnímu použití. Knihovnu proto využívají nejen univerzity pro vědecké účely, ale i malé, střední a velké korporátní společnosti, jako jsou Google, Yahoo, Microsoft, Intel, IBM, Toyota [2].

### 1.3.2 Proč OpenCV

*OpenCV* je vhodným řešením pro zpracování obrazu. Jsou v ní hotové nástroje a metody určené pro zpracování obrazu a jeho analýzy. Celkem existuje kolem 2 500 algoritmů. Nyní pracují na vývoji nástrojů *CUDA* a *OpenCL*, již teď existuje kolem 500 algoritmů a cca 10 krát více funkcí podporujících tyto algoritmy. Knihovna je využita týmem lidí, proto je dobře dokumentována, podporována a aktualizována [2].

### 1.3.3 Struktura OpenCV

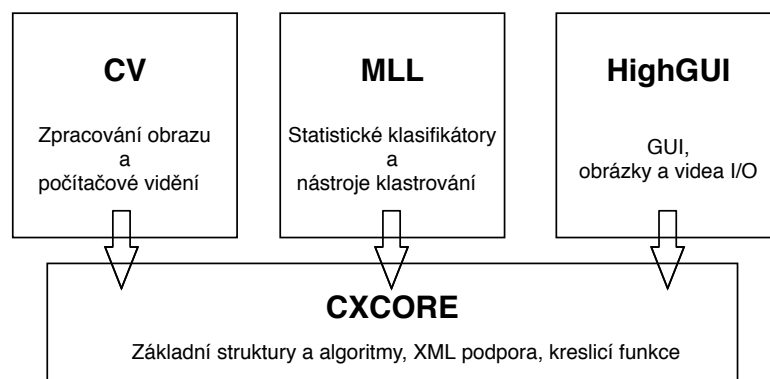
Teoreticky můžeme říct, že *OpenCV* je balík obsahující sadu datových typů, funkcí a tříd pro zpracování obrazu pomocí algoritmů počítačového vidění. Základní moduly knihovny jsou:

- *CXCORE* – jádro obsahuje základní datové struktury a algoritmy, například: maticová algebra, matematické funkce, generátory náhodných čísel, základní funkce 2D grafiky,
- *CV* – modul pro zpracování obrazu a počítačové vidění, využívá se pro detekci objektu a jeho analýzu (například morfologie, vyhledávání obrysů, histogramy). Také tento modul můžeme využít pro kalibrace kamery nebo základní operace s obrázky (filtrování, geometrické transformace, konverze barevného prostoru atd.),
- *HighGUI* – modul pro vstup nebo výstup obrázků a videa, načtení videa z kamery nebo video souboru,
- *Machine Learning (ML)* – knihovna, ve které je spousta funkcí pro analýzu a klasifikaci dat, tato knihovna rovněž obsahuje algoritmy pro trénování dat [7].

Od verze 2.2 se struktura trochu změnila. Již se nejedná o tak velké moduly jako *CXCORE*, *CV* nebo *HighGUI*. Tyto moduly byly rozděleny na menší pro funkční použití. Jsou to:

- *opencv\_core* – jádro, základní struktury, výpočty,
- *opencv\_imgproc* – práce s obrazy (filtry, transformace),
- *opencv\_highgui* – vstup nebo výstup obrázků a videa,
- *opencv\_ml* – metody a modely strojového učení,
- *opencv\_features2d* – různé deskriptory (SURF, VGG),
- *opencv\_video* – analýza pohybu a sledování objektů,
- *opencv\_objdetect* – detekce objektů v obraze (Haar-like feature, HOG),
- *opencv\_calib3d* – kalibrace kamery, prvky pro zpracování 3D dat,

- *opencv\_gpu* – zrychlení některých funkcí OpenCV s pomocí CUDA vyvinutou firmou NVidia [3].



Obr. 1.1: Struktura OpenCV, převzato z [7].

## 1.4 Teorie zpracování obrazu

V dnešní době získává počítačové vidění velkou popularitu. Využívá se v různých oblastech, jako je například medicína, průmysl, často v oblasti ochrany pro detekci a rozpoznání obličeje, aut apod. Pro ukázkou využití počítačového vidění může sloužit auto od firmy Tesla. Auto je vybaveno kamerami, pomocí kterých umí rozpoznat: znaky, vedlejší auta, silniční pruhy, semaforey atd. Odpověď na otázku, jak to celé funguje, je taková, že je to právě za pomoci počítačového vidění.

Zpracování obrazu je jednou z částí počítačového vidění. Celý proces je možné rozdělit na následující části:

- snímání a digitalizace obrazu,
- předzpracování obrazu,
- segmentace obrazu,
- popis objektů,
- klasifikace objektů [8].

### 1.4.1 Snímání a digitalizace obrazu

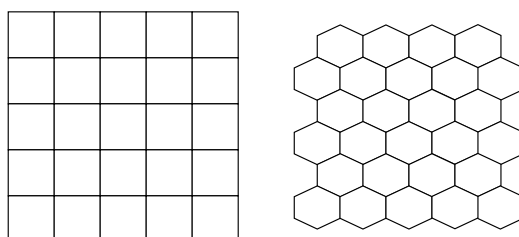
Dnes má každý člověk ve svém mobilu kameru, takže může v jakýkoliv moment fotit nebo natočit video. Všechny fotografie nebo videa se ukládají do paměti telefonu a lze je chránit po dlouhou dobu. V minulosti, když se využívaly kazety, se video zapisovalo pomocí analogového signálu. Ale kvůli demagnetizaci kazety mohou být některé soubory ztraceny bez možnosti obnovení. Pro uložení a obnovu souboru na dlouhou dobu je vhodným řešením využití digitálního signálu. Proto byl vymyšlen

proces digitalizace. Je to proces převodu spojitého analogového signálu na signál digitální [9]. Mezi postupy digitalizace patří: vzorkování obrazových funkcí do číslicové matice o rozměrech  $M \times N$  a kvantování do  $K$  stejně velkých intervalů.  $K$  - počet intervalů se vypočítá pomocí vztahu:

$$K = b^2, \quad (1.1)$$

kde  $b$  je počet bitů, do kterých je hodnota kvantována [10].

Dalším krokem digitalizace je volba obrazové matice. Nejčastěji se využívají dva typy: hexagonální a čtvercová (viz obrázek 1.2). Tato matice je výsledkem digitalizovaného obrazu. Prvkem této matice je tzv. pixel.



Obr. 1.2: Čtvercová a hexagonální obrazová matice, převzato z [9].

## 1.4.2 Předzpracování obrazu

Cílem předzpracování obrazu je jeho zlepšení pro další zpracování. Existuje několik metod, které můžeme potřebovat pro opravu nebo zlepšení původního obrázku:

- filtrace šumu,
- jasová transformace,
- geometrická transformace.

### Filtrace šumu

Šum je ten nejčastější problém v obrázku. Může vzniknout při digitalizaci, přenosu obrazu nebo v momentu snímání. Nejčastěji se pro jeho odstranění používá filtrace šumu. Často se v praxi setkáváme s tzv. bílým a Gaussovým šumem. Pro odstranění šumu se využívá Gaussův filtr. Existují také různé další metody, ale metoda použití Gaussova filtru patří mezi ty nejpoužívanější.

Gaussův filtr řadíme mezi nelineární filtry, slouží pro detekci hran a odstranění šumu z obrazu. Realizovat tento filtr lze pomocí Fourierovy transformace, která je vhodná pro využití ve frekvenční oblasti, nebo použitím operace konvoluce v prostoro-  
vé oblasti. Konvoluční jádro tohoto filtrování je představováno Gaussovou funkcí



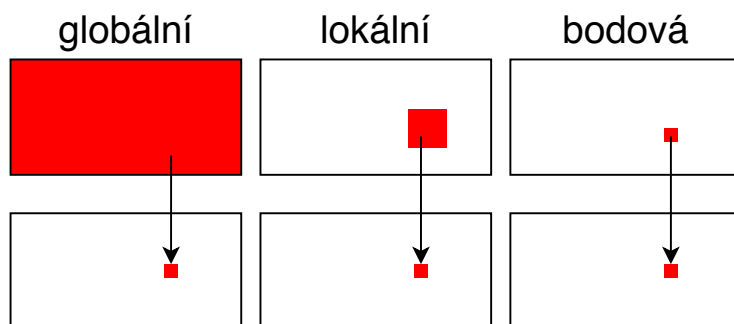
Obr. 1.3: Výsledek aplikace Gaussova rozostření na vstupní obraz, převzato z [11].

(viz rovnice 1.2). Gaussův filtr patří mezi filtry typu dolní propust, takže výsledkem této operace může být rozmazání obrazu (viz obrázek 1.3) [11].

$$I(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \quad (1.2)$$

### Jasová transformace

Jasová transformace patří mezi často používané operace v oblasti zpracování obrazu. Tato metoda pomáhá lépe pochopit obraz. Hezkým příkladem může být lehce rozmazaný rentgenový snímek. Po aplikování jasové transformace může takový snímek dosáhnout lepší pozorovací kvality. Ale při nevhodném použití jasové transformace se může stát, že některá informace bude ztracena. Jsou tři typy jasové transformace: globální, lokální a bodová (viz obrázek 1.4). Po transformaci má výstupní obrázek stejné rozlišení a bitovou hloubku jako vstupní obrázek [12].

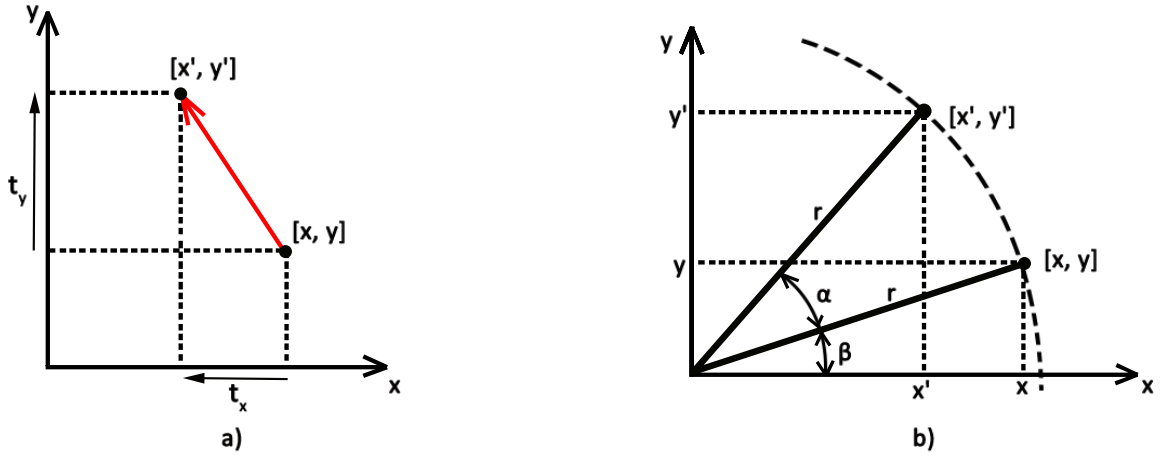


Obr. 1.4: Typy jasové transformace, převzato z [12].

### Geometrická transformace

Při transformaci se obsah obrazu nemění, hodnota pixelu tedy zůstává stejná, mění se jenom matice obrazu. To znamená, že hodnota pixelu se nemění, mění se jenom jeho poloha. Je několik typů transformací, které jsou určeny pro různé úkoly.

Nejpoužívanějším typem lineární transformace je tzv. *euklidovská transformace*. Využívá se pro translaci (posunutí) nebo rotaci obrazu. Příklad translace bodu je znázorněn na obrázku 1.5 (a).



Obr. 1.5: Translace bodu a rotace bodu kolem počátku, převzato z [15].

Po aplikování transformace daný bod změní svou polohu, tu je možné zapsat jako:

$$x' = x + t_x, \quad (1.3)$$

$$y' = y + t_y, \quad (1.4)$$

kde  $x'$ ,  $y'$  vyjadřují novou polohu bodu a  $t_x$ ,  $t_y$  velikost posunutí tohoto bodu. Vektorově je možné zapsat tento tvar jako:

$$x' = x + t, \text{ kde } t = \begin{bmatrix} t_x \\ t_y \end{bmatrix}. \quad (1.5)$$

Pro rotaci bodu v rovině kolem počátku z obrázku 1.5 (b) platí rovnice:

$$x' = x \cos \alpha - y \sin \alpha, \quad (1.6)$$

$$y' = x \sin \alpha + y \cos \alpha, \quad (1.7)$$

kde  $\alpha$  je úhel rotace obrazu. Vektorově má tvar:

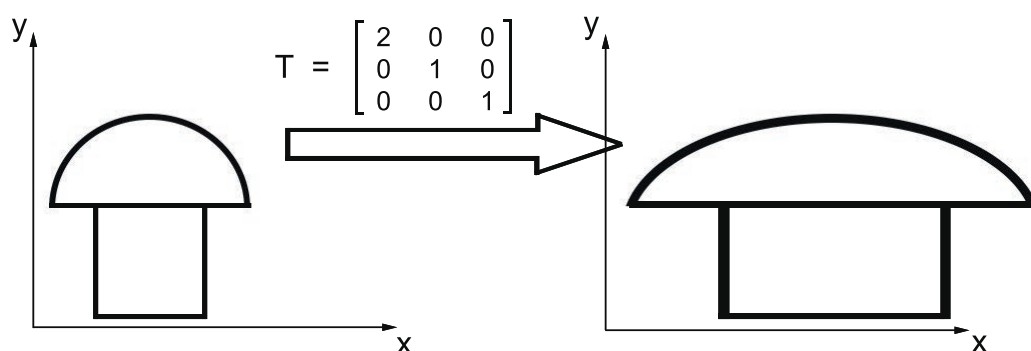
$$x' = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix} x. \quad (1.8)$$

Matice 1.5 a 1.8 představují dvě zvláštní operace: translace – součet a rotace – součin. Pro získání tzv. *euklidovské transformační matice* je potřebné výše uvedené matice

sloučit pomocí homogenních souřadnic [15]. Výsledný vztah může mít následující tvar:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & t_x \\ \sin\alpha & \cos\alpha & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \quad (1.9)$$

Dalším a také často používaným typem lineární transformace je *afinní transformace*. Tento typ transformace se využívá, když je například potřeba obrázky naklonit nebo změnit jeho velikost (roztáhnout). V případě změny velikosti si je potřeba uvědomit, že změna měřítka má vliv na celý obrázek nebo objekt v tomto obrázku. Výsledek této transformace je možné sledovat na obrázku 1.6.



Obr. 1.6: Příklad změny měřítka ve směru x, převzato z [15].

Obecný tvar této transformace lze znázornit následujícím vztahem:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad (1.10)$$

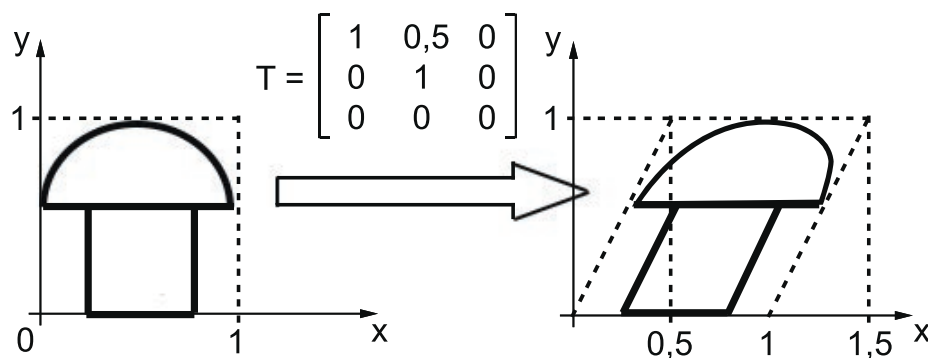
kde  $s_x$  a  $s_y$  představují parametry změny měřítka. Jsou to dva důležité koeficienty, pomocí kterých je možné pochopit, jak se obrázek mění. V případě, kdy se  $s_x$  rovná  $s_y$  nebo naopak, se v obrázku zachová poměr stran a obrázek se mění proporcionalně. Obrázek se zvětšuje, jestliže je jeden z těchto koeficientů větší než hodnota 1 ( $s > 1$ ). Když je hodnota koeficientu v rozsahu  $[0;1]$  – obrázek se zmenšuje. Hodnota koeficientu menší než 0 znamená převrácení obrázku [15].

Příklad zkosení je možné vidět na obrázku 1.7. Z obrázku je patrné, že zkosení probíhá nezávisle ve směru každé osy zvlášť. Zkosení navíc nemusí být jenom podle vodorovné nebo svislé osy, ale může být i pod určeným úhlem vzhledem k určené ose.

Obecný tvar této transformace vypadá takto:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & k_x & 0 \\ k_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad (1.11)$$

kde  $k_x$  a  $k_y$  jsou koeficienty míry zkosení příslušné osy [15].



Obr. 1.7: Příklad zkosení ve směru osy  $x$ , převzato z [15].

Po skládání těchto transformací bude mít, výsledná matice *afinní transformace* tvar:

$$T = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}. \quad (1.12)$$

### 1.4.3 Segmentace obrazu

Segmentace slouží pro analýzu obrazu a popis jeho obsahu pro další zpracování, může být kompletní nebo částečná. Rozdílem je jen to, že při částečné segmentaci nemusí části obrazu (segmenty) přímo korespondovat s objekty. Mezi běžně používané metody segmentace patří:

- prahování obrazu,
- detekce hran.

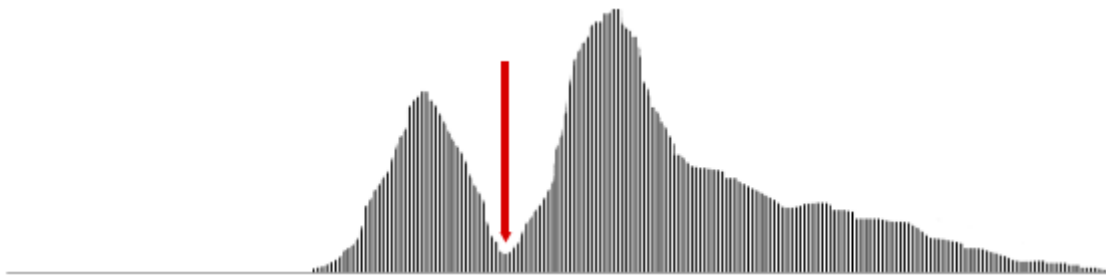
#### Prahování obrazu

První a velmi často používanou metodou segmentace je prahování obrazu. Často se používá díky své jednoduché implementaci a malé časové náročnosti při výpočtech. Jejím principem je nalezení hodnoty prahu v histogramu (viz obrázek 1.8). Pro nalezenou hodnotu bude platit, že každá hodnota, která bude větší než hodnota

prahu bude odpovídat popředí obrazu. Hodnoty, které budou menší než hodnota prahu budou odpovídat pozadí obrazu. Výsledkem prahování je binární obraz, ve kterém hodnota popředí se rovná 1, zatímco hodnota pozadí je 0 [8]. Prahování lze popsat pomocí rovnice:

$$g(x, y) = \begin{cases} 1 & \text{pro } f(x, y) > T \\ 0 & \text{pro } f(x, y) \leq T \end{cases}, \quad (1.13)$$

kde  $T$  je prahová hodnota,  $f(x, y)$  původní obraz,  $g(x, y)$  binární obraz.



Obr. 1.8: Histogram s viditelným prahem, převzato z [8].

## Detekce hran

Druhou metodou je detekce hran, jejím principem je vyrazení hranic objektu. Vyrazit hranici objektu můžeme pomocí Cannyho hranového detektoru. Tato metoda je velice citlivá na šum, proto je nutné provést před využitím této metody filtraci šumu (nejlépe Gaussovým filtrem). Pro zmenšení náročnosti při výpočtech musí být obrázek v odstínech šedi. Pak je potřeba najít gradienty funkce. V tomto nám pomůže *Sobel operator*.

*Sobel* funguje pomocí využití metody konvolucí obrazu pro výpočet aproximací derivátu – jedna pro horizontální změny, druhá pro vertikální. Je velice dobrou a často používanou metodou pro nalezení hranic. Filtrační masky *Sobelova operátoru* mají tvar matice  $3 \times 3$  a vypadají jako na obrázku 1.9. Hodnotu gradientu je možné vypočítat pomocí vztahu 1.14, dále je potřeba se dozvědět směr vektoru, na to je vhodný vztah 1.15.

$$G = \sqrt{G_x^2 + G_y^2}. \quad (1.14)$$

$$\theta = \left( \frac{G_y}{G_x} \right) \quad (1.15)$$

Dalším krokem je tzv. metoda *Non-maximum Suppression*. Jinými slovy je to odstranění nepotřebných proměnných, které nejsou hranicí objektu. Poznáme je tak,

-1	0	1
-2	0	2
-1	0	1

směr  $x$

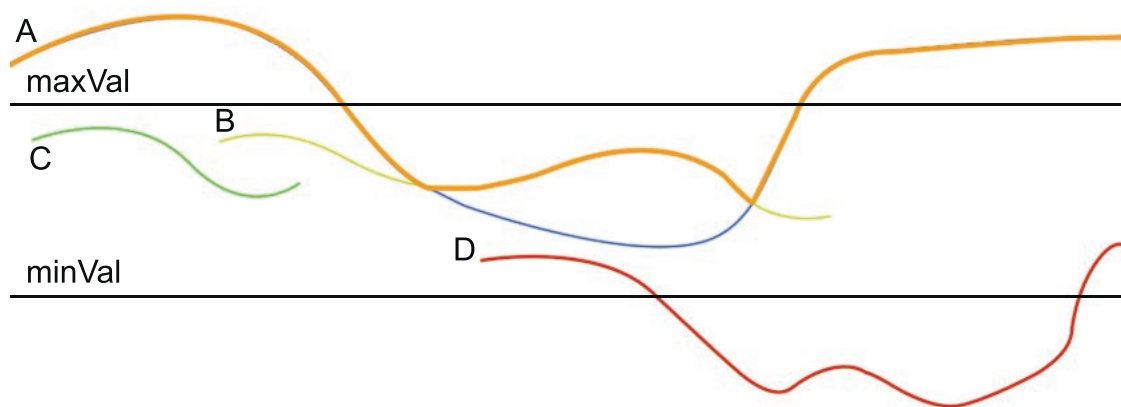
-1	-2	-1
0	0	0
1	2	1

směr  $y$

Obr. 1.9: Sobelovy gradientní operátory

že zvolíme dvě hodnoty  $maxVal$  a  $minVal$ . Všechny křivky, které budou nad hodnotou  $maxVal$ , budou hranicí objektu, křivky, které budou menší než hodnota  $minVal$ , budou vymazány, a nakonec křivky, které budou mezi výše řečenými hodnotami, budou rozpracovány podle dalšího principu.

Jestli se křivka nachází mezi hodnotami  $maxVal$  a  $minVal$  a je spojena s křivkou, která má hodnotu nad  $maxVal$ , můžeme tuto křivku považovat za hranici objektu. Všechny ostatní křivky budou vymazány. Obrázek 1.10 znázorňující čtyři křivky A, B, C a D slouží k hezkému příkladu detekcí hranic. V tomto příkladu křivka A (modrá) protíná křivku B (žlutá). Z výše řečených důvodů křivky C a D nejsou hranicí objektu. V důsledku toho máme oranžovou křivku, která ukazuje hranici objektu [14].



Obr. 1.10: Ukázka detekce hranic.

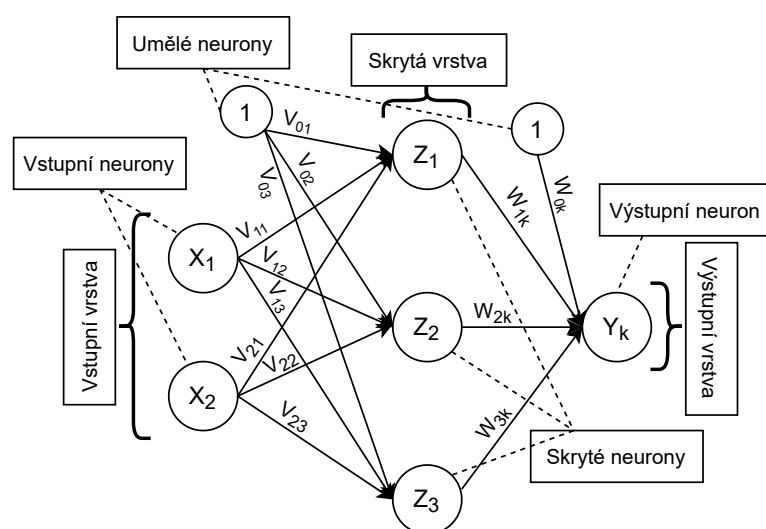
#### 1.4.4 Popis a klasifikace objektu

Po segmentaci je potřeba objekt co nejlépe popsat. Pro jeho popis může sloužit charakteristika objektu. Může to být například: jas, kontrast, rotace objektu apod. Je velmi důležité najít co nejvíce příznaků, které budou sloužit pro klasifikaci objektu.

Klasifikace objektu prochází podle nalezených příznaků. Jejím principem je rozdělení příznaků do soukromých tříd. Pomocí těchto tříd je neuronová síť schopna se učit (viz kapitola 1.5.5). Úkolem klasifikace obrázků je pořídit původní obrázek a přiřadit mu jeho třídu.

## 1.5 Neuronová síť

Teoreticky můžeme říct, že neuronová síť (zkráceno NS) je jako mozek člověka, kde jsou miliony neuronů, které si sdělují informaci pomocí elektronických impulzů. Neurony se spojují mezi sebou pomocí synapsí. O neuronech a synapsích bude řečeno níže v práci. Struktura neuronových sítí je skoro stejná jako v biologii. Pomocí této struktury mohou stroje analyzovat, pamatovat si a reprodukovat informaci z paměti. Na obrázku 1.11 je vidět struktura neuronové sítě [4].

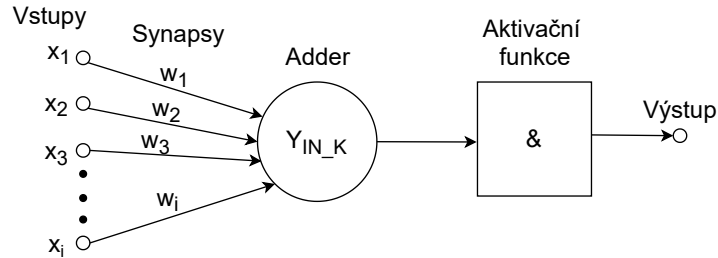


Obr. 1.11: Struktura neuronové sítě.

### 1.5.1 Neuron

Neuron je buňka, která přijímá informaci, provádí nad ní základní výpočty a rozpracovanou informaci posílá dál. Jsou tři typy neuronu: vstupní neuron (anglicky input neuron), skrytý neuron (anglicky hidden neuron) a výstupní neuron (anglicky output neuron). Všechny tři typy je možno vidět na obrázku 1.11. Když je využito velké množství neuronů, můžeme použít takzvané vrstvy. První vrstvu nazýváme vstupní, tato vrstva přijímá informaci. Druhá – skrytá vrstva, zpracovává informaci předanou od první vrstvy a posílá ji dál. V případě zpracování velkého množství informací může být využito několik skrytých vrstev. Čím více informací, tím více

vrstev, běžně se v praxi nevyužívá více než tři vrstvy. Třetí výstupní vrstva zobrazuje výsledek. Výstupní hodnota neuronu může být v rozsahu  $[0; 1]$ , nebo  $[-1; 1]$ . Dále můžeme na obrázku 1.11 pozorovat dva neurony, které se nazývají umělé neurony (anglicky artificial neurons). Ty mají vždy hodnotu 1 a slouží pro opravu NS [4].



Obr. 1.12: Struktura neuronu, převzato z [4].

### Jak funguje neuron?

Nejprve vstupní neuron přijímá informaci a posílá ji všem skrytým neuronům. Informace přechází od vstupního neuronu k výstupnímu pomocí tzv. synapse, o kterém bude zmíněno později. Na vstupu má neuron tzv. adder, v něm se informace zpracovává pomocí vztahu:

$$y_{in\_k} = \sum_{i=1}^n w_i x_i, \quad (1.16)$$

kde  $w_i$  je hodnota synapse a  $x_i$  vstupní hodnota. Potom vypočítaná hodnota musí projít přes funkci aktivace (viz kapitola 1.5.4), aby byla výstupní hodnota v námi potřebném rozsahu [4].

## 1.5.2 Fungování neuronové sítě

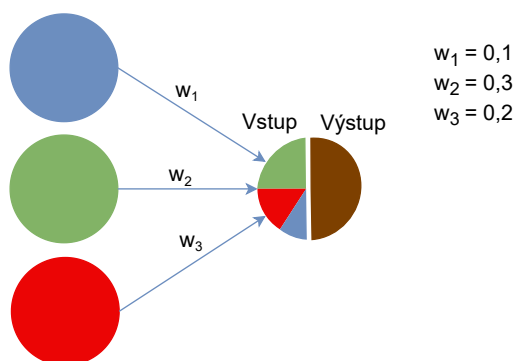
Každý vstupní neuron přijímá informaci a posílá ji všem skrytým neuronům (viz obrázek 1.11). Skryté neurony sčítají všechny vstupní signály vynásobené hodnotou váhy (vztah 1.17). Z obrázku 1.12 teď známe hodnotu v adderu. Abychom obdrželi výstupní hodnotu, musí hodnota z adderu projít přes funkci aktivace (vztah 1.19 nebo 1.20). Pak se výstupní hodnota skrytého neuronu posílá dál do výstupního neuronu. Výstupní neuron dělá to samé co skrytý (vztah 1.18), na výstupu obdržíme hodnotu  $y_k$ , což bude i výsledkem naší neuronové sítě.

$$z_{in\_2} = v_{02} + x_1 v_{12} + x_2 v_{22}, \quad (1.17)$$

$$y_{in\_k} = w_{0w} + z_1 w_{1v} + z_2 w_{2v} + z_3 w_{3v}. \quad (1.18)$$

### 1.5.3 Synapse

Jak bylo řečeno výše, synapse je spoj mezi dvěma neurony. Každá synapse má různou váhu (anglicky weights), váha je jediný a nejdůležitější parametr. Čím přesněji je vybraná váha, tím lepší je i výsledek. Váha se vybírá náhodně. Princip výběru váhy bude představen v kapitole 1.5.6. Pomocí synapse se vstupní informace po přechodu od jednoho neuronu k druhému mění. Máme například tři různé neurony a každý má různou informaci, s tím i různou váhu synapse. Vezmeme například barvy: červená, zelená a modrá, váhy neuronů v tomto případě budou kupříkladu: 0.1, 0.2, 0.3. Neuron s největší váhou bude dominantní vůči ostatním neuronům.

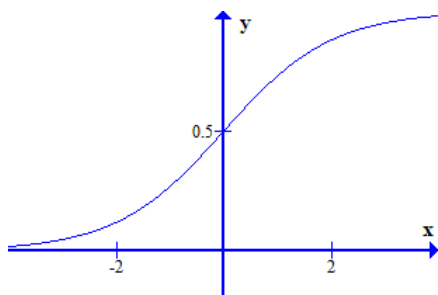


Obr. 1.13: Ukázka, jak pracuje synapse.

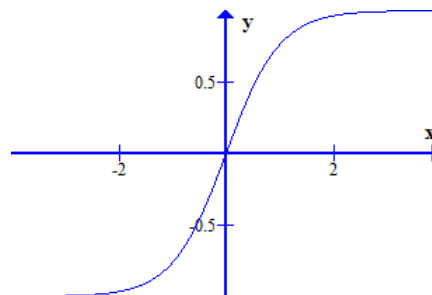
### 1.5.4 Funkce aktivace

Funkce aktivace pomáhá normovat vstupní hodnotu. Používáme ji, když je na vstupu hodnota, která je mimo náš potřebný rozsah. Je několik typů takových funkcí. První se nazývá *lineární funkce*. Tato funkce se používá pouze pro základní situace. Například když potřebujeme otestovat správnou funkčnost NS nebo když je třeba předat nějaký parametr beze změny. Lineární funkce neboli ReLU (Rectified Linear Units) se využívá v konvolučních neuronových sítích, více o tom v kapitole 1.6.

Druhá se nazývá *sigmoid*, kterou je možné vyjádřit vztahem 1.19. Je to nejpožívanější funkce v neuronových sítích, pomocí ní můžeme normalizovat hodnotu pro správný rozsah. Rozsah hodnot v této funkci je od 0 do 1. Pokud potřebujeme rozsah od  $-1$  do 1, můžeme využít funkci, která se nazývá *hyperbolický tangens*. Tuto funkci lze vyjádřit vztahem 1.20 [4].



Obr. 1.14: Graf Sigmoid.



Obr. 1.15: Graf Hyperbolicky tangens.

$$f(x) = \frac{1}{1 + e^{-x}} = y_k. \quad (1.19)$$

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} = y_k. \quad (1.20)$$

### 1.5.5 Učení neuronových sítí

Učení sítě je proces výběru správných parametrů – *weights*. Výběr parametrů se obvykle provádí pomocí adaptačních algoritmů, které můžeme rozdělit na dvě skupiny: učení s učitelem a učení bez učitele.

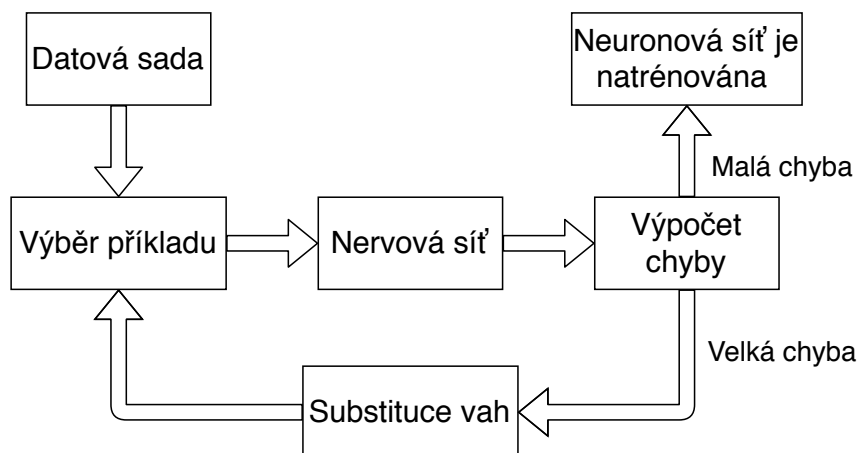
#### Učení s učitelem

Pro učení s učitelem potřebujeme dataset, ve kterém budou rozpracovány příklady, tzn. že budou připravené úkoly a správné odpovědi. NS probírá všechny správné odpovědi a nachází mezi nimi závislosti. Datová sada (anglicky dataset) slouží jako učitel a pomáhá trénovat NS. Po době, kdy se data z datasetu dostala na vstup a byla zpracována neuronovou sítí, se vypočítá chyba (viz kapitola 1.5.6). Při velké hodnotě chyby se mění parametr *weights* a stejný proces probíhá znovu, dokud hodnota chyby nebude v námi požadovaném rozsahu.

#### Učení bez učitele

Jak plyne z názvu, učení probíhá bez učitele, to znamená, že na vstupu už není rozpracován dataset, podle kterého by se mohla NS učit. V tomto případě učení probíhá pomocí shlukování objektů do různých tříd. Například na vstup pošleme 10 000 různých objektů s charakteristikou, budou to například studenti. Program je rozdělí na různé třídy podle známých parametrů získaných při popisu a klasifikace

objektu (viz kapitola 1.4.4). Může to být průměr, věk, ročník apod. Tento typ se nevyužívá často k učení neuronové sítě.



Obr. 1.16: Učení neuronové sítě s učitelem.

### 1.5.6 Výpočet chyby NS

Pomocí výpočtu chyby v neuronové síti se můžeme dozvědět funkčnost NS a zvolit vhodné váhy synapse. Chyba neuronové sítě se vypočítá pro každou epochu, tzn. pokaždé, když vstupní data projdou přes celou neuronovou síť. Počet epoch zadává uživatel, čím více epoch, tím lepší výsledek. Chyba se po každé epoše musí zmenšovat. Pro výpočet chyby se nejčastěji používá metoda *Mean Squared Error* (zkráceno *MSE*), tedy vztah:

$$f(x) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2, \quad (1.21)$$

kde  $y_i$  je očekávaný rezultat,  $\hat{y}_i$  je rezultátem neuronové sítě a  $N$  je počet příkladu. Je to velmi jednoduchá a často používaná, ale neúčinná metoda. Není užitečná, protože jestli vygenerované váhy budou špatné, tak druhá mocnina chyby NS zhorší výsledek ještě více. Tím samým může pokazit metriku NS. Stává se to velice často kvůli tomu, že některá data nejsou spolehlivá [6]. Celkem vhodným algoritmem pro výpočet chyby je tzv. algoritmus *backpropagation* (zkráceno BP), který bude představen níže.

### Backpropagation

Algoritmus BP funguje přes celou neuronovou síť pomocí učení s učitelem (viz kapitola 1.5.5). Algoritmus se zabývá výpočtem chyby a korekcí váhy synapse. Začátek

algoritmu začíná inicializací váhy synapse. Jsou to čísla v rozsahu  $[-0,5; 0,5]$  a vybírána jsou náhodně. Dále ověřujeme zda nejsou splněny podmínky pro ukončení algoritmu. Podmínkou skončení algoritmu může být počet iterací nebo předem definovaná minimální hodnota chyby. Hlavní myšlenkou tohoto algoritmu je dodržení metody *gradient descent*. Tato metoda pomáhá najít minimální hodnotu chyby. Dále pokračujeme v algoritmu pomocí kapitoly 1.5.2. Ve chvíli, kdy data dorazila do výstupního neuronu, se vypočítá chyba:

$$\sigma_k = (t_k - y_k) \cdot f'(y_{in\_k}), \quad (1.22)$$

kde  $t_k$  je hodnota nabídnutá učitelem. Dále pomocí vztahu 1.23 se vypočítá rozdíl váhy synapse  $w_{2k}$  z obrázku 1.11.

$$\Delta_{w_{2k}} = a\sigma_k z_2, \quad (1.23)$$

kde  $a$  je rychlost NS. Ještě k tomu se vypočítá tzv. bias, který je zodpovědný za korekci posunutí neuronové sítě (viz vztah 1.24) a hodnota chyby  $\sigma_k$  přechází do předchozí vrstvy.

$$\Delta_{w_{0k}} = a\sigma_k. \quad (1.24)$$

V tento okamžik se nacházíme na skryté vrstvě (*hidden layer*). Z obrázku 1.11 je zřejmé, že jeden neuron má několik synapsí, tím pádem je potřeba vynásobit každou váhu synapse s chybou  $\sigma_k$  a sečíst je (vztah 1.25). Dále se vypočítá chyba vrstvy podle vztahu 1.26, rozdíl váhy synapse 1.27 a bias 1.28 [13].

$$\sigma_{in_2} = \sum_{k=1}^n \sigma_k \cdot w_{2k}, \quad (1.25)$$

$$\sigma_2 = \sigma_{in_2} \cdot f'(z_{in_2}), \quad (1.26)$$

$$\Delta_{v_{22}} = a\sigma_2 x_2, \quad (1.27)$$

$$\Delta_{v_{02}} = a\sigma_2. \quad (1.28)$$

V posledním kroku vypočítáme výchozí hodnotu synapse. Pro novu hodnotu  $w_{2k}$  platí vztah 1.29 a pro  $v_{22}$  vztah 1.30.

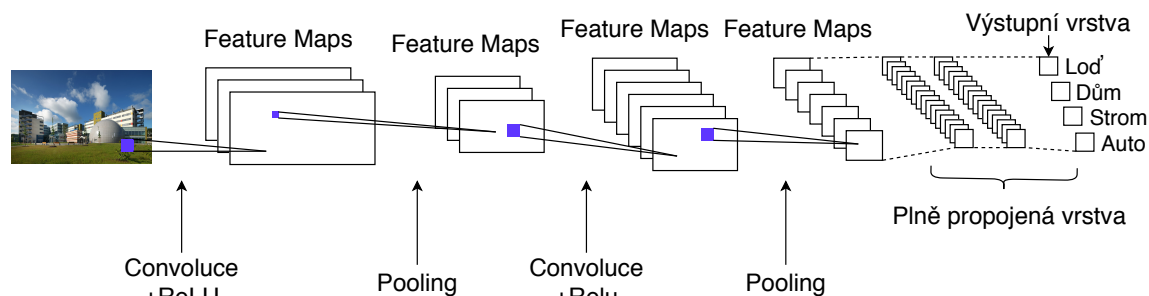
$$w'_{2k} = w_{2k} + \Delta w_{2k}, \quad (1.29)$$

$$v'_{22} = v_{22} + \Delta v_{22}. \quad (1.30)$$

## 1.6 Konvoluční neuronová síť

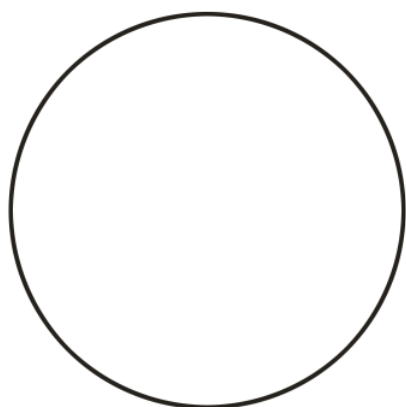
Konvoluční neuronová síť (zkráceno CNN) je část hloubkového učení (anglicky deep learning) neuronové sítě. Díky své metodě konvolucí a struktuře je vhodná pro rozpoznání obrazu, jeho detekci a klasifikaci. Struktura CNN se skládá z mnoha vrstev:

konvoluční vrstva, pooling vrstva a plně propojená vrstva (viz obrázek 1.17). Co se týká aktivačních funkcí, tak CNN využívá *lineární funkce*, protože je mnohem rychlejší na rozdíl od jiných typů funkcí (Sigmoid, Hyperbolický tangens). Učí se CNN pomocí metody backpropagation (viz kapitola 1.5.6).



Obr. 1.17: Struktura CNN, převzato z [16].

Počítač nerozpoznává obraz stejně jako člověk, vidí ho jako matici čísel (viz obrázek 1.18). Aby to bylo pochopitelnější, představme si, že máme barevný obrázek ve formátu JPG a jeho velikost je  $240 \times 240$ . Matice bude v tomto případě velikosti  $240 \times 240 \times 3$ , kde 3 je hodnota kanálů RGB. Navíc každý pixel získá hodnotu mezi 0 a 255, tato hodnota popisuje intenzitu pixelu v tomto bodě.



Jak vidí člověk.

0	0	0	0	0	0	0	0	0	0
0	0	0	32	32	32	32	0	0	0
0	0	32	32	0	0	32	32	0	0
0	32	32	0	0	0	0	32	32	0
0	32	0	0	0	0	0	0	32	0
0	32	0	0	0	0	0	0	32	0
0	32	0	0	0	0	0	0	32	0
0	32	0	0	0	0	0	0	32	0
0	32	0	0	0	0	0	0	32	0
0	32	0	0	0	0	0	0	32	0
0	32	32	0	0	0	0	32	32	0
0	0	32	32	0	0	32	32	0	0
0	0	0	32	32	32	32	0	0	0
0	0	0	0	0	0	0	0	0	0

Jak vidí počítač.

Obr. 1.18: Rozpoznávání obrazu člověkem a počítačem.

Hlavní funkcí CNN je proces konvoluce. Tento proces je doprovázen pomocí tzv. filtru. Tímto filtrem je matice s váhami, která hraje nejdůležitější roli v konvoluci a učení CNN. Metoda konvoluce pracuje pomocí násobení každého elementu matice obrazu na matici filtru (viz obrázek 1.19). Následující část této kapitoly popisuje existující modely CNN: *Faster R-CNN*, *SSD*, *YOLO*.

<table><tr><td>3<sub>2</sub></td><td>3<sub>2</sub></td><td>2<sub>2</sub></td><td>1</td><td>0</td></tr><tr><td>0<sub>2</sub></td><td>0<sub>2</sub></td><td>1<sub>0</sub></td><td>3</td><td>1</td></tr><tr><td>3<sub>0</sub></td><td>1<sub>1</sub></td><td>2<sub>2</sub></td><td>2</td><td>3</td></tr><tr><td>2</td><td>0</td><td>0</td><td>2</td><td>2</td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table> <table><tr><td>12</td><td>12</td><td>17</td></tr><tr><td>10</td><td>17</td><td>19</td></tr><tr><td>9</td><td>6</td><td>14</td></tr></table>	3 <sub>2</sub>	3 <sub>2</sub>	2 <sub>2</sub>	1	0	0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1	3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3	2	0	0	2	2	2	0	0	0	1	12	12	17	10	17	19	9	6	14	<table><tr><td>3</td><td>3<sub>2</sub></td><td>2<sub>2</sub></td><td>1<sub>2</sub></td><td>0</td></tr><tr><td>0</td><td>0<sub>2</sub></td><td>1<sub>2</sub></td><td>3<sub>0</sub></td><td>1</td></tr><tr><td>3</td><td>1<sub>0</sub></td><td>2<sub>1</sub></td><td>2<sub>2</sub></td><td>3</td></tr><tr><td>2</td><td>0</td><td>0</td><td>2</td><td>2</td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table> <table><tr><td>12</td><td>12</td><td>17</td></tr><tr><td>10</td><td>17</td><td>19</td></tr><tr><td>9</td><td>6</td><td>14</td></tr></table>	3	3 <sub>2</sub>	2 <sub>2</sub>	1 <sub>2</sub>	0	0	0 <sub>2</sub>	1 <sub>2</sub>	3 <sub>0</sub>	1	3	1 <sub>0</sub>	2 <sub>1</sub>	2 <sub>2</sub>	3	2	0	0	2	2	2	0	0	0	1	12	12	17	10	17	19	9	6	14	<table><tr><td>3</td><td>3</td><td>2<sub>2</sub></td><td>1<sub>2</sub></td><td>0<sub>2</sub></td></tr><tr><td>0</td><td>0</td><td>1<sub>2</sub></td><td>3<sub>2</sub></td><td>1<sub>0</sub></td></tr><tr><td>3</td><td>1</td><td>2<sub>0</sub></td><td>2<sub>1</sub></td><td>3<sub>2</sub></td></tr><tr><td>2</td><td>0</td><td>0</td><td>2</td><td>2</td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table> <table><tr><td>12</td><td>12</td><td>17</td></tr><tr><td>10</td><td>17</td><td>19</td></tr><tr><td>9</td><td>6</td><td>14</td></tr></table>	3	3	2 <sub>2</sub>	1 <sub>2</sub>	0 <sub>2</sub>	0	0	1 <sub>2</sub>	3 <sub>2</sub>	1 <sub>0</sub>	3	1	2 <sub>0</sub>	2 <sub>1</sub>	3 <sub>2</sub>	2	0	0	2	2	2	0	0	0	1	12	12	17	10	17	19	9	6	14
3 <sub>2</sub>	3 <sub>2</sub>	2 <sub>2</sub>	1	0																																																																																																				
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1																																																																																																				
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3																																																																																																				
2	0	0	2	2																																																																																																				
2	0	0	0	1																																																																																																				
12	12	17																																																																																																						
10	17	19																																																																																																						
9	6	14																																																																																																						
3	3 <sub>2</sub>	2 <sub>2</sub>	1 <sub>2</sub>	0																																																																																																				
0	0 <sub>2</sub>	1 <sub>2</sub>	3 <sub>0</sub>	1																																																																																																				
3	1 <sub>0</sub>	2 <sub>1</sub>	2 <sub>2</sub>	3																																																																																																				
2	0	0	2	2																																																																																																				
2	0	0	0	1																																																																																																				
12	12	17																																																																																																						
10	17	19																																																																																																						
9	6	14																																																																																																						
3	3	2 <sub>2</sub>	1 <sub>2</sub>	0 <sub>2</sub>																																																																																																				
0	0	1 <sub>2</sub>	3 <sub>2</sub>	1 <sub>0</sub>																																																																																																				
3	1	2 <sub>0</sub>	2 <sub>1</sub>	3 <sub>2</sub>																																																																																																				
2	0	0	2	2																																																																																																				
2	0	0	0	1																																																																																																				
12	12	17																																																																																																						
10	17	19																																																																																																						
9	6	14																																																																																																						
<table><tr><td>3</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0<sub>2</sub></td><td>0<sub>2</sub></td><td>1<sub>2</sub></td><td>3</td><td>1</td></tr><tr><td>3<sub>2</sub></td><td>1<sub>2</sub></td><td>2<sub>0</sub></td><td>2</td><td>3</td></tr><tr><td>2<sub>0</sub></td><td>0<sub>1</sub></td><td>0<sub>2</sub></td><td>2</td><td>2</td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table> <table><tr><td>12</td><td>12</td><td>17</td></tr><tr><td>10</td><td>17</td><td>19</td></tr><tr><td>9</td><td>6</td><td>14</td></tr></table>	3	3	2	1	0	0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>2</sub>	3	1	3 <sub>2</sub>	1 <sub>2</sub>	2 <sub>0</sub>	2	3	2 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	2	2	2	0	0	0	1	12	12	17	10	17	19	9	6	14	<table><tr><td>3</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0<sub>2</sub></td><td>1<sub>2</sub></td><td>3<sub>2</sub></td><td>1</td></tr><tr><td>3</td><td>1<sub>2</sub></td><td>2<sub>2</sub></td><td>2<sub>0</sub></td><td>3</td></tr><tr><td>2</td><td>0<sub>0</sub></td><td>0<sub>1</sub></td><td>2<sub>2</sub></td><td>2</td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table> <table><tr><td>12</td><td>12</td><td>17</td></tr><tr><td>10</td><td>17</td><td>19</td></tr><tr><td>9</td><td>6</td><td>14</td></tr></table>	3	3	2	1	0	0	0 <sub>2</sub>	1 <sub>2</sub>	3 <sub>2</sub>	1	3	1 <sub>2</sub>	2 <sub>2</sub>	2 <sub>0</sub>	3	2	0 <sub>0</sub>	0 <sub>1</sub>	2 <sub>2</sub>	2	2	0	0	0	1	12	12	17	10	17	19	9	6	14	<table><tr><td>3</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1<sub>2</sub></td><td>3<sub>2</sub></td><td>1<sub>2</sub></td></tr><tr><td>3</td><td>1</td><td>2<sub>2</sub></td><td>2<sub>2</sub></td><td>3<sub>0</sub></td></tr><tr><td>2</td><td>0</td><td>0<sub>0</sub></td><td>2<sub>1</sub></td><td>2<sub>2</sub></td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table> <table><tr><td>12</td><td>12</td><td>17</td></tr><tr><td>10</td><td>17</td><td>19</td></tr><tr><td>9</td><td>6</td><td>14</td></tr></table>	3	3	2	1	0	0	0	1 <sub>2</sub>	3 <sub>2</sub>	1 <sub>2</sub>	3	1	2 <sub>2</sub>	2 <sub>2</sub>	3 <sub>0</sub>	2	0	0 <sub>0</sub>	2 <sub>1</sub>	2 <sub>2</sub>	2	0	0	0	1	12	12	17	10	17	19	9	6	14
3	3	2	1	0																																																																																																				
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>2</sub>	3	1																																																																																																				
3 <sub>2</sub>	1 <sub>2</sub>	2 <sub>0</sub>	2	3																																																																																																				
2 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	2	2																																																																																																				
2	0	0	0	1																																																																																																				
12	12	17																																																																																																						
10	17	19																																																																																																						
9	6	14																																																																																																						
3	3	2	1	0																																																																																																				
0	0 <sub>2</sub>	1 <sub>2</sub>	3 <sub>2</sub>	1																																																																																																				
3	1 <sub>2</sub>	2 <sub>2</sub>	2 <sub>0</sub>	3																																																																																																				
2	0 <sub>0</sub>	0 <sub>1</sub>	2 <sub>2</sub>	2																																																																																																				
2	0	0	0	1																																																																																																				
12	12	17																																																																																																						
10	17	19																																																																																																						
9	6	14																																																																																																						
3	3	2	1	0																																																																																																				
0	0	1 <sub>2</sub>	3 <sub>2</sub>	1 <sub>2</sub>																																																																																																				
3	1	2 <sub>2</sub>	2 <sub>2</sub>	3 <sub>0</sub>																																																																																																				
2	0	0 <sub>0</sub>	2 <sub>1</sub>	2 <sub>2</sub>																																																																																																				
2	0	0	0	1																																																																																																				
12	12	17																																																																																																						
10	17	19																																																																																																						
9	6	14																																																																																																						
<table><tr><td>3</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>3</td><td>1</td></tr><tr><td>3<sub>2</sub></td><td>1<sub>2</sub></td><td>2<sub>2</sub></td><td>2</td><td>3</td></tr><tr><td>2<sub>2</sub></td><td>0<sub>2</sub></td><td>0<sub>0</sub></td><td>2</td><td>2</td></tr><tr><td>2<sub>0</sub></td><td>0<sub>1</sub></td><td>0<sub>2</sub></td><td>0</td><td>1</td></tr></table> <table><tr><td>12</td><td>12</td><td>17</td></tr><tr><td>10</td><td>17</td><td>19</td></tr><tr><td>9</td><td>6</td><td>14</td></tr></table>	3	3	2	1	0	0	0	1	3	1	3 <sub>2</sub>	1 <sub>2</sub>	2 <sub>2</sub>	2	3	2 <sub>2</sub>	0 <sub>2</sub>	0 <sub>0</sub>	2	2	2 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	0	1	12	12	17	10	17	19	9	6	14	<table><tr><td>3</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>3</td><td>1</td></tr><tr><td>3</td><td>1<sub>2</sub></td><td>2<sub>2</sub></td><td>2<sub>2</sub></td><td>3</td></tr><tr><td>2</td><td>0<sub>2</sub></td><td>0<sub>2</sub></td><td>2<sub>0</sub></td><td>2</td></tr><tr><td>2</td><td>0<sub>0</sub></td><td>0<sub>1</sub></td><td>0<sub>2</sub></td><td>1</td></tr></table> <table><tr><td>12</td><td>12</td><td>17</td></tr><tr><td>10</td><td>17</td><td>19</td></tr><tr><td>9</td><td>6</td><td>14</td></tr></table>	3	3	2	1	0	0	0	1	3	1	3	1 <sub>2</sub>	2 <sub>2</sub>	2 <sub>2</sub>	3	2	0 <sub>2</sub>	0 <sub>2</sub>	2 <sub>0</sub>	2	2	0 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	1	12	12	17	10	17	19	9	6	14	<table><tr><td>3</td><td>3</td><td>2</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>3</td><td>1</td></tr><tr><td>3</td><td>1</td><td>2<sub>2</sub></td><td>2<sub>2</sub></td><td>3<sub>2</sub></td></tr><tr><td>2</td><td>0</td><td>0<sub>2</sub></td><td>2<sub>2</sub></td><td>2<sub>0</sub></td></tr><tr><td>2</td><td>0</td><td>0<sub>0</sub></td><td>0<sub>1</sub></td><td>1<sub>2</sub></td></tr></table> <table><tr><td>12</td><td>12</td><td>17</td></tr><tr><td>10</td><td>17</td><td>19</td></tr><tr><td>9</td><td>6</td><td>14</td></tr></table>	3	3	2	1	0	0	0	1	3	1	3	1	2 <sub>2</sub>	2 <sub>2</sub>	3 <sub>2</sub>	2	0	0 <sub>2</sub>	2 <sub>2</sub>	2 <sub>0</sub>	2	0	0 <sub>0</sub>	0 <sub>1</sub>	1 <sub>2</sub>	12	12	17	10	17	19	9	6	14
3	3	2	1	0																																																																																																				
0	0	1	3	1																																																																																																				
3 <sub>2</sub>	1 <sub>2</sub>	2 <sub>2</sub>	2	3																																																																																																				
2 <sub>2</sub>	0 <sub>2</sub>	0 <sub>0</sub>	2	2																																																																																																				
2 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	0	1																																																																																																				
12	12	17																																																																																																						
10	17	19																																																																																																						
9	6	14																																																																																																						
3	3	2	1	0																																																																																																				
0	0	1	3	1																																																																																																				
3	1 <sub>2</sub>	2 <sub>2</sub>	2 <sub>2</sub>	3																																																																																																				
2	0 <sub>2</sub>	0 <sub>2</sub>	2 <sub>0</sub>	2																																																																																																				
2	0 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	1																																																																																																				
12	12	17																																																																																																						
10	17	19																																																																																																						
9	6	14																																																																																																						
3	3	2	1	0																																																																																																				
0	0	1	3	1																																																																																																				
3	1	2 <sub>2</sub>	2 <sub>2</sub>	3 <sub>2</sub>																																																																																																				
2	0	0 <sub>2</sub>	2 <sub>2</sub>	2 <sub>0</sub>																																																																																																				
2	0	0 <sub>0</sub>	0 <sub>1</sub>	1 <sub>2</sub>																																																																																																				
12	12	17																																																																																																						
10	17	19																																																																																																						
9	6	14																																																																																																						

Obr. 1.19: Konvoluce vrstvy, převzato z [5].

### 1.6.1 Faster R-CNN

*Faster R-CNN* je síť, která obsahuje dvě sítě. První síť se nazývá *RPN* – *Region Proposal Network*, tato síť se používá pro generování návrhů regionů. Zatím co druhá využívá tyto návrhy pro detekci objektu. *Faster R-CNN* se liší od předchozích verzí (*Fast R-CNN*, *R-CNN*) tím, že později používá selektivní vyhledávání (anglický *Selective Search*) pro generování návrhů ohraničujících rámečků (anglický *bounding boxes*) objektu. *RPN* pracuje na principu posuvného okna, které se pohybuje nad regiony a u každého regionu vrací počet bounding boxů a hodnotu pravděpodobnosti, že ohraničující rámeček obsahuje nám potřebný objekt. Bounding boxy s největší pravděpodobností, *RPN* sdílí s druhou sítí pro detekci a klasifikaci objektu [23].

### 1.6.2 SSD: Single Shot MultiBox Detector

*SSD* je síť, která používá pouze jednu konvoluční síť (anglický *feedforward convolutional network*), což pomáhá urychlit proces předpovědi umístění bounding boxů a jejich tříd, aniž by byla použita druhá fáze klasifikace. V této metodě je generováno několik tisíc předpovědí na výstupu neuronové sítě pro možné regiony umístění objektů různých tvarů v různých stupnicích, pak potlačením maxima (*Non-maximum Suppression*) síť dokáže vybrat několik nejpravděpodobnějších oblastí, ve kterých může nacházet požadovaný objekt detekci.

Vstupem tohoto modelu je obrázek o rozměrech  $300 \times 300$  pixelů, pak se na obraz aplikují konvoluční vrstvy ze standardního modelu VGG-16. Následně se do výstupní vrstvy přidávají speciální konvoluční vrstvy představující obraz v různých měřítcích. Prostorová dimenze se zmenšuje, dokud nebude rovna jedné. Tento typ CNN je velmi vhodný pro detekci objektů v reálném čase [19].

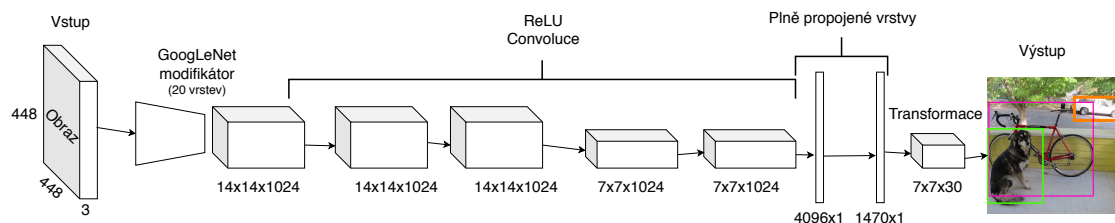
### 1.6.3 You Only Look Once — YOLO

*YOLO* je velice známá konvoluční neuronová síť. Slávu získala díky své architektuře. Hlavní rys architektury ve srovnání s ostatními sítěmi je v tom, že většina z nich používá CNN několikrát pro různé regiony, zatímco *YOLO* pouze jednou pro celý obraz. Síť rozděluje obrázek na mřížky a předpovídá ohraničující rámečky a pravděpodobnost toho, že v tomto rámečku je hledaný objekt.

Výhodou této sítě je to, že *YOLO* se dívá na celý obraz najednou (což plyne z názvu) a při detekci a rozpoznávání objektu bere v úvahu kontext obrazu.

### 1.6.4 Popis jak pracuje YOLO

Vstupem je obrázek s rozměrem  $448 \times 448 \times 3$ . Následně tento obraz prochází přes modifikovanou architekturu *GoogLeNet*, která obsahuje 20 konvolučních vrstev. Místo *GoogLeNet* je možné využít *VGG* nebo jinou modifikovanou architekturu. Na výstupu obdržíme takzvané hlavní mapy (Feature Maps) s rozměrem  $14 \times 14 \times 1024$ . Pak se používají konvoluční vrstvy a aktivační funkce ReLU. Čím více konvolučních vrstev obsahuje model detektoru, tím lepší je výsledek. Dále je využita plně propojená vrstva (anglicky Fully Connected Layers). Tahle vrstva spojuje každý jeden neuron v jedné vrstvě s každým neuronem v jiné vrstvě. Potom výsledná matice prochází přes plně propojenou vrstvu pro klasifikaci obrázků. Výstupem plně propojené vrstvy je vektor s rozměrem  $1470 \times 1$ . Po transformaci se objeví tensor s velikostí  $7 \times 7 \times 30$ . Na konci probíhá proces detekce, po kterém je vidět výsledek [21].



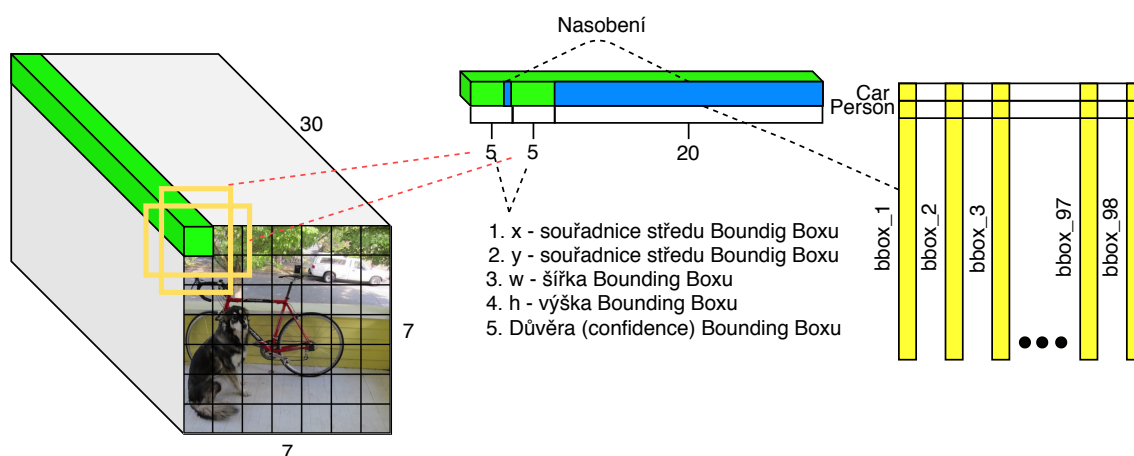
Obr. 1.20: Ukázka, jak pracuje YOLO, převzato z [21].

## Proces detekce

Teoreticky všechna detekce je kódována v tenzoru  $7 \times 7 \times 30$ . Na původní obrázek se použije mřížka  $7 \times 7$ , což odpovídá velikosti tensoru. Každá buňka v mřížce obsahuje vektor velikosti 30, ve kterém prvních 5 čísel značí šířku, výšku, centry hran ohraničujícího rámečku a jistotu, že Bounding Box označil objekt správně (viz obrázek 1.21). Dalších 5 čísel obsahuje stejné hodnoty pro druhý bounding box (*YOLO* vždy vytvoří dva ohraničující rámečky pro jednu buňku). Poslední hodnoty vektoru značí počet tříd využitých při trénování, každá z těchto hodnot obsahuje pravděpodobnost toho, že centr objektu se nachází ve zvolené buňce. Ale tyto hodnoty však nejsou vázány na žádné ze dvou existujících ohraničujících rámečků. Ke zjištění toho, zda v rámečku existuje nějaký objekt, je potřeba znásobit hodnotu jistoty ohraničujícího rámečku se všemi jistotami tříd.

Každá tenzorová buňka musí obsahovat 2 ohraničující rámečky, z toho dostaneme 98 rámečků, které obsahují hodnoty pravděpodobnosti umístění objektu pro každou třídu. Pak existuje proces vynulování hodnot, které jsou menší než 0,2, poté dochází ke třídění rámečků. Bounding box s největší hodnotou pravděpodobnosti bude na prvním místě. V tuto chvíli jedna třída bude obsahovat několik ohraničujících rámečků, proto je potřeba použít algoritmus *Non-maximum Suppression*. Tento algoritmus eliminuje duplicitní detekce.

Výše popsany proces se provádí pro každou třídu, na konci *YOLO* probere všechny ohraničující rámečky a vybere jenom ty které obsahují hodnotu pravděpodobnosti větší než 0. Tyto rámečky namaluje na obrazu, čímž označí objekt detekcí [21].



Obr. 1.21: Ukázka, jak vypadá a co v sobě obsahuje tenzor  $7 \times 7 \times 30$ .

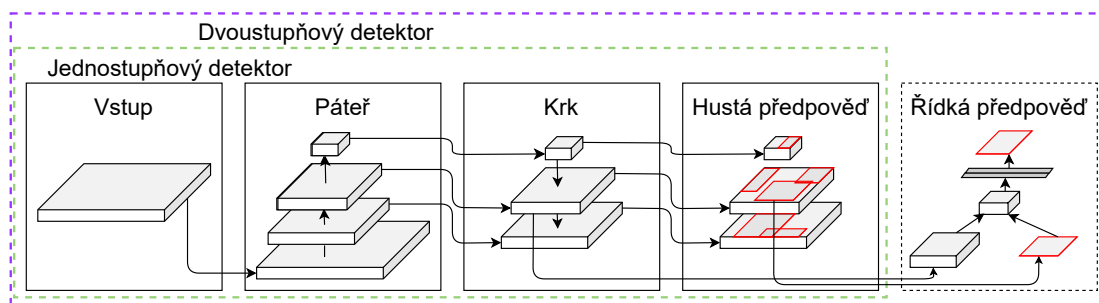
V současné době existuje několik verzí *YOLO*: *YOLOv1*, *YOLOv2*, *YOLOv3*, *YOLOv4* a existují i menší modely, jako jsou například *YOLOv3-tiny*, *YOLOv4-tiny* atd. Níže bude popsána struktura jednoho z modelů *YOLO*, který během trénování dosáhl nejlepších hodnot na datové sadě Microsoft COCO.

## YOLOv4

*YOLOv4* je nyní jedním z nejpresnějších modelů detektorů, její přesnost dosahuje až 43,5 % AP (Average precision) pro datovou sadu Microsoft COCO na grafické kartě Tesla V100 s rychlostí výstupu přibližně 65 FPS. Používání sítě *YOLO* nevyžaduje vysoce efektivní grafické karty a je relativně levným řešením.

Model detektoru *YOLOv4* se skládá z několika částí:

- vstup: obrázek, opravy, pyramida obrázku,
- páteř: VGG16, ResNet-50, SpineNet, EfficientNet-B0/B7, CSPResNeXt50, CSP-Darknet53,
- krk:
  - bloky agregace cest: FPN, PAN, NAS-FPN, plně připojené FPN, BiFPN, ASFF, SFAM,
  - další bloky: SPP, ASPP, RFB, SAM,
- hlava:
  - hustá předpověď (jednostupňová):
    - \* RPN, SDD, YOLO, RetinaNet (na základě kotvy),
    - \* CornerNet, CenterNet, MatrixNet, FCOS (bez kotvy),
  - řídká předpověď (dvoustupňová):
    - \* Faster R-CNN, R-FCN, Mask R-CNN (na základě kotvy),
    - \* RepPoints (bez kotvy) [22].



Obr. 1.22: Struktura sítě YOLOv4, převzato z [22].

## 2 Návrh a realizace řešení práce

V této kapitole bude zaznamenána praktická část bakalářské práce. Zde je možné zjistit, jaký software a hardware byly použity k vyřešení problematiky této práce, dále také to, jaký byl vybrán model detektoru a framework pro implementaci tohoto modelu. Také zde jsou informace o datových sadách (příklady vhodných, ale i naopak nevhodných datových sad), návod, jak připravit vlastní datovou sadu nebo jak používat již existující pro vlastní potřeby.

### 2.1 Příprava softwaru

Nejprve pro řešení úkolu bakalářské práce je nutné správně připravit software. V první řadě nainstalovat vývojové prostředí Visual Studio (nejlépe od verze 2017) a implementovat knihovnu *OpenCV* do vývojového prostředí.

Po implementaci knihovny je potřeba připravit software pro detekci a trénování modelu detektoru. Před začátkem trénování modelu je zapotřebí vybrat nástroj. To může být například: *SSD – Single Shot Detector*, *R-CNN – Region-based Convolutional Neural Network* nebo *YOLO – You Only Look Once*. Informace o těchto detektorech najdete v kapitolách 1.6.1, 1.6.2, 1.6.3. Z těchto třech jsem vybral model, který v současnosti nejrychlejší a má největší hodnotu přesnosti, jde o *YOLOv4* (viz kapitola 1.6.4). S tímto detektorem budou spojena všechna následující nastavení.

Pro implementaci tohoto nástroje jsem využil framework Darknet. Tento framework je open source, napsaný v jazyce *C* a *CUDA*. Výhodou tohoto frameworku je možnost provádět výpočty na CPU a GPU, což je velmi důležité a urychluje to práci. Aby bylo možné provádět výpočty na GPU, je třeba dát pozor na to, aby byla na počítači nainstalovaná grafická karta firmy NVIDIA. Při své práci jsem použil kartu NVIDIA GTX 1050. Pro využití tohoto frameworku na grafické kartě je navíc potřeba nainstalovat *NVIDIA CUDA Toolkit* a *NVIDIA CUDA Deep Neural Network* (cuDNN). Instalaci jsem provedl podle návodu dostupného na webových stránkách společnosti NVIDIA [17] [18].

Framework je možné stáhnout z webu <https://github.com/AlexeyAB/darknet>. Výše uvedený odkaz je vhodný pro ty, kdo využívá Windows. Pro Linux je lepší použít následující odkaz: <https://github.com/pjreddie/darknet>. Stáhnout obě verze je možné prostřednictvím nástroje GIT.

#### Windows instalace

Po stažení frameworku je nejprve třeba zkompilevat soubor `darknet.sln`, který se nachází ve složce `/build/darknet`. Pro kompilaci do vývojového prostředí je nutné

implementovat knihovny *OpenCV* a *cuDNN*. V průběhu kompilace jsem narazil na chybu. Chyba byla způsobena tím, že jsem použil jinou verzi *CUDA* (na rozdíl od té, která byla v projektu). Řešením bylo opravit soubor `darknet.vcxproj`. V tomto souboru jsem vyhledal místa, kde byla označena verze *CUDA*, následně jsem ji změnil na svou. Po úspěšné kompilaci se ve složce `/darknet/build/darknet/x64` objeví soubor `darknet.exe`, tento soubor je potřebný pro trénování, testování a provádění různých výpočtů konvoluční sítě.

## 2.2 Datová sada

V této části bakalářské práce se popisuje, z čeho se skládá datová sada (anglický dataset) pro trénování modelu detektoru. Jsou zde i ukázky vhodných datových sad, které je možné nalézt na internetu. Navíc je popsáno, jak vytvořit vlastní datovou sadu a jaký program k tomu využít.

### 2.2.1 Co v sobě obsahuje dataset

Datová sada je jedním z nejdůležitějších bodů, kterému musíme dát maximální pozornost. Dataset musí obsahovat sadu snímků s anotací nám potřebných objektů pro učení CNN. Ke každému snímku je potřeba vytvořit soubor `.txt`, který bude obsahovat informace s anotacemi pro každou třídu a objekt, který na snímku existuje. Každý řádek tohoto souboru má tvar:

`<číslo_třídy> <x_center> <y_center> <šířka> <výška>`

`<číslo_třídy>` – v souboru s příponou `.names` (více o využití tohoto souboru bude popsáno v kapitole 2.3.1) jsou zapsaná jména tříd, které budou detekovány v průběhu trénování. Každé jméno se zapisuje z nového řádku a hodnota `<číslo_třídy>` odpovídá číslu řádku (numerace řádku se začíná od nuly).

`<x_center>`, `<y_center>` – centry hran ohraničujícího rámečku.

`<šířka>`, `<výška>` – šířka a výška ohraničujícího rámečku.

Hodnoty `<x_center>`, `<y_center>` a `<šířka>`, `<výška>` jsou v rozsahu od 0 do 1 a vypočítají se pomocí vztahů:

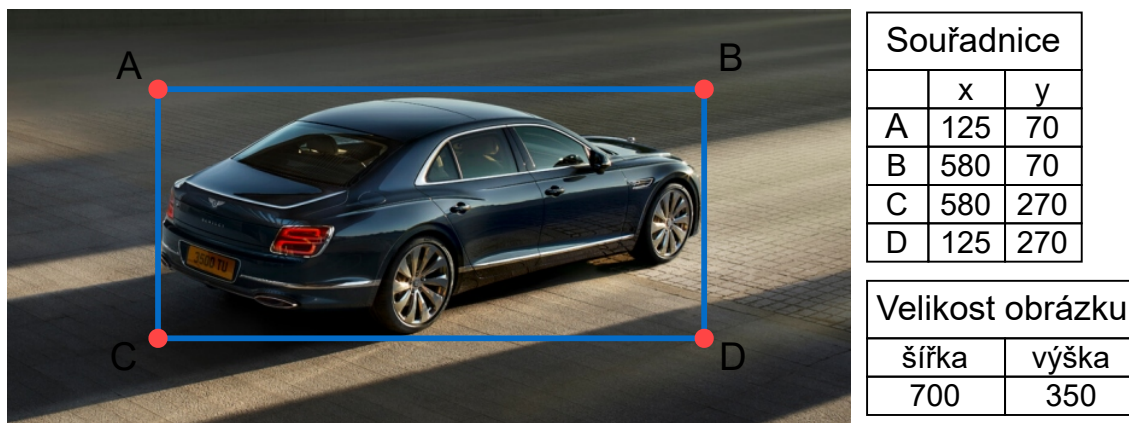
$$\langle x\_center \rangle = \text{abs\_x} / \text{šířka\_snímku}, \quad (2.1)$$

$$\langle y\_center \rangle = \text{abs\_y} / \text{výška\_snímku}, \quad (2.2)$$

$$\langle \text{šířka} \rangle = \text{abs\_šířka} / \text{šířka\_snímku}, \quad (2.3)$$

$$\langle \text{výška} \rangle = \text{abs\_výška} / \text{výška\_snímku}. \quad (2.4)$$

kde  $abs\_x$ ,  $abs\_y$ ,  $abs\_šířka$ ,  $abs\_výška$  ukazují absolutní hodnotu výšky, šířky a středů hranic objektu, který bude detekován.  $Šířka\_snímku$ ,  $výška\_snímku$  ukazují celkovou velikost obrazu v pixelech. Pro lepší pochopení viz obrázek 2.1 a pod ním příklad výpočtu.



Obr. 2.1: Příklad anotovaného snímku.

$$\begin{aligned}
 abs\_šířka &= B_x - A_x = 580 - 125 = 455 \\
 abs\_výška &= D_y - B_y = 270 - 70 = 200 \\
 abs\_x &= abs\_šířka/2 + A_x = 455/2 + 125 = 325,5 \\
 abs\_y &= abs\_výška/2 + A_y = 200/2 + 70 = 170 \\
 < šířka > &= abs\_šířka/šířka\_snímku = 0,65 \\
 < výška > &= abs\_výška/výška\_snímku = 0,571 \\
 < x\_center > &= abs\_x/šířka\_snímku = 0,465 \\
 < y\_center > &= abs\_y/výška\_snímku = 0,486
 \end{aligned}$$

Po výpočtu do souboru `.txt` je potřeba zapsat řádek: 0 0.465 0.486 0.65 0.571, kde 0 je číslo třídy.

## 2.2.2 Příklady datových sad

V současnosti je na prostorech internetu možné nalézt spoustu připravených datových sad pro různé třídy. Mezi vhodné datasety, můžeme zařadit například *Berkley Deep Drive (BDD)* nebo velice známou datovou sadu *COCO*, více o ní je možné nalézt v kapitole níže. Ale máme i nevhodné datasety jako například *Stanford AI Lab* (nevhodné, pouze pokud úkol řešíme pomocí *YOLO*). Snímky z této datové sady můžeme vidět na obrázku 2.2. Tento dataset obsahuje obyčejné snímky aut z blízkosti. V průběhu trénování *YOLO* sleduje celý snímek a bere v úvahu i kontext snímku. Proto je lepší použít snímky, ve kterých je objekt umístěn za různých okolností jako na obrázku 2.3.



Obr. 2.2: Příklad nevhodné datové sady pro síť YOLO, převzato z datové sady Stanford AI Lab.



Obr. 2.3: Příklad vhodné datové sady pro síť YOLO, převzato z datové sady Berkley Deep Drive.

Pro řešení úkolu bakalářské práce jsem vytvořil vlastní datovou sadu pomocí programu *YOLO\_mark*. Více o tomto programu je možné se dočíst v kapitole 2.2.4. Dataset byl vytvořen pro 7 tříd: auta, autobusy, dodávky, kamiony, motorky, jízdní kola a lidí. Celkem bylo anotováno více než 1 000 snímků a k tomu bylo provedeno více než 13 000 anotací. Snímky pro trénovací sadu byly pořízeny z dopravních online kamer.

Kvůli tomu, že datové sady z internetu neodpovídají formátu *YOLO*, jsem vytvořil kapitolu která popisuje, jak přečíst datovou sadu z internetu na příkladu datasetu *COCO* (viz kapitola 2.2.3).

### 2.2.3 Jak přečíst datovou sadu COCO

*COCO* je velice známá a často používaná datová sada pro trénování různých modelů detektoru. Výsledky trénování z této datové sady je možné najít na oficiálních stránkách *COCO*.

Na trénovací sadě *COCO* konvoluční neuronová síť *YOLOv4* ukazuje hodnotu mAP (anglicky mean Average Precision) až 62,8 % s velikosti vstupního obrazu  $416 \times 416$ . Což je nyní nejlepší hodnota v porovnání všech existujících modelů detektorů. Konvoluční architektura *YOLOv4-tiny* kvůli tomu, že je menší oproti *YOLOv4*, za stejných podmínek trénování dosahuje hodnoty mAP 40,2 %.

Pro přečtení datové sady *COCO* je potřeba využít OS Linux. V případě, že pracujete na operačním systému Windows, doporučuji využít prostředí *GoogleColab*. Nejprve je potřeba stáhnout *COCO* dataset, který je k dispozici na oficiálním webu *COCO*. Konkrétně soubory pro trénování, testování a anotační soubory. Pro zpracování obrazu se používá knihovna *GluonCV*, proto je potřeba ji nainstalovat. Spolu s tím se instaluje program *MXNet* určený k trénování a nasazení hlubokých neuronových sítí. Dalším krokem je příprava datové sady pomocí knihovny *GluonCV* a skriptu převzatého z oficiální stránky *GluonCV* [20]. Skript je ukázán níže.

Výpis 2.1: Ukázka skriptu pro přípravu datové sady.

```
from gluoncv import data, utils
from matplotlib import pyplot as plt

train_dataset=data.COCODetection(splits=['instances_train2017'])
val_dataset=data.COCODetection(splits=['instances_val2017'])
print('Počet_traininkových_obrázků:', len(train_dataset))
print('Počet_ověřovacích_obrázků:', len(val_dataset))
```

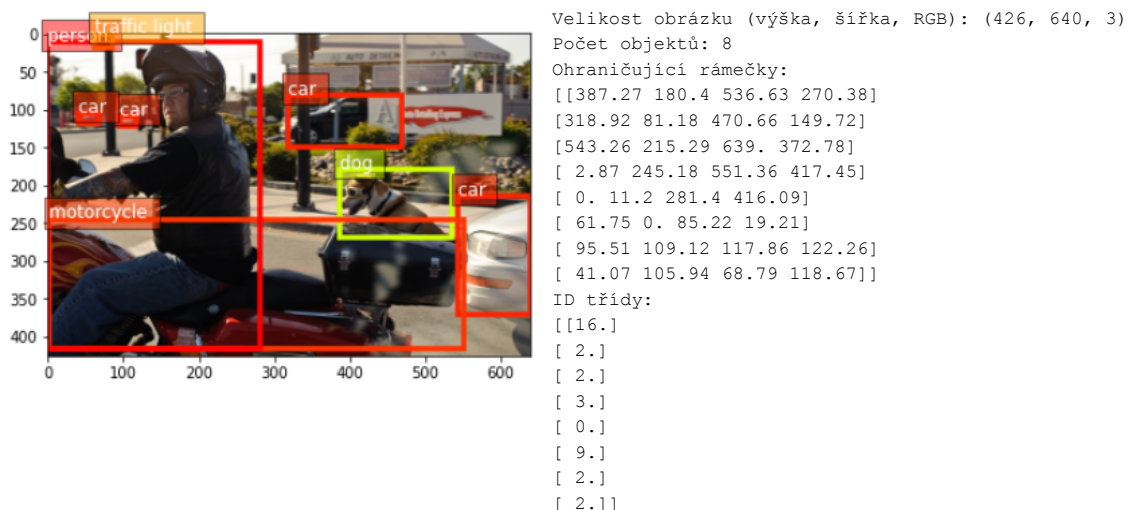
Pomocí následujícího skriptu převzatého ze stejné webové stránky, je poté možné zobrazit rozpracovaný snímek.

Výpis 2.2: Ukázka skriptu pro zobrazení rozpracovaného snímku.

```
train_image, train_label = train_dataset[1]
bounding_boxes = train_label[:, :4]
class_ids = train_label[:, 4:5]
print('Velikost_obrázku(výška,šířka,RGB):', train_image.shape)
print('Počet_objektů:', bounding_boxes.shape[0])
print('Ohraničující_rámečky:\n', bounding_boxes)
print('ID_třídy:\n', class_ids)

utils.viz.plot_bbox(train_image.asnumpy(), bounding_boxes,
                    scores=None, labels=class_ids,
                    class_names=train_dataset.classes)
plt.show()
```

Na výstupu je následně vidět zpracovaný obraz a souřadnice ohraničujících rámečků. Z obrázku 2.4 je patrné, že souřadnice ohraničujícího rámečku ne odpovídají *YOLO* formátu. K překladi do formátu, který potřebujeme, je potřeba provést výpočty z kapitoly 2.2.1.



Obr. 2.4: Ukázka zpracovaného obrazu v GoogleColab.

## 2.2.4 Yolo\_mark

Tato aplikace slouží k anotaci objektů na snímcích, potřebných pro učení neuronové sítě. Aplikaci lze stáhnout pomocí nástroje GIT z webu uvedeného v popisu instalace aplikace níže. Instalace musí být provedena pomocí následujících kroků.

1. Stáhnout aplikaci z webu [https://github.com/AlexeyAB/Yolo\\_mark](https://github.com/AlexeyAB/Yolo_mark).
2. Po stažení je potřeba otevřít soubor `yolo_mark.sln` prostřednictvím vývojového prostředí Visual Studio IDE a implementovat knihovnu *OpenCV* do otevřeného projektu.
3. Ve VisualStudio přepnout konfigurace řešení do modu **Release**.
4. Zkompilovat projekt.

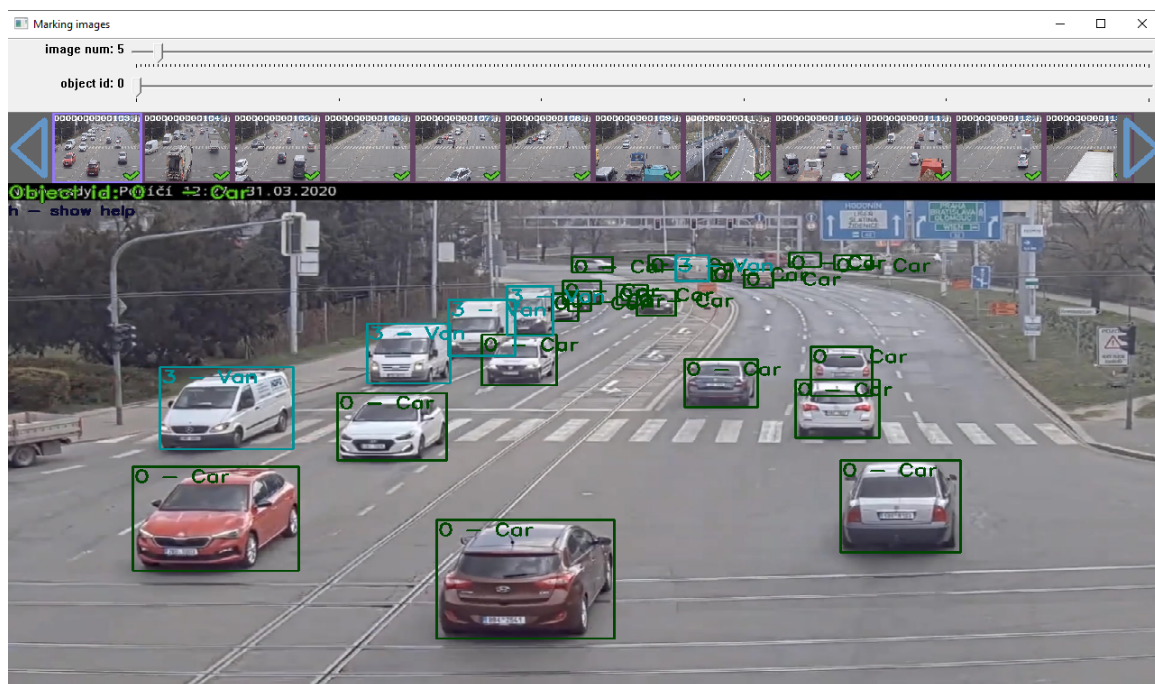
Po instalaci musí být obrázky, které mají být anotovány, zkopírovány do složky `Yolo_mark\x64\Release\data\img`. V předchozí složce `data` je potřeba opravit specifické soubory: `obj.data` a `obj.names`. Více o těchto souborech je zmíněno v kapitole 2.3.1. Následně ve složce **Release** můžeme spustit skript `yolo_mark.cmd` a začít anotovat obrázky. Jak aplikace vypadá můžeme vidět na obrázku níže.

Tento program vytvoří pro každý snímek `.txt` soubor (více o souboru v kapitole 2.2.1). Program navíc vytvoří soubor `train.txt`, ve kterém bude uveden seznam všech anotovaných snímků a cesta k nim. Zmíněný soubor je pak používán během trénování.

## 2.2.5 Automatická anotace programem CVAT

CVAT je bezplatný interaktivní nástroj k anotaci obrázků pro počítačové vidění, podporuje možnosti automatické a ruční anotace. Tento anotační nástroj lze pou-

žít na webové stránce [cvat.org](https://cvat.org) nebo nainstalovat do vlastního počítače. V případě použití webové verze stačí pouze vytvořit účet na výše uvedené stránce a pak je možné aplikaci používat. Pokud chcete získat lepší výsledky než výsledky z webové verze, můžete si program nainstalovat do svého počítače. Pro lepší výsledek budete potřebovat efektivnější grafickou kartu a implementovat vhodnější model detektoru. Webová verze tohoto programu podporuje anotace pomocí třech modelů detektorů: *Mask R-CNN*, *Faster R-CNN* a *YOLOv3*.



Obr. 2.5: Ukázka aplikaci yolo mark.

## 2.3 Trénování

Tato část bakalářské práce popisuje postup přípravy potřebných souborů a spuštění trénování dvou modelů detektorů *YOLOv4-tiny* a *YOLOv4*.

Na začátku této kapitoly je popsán proces přípravy souboru pro trénování modelu detektoru *YOLOv4-tiny* a potom spuštění trénování tohoto modelu. Další část této kapitoly také popisuje přípravu souboru pro trénování a zahájení trénování, ale pro model *YOLOv4*. Na konci této kapitoly budou uvedeny výsledky učení obou modelů.

### 2.3.1 Příprava souboru pro trénování modelu detektoru YOLOv4-tiny

Nejprve jsem pro detekci zvolil síť *YOLOv4-tiny*. Výhodou této sítě je, že je menší, tj. obsahuje méně konvolučních vrstev než *YOLOv4*, a proto je rychlejší a k jejímu trénování nemusím používat velmi drahé grafické karty. Ale přesnost tohoto modelu je samozřejmě mnohem nižší. K implementaci tohoto modelu bylo nutné stáhnout soubor `yolov4-tiny-custom.cfg`, který je dostupný na oficiálním webu frameworku *Darknet*. Tento soubor zahrnuje veškeré informace o modelu, jako například: počet výstupních vrstev, počet tříd, počet filtrů využitých pro tyto třídy, velikost vstupního obrazu atd. V tomto souboru bylo potřeba provést několik úprav.

Na začátku jsem upravil počet snímků, které budou zpracovány na GPU během jedné iterace. Za tuto informaci odpovídá řádek číslo 6 a 7, kde musím nastavit hodnoty `batch` a `subdivisions`. Hodnota `batch` odpovídá za celkový počet snímků, které budou zpracovány během 1 iterace. Hodnota `subdivisions` rozdělí `batch` na tzv. *mini-batches* a pošle tuto hodnotu (tj. snímky) na zpracování do GPU. Po zpracování prvních *mini-batches* se do GPU přidají další. Tento proces se opakuje, dokud nebude zpracován celkový počet snímků uvedený v `batch`. Pokud během trénování dojde k chybě "CUDA Error: out of memory", znamená to, že grafická karta nemá dostatečnou kapacitu. Řešením může být zvětšení hodnoty `subdivisions` nebo zmenšení velikosti vstupního obrazu. Doporučená hodnota pro `batch` je 64 a pro `subdivisions` 16. Tyto hodnoty jsem zvolil pro trénování.

Další, stejně důležité hodnoty musí být zapsány do řádků 212 a 220. Jsou to `filters` a `classes`. Počet filtrů je přímo závislý na počtu tříd. Vzorec pro výpočet filtrů vypadá následovně:  $filters = (classes + 5) \cdot 3$ . Můj model umí detekovat 7 tříd, proto jsem počet filtrů nastavil na 36. Stejně hodnoty je potřeba uvést na řádcích 263, 269. Pro kvalitnější detekci pak můžeme opravit hodnoty `width` a `height` (řádky 8 a 9), ty jsou zodpovědné za šířku a výšku vstupního obrazu. Tyto hodnoty musí být dělitelné 32. Pro přesnější detekci raději tyto hodnoty zvětšit, ale při tom bude se využívat více grafické paměti a je možná natrápit na výše uvedenou chybu "CUDA Error: out of memory". Pro svůj model jsem nastavil tyto hodnoty na 416.

Následující není povinné hodnoty jsou: `max_batches`, `anchors`:

- `max_batches` – označuje maximální počet iterací během trénování. Autor doporučuje nastavit tuto hodnotu podle vzorce  $max\_batches = classes \cdot 2000$ ,
- `anchors` – předpovídá přibližnou velikost hledaného případně hledaných objektů. Vypočítat je možná pomocí příkazu:

```
darknet.exe detector calc_anchors data/obj.data -  
num_of_clusters 9 -width 416 -height 416
```

Po postupu uvedeném výše je soubor `.cfg` považován za připravený a musí být umístěn vedle souboru `darknet.exe` podél cesty `/darknet/build/darknet/x64`.

Následující soubor je potřeba vytvořit s příponou `.names`. V tomto souboru musí být zapsána jména všech tříd a každé jméno by mělo být zapsáno do nového řádku. Další soubor, který je potřeba vytvořit, musí být s příponou `.data`. Struktura tohoto souboru je následující:

```
classes= 7
train   = data/train.txt
valid   = data/valid.txt
names   = data/obj.names
backup  = backup/
```

V prvním řádku je uveden počet tříd. Druhý, třetí a následující řádek ukazuje umístění textového souboru pro trénování (`train.txt`), testování (`valid.txt`) a cestu k souboru `obj.names`, který obsahuje seznam tříd. Poslední řádek ukazuje, kam se budou ukládat váhy vypočtené během trénování.

Následující, povinný textový soubor `train.txt` obsahuje seznam všech snímků, které budou zpracovány během trénování a cestu k nim, každá nová cesta by měla být zapsaná do nového řádku. Pokud testování nebude probíhat na stejných snímcích jako pro trénování je potřeba vytvořit soubor `valid.txt` ve kterém budou popsány cesty ke každému snímku pro testování (každá cesta se zapisuje z nového řádku, stejně jako a i v souboru `train.txt`). Soubory `obj.names`, `obj.data` a `train.txt` je potřeba umístit ve složce podél cesty `/darknet/build/darknet/x64/data`.

### 2.3.2 Trénování sítě YOLOv4-tiny

Trénování je možná spustit s využitím již existujících, tzv. vah. Tento způsob je efektivnější a během tréninku ušetří trochu času. Soubor s váhy možná stáhnout na oficiálních stránkách frameworku *Darknet* a on se jmenuje `yolov4-tiny.weights`. Tento soubor s váhy je potřeba umístit vedle souboru `darknet.exe`. Pomocí příkazu uvedeného níže pak získáme soubor s předem trénovanými váhami. Soubor musí být pojmenován `yolov4-tiny.conv.29`.

Výpis 2.3: Příkaz pro získání souboru s předem trénovanými váhy.

```
darknet.exe partial cfg/yolov4-tiny-custom.cfg yolov4-tiny.
weights yolov4-tiny.conv.29 29
```

Nakonec je pomocí příkazu níže možné spustit trénování.

Výpis 2.4: Příkaz pro spuštění trénování.

```
darknet.exe detector train data/obj.data yolov4-tiny-custom.cfg
yolov4-tiny.conv.29
```

## YOLOv4

*YOLOv4* se od *YOLOv4-tiny* liší tím, že je mnohem větší a lepší pro detekci menších objektů (více o modelu *YOLOv4* v kapitole 1.6.3). Pro implementaci trénování je potřeba stáhnout jiný soubor `.cfg`, který se jmenuje `yolov4-custom.cfg`. Je možné ho stáhnout ze stejné stránky jako soubor `yolov4-tiny-custom.cfg`.

Proces přípravy souboru pro *YOLOv4* je skoro stejný jako pro *YOLOv4-tiny*. Rozdílem je to, že model detektoru obsahuje tři výstupní vrstvy, zatímco *YOLOv4-tiny* pouze dvě. Kvůli tomu je v souboru `.cfg` potřeba tři krát uvést počet tříd (řádky: 970, 1058, 1146) a počet filtrů (řádky: 963, 1058, 1139).

Dalším rozdílem v nastavení je instalace souboru s předem trénovanými váhy. Tento soubor se jmenuje `yolov4.conv.137` a umísťuje se vedle souboru `darknet.exe`. Pro spuštění trénování je možná využít příkaz s výpisu 2.4, ale místo souboru `yolov4-tiny-custom.cfg` třeba zapsat `yolov4-custom.cfg` a následující položku `yolov4-tiny.conv.29` změnit na `yolov4.conv.137`.

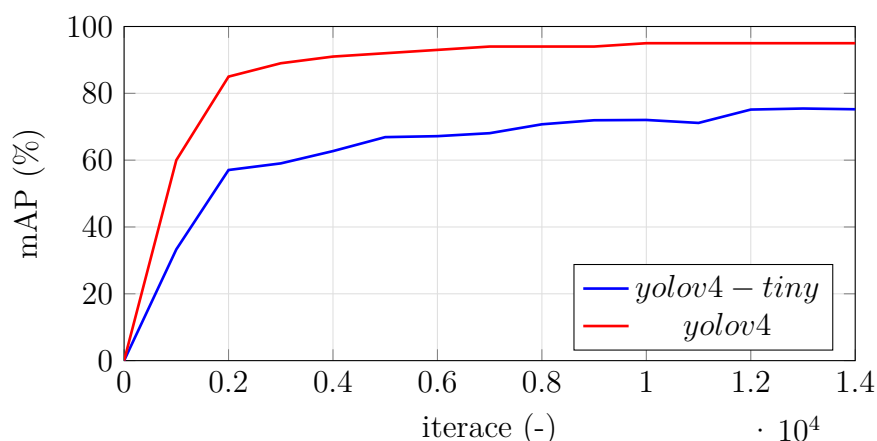
### 2.3.3 Výsledky a porovnání natrénovaného modelu yolov4-tiny a yolov4

Při trénování oba dva modely měly stejnou konfiguraci `.cfg` souboru a trénovali se ze stejných podmínek. Jediný rozdíl byl v tom, že při trénování modelu *yolov4* byla použita grafická karta s větší kapacitou. Pro trénování *yolov4-tiny* jsem využil grafickou kartu NVIDIA GTX 1050 (4 Gb), procesor AMD Ryzen 5 1400 a 8 Gb RAM, zatímco pro trénování *yolov4* byla použita grafická karta NVIDIA RTX 2080ti (11 Gb), procesor AMD Ryzen 9 3900X a 32 Gb RAM. Změna výpočtového zařízení byla způsobena tím, že model neuronové sítě *yolov4* je robustnější ve srovnání s *yolov4-tiny* a tak při trénování vyžaduje více operační paměti grafické karty.

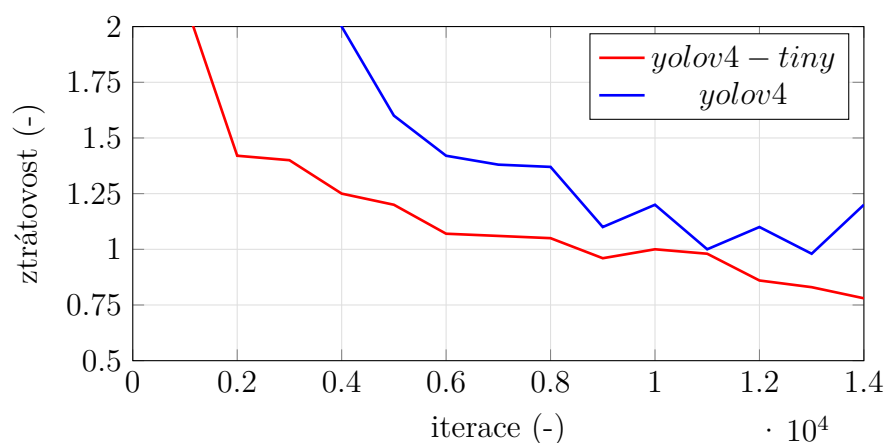
Následující dva grafy (obrázek 2.6 a 2.7) znázorňují, jak se chovala metrika mean Average Precision (zkráceně mAP), která je výchozí metrikou přesnosti, a jaká byla ztrátovost během procesu trénování.

Jak bylo řečeno výše, *yolov4* obsahuje více konvolučních vrstev, s toho důvodu je oproti *yolov4-tiny* přesnější. Tuto teorii dokazuje graf na obrázku 2.6.

Poté jsem pomocí programu *Darknet* porovnal hodnoty FPS na grafické kartě NVIDIA GTX 1050. Pro *yolov4* FPS se rovna 12.9 snímků za sekundu a pro model *yolov4-tiny* FPS se rovna až 64.3 snímků za sekundu. Vzhledem k tomu, že model *yolov4-tiny* vykazuje lepší hodnotu FPS, bylo by lepší použít jej pro detekci v reálném čase, zatímco *yolov4* bude lepší pro detekci objektů ze stávajících videí.



Obr. 2.6: Priebeh mAP počas procesu trérování dvou modelu detektoru YOLO.



Obr. 2.7: Priebeh strátovosti počas procesu trérování dvou modelu detektoru YOLO.

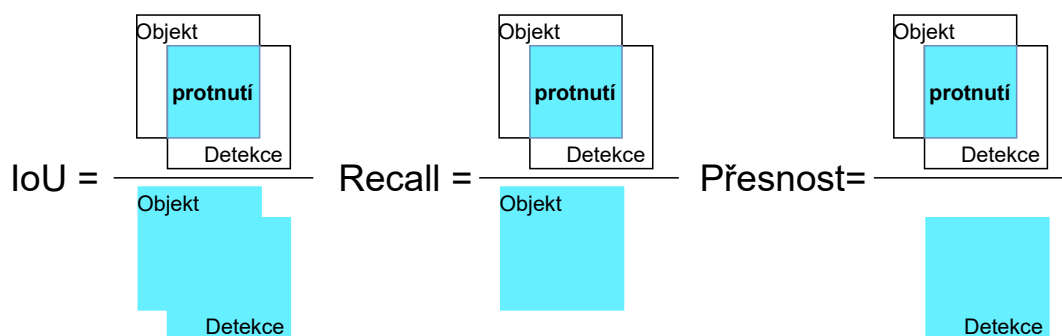
Nejvhodnější natrénovaný model *yolo4-tiny* dosahuje průměrnou přesnost mAP 61,24 %, recall 0,75, f-score 0,73 a průměrnou hodnotu IoU 55,12 %. U nejvhodnějšího modelu *yolo4* tyto metriky se rovnají: mAP 95,44 %, recall 0,94, f-score 0,94 a hodnota IoU 79,82 %. Jak se vypočítají tyto metriky znárodněno na obrázku 2.8.

Další tabulka porovnává metriky přesnosti jednotlivých tříd pro oba dva naučené modely. Metrika TP (True Positive) ukazuje, kolik anotací bylo úspěšně rozpoznáno, zatímco metrika FP (False Positive) udává, kolik jich selhalo. Objekt je považován za úspěšně rozpoznaný, pokud je hodnota IoU (Intersect over Union) větší nebo rovna 0,5. Hodnota IoU se vypočítá podle obrázku 2.8. Z tabulky je vidět, že nejlepší hodnoty přesnosti dosahuje třída auto u modelu *yolo4-tiny* a třídy kamiony, dodávky a autobusy u modelu *yolo4*, jejich přesnost přesahuje 99 %. Nejmenší hodnoty přesnosti v obou modelech dosáhla třída člověk. Je to pravděpodobně z toho

důvodu, že vytvořený dataset obsahuje nedostatečný počet snímků, kde se zmíněná třída člověk vyskytuje. Problém může být také v tom, že objekty člověk v datové sadě má velmi malou velikost. Řešení tohoto problému je popsáno v kapitole 2.3.4.

Tab. 2.1: Přesnost jednotlivých tříd pro dva naučené modely YOLO.

YOLOv4-tiny				YOLOv4			
nazev třídy	AP[%]	TP	FP	nazev třídy	AP[%]	TP	FP
auto	79.15	6815	1363	auto	95.83	8081	543
člověk	57.4	1290	570	člověk	86.72	1696	239
kamiony	91.68	819	132	kamiony	99.66	912	13
dodávky	90.72	720	161	dodávky	99.23	795	18
jízdní kola	57.76	205	122	jízdní kola	93.77	273	35
motorky	64.82	151	71	motorky	92.49	181	17
autobusy	89.02	338	52	autobusy	99.84	367	6



Obr. 2.8: Výpočet metrik CNN.

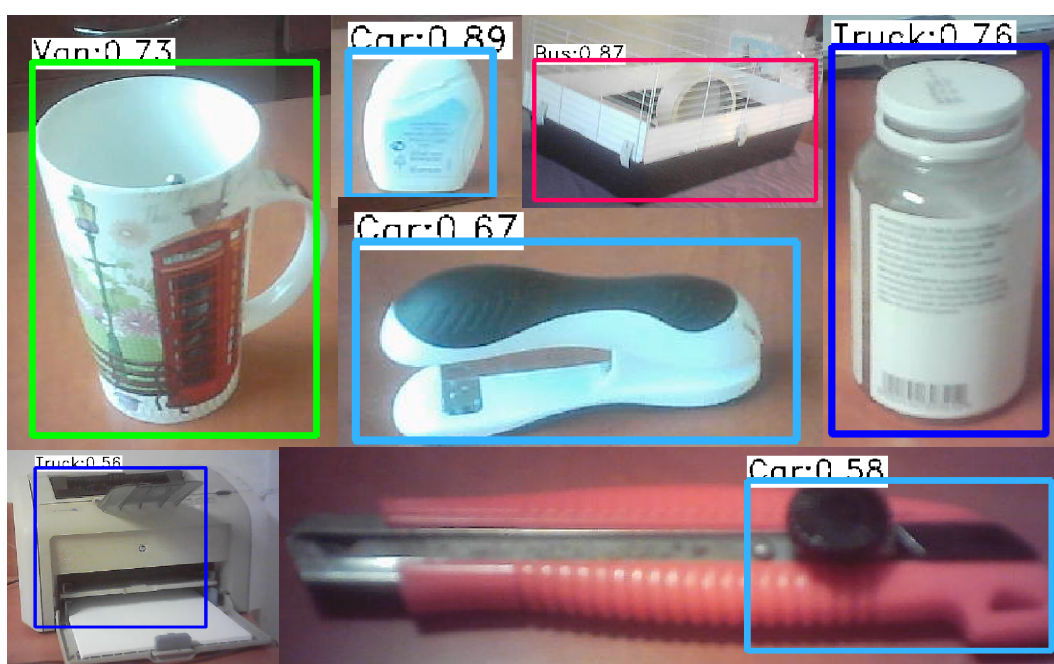
### 2.3.4 Chyby v aplikaci a jak zlepšit výsledky

Ve většině případů *yolov4* úspěšně dokázala detekovat objekt, zatímco *yolov4-tiny* častěji detekovala objekt nesprávně. Pro zlepšení výsledku by bylo potřeba přidat více snímků s problematickými objekty, tedy s objekty, které model detekuje chybně, a zvětšit velikost vstupního obrazu, pokud to grafická karta umožňuje. Také v souboru *.cfg* možná nastavit hodnotu **random** v každé výstupní vrstvě na 1, tím se zvyšuje přesnost při tréninku modelu *Yolo* v různých rozlišeních.

Dalším problémem aplikace bylo, že oba modely vykazovaly nízké hodnoty přesnosti pro třídu člověk. V mé datové sadě je objekt člověk považován za malý objekt.

Objekt je malý, pokud je jeho velikost  $16 \times 16$  s velikostí obrazu  $416 \times 416$ . Výše uvedený popis zlepšení výsledku detekce by podobně zlepšil výsledky i pro detekci objektu člověk, ale navíc jeden z autorů *YOLO* Alexey Bochkovskiy doporučuje nastavit hodnotu **stride** na 4 v *.cfg* souboru. Tuto hodnotu je třeba nastavit pouze v případě, když datová sada obsahuje malé objekty. Pro *yolov4-tiny* se tato hodnota nastavuje na řádcích 246 a 261 a pro *yolov4* odpovídají řádky 892 a 989.

Poslední chyba, se kterou jsem se setkal, je vidět na obrázku 2.9. Oba modely velmi často nesprávně detekovaly různé objekty v reálném čase. Řešením tohoto problému by bylo přidání obrázků do datové sady, které neobsahují žádné objekty detekce. Pro lepší výsledky by se měl počet takových snímků rovnat počtu snímků v datasetu.



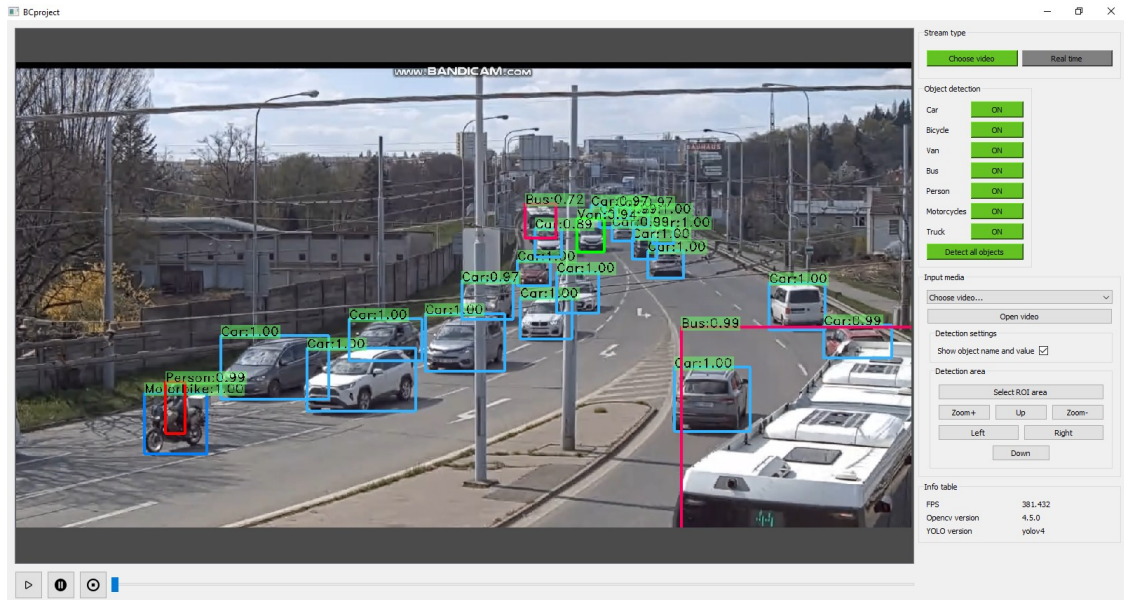
Obr. 2.9: Chyby detekcí.

### 2.3.5 Grafické uživatelské rozhraní aplikace

Vyvinutá aplikace má jednoduché uživatelské rozhraní, které je srozumitelné každému uživateli.

V aplikaci můžeme vybrat režim detekce. Celkem existují 2 režimy: v reálném čase nebo z již existujícího videa. Pomocí tlačítek na pravé straně aplikace může uživatel vybrat objekt detekce, zobrazit jeho jméno a pravděpodobnost detekce. Při stisknutí tlačítka „Select ROI area“ se objeví tzv. ROI (Region Of Interest) oblast a objekty budou detekovány pouze v této oblasti.

V podstatě aplikace umí detekovat celkem 7 tříd: auta, motorky, jízdní kola, autobusy, kamiony, dodávky a lidi. Aplikaci lze také spustit pomocí dvou konvolučních modelů: *yolov4*, *yolov4-tiny*. Výběr detekčního modelu musí být proveden před spuštěním programu pomocí programového kódu.



Obr. 2.10: Ukázka aplikace.

# Závěr

Cílem této práce bylo seznámit se s problematikou zpracování obrazů a naučit vytvořenou aplikaci detekovat pohyblivé objekty s využitím konvolučních neuronových sítí. V úvodu práce je popsán teoretický základ potřebný pro pochopení zpracování obrazů. Tento základ obsahuje snímání a digitalizaci obrazu, předzpracování obrazu, jeho segmentace a nakonec popis a klasifikaci objektu v obraze. V práci se navíc popisuje, co je to neuronová síť, konvoluční neuronová síť, jaké typy neuronových sítí existují, z čeho se skládají a jak fungují.

Pro řešení úkolu bakalářské práce byla použita neuronová síť *YOLO* a byli porovnaní dvě její konvoluční architektury *YOLOv4* a *YOLOv4-tiny*. Tyto architektury byli vybráni z důvodu rychlosti, přesnosti detekcí a ekonomické částí. Pro trénování těchto modelů byla vytvořena vlastní datová sada. Snímky pro kterou byli pořízeni s dopravních online kamer.

Po porovnání dvou modelů detektorů jsem se ujistil, že počet konvolučních vrstev ovlivňuje přesnost detekce a má vliv na rychlost zpracování obrazu. Konvoluční architektury *YOLOv4* obsahovala více konvolučních vrstev a byla přesnější oproti konvoluční architektury *YOLOv4-tiny*. U nejvhodnějšího modelu *yolo4* hodnota přesnosti dosáhla 95,44 % a byla o 34,2 % vyšší než u druhého modelu. Zatímco *YOLOv4-tiny* ukázal lepší hodnotu FPS, která se rovnala 64,3 snímku za sekundu a která byla o 51,4 snímku rychlejší. Výše uvedené metriky byly spočítány programem *Darknet* na grafické kartě NVIDIA GTX 1050.

Vyvinutá aplikace má implementované grafické uživatelské rozhraní, pomocí kterého je možné jednoduše celou aplikaci ovládat. Aplikace je schopna detekovat celkem 7 tříd: auta, motorky, jízdní kola, autobusy, kamiony, dodávky a lidi. Uvedené objekty umí aplikace detekovat v reálném čase nebo z již existujícího videa. V aplikaci je možné nastavit třídu, kterou chceme detekovat, vypsat její název a hodnotu pravděpodobnosti detekce. Aplikace navíc umožňuje vybrat tzv. ROI oblast, přičemž objekty budou detekovány pouze v této oblasti.

# Literatura

- [1] BLANCHETTE J., SUMMERFIELD M. *C++ GUI Programming with Qt 4*. Pearson Hall ve spojení s Trolltech Press, 21.06.2006 [cit. 10.11.2019]. ISBN-10: 0-13-187249-4.
- [2] OpenCV. *OpenCV library* [online]. California: OpenCV team, 2019 [cit. 10.11.2019]. Dostupné z: <https://opencv.org/about/>
- [3] DOXYGEN. *OpenCV modules* [online]. Technická dokumentace, [cit. 10.11.2019]. Dostupné z: <https://docs.opencv.org/master/>.
- [4] HAYKIN, Simon. *Neural networks: a comprehensive foundation*. Second Edition. Ontario: Prentice Hall PTR, 1994, [cit. 15.11.2019]. ISBN 978-0132733502.
- [5] DUMOULIN, Vincent a Francesco VISIN. *A guide to convolution arithmetic for deep learning*. Cornell University [online]. Mila: Université de Montréal, 2018 [cit. 09.12.2019]. Dostupné z: <https://arxiv.org/abs/1603.07285>.
- [6] GEORGIOS, D. *How to select the Right Evaluation Metric for Machine Learning Models: Part 1 Regression Metrics*. Medium [online]. San Francisco(California): The Obvious Corporation, 2017 [cit. 10.11.2019]. Dostupné z: <https://towardsdatascience.com/>.
- [7] BRADSKI, G., KAEHLER, A. *Learning OpenCV*. Sebastopol: O'Reilly Media, 2008. ISBN 978-0-596-51613-0.
- [8] KIAC, M. *Využití moderních metod zpracování obrazu při kontrole laboratorních procesů*. Brno, 2019, [cit. 19.11.2019].. Diplomová práce. FEKT VUT Brno.
- [9] FIŘT, J., HOLOTA R. *Digitalizace a zpracování obrazu*. Nové technologie - výzkumné centrum [online]. Plzeň, [cit. 20.11.2019]. Dostupné z: <http://home.zcu.cz/>
- [10] HORÁK, K., KALOVA, I. *Počítačové vidění* BRNO, 2008. Skripta. FEKT VUT Brno.
- [11] KIAC, M. *Aplikace pro zpracování obrazu na platformě Android*. Brno, 2017. Bakalářská práce. FEKT VUT Brno.
- [12] HORAK, K. *Jasové transformace* Presentace [online]. FEKT VUT Brno. Dostupné z: <http://midas.uamt.feec.vutbr.cz/>

- [13] FAUSET, L. *Fundamentals of neural networks architectures algorithms and applications* [online]. Hall, 1994 [cit. 29.11.2019]. Dostupné z: <http://dl.matlabyar.com/>
- [14] Knižnice OpenCV. *Metod Canny* [online]. California: OpenCV team, 2019 [cit. 20.11.2019]. Dostupné z: <https://docs.opencv.org/>
- [15] ŘIHA, K. *Pokročilé techniky zpracování obrazu*. Brno, 2012. Skripta. FEKT VUT Brno. ISBN 978-80-214-4894-0.
- [16] CHOCHOLATÝ, T. *Detekce dopravních značek a semaforů*. Brno, 2019. Bachelářská práce. FEKT VUT Brno.
- [17] NVIDIA Corporation. CUDA toolkit documentation. *Instalace CUDA Toolkit* [online]. Santa Clara (Kalifornie) [cit. 09.03.2020]. Dostupné z: <https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html>
- [18] NVIDIA Corporation. NVIDIA cuDNN documentation. *Instalace CuDNN* [online]. Santa Clara (Kalifornie) [cit. 09.03.2020]. Dostupné z: <https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html#install-windows>
- [19] Liu, W.; Anguelov, D.; Erhan, D. *SSD: Single Shot MultiBox Detector* [online]. Cornell University. [cit. 07.03.2020]. Dostupné z: <https://arxiv.org/pdf/1512.02325.pdf>.
- [20] GluonCV. *Prepare COCO datasets*. [online]. [cit. 23.03.2020]. Dostupné z: [https://gluon-cv.mxnet.io/build/examples\\_datasets/mscoco.html#sphx-glr-download-build-examples-datasets-mscoco-py](https://gluon-cv.mxnet.io/build/examples_datasets/mscoco.html#sphx-glr-download-build-examples-datasets-mscoco-py)
- [21] TAEGYUN, J. Prezentace: *You Only Look Once (YOLO): Unified Real-Time Object Detection* [online]. [cit. 03.08.2020]. Dostupné z: <https://www.slideshare.net/>.
- [22] BOCHKOVSKIY, A., WANG, C. *YOLOv4: Optimal Speed and Accuracy of Object Detection* [online]. Cornell University [cit. 01.02.2021]. Dostupné z: <https://arxiv.org/pdf/2004.10934.pdf>.
- [23] SHAOQING R., KAIMING H., GIRSHICK R. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* [online]. Cornell University [cit. 03.09.2020]. Dostupné z: <https://arxiv.org/pdf/1506.01497.pdf>.

# Seznam symbolů, veličin a zkratek

<b>IDE</b>	Integrated Development Environment – vývojové prostředí
<b>BSD</b>	Berkeley Software Distribution – distribuce softwaru Berkeley
<b>OpenCV</b>	Open Source Computer Vision Library – volně dostupná knihovna počítačového vidění
<b>CUDA</b>	Compute Unified Device Architecture – hardwarová a softwarová architektura
<b>OpenCL</b>	Open Computing Language – průmyslový standard pro paralelní programování heterogenních počítačových systémů
<b>CNN</b>	Convolutional Neural Network – konvoluční neuronová síť
<b>MSE</b>	Mean Squared Error – střední kvadratická chyba
<b>SURF</b>	Speeded-Up Robust Features – zrychlené robustní funkce
<b>VGG</b>	Visual Geometry Group
<b>HOG</b>	Histogram of Oriented Gradients – histogram orientovaných přechodů
<b>CNN</b>	Convolutional Neural Network – konvoluční neuronová síť
<b>GPU</b>	Graphics Processing Unit – grafická výpočetní jednotka
<b>CPU</b>	Central Processing Unit – centrální procesorová jednotka
<b>cuDNN</b>	CUDA Deep Neural Network library
<b>YOLO</b>	You Only Look Once
<b>SSD</b>	Single Shot Detector
<b>R-CNN</b>	Regionbased Convolutional neural network
<b>RPN</b>	Region Proposal Network – síť pro návrh regionu
<b>PAN</b>	Path Aggregation Network
<b>FPN</b>	Feature Pyramid Networks – pyramidové sítě pro detekci objektů
<b>ASFF</b>	Adaptively Spatial Feature Fusion – adaptivní fúze prostorových funkcí
<b>SFAM</b>	Scale-wise Feature Aggregation Module – modul agregace funkcí v měřítku
<b>SPP</b>	Spatial Pyramid Pooling – kombinace prostorových pyramid v hlubokých konvolučních sítích pro vizuální rozpoznávání
<b>ASPP</b>	Atrous Spatial Pyramid Pooling – silné spojení prostorové pyramidy
<b>RFB</b>	Receptive Fields Block – bere v úvahu vztah mezi velikostí a výstředností přijímajících polí
<b>FPS</b>	Frames Per Second
<b>CVAT</b>	Computer Vision Annotation Tool – nástroj pro anotaci snímků
<b>ROI</b>	Region Of Interest

## A Obsah přiloženého CD

```
/
├── Dokumentace
│   └── Bakalářská práce - Ivan Medynskyi.pdf
├── Zdrojové kódy
│   ├── BCproject.cpp
│   ├── main.cpp
│   ├── BCproject.h
│   └── BCproject.ui
├── YOLO Model
│   ├── yolov4-obj.cfg
│   ├── yolov4-tiny-obj.cfg
│   ├── obj.names
│   └── YOLO_weights.txt.....textový soubor s odkazem na stažení yolo.weights
└── Ukázky aplikace
    ├── Aplikace_1.jpg
    ├── Aplikace_2.jpg
    ├── Aplikace_3.jpg
    ├── Aplikace_4.jpg
    ├── Aplikace_5.jpg
    ├── Aplikace_6.jpg
    └── Aplikace_7.jpg
```