VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE OSOB VE VIDEOZÁZNAMU

PERSON DETECTION IN VIDEO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LUKÁŠ MAREK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ DYK**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Marek Lukáš**
Program: Informační technologie
Název: **Detekce osob ve videozáznamu**
Person Detection in Video
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte základy zpracování obrazu. Zaměřte se zejména na problematiku obecné detekce objektů ve videozáznamu.
2. Vyberte vhodné metody a navrhněte řešení problému detekce postav z videozáznamu.
3. Navržené řešení implementujte v programovacím jazyce Python.
4. Experimentujte s vaší implementací a případně navrhněte vlastní modifikace vybraných metod.
5. Porovnejte dosažené výsledky a navrhněte možnosti budoucího vývoje.

Literatura:

- ANDRILUKA, Mykhaylo; ROTH, Stefan; SCHIELE, Bernt. People-tracking-by-detection and people-detection-by-tracking. In: 2008 IEEE Conference on computer vision and pattern recognition. IEEE, 2008. p. 1-8.
- LANIR, Joel, et al. Visualizing museum visitors' behavior: Where do they go and what do they do there?. *Personal and Ubiquitous Computing*, 2017, 21.2: 313-326.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Dyk Tomáš, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Tato bakalářská práce se zabývá implementací aplikace pro detekci osob ve videozáznamu. Jak aplikace tak detekce jsou implementovány v jazyce Python. Pro vytvoření aplikace je použita knihovna TKinter. Pro detekci je použita knihovna OpenCV a detekční algoritmus YOLOv4, který běží na CUDA backendu, ale může běžet i na procesoru.

Abstract

This bachelor thesis deals with the implementation of an application for detecting people in video footage. Both the application and the detection are implemented in Python. TKinter library is used to create the application. The OpenCV library and the YOLOv4 detection algorithm, which runs on a CUDA backend but can also run on the CPU, are used for detection.

Klíčová slova

detekce, detekce osob, detekce ve videozáznamu, opencv, python, tkinter, zpracování obrazu, yolov4, CUDA

Keywords

detection, person detection, detection in video, opencv, python, tkinter, image processing, yolov4, CUDA

Citace

MAREK, Lukáš. *Detekce osob ve videozáznamu.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Dyk

Detekce osob ve videozáznamu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Dyka. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Lukáš Marek
6. května 2022

Poděkování

Chtěl bych poděkovat svoji rodině za podporu při studiu. Hlavně bych chtěl poděkovat slečně Ing. Aleně Tesařové za to, jaký skvělý člověk je. Tuto práci jsem dokončil hlavně díky ní. A také chci poděkovat svému vedoucímu Ing. Tomášovi Dykovi za přátelský a otevřený přístup.

Obsah

1 Úvod **2**

2 Zpracování obrazu **3**
2.1 Převod na stupně šedi . . . 3
2.2 Gaussovo rozostření . . . 4
2.3 Prahování obrazu . . . 5
2.4 Morfologické transformace . . . 5
2.5 Detekce hran . . . 7
2.6 Houghova transformace . . . 8

3 Detekce objektů ve videozáznamu **11**
3.1 Postprocesní metody . . . 11
3.2 Více snímkové metody . . . 13
3.3 Metody optického toku . . . 14
3.4 Detekce osob a nejpoužívanější metody . . . 17

4 Návrh **27**
4.1 Technologie . . . 27
4.2 Hlavní program . . . 27

5 Implementace **32**
5.1 Instalace prerekvizit . . . 32
5.2 Sestavení . . . 33
5.3 Aplikace a UI . . . 34
5.4 Detekce a sledování . . . 37

6 Testování **41**
6.1 Výsledky . . . 41
6.2 Vlastní model . . . 42
6.3 Grafické porovnání detekce . . . 44

7 Závěr **47**

Literatura **48**

A Obsah přiloženého paměťového média **51**

Kapitola 1

Úvod

Detekce objektů je technologie spojována nejčastěji s počítačovým viděním a zpracováním obrazu. Hlavním účelem je detekce instance objektu nějaké konkrétní třídy, např. osoby, auta, zvěř, budovy apod. Mezi pokročilejší detekce patří detekce obličejů a osob. Detekce objektů má zastoupení v mnoha odvětvích, kde se používá počítačové vidění. Častými příklady využití detekce objektů jsou detekce pohybu u bezpečnostních kamer, kontrola kvality výrobků, počítání dopravních prostředků na komunikacích, sledování míče při fotbalovém zápase a mnoho dalšího.

Cílem této bakalářské práce je program, který bude zpracovávat videozáznamy a detekovat na nich všechny osoby. Detekce by měla probíhat, pokud možno bez chyb a s co největší přesností. Nad rámec základního zadání bude implementováno i sledování osob. To znamená, že sledujeme odkud, kam daná osoba šla a můžeme tak určit třeba za jak dlouho tuto vzdálenost ušla, jak rychle, jaká je nejčastější cesta apod. Výhodu mého řešení je jednoduché použití. Spuštění programu je velmi jednoduché, v podstatě pro něj stačí jakýkoliv stroj, na kterém běží Python. Výsledný obraz (s detekovanými osobami) bude možno sledovat přímo za běhu programu.

Práce je rozdělena na 7 kapitol. První je úvod. Druhá a třetí se zaměřují na teoretickou část práce. Popisují základy zpracování obrazu, detekce objektů a používané metody. Čtvrtá kapitola se zaobírá návrhem programu. Pátá kapitola pak implementací návrhu. Popisuje jednotlivé použité prostředky, knihovny, funkce, dále rozdělení programu a jeho hlavní části. V šesté kapitole je pak popsáno testování, jak probíhalo a jaké byly jeho výsledky. Poslední sedmou kapitolou je pak závěr shrnující celkový výsledek práce.

Kapitola 2

Zpracování obrazu

V dnešní době je zpracování obrazu jedním z nejdůležitějších pomocných nástrojů v mnoha odvětvích (počítačová věda, elektrické a elektronické inženýrství, robotika, fyzika, chemie, biologie). Zpracování obrazu je proces, při kterém se provádí specifické operace nad vybraným snímkem za účelem získání buď to vylepšeného snímku (s lepším rozlišením, zaostřením, apod.) nebo získání důležitých informací (přítomnost osob, aut, atd.). Jedná se o typ signálového zpracování, kde vstupem je snímek a výstupem může být taktéž snímek nebo charakteristika/vlastnost spojená s tímto snímkem [25].

Zpracování obrazu se v základu skládá z těchto tří kroků:

- Vstup – Import snímku skrze nástroj pro jejich manipulaci
- Analýza – Manipulace se snímkem, úprava do požadované formy
- Výstup –Může se jednat o upravený vstupní snímek nebo report založený na jeho analýze

S obrazem se pracuje jako s 2D polem prvků, kde každý prvek je jeden pixel složený ze tří pod-pixelů (výsledné pole pak vypadá takto [[R,G,B],[R,G,B],...]). To urychluje a usnadňuje celkovou práci s obrazem.

2.1 Převod na stupně šedi

Jedna ze základních a nejdůležitějších metod pro zpracování obrazu. Pro převod barevného obrázku na stupně šedi stačí pouhý obrázek. Barva pixelu obrázku je kombinací barev červené, zelené a modré (RGB). Kvalita barvy závisí taktéž na bitové hloubce, kterou dané zařízení podporuje. Základem je 8 bitů. Čím více bitů máme, tím kvalitnější, ale i větší, obrázek je. Pro tři barevné složky by to tak bylo 24 bitů, jejichž kombinací lze získat 16 777 216 různých barev.

Šedý obraz je však prezentován pouze intenzitou a ta je 8 bitová. Je nutné tak převést 24bitů na 8. Pro převod se používá následující rovnice:

$$intenzita = 0.299 * R + 0.587 * G + 0.114 * B \tag{2.1}$$

Každá barevná složka je vynásobena vlastní konstantou, protože každá barva je pro člověka jinak viditelná než jiná (citlivost lidského oka pro každou barvu je jiná). Barevné složky jsou tak nahrazeny jednou hodnotou a to intenzitou. Výsledkem je obrázek ve stupních šedi.[28][6]

Obrázek 2.1: Převod obrázku do stupňů šedi

2.2 Gaussovo rozostření

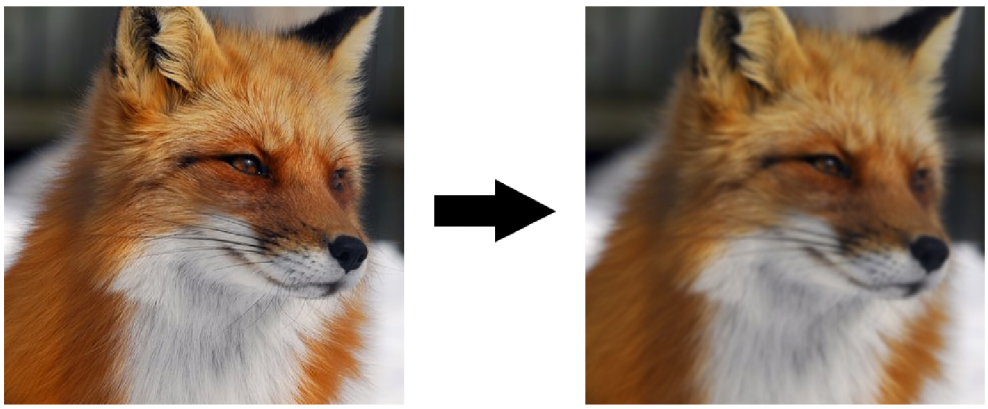
Gaussovo rozostření se získá použitím Gaussovy funkce. Výsledkem je pak vyhlazení výsledného obrazu (rozostření), ale i snížení šumu. Jedná se prakticky o filtr typu dolní propusti, který zachovává nízkou prostorovou frekvenci, redukuje obrazový šum a zanedbatelné detaily. Typicky je tohoto dosáhnuto konvolucí obrazu s Gaussovým jádrem. Gaussovo jádro je v 2D prostoru definováno jako:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.2}$$

kde:

- σ – směrodatná odchylka rozdělení
- (x, y) – souřadnice pixelu

Hodnota σ řídí rozptyl kolem střední hodnoty Gaussova rozdělení, což určuje sílu rozostření kolem daného pixelu. [7]



Obrázek 2.2: Převod obrázku pomocí Gaussova rozostření

2.3 Prahování obrazu

Vzhledem k intuitivním vlastnostem, jednoduchosti implementace a výpočetní rychlosti je prahování jeden z hlavních způsobů jak segmentovat obraz. Vstupem prahování je obrázek převedený do stupňů šedi.

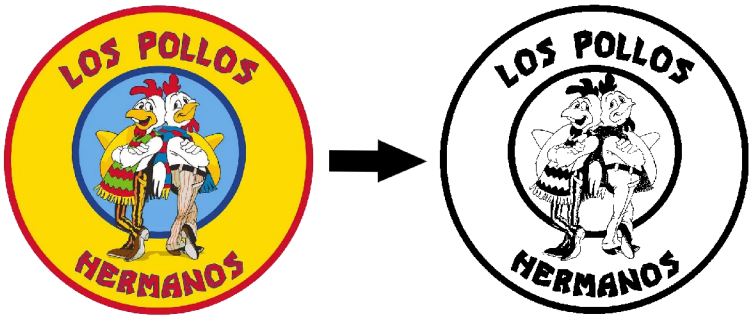
Obraz $f(x, y)$, kde x a y jsou souřadnice pixelu v 2D poli, je tvořen světlými objekty na tmavém pozadí tak, že intenzita jejich pixelů je seskupená do dvou dominantních skupin. Jedním z řešení, jak extrahovat objekty z pozadí, je zvolení prahu T, který tyto skupiny navzájem separuje. Potom jakýkoliv bod (x, y) obrazu, kdy $f(x, y) > T$, je bod objektu. Jinak se jedná o bod pozadí. Jinými slovy se segmentovaný obraz $g(x, y)$ dá získat následující funkcí:

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T \end{cases} \tag{2.3}$$

kde:

- $g(x, y)$ – výsledný segmentovaný obraz
- $f(x, y)$ – funkce prahování
- T – hodnota prahu
- 0 a 1 – černá a bílá barva

V případě, že T je konstantní a nemění se během aplikace prahování, jedná se o globální prahování. Naopak v případě, že se T mění během aplikace prahování, jedná se o variabilní prahování. Lze se i setkat s pojmy jako lokální či regionální prahování. V tomto případě se hodnota prahu mění podle hodnot sousedních pixelů. Nebo když záleží na poloze pixelu, tedy přímo na hodnotách (x, y), jedná se o dynamické neboli adaptivní prahování. [26]



Obrázek 2.3: Převod obrázku do černobílé pomocí prahování

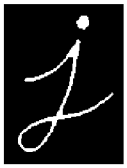
2.4 Morfologické transformace

Morfologické transformace jsou jednoduché operace založené výhradně na několika základních množinových operátorech. Ty se aplikují na binární podobu obrazu. Jsou zapotřebí dva vstupy a to původní obrázek a strukturující element nebo jádro (*kernel*), který rozhoduje o typu operace. Dvě základní operace jsou eroze a dilatace. Následně jejich varianty jako třeba otevření, uzavření atd.

Obrázek 2.4: Ukázkový obrázek bez úprav [1]

2.4.1 Eroze

Základní myšlenka eroze je jako půdní eroze. Eroduje kraje objektu v popředí (je dobré udržovat popředí bílé). Princip je, že ono jádro projde skrze obrázek jako při 2D konvoluci. Pixel v originálním obrázku (0 nebo 1) bude považován za 1, pouze pokud všechny pixely pod jádrem jsou 1, jinak je pixel erodován (přeměněn na nulu). Výsledkem je, že všechny pixely, poblíž krajů, jsou erodovány v závislosti na velikosti jádra. Zjednodušeně se bílý objekt v popředí zeštíhlí nebo zmenší. Eroze je tak vhodná na odstranění šumu, rozpojení dvou objektů apod.



Obrázek 2.5: Ukázkový obrázek po aplikaci eroze [1]

2.4.2 Dilatace

Dilatace je přesný opak eroze. Zde je pixel v originálním obrázku považován za 1, když aspoň jeden pixel pod jádrem je taky 1. Výsledkem je tak zesílení nebo zvětšení objektu v popředí. V případech jako při odstranění šumu, běžně dilatace následuje erozi. Dělá se to proto, jelikož eroze odstraní bílý šum, ale taky zmenší obrázek. Proto použijeme dilataci. Bílý šum je odstraněn a obrázek se vrátí do původní velikosti. Dilatace se také používá na spojení rozpojených částí objektu.

Obrázek 2.6: Ukázkový obrázek po aplikaci dilatace [1]

[1] Obrázky 2.4, 2.5 a 2.6 jsou převzaty z [5]

2.4.3 Kombinace předchozích operací

Otevření

Otevření je jen jiné pojmenování eroze následované dilatací. Klasické použití je odstranění šumu.

Obrázek 2.7: Ukázkový obrázek po aplikaci otevření [2]

Uzavření

Uzavření je zase opak otevření. Dilatace následována erozí. Hodí se na odstranění malých děr uvnitř objektu v popředí.

Obrázek 2.8: Ukázkový obrázek po aplikaci eroze [2]

2.5 Detekce hran

Detekce hran jakýmkoliv způsobem je esenciální součástí mnoha systému založených na počítačovém vidění. Detekce hran slouží ke zjednodušení analýzy drastickým zmenšením zpracovávaných dat a zároveň zachování potřebných informací o hranicích objektů. Ty jsou potřeba ze všeho nejvíc, jelikož určují umístění objektu, jeho rozpoznání oproti pozadí nebo jinému objektu. Detektorů hran existuje nepřeberné množství. Jeden z těch důležitějších je Cannyho hranový detektor [10]. Jeho autorem je John Canny, který roku 1986 definoval určité cíle, co by měl detektor hran plnit. Cíle jsou následující [24]:

- Chybovost – Detektor hran by měl reagovat pouze na hrany a měl by je najít všechny.
- Umístění – Vzdálenost, mezi hranovými pixely nalezenými detektorem a skutečnými hranovými pixely, by měla být co nejmenší.
- Odezva – Detektor by neměl identifikovat více hranových pixelů, kde existuje jen jeden.

[2] Obrázky 2.7 a 2.8 jsou převzaty z [5]

Cannyho detektor hran je více stupňový proces. Tyto procesy [1] jsou:

- Redukce šumu – šum je odstraněn pomocí Gaussova filtru.
- Nalezení gradientu obrazu – vyhlazený obraz je vyfiltrován Sobelovým jádrem v obou směrech pro získání prvního derivátu. Z toho lze pak zjistit hranový gradient a směr pro každý pixel.
- Nalezení lokálních maxim – získané gradienty se využijí pro odstranění nežádoucích pixelů, které nemusí být součástí hrany. Tj. když gradient nedosahuje lokálního maxima.
- Odstranění přebytečných hran – tato část určuje co je vlastně hrana a co ne. K tomuto jsou zapotřebí dvě hodnoty a to `maxVal` a `minVal`. Všechny hrany s gradientem větším jak `maxVal` jsou určitě hrany a s menším než `minVal` určitě hrany nejsou a jsou odstraněny. Hrany co leží mezi, jsou kontrolovány, zda jsou spojeny s určitými hranami (jsou součástí hrany). Když nejsou propojeny, jsou odstraněny.

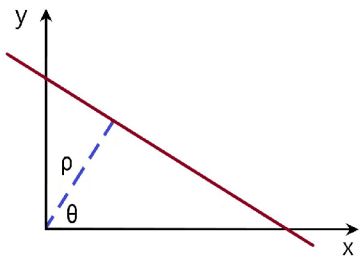
2.6 Houghova transformace

Originální forma Houghovi transformace je primárně zaměřena na identifikaci rovných čar v obraze. S úpravou se dá také použít na identifikaci dalších tvarů (kružnic, elips). Aby šla transformace použít, musí být zpracovávaný snímek v binární podobě. Hledat rovné čáry v barevném obraze hned nejde. Nejběžnější přístup je před aplikací transformace snímek převést na stupně šedi a následně detekovat hrany. Výslednou masku lze bezpečně předhodit Houghově metodě, která by měla vrátit všechny rovné čáry na snímku.

2.6.1 Reprezentace přímek

Reprezentace těchto čar je nejsnadnější pomocí páru parametrů (a, b). Výsledkem je základní rovnice pro přímku: $y = ax + b$. Nebo je lze také popsat souřadnicemi v polárním systému pomocí páru (ρ, θ). První parametr je kolmá vzdálenost od počátku k přímce. Druhý parametr je úhel mezi osou x a linkou značící vzdálenost k přímce. Jednou z výhodou tohoto přístupu je, že umožňuje vertikální linie popsat těmito parametry v Kartézském systému, což je nemožné u parametrů (a, b). Vznikne tak rovnice:

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{2.4}$$

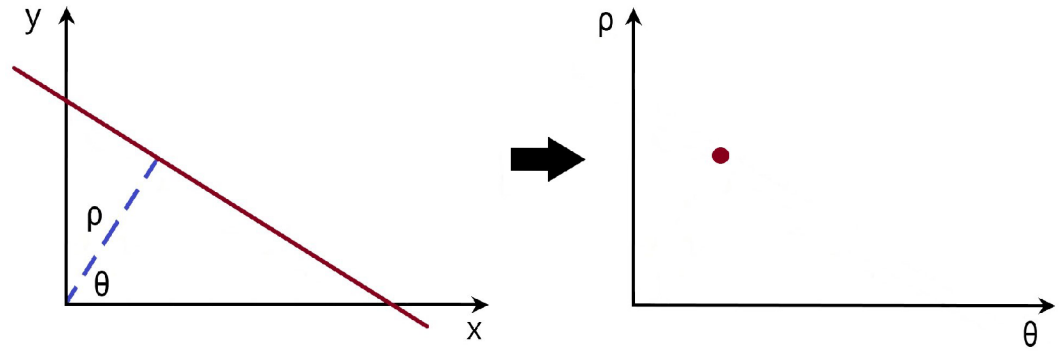


Obrázek 2.9: Přímka v polární soustavě [3]

[3] Obrázek 2.9 je převzat z [17]

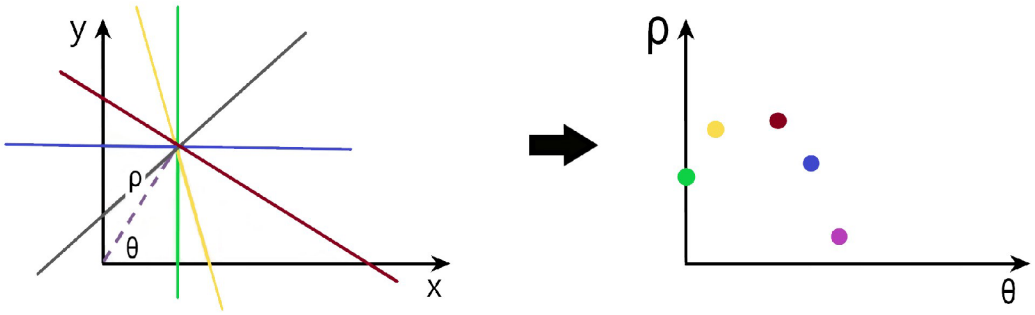
2.6.2 Mapování z obrazového prostoru do Houghova prostoru

Už je definované, že lze přímku reprezentovat parametry ρ a θ. Do Houghova prostoru tak jde zakreslit bod s těmito souřadnicemi.



Obrázek 2.10: Přemapování do Houghova prostoru [4]

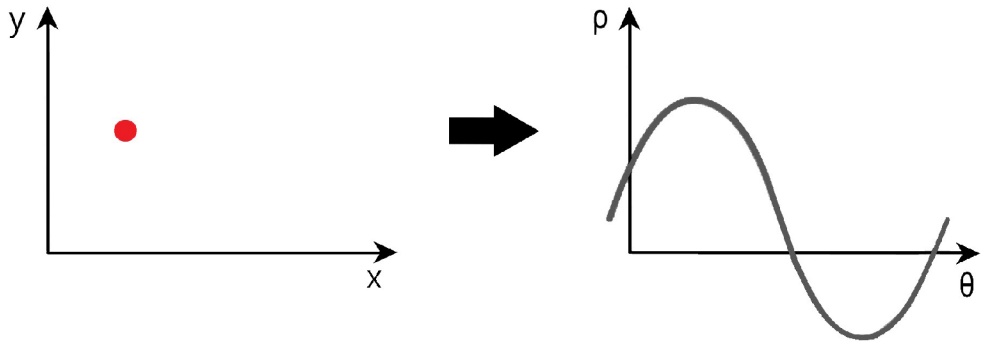
Do prostoru obrázku se přidají další přímky, které se ale všechny protínají ve společném bodě. Opět se tyto přímky převedou na body do Houghova prostoru.



Obrázek 2.11: Přemapování více přímek do Houghova prostoru začne tvořit sinusoidu [4]

Je zřejmé, že tyto body tvoří sinusoidu. Vykreslení nekonečně mnoha dalších přímek, procházejícím v jednom stejném bodě, by vyústilo v nepřerušovanou sinusoidu v Houghově prostoru. Při opětovném pohledu na rovnici 2.4 popisující vztah těchto parametrů, lze vidět, že při fixních parametrech (x, y) a změně všech možných hodnot parametru θ v určitém rozsahu, vznikají hodnoty parametru ρ, které tvoří sinusoidu.

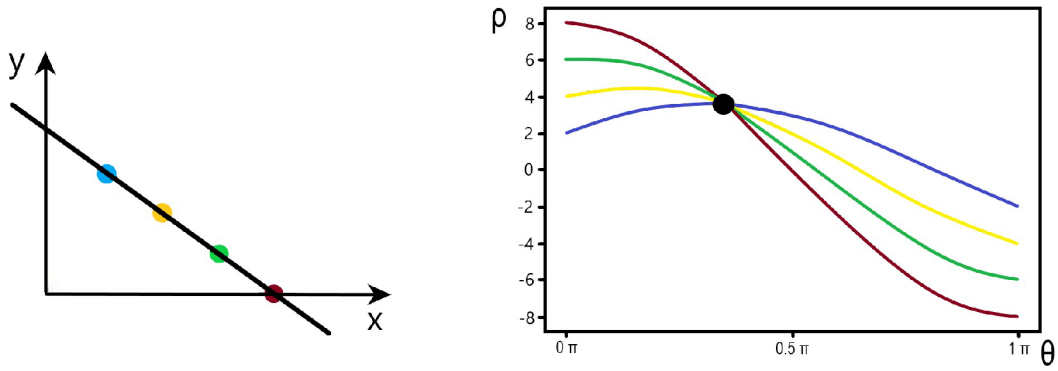
[4] Obrázky 2.10 a 2.11 jsou převzaty z [17]



Obrázek 2.12: Pro daný bod máme v Houghově prostoru kompletní sinusoidu [5]

2.6.3 Nalezení Houghových čar

Při nakreslení několik bodů v obrazovém prostoru, které tvoří přímku, v se Houghově prostoru vytvoří několik sinusoid. Ty ovšem prochází jedním společným bodem.



Obrázek 2.13: Sinusoidy procházející jedním společným bodem [5]

To znamená, že pro hledání bodů, jež jsou kandidáty na přímku, je nutné hledat průniky v Houghově prostoru.

[5]Obrázky 2.12 a 2.13 jsou převzaty z [17]

Kapitola 3

Detekce objektů ve videozáznamu

Za poslední desetiletí byl zaznamenán velký pokrok týkající se strojového učení, zvláště pak v počítačovém vidění. Pokrok byl hlavně ve zpracování samotného obrazu (klasifikace, detekce, segmentace a generace), zatímco zpracování videa bylo daleko méně prozkoumáno. Video je v podstatě sekvence obrázků. Avšak tato definice sama o sobě nedokáže pokrýt, co zpracování videa samo o sobě znamená. Zpracování videa totiž přidává úplně nový rozměr této problematice a to **časový rozměr**. Videa jsou sekvence **souvisejících** obrázků. Ač se to může zdát jako minimální rozdíl, lze tento rozměr využít mnoha způsoby, které na samostatný obrázek aplikovat nelze. Prorazit na tomto poli bylo také těžší z dvou důvodů: výpočetní náročnost a komplexnost videa. To se změnilo s vydáním *ImageNet VID*[6] a jiných masivních data setů během druhé půlky dekády. Od té doby vzniká více a více studií věnovaných zpracování videí.

3.1 Postprocesní metody

Jedná se o modifikace aplikované v post procesu detekce objektu. To proto, že vyžadují méně infrastruktury a nevyžadují žádné změny v architektuře modelu. Tyto metody jsou v základu stále detekcí po snímcích, takže nepřináší žádné viditelné zrychlení, mohou být dokonce i pomalejší. Avšak dosahují znatelných zlepšení, co se přesnosti týče.

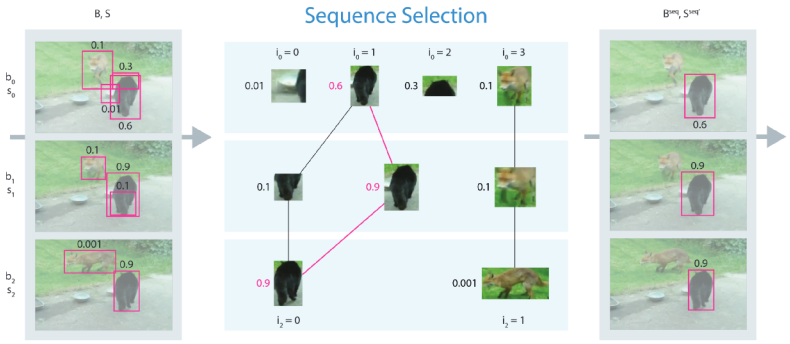
Metoda Seq-NMS

Sequence Non-Maximal Suppression[13] je metoda, která používá modifikaci důvěryhodnosti detekce na základě jiných detekcí. Například slabší návrhy pozitivního objektu mohou být způsobeny okluzí, pohybovou neostrostí nebo jinými vadami, ale protože budou extrahované z předchozích snímků, důvěra se zvýší. To efektivně minimalizuje počet chybných detekcí mezi snímky nebo výskyt náhodných skokových detekcí a stabilizuje se konečný výsledek. Princip algoritmu NMS v krocích (převzato od [16]):

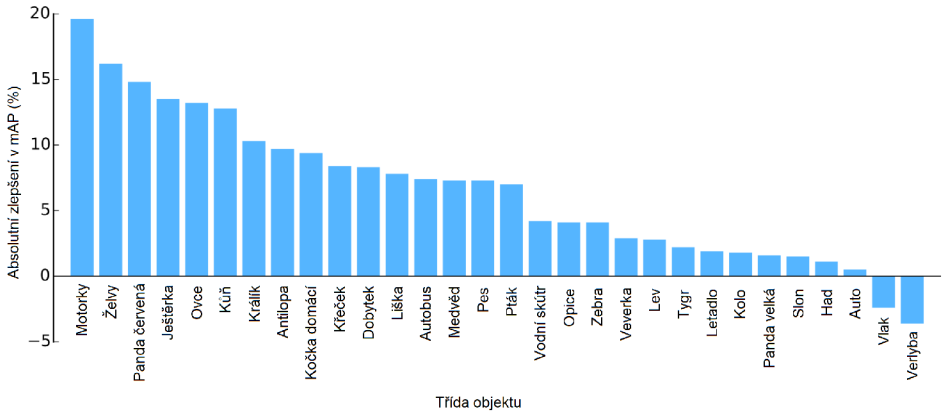
- Vstup: List navrhnutých rámců `B` k nim odpovídající důvěryhodnostní hodnota `S` a práh pro překrytí `N`
- Výstup: List vyfiltrovaných návrhů `D`
- Algoritmus:

[6] Odkaz na data sety zde: https://bvisionweb1.cs.unc.edu/ilsvrc2015/download-videos-3j16.php

1. Vybrání návrhu s největší důvěryhodností hodnotou, jeho odstranění z listu `B` a vložení do listu `D`.
2. Porovnání tohoto návrhu s ostatními návrhy – vypočítání IoU[7] tohoto návrhu s každým návrhem, pokud $IoU > N$ tak se návrh odstranění z listu `B`.
3. Opět vezmeme návrh s největší důvěryhodností hodnotou ze zbývajících návrhů v listu `B`, odstraníme ho z `B` a přidáme do `D`.
4. Opět vypočítání IoU tohoto návrhu s každým návrhem, pokud $IoU > N$ tak se návrh odstranění z listu `B`.
5. Tento proces je opakován, dokud není list `B` prázdný.



Obrázek 3.1: Ilustrace sekvenčního výběru. Je sestaven graf, kde jsou políčka v sousedních rámcích propojena, pokud je jejich $IoU > 0,5$ [8]



Obrázek 3.2: Graf absolutního zlepšení v mAP (průměrná střední přesnost) po aplikaci Seq-NMS [8]

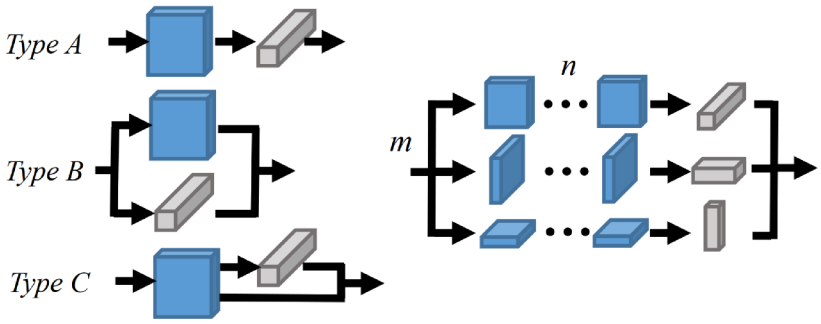
Je vidět, že po aplikaci metody Seq-NMS se zvětšila přesnost detekce o velmi relevantní hodnotu. U sedmi objektů se přesnost zvýšila o více jak 10% a jenom u dvou došlo k poklesu přesnosti. Ačkoli je tato metoda znatelně přesnější, neukázala se jako nejlepší jak z hlediska přesnosti, tak z hlediska výkonu. Viditelnou výhodou však je, že nevyžaduje vlastní trénink a funguje spíše jako doplněk, který lze zapojit do jakéhokoli objektového detektoru.

[7] IoU (Intersection over Union) – podíl průniku a sjednocení dvou navrhnutých snímků
[8] Obrázky 3.1 a 3.2 jsou převzaty a přeloženy z [13]

3.2 Více snímkové metody

3D Konvoluce

Myšlenkou je 3D konvoluce, založená na 2D konvoluci, která se provádí u obrázků. Tuto myšlenku lze provést: iterací přes n snímků jakožto vstupů do modelu a výstup sekvenčních detekcí na po sobě jdoucích snímcích. Rozhodně se tak jedná o potencionální směr pro detekci, jelikož takhle může extrahovat nízko úrovňové rysy pro časoprostorová data. Problém je, že konvoluční neuronová síť s 3D konvolucí se většinou ukázala jako užitečná, pokud jde o zpracování 3D snímků, například na *3D MNIST* nebo skenech *MRI*. Proto jsou tyto modely více užitečné v oblasti lékařského zobrazování a méně tak pro detekci ve videu.



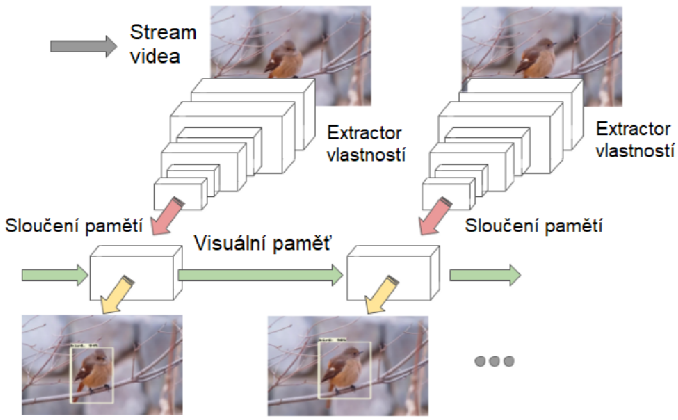
Obrázek 3.3: Strukturované 2D a 1D konvoluce [9]

Jedním z příkladů výzkumné práce, která se tímto zabývá, je [15]. V této výzkumné práci je video nejprve rozděleno na stejně dlouhé klipy a poté je pro každý klip vygenerována sada návrhů založená na vlastnostech *3D CNN* (3D konvoluční neuronové sítě). Následně jsou návrhy různých klipů spojeny dohromady a s jejich pomocí je provedena časoprostorová detekce akcí. To ovšem neznamená, že by se tato architektura nedala přeučit na detekci objektů. Tato metoda ovšem nevyniká ve směru výkonu. Vzhledem k velkému objemu výpočtů, s vícerozměrnými maticemi, nemůže být doba zpracování za současného stavu v reálném čase. 3D konvoluce není snadným úkolem, takže k brzkému navýšení výkonu jen tak nedojde. Přesnost a stabilitu však s 3D konvolucí výrazně zlepšit. To je možné díky efektivnímu využití časového rozměru, konkrétně agregace rysů mezi snímky. V současné době se však jedná pouze o spekulaci založenou na jiných nejmodernějších 3D konvolučních modelech.

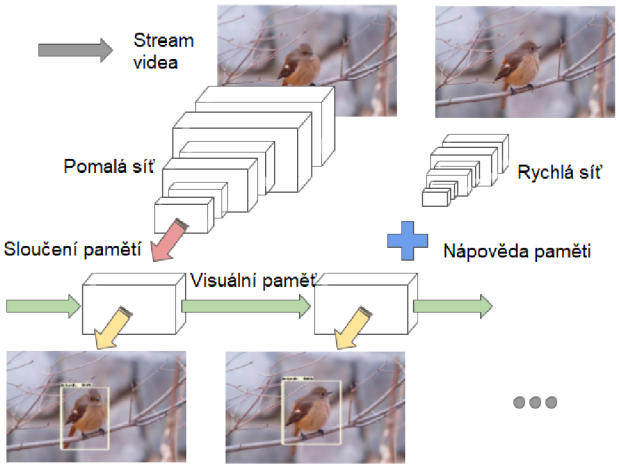
Rekurentní neuronové sítě

RNN (Recurrent Neural Networks) jsou speciální typy sítí vytvořené za účelem zpracování sekvenčních a dočasných dat. Nejčastější využití je v oblasti zpracování přirozeného jazyka. *RNN* se dá využít i u zpracování obrazu, jelikož video je sekvence snímků. Tímto se zabývají například práce: [19] a [20]. Práce kombinují rychlou detekci jednoho obrazu s konvolučními vrstvami s dlouhou krátkodobou pamětí (*LSTM*) pro vytvoření propletené rekurentně-konvoluční architektury. *LSTM* vrstva snižuje výpočetní náročnost při zachování map objektů napříč snímky.

[9] Obrázek 3.3 je převzat z [12]



Obrázek 3.4: LSTM vrstvy pro zlepšení výkonu [9]



Obrázek 3.5: LSTM vrstvy s pomalou a rychlou neuronovou sítí [10]

Je možné také využít podstatu scény. Architektura modelu spočívá ve střídání konvenčních extraktorů vlastností s odlehčenými, které musí pouze rozpoznat podstatu scény (minimální výpočetní zátěž). Tím se efektivně vytváří dlouhodobá paměť architektury z klíčového snímku, který zachycuje podstatu scény a vede pomalejší síť pouze k tomu, co má detekovat. Problém nastává při trénování architektury, ta si totiž bere na vstupu obraz a na výstupu nám dává masku. Z toho vyplývá, že musí být trénována celá architektura, což u videozáznamů znamená obrovské množství dat.

3.3 Metody optického toku

Samozřejmě lze pro detekci použít i metody používané u zpracování obrazu, jakými jsou např. *YOLOv3*, *RetinaFace* nebo *DSFD*. Princip je, že aplikujeme vybranou metodu na každý snímek videa zvlášť. Přesnost výsledku je pak průměr přesností jednotlivých snímků. Přímé aplikování těchto metod je jednoduché, trpí tím však rychlost a přesnost.

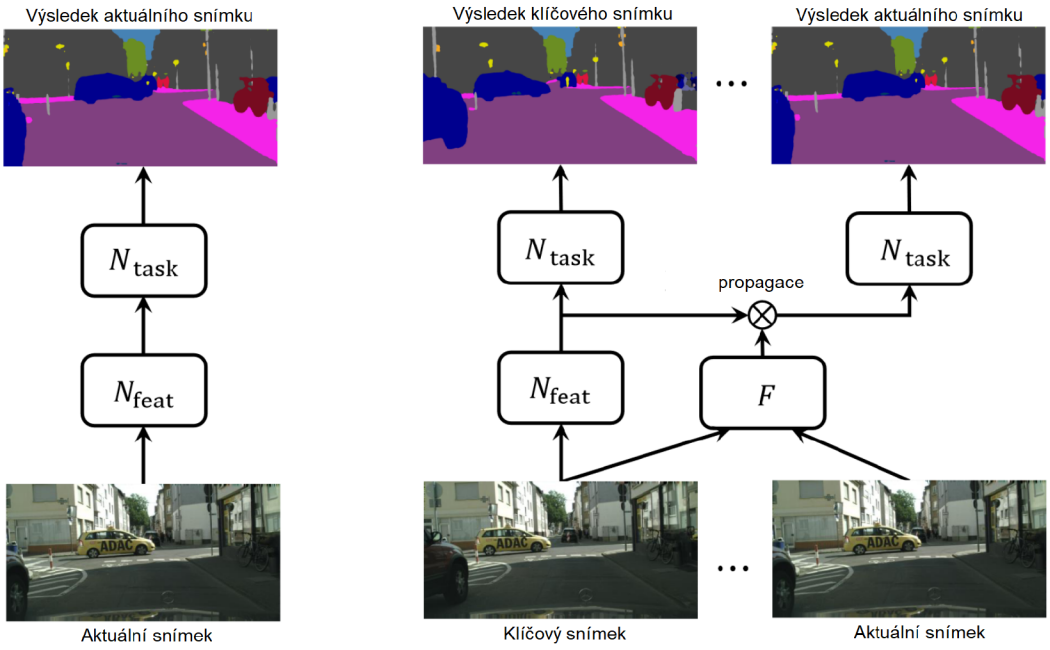
[10] Obrázek 3.4 je převzat a přeložen z [19] a 3.5 je převzat a přeložen z [20]

Rychlost trpí tím, že pro každý snímek se musí daná metoda spouštět znovu, což stojí čas a navíc u videí se ihned za sebou jdoucí snímky moc neliší, což vede na zbytečné výpočty. Přesnost trpí na zhoršené podmínky, jakými jsou: pohybové rozostření, rozostření videa nebo zvláštní pózy. Ty mohou detekci na některých snímcích zcela znehodnotit a snížit tím celkovou přesnost.

Odhad na základě optického toku je metoda odhadu zdánlivého pohybu objektů mezi dvěma snímky. Ten může být zapříčiněn pohybem kamery nebo samotného objektu. Výstupem je obvykle 2D vektorové pole, kde každý vektor představuje vektor posunu pixelu z prvního snímku na druhý.

Propagace pro výkon

Architektura pracuje s konceptem řídkého klíčového rámce. Jelikož síť optických toků může být relativně malá, doba zpracování a výpočetní výkon potřebný pro takové sítě jsou menší než u detektorů objektů. Z toho důvodu pipeline funguje jako cyklus n snímků. První snímek se nazývá klíčový snímek. Jedná se o snímek, který je detekován detektorem objektů. Protože nyní detektory poskytují přesnou detekci všech objektů, budou detekce podléhat algoritmům optického toku. Po získání vektorů posunutí je známa detekce dalších n-1 snímků a cyklus se opakuje.

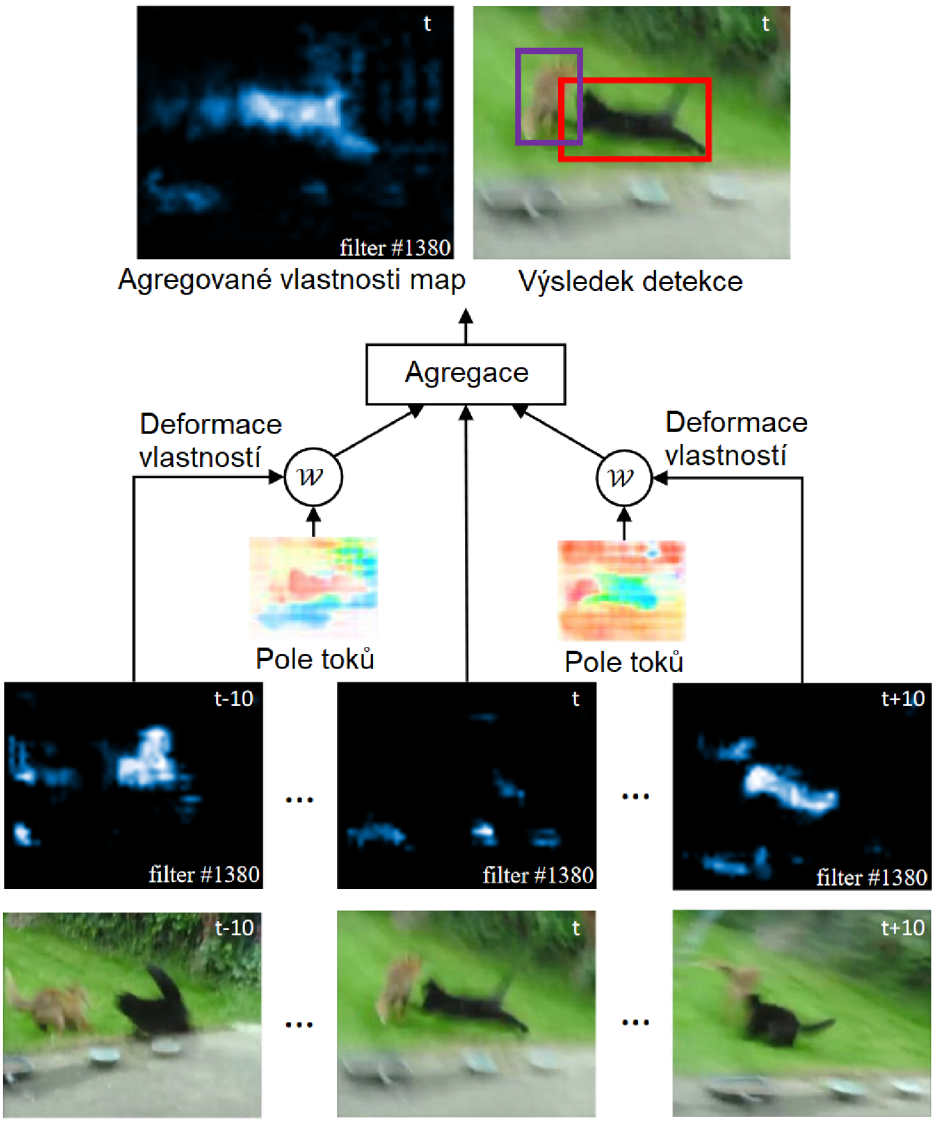


Obrázek 3.6: Síť pro jeden snímek a síť pro DFF (Deep Feature Flow) [11]

[11] Obrázek 3.6 je převzat a přeložen z [31]

Agregace pro přesnost

Výkon je vyřešený pomocí propagace, zbývá jen přesnost a ta se řeší pomocí agregace více snímků. Existují různé způsoby její implementace, ale všechny mají stejnou myšlenku. Hustě vypočtené detekce, při deformaci rysů ze sousedních snímků, na aktuální snímek a agregaci pomocí váženého průměrování. Aktuální snímek tedy využije bezprostřední snímky i některé další snímky k získání lepší detekce. To by pak mohlo vyřešit problémy s pohybem a oříznutými objekty ze snímku videa.



Obrázek 3.7: Příklad agregace vlastností více snímků [12]

[12] Obrázek 3.7 je převzat a přeložen z [30]

3.4 Detekce osob a nejpoužívanější metody

Detekce osob je konkrétní případ detekce objektů, jelikož osoba je jednou z mnoha tříd, kterou lze detekovat. Detekování osob je v současnosti jednou z nejrozšířenějších typů detekcí. Hlavní využití je u monitorovacích systémů, takže nároky na přesnost jsou více než žádoucí. Dnešní *deep learning* algoritmy nabízejí robustní detekci osob a jsou nejčastěji trénované na čelních a asymetrických pohledech. Algoritmy trénované jen čelních pohledech mají i tak výborné výsledky, když jsou správně aplikované, např. na počítaní osob.

Všechny moderní algoritmy jsou založené na metodě *deep learning*. V kombinaci s čím dál tím výkonnějšími GPU se výkon objektových detektorů a sledovačů rapidně zvýšil a pokrok v tomto odvětví se jen tak nezastaví. Strojové učení je součástí umělé inteligence a základem je učení se vzorců z ukázkových dat s vlastností se z toho poučit. *Deep learning* je speciální případ strojového učení s možností učení se v různých fázích. Oproti obyčejnému postupu přináší tento přístup mnoho výhod, díky dobrým a velkým data setům jsou tyto metody mnohem odolnější proti změnám pozadí, okluzi, komplexnějším scénám apod. [9]

3.4.1 Dvoufázové algoritmy

Základní princip spočívá v tom, že ve dvoustupňových detektorech objektů se navrhují přibližné oblasti objektů pomocí hloubkových rysů předtím, než se tyto rysy použijí pro klasifikaci a regresi ohraničujícího pole pro kandidáta na objekt.

Mask R-CNN

Mask R-CNN je nejmodernější model pro segmentaci obrazu vyvinutý na základě *Faster R-CNN*. Prvně by bylo dobré vysvětlit jednotlivé části této metody.

Convolutional Neural Network

Jedná se o typ umělé neuronové sítě používané při rozpoznávání a zpracování obrazu, která je optimalizována pro zpracování pixelových dat. Kombinace vrstev *CNN* umožňuje navržené neuronové síti naučit se identifikovat a rozpoznat objekt zájmu v obraze. Jednoduché konvoluční neuronové sítě jsou konstruovány pro klasifikaci obrazu a detekci objektů s jediným objektem v obraze. Architektura *CNN* se skládá ze tří hlavních vrstev[23]:

1. **Konvoluční vrstva** – pomáhá abstrahovat vstupní obraz jakož to mapu prvků pomocí filtrů a jader.
2. **Sdružovací vrstva** – pomáhá zmenšovat vzorky map prvků tím, že shrnuje přítomnost prvků v políčkách mapy prvků.
3. **Plně propojená vrstva** – propojuje každý neuron v jedné vrstvě s každým neuronem v jiné vrstvě.

Region-Based Convolutional Neural Network

Regionálně založené konvoluční neuronové síti neboli *RCNN* (**Region-Based Convolutional Neural Network**) je typ modelu strojového učení, který se používá pro úlohy počítačového vidění, konkrétně pro detekci objektů. Tento přístup využívá ohraničující rámce napříč regiony objektů, které pak vyhodnocují konvoluční sítě nezávisle na všech regionech zájmu (`RoI`), aby klasifikovaly více oblastí obrazu do navrhované třídy. Architektura *RCNN*

byla navržena pro řešení úloh detekce obrazu. Tvoří základ *Mask R-CNN* a je zdokonalena do podoby, známou jako *Faster R-CNN*.

Faster R-CNN

Faster R-CNN je konvoluční neuronová síť založená na regionech, která vrací ohraničující rámce pro každý objekt a jeho značku třídy se skóre důvěryhodnosti. Architektura je složena ze dvou částí[2]:

1. Fáze – skládá se ze dvou sítí, páteřní (*ResNet*, *VGG*, *Inception* atd.) a sítě pro návrh regionu. Tyto sítě se spustí jednou na každý snímek a poskytnou sadu návrhů regionů. Návrhy regionů jsou oblasti v mapě prvků, které obsahují objekt.

2. Fáze – síť předpovídá ohraničující rámce a třídy objektů pro každou z navržených oblastí získaných v první fázi. Každá navržená oblast může mít různou velikost, zatímco plně propojené vrstvy v sítích vyžadují k předpovědím vždy vektor pevné velikosti. Velikost těchto navržených oblastí se stanoví buď pomocí metody `RoI`[13] `pool`[13] (která je velmi podobná metodě `MaxPooling`), nebo `RoIAlign`.

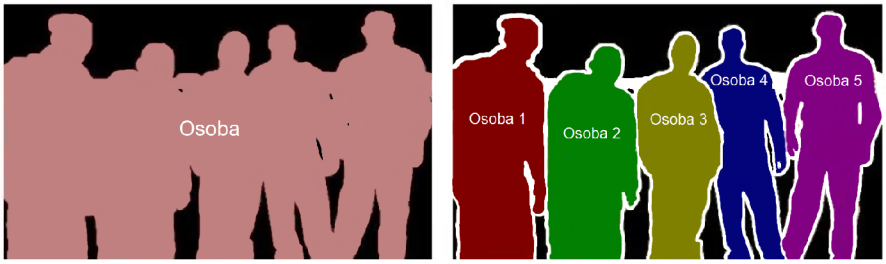
Ve druhé fázi je `RoI pool` nahrazen `RoIAlign`, který pomáhá zachovat prostorové informace, které jsou v případě `RoI pool` špatně zarovnány. `RoIAlign` používá binární interpolaci k vytvoření mapy prvků, která má pevnou velikost. Výstup z vrstvy `RoIAlign` je poté přiveden do maskovací hlavy, která se skládá ze dvou konvolučních vrstev. Ta generuje masku pro každý `RoI`, čímž segmentuje obraz způsobem pixel po pixelu. Viz obrázek 3.8

Mask R-CNN

Pod tuto metodu spadají dva typy obrazové segmentace[23]:

- **Sémantická** – zařazuje každý pixel do pevné sady kategorií, aniž by rozlišovala instance objektů. Jinými slovy, sémantická segmentace se zabývá identifikací/klasifikací podobných objektů jako jedné třídy z úrovně pixelu.

- **Instanční** – zabývá se správnou detekcí všech objektů v obraze a zároveň přesnou segmentací každé instance. Jedná se tedy o kombinaci detekce objektů, lokalizace objektů a klasifikace objektů. Tento typ segmentace tak jde ještě dále a jasně rozlišuje jednotlivé objekty klasifikované jako podobné instance.

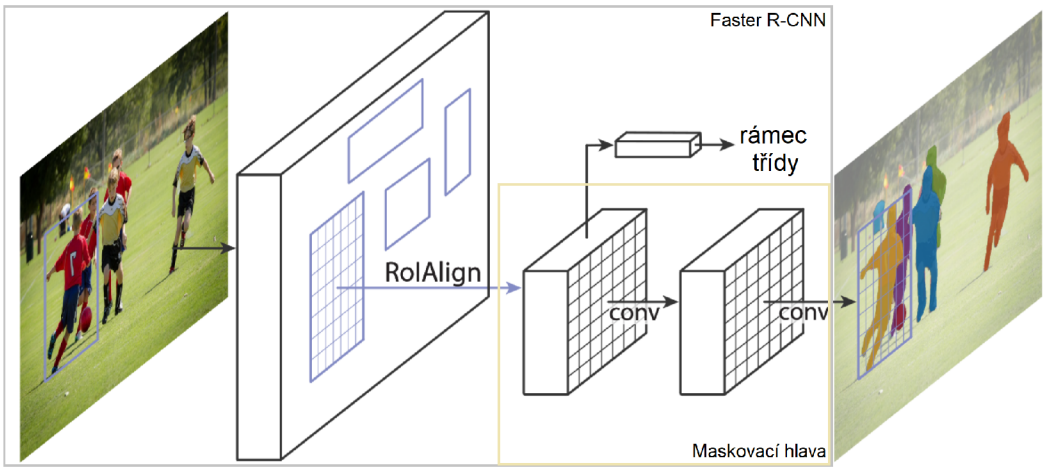


Obrázek 3.8: Sémantická segmentace a instanční segmentace [14]

[13] RoI – (Regions of Interest) regiony zájmu
[14] Obrázek 3.8 je převzat, upraven a přeložen z [23]

Jak již bylo řečeno *Mask R-CNN* je vytvořen s použitím *Faster R-CNN*. Zatímco *Faster R-CNN* má pro každý kandidátský objekt dva výstupy, označení třídy a posunutí ohraničujícího pole, *Mask R-CNN* je doplněn o třetí větev, která vypisuje masku objektu. Dodatečný výstup masky je odlišný od výstupů třídy a ohraničujícího rámečku a vyžaduje extrakci mnohem jemnějšího prostorového rozložení objektu. *Mask R-CNN* je tak rozšířením *Faster R-CNN* a funguje tak, že ke stávající větvi pro rozpoznávání ohraničujícího rámečku přidává paralelně větev pro předpovídání masky objektu (`RoI`).

Klíčovým prvkem je zarovnání mezi pixely, což je hlavní chybějící prvek *Fast/Faster R-CNN*. *Mask R-CNN* ale i tak používá stejný dvoustupňový postup s identickým prvním stupněm (*RPN*). Ve druhém stupni, souběžně s předpovídáním třídy a posunu políčka, *Mask R-CNN* také vypisuje binární masku pro každý `RoI`. To je velký rozdíl od většiny nejnovějších systémů, kde klasifikace závisí na předpovědi masky. *Mask R-CNN* usnadňuje širokou škálu flexibilních návrhů architektury díky tomu, že je jednoduchý na implementaci a má jednoduší trénovací proces než třeba *Faster R-CNN*. Navíc maskovací větev přidává pouze malou výpočetní režii, což umožňuje rychlý systém a tak i rychlé experimentování.



Obrázek 3.9: Framework Mask R-CNN pro segmentaci instancí [15]

3.4.2 Jednofázové algoritmy

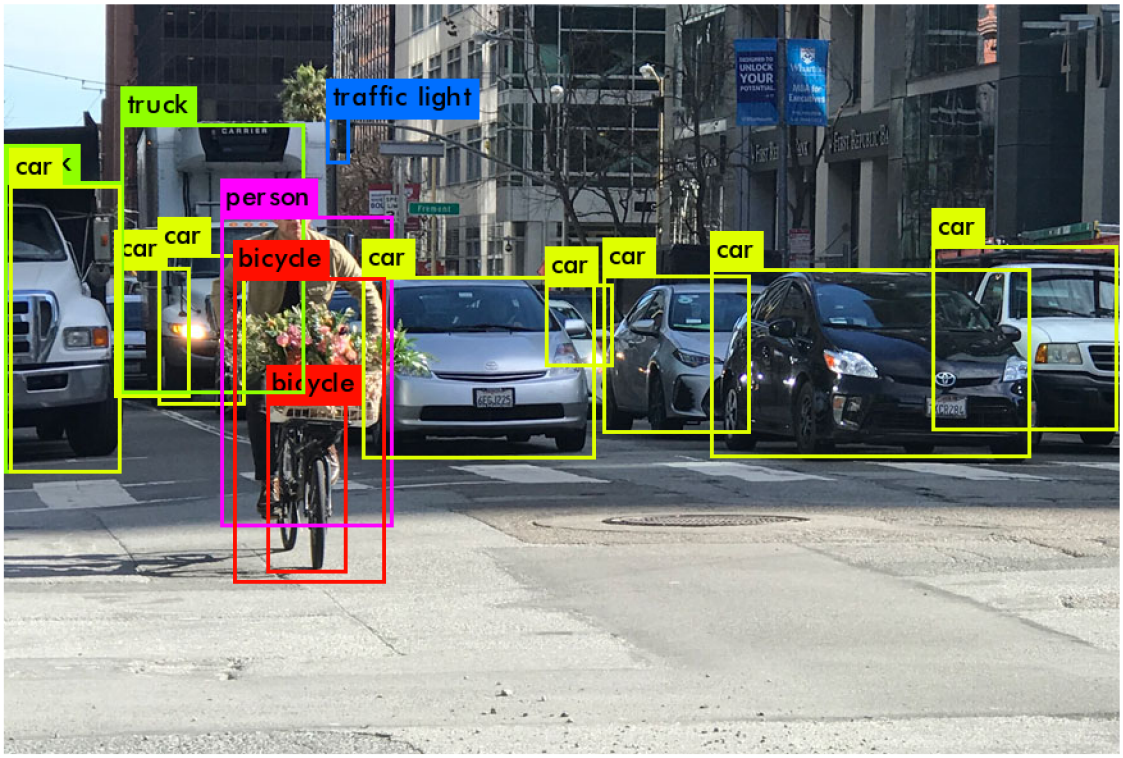
Jejich princip je jednoduší díky využití jednostupňových detektorů, protože pro předpovídaní ohraničujících boxů nepoužívají návrh oblasti. Díky tomu nejsou tak časově náročné a běžně se využívají při zpracování v reálném čase.

YOLOv3

You Only Look Once (*YOLO*) je algoritmus pro detekci objektů v reálném čase, který dokáže detekovat mnoho různých tříd objektů v jednom okamžiku. Algoritmus nabízí kvalitní detekci díky tomu, že používá rysy naučené pomocí hloubkových konvolučních neuronových sítí (deep convolutional neural network). *CNN* jsou systémy založené na klasifikátorech, mohou tak zpracovávat vstupní obrázky jakožto strukturovaná pole dat a identifikovat mezi nimi vzory. *YOLO* je tak mnohem rychlejší než jiné sítě při zachování přesnosti. Což umožňuje, že se model dívá na celý obrázek, takže jeho následné předpovědi jsou založeny na

[15] Obrázek 3.9 je převzat, upraven a přeložen z [14]

globálním kontextu obrázku. *YOLO* a další algoritmy konvolučních neuronových sítí tak hodnotí oblasti v obrázku na základě jejich podobnosti s předem definovanými třídami. Oblasti s vysokými hodnotami jsou označeny jako pozitivní objekt třídy, které je nejvíc objekt podobný. [22]



Obrázek 3.10: Příklad výstupu YOLOv3 algoritmu [16]

Algoritmus prvně obrázek rozdělí do mřížky a následně každá buňka odhadne počet ohraničujících rámců kolem objektů s nejvyšší hodnotou. Každému ohraničujícímu rámci je přiřazena příslušná hodnota důvěryhodnosti, které udává, jak přesná by předpověď měla být a detekuje tak pouze jeden objekt na daný rámec. Ohraničující rámce jsou generovány shlukováním rozměrů základních rámců z původní sady dat s cílem najít nejčastější tvary a velikosti.

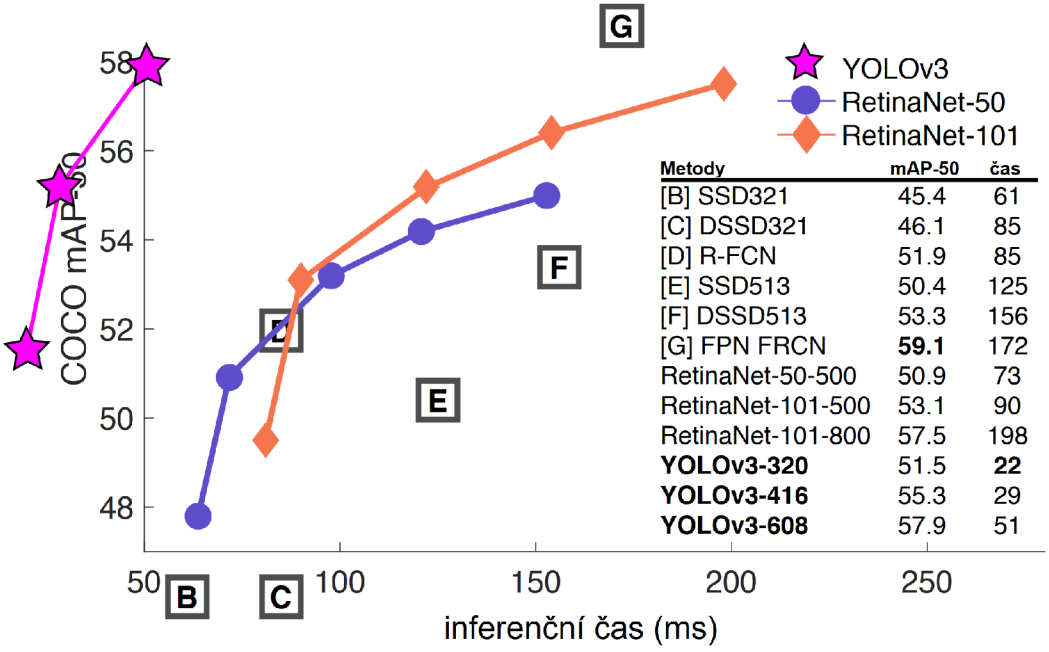
YOLOv3 používá, pro extrakci rysů, páteřní síť *Darknet-53*, zatímco verze 2 používala *Darknet-19*. Číslo v názvu znamená, že používá 53 konvolučních vrstev, což z něj dělá silnější nástroj než předchozí verze a zároveň efektivnější než konkurenční sítě, jakými jsou např. *ResNet-101* a *ResNet-152*. *YOLOv3* algoritmus je stejně přesný jako *RetinaNet-101* a zároveň je skoro 4x rychlejší. Z testovaných algoritmů není nejpřesnější, ale v poměru rychlost/přesnost je zaručeně nejlepší. Viz. graf 3.11. *YOLOv3* používá při trénování nezávislé logistické klasifikátory a binární ztrátu křížové entropie pro předpovědi tříd. Tyto úpravy umožňují používat pro trénování modelu složité datové sady, jako je například *Open Images Dataset*[17], ten obsahuje desítky překrývajících se názvů tříd, např. člověk a osoba.

YOLOv3 umožňuje, aby třídy byly konkrétnější a bylo jich více pro jednotlivé ohraničující rámce díky více názvovému přístupu. Zatím co *YOLOv2* používal `softmax`, což je

[16] Obrázek 3.10 je převzat z [27].
[17] Odkaz na data set zde: https://storage.googleapis.com/openimages/web/index.html

Páteřní síť	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19	74.1	91.8	7.29	1246	**171**
ResNet-101	77.1	93.7	19.7	1039	53
ResNet-152	**77.6**	**93.8**	29.4	1090	37
DarkNet-53	77.2	**93.8**	18.7	**1457**	78

Tabulka 3.1: Porovnání páteřních sítí. Top-1 a Top-5 – přesnost, Bn Ops – biliony operací, BFLOP/s – biliony operací s plovoucí desetinnou čárkou za sekundu [18]



Obrázek 3.11: Porovnání rychlosti s přesností různých algoritmů [18]

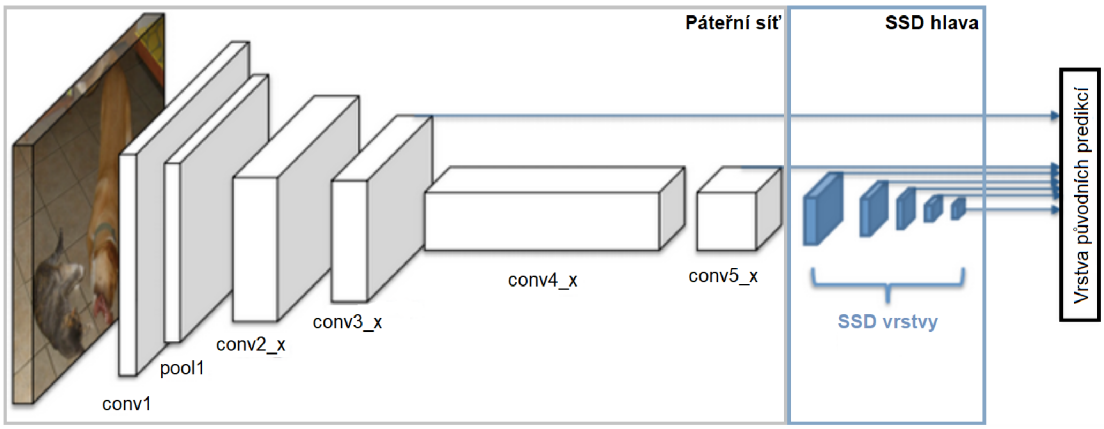
matematická funkce, která převádí vektor čísel na vektor pravděpodobností, kde jsou pravděpodobnosti jednotlivých hodnot úměrné relativnímu měřítku každé hodnoty ve vektoru. *YOLOv3* jej nepoužívá kvůli tomu, že každý ohraničující rámec může patřit pouze do jedné třídy, což né vždy platí.

[18] Tabulka 3.1 a obrázky 3.10, 3.11 jsou převzaty a přeloženy z [27].

Single Shot Detector

Ten dokáže, stejně tak jako *YOLOv3*, předpovídat více tříd objektů. Metoda detekuje objekty na snímcích pomocí jedné hloubkové neuronové sítě diskretizací výstupního prostoru ohraničujících rámců do sady výchozích rámců v různých poměrech stran a měřítkách na umístění mapy prvků. Detektor objektů generuje skóre pro přítomnost jednotlivých kategorií objektů v každém výchozím rámci a upravuje rámce tak, aby lépe odpovídal tvaru objektu. Síť také kombinuje předpovědi z více map prvků s různým rozlišením, aby si poradila s objekty různých velikostí. Detektor *SSD* lze snadno trénovat a integrovat do softwarových systémů, které vyžadují komponentu detekce objektů. Ve srovnání s jinými jednostupňovými metodami má *SSD* mnohem lepší přesnost, a to i při menších velikostech vstupního obrazu.[21]

SSD má dvě součásti: páteřní model a hlavu *SSD*. Páteřní model je obvykle předem natrénovaná síť pro klasifikaci obrazu jako extraktor příznaků. Obvykle se jedná o síť podobnou síti *ResNet* vycvičené na síti *ImageNet*, ze které byla odstraněna závěrečná plně propojená klasifikační vrstva. Zůstává tedy hloubková neuronová síť, která je schopna extrahovat sémantický význam ze vstupního obrazu při zachování prostorové struktury obrazu, i když v nižším rozlišení. Pro *ResNet34* je výsledkem páteřní sítě 256 příznakových map 7x7 pro vstupní obrázek. Hlava SSD je pouze jedna nebo více konvolučních vrstev přidaných k této páteři a výstupy jsou interpretovány jako ohraničující rámce a třídy objektů v prostorovém umístění aktivace konečných vrstev.

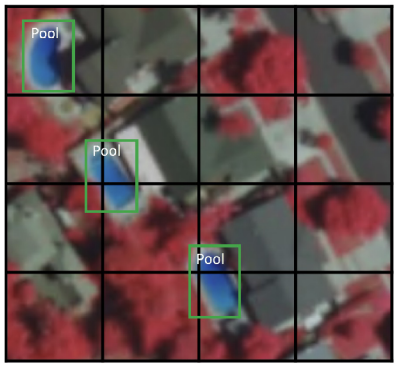


Obrázek 3.12: Architektura konvoluční neuronové sítě s SSD detektorem [19]

Důležité parametry metody *SSD*[4]:

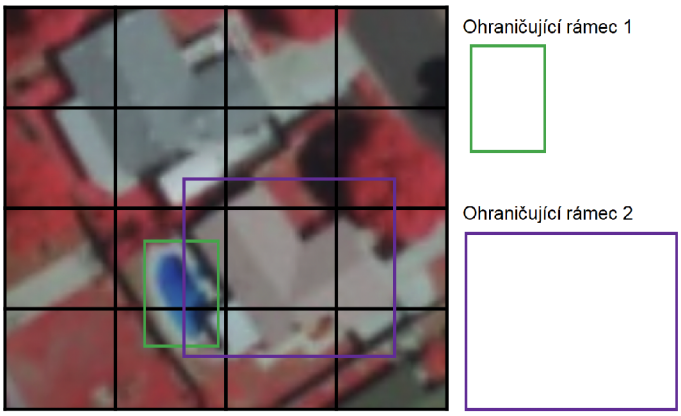
- **Buňka mřížky** – *SSD* rozděluje obraz pomocí mřížky a každá buňka mřížky je zodpovědná za detekci objektů v dané oblasti obrazu. Detekce objektů jednoduše znamená předpověď třídy a umístění objektu v dané oblasti. Pokud není přítomen žádný objekt, považujeme jej za třídu pozadí a umístění se ignoruje. Každá buňka mřížky je schopna vypsat polohu a tvar objektu, který obsahuje. Obr. 3.13a.
- **Ohraničující rámce** – Každé buňce mřížky lze přiřadit více ohraničujících/prioritních rámců. Tyto ohraničující rámce jsou předem definovány a každý z nich odpovídá za velikost a tvar v rámci buňky. *SSD* používá při trénování fázi porovnávání, kdy se odpovídající ohraničující rámce porovnávají s ohraničujícími rámci každého objektu

[19] Obrázek 3.12 je převzat a přeložen z [4].



(a) Ukázka mřížky 4x4

(b) Ukázka poměru stran



(c) Ukázka dvou ohraničujících rámců

Obrázek 3.13: Ukázka vybraných parametrů metody SSD [20]

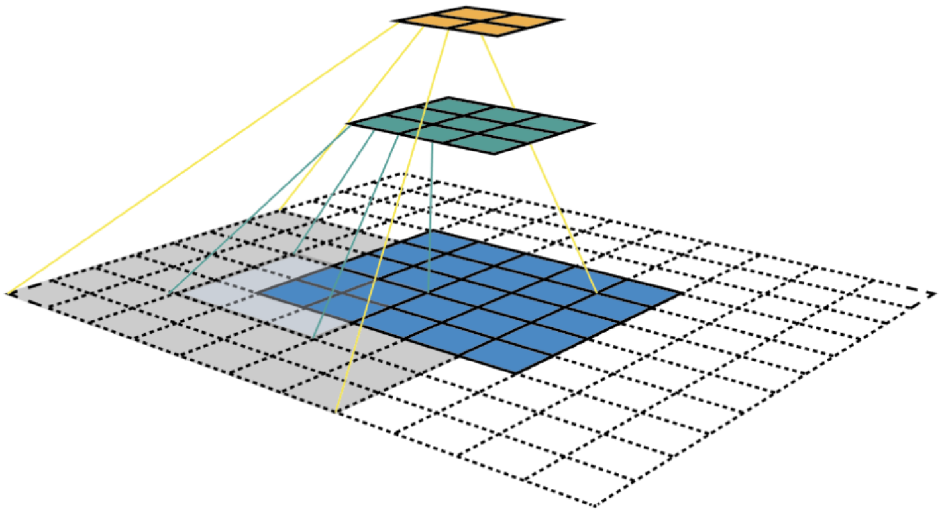
v obraze. Za rozpoznání třídy objektu a jeho umístění je v podstatě zodpovědný ohraničující rámec s nejvyšší mírou překrytí s objektem. Tato vlastnost se používá pro trénování sítě a pro předpovídání detekovaných objektů a jejich umístění poté, co byla síť natrénována. V praxi je každý ohraničující rámec specifikován poměrem stran a úrovní přiblížení. Obr. 3.13c.

- **Poměr stran** – Kvůli ne symetrii různých objektů, umožňuje architektura *SSD* předdefinovat poměry stran ohraničujících rámců, které to zohledňují. Parametr `ratios` lze použít k určení různých poměrů stran ohraničujících rámců přidružených ke každé buňce mřížky na každé úrovni přiblížení/měřítka. Obr. 3.13b.
- **Úroveň přiblížení** – Není nutné, aby ohraničující rámce měly stejnou velikost jako buňka mřížky. Může nás zajímat nalezení menších nebo větších objektů v buňce mřížky. Parametr `zooms` slouží k určení, jak moc je třeba ohraničující rámce zvětšit nebo zmenšit vzhledem ke každé buňce mřížky.

Nejdůležitější je však **Recepční pole**. To je definováno jako oblast ve vstupním prostoru, na kterou se dívá (tj. kterou ovlivňuje) určitá funkce/operace *CNN*. Vzhledem k operaci konvoluce představují rysy v různých vrstvách různě velké oblasti ve vstupním obraze. S postupující hloubkou se velikost reprezentovaná rysem zvětšuje.

[20] Obrázky 3.13a, 3.13b, 3.13c jsou převzaty a přeloženy z [4].

Recepční pole je hlavním předpokladem architektury *SSD*, protože umožňuje detekovat objekty v různých měřicích a vytvářet užší ohraničení. Například při použití mřížky 4x4 pro již zmíněný *ResNet34* (příznakové mapy 7x7), nejjednodušším přístupem je pouze aplikovat konvoluci na tuto mapu příznaků a převést ji na 4x4. Další krok, který *SSD* provádí, spočívá v tom, že na páteřní mapu prvků aplikuje více konvolučních vrstev a každá z těchto konvolučních vrstev má na výstupu výsledky detekce objektu. Protože dřívější vrstvy nesoucí menší receptivní pole mohou reprezentovat objekty menších rozměrů, předpovědi z dřívějších vrstev pomáhají při řešení objektů menších rozměrů. Z tohoto důvodu *SSD* umožňuje definovat hierarchii buněk mřížky v různých vrstvách. Můžeme například použít mřížku 4x4 pro vyhledávání menších objektů, mřížku 2x2 pro vyhledávání objektů střední velikosti a mřížku 1x1 pro vyhledávání objektů, které pokrývají celý obraz.



Obrázek 3.14: Příklad vizualizace map funkcí CNN a receptivního pole [21]

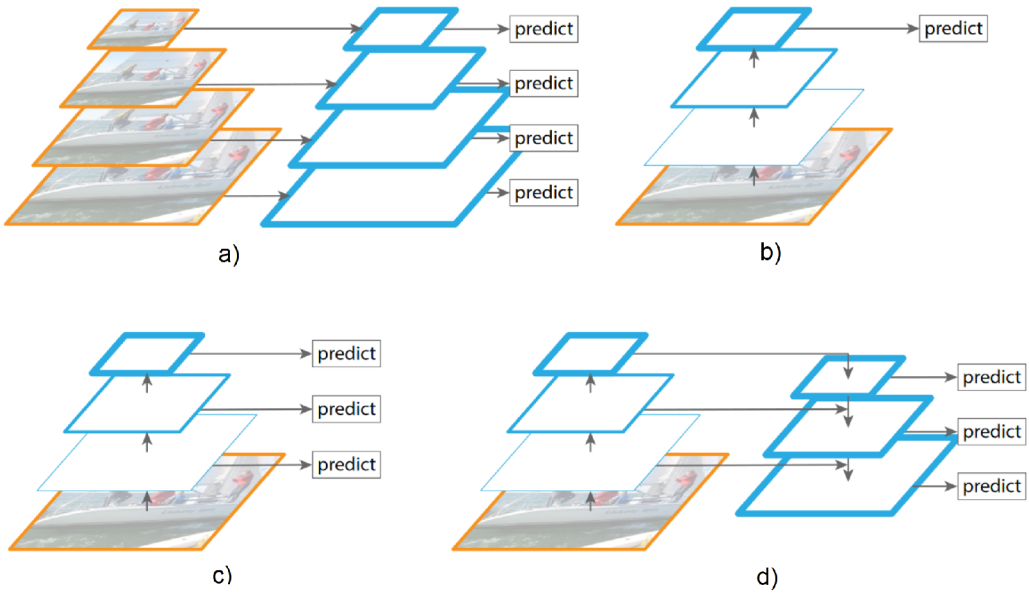
Na příkladu, znázorněném obrázkem 3.14, začínáme spodní vrstvou (5x5) a poté aplikujeme konvoluci, jejímž výsledkem je střední vrstva (3x3), kde jeden rys (zelený pixel) představuje oblast 3x3 vstupní vrstvy (spodní vrstva). A po aplikaci konvoluce na střední vrstvu a získáme horní vrstvu (2x2), kde každý rys odpovídá oblasti 7x7 na vstupním obrázku. Tento druh zeleného a oranžového 2D pole se také nazývá mapa příznaků, která označuje sadu příznaků vytvořenou použitím stejného extraktoru příznaků na různých místech vstupní mapy v rychlém posuvném okně. Rysy ve stejné mapě rysů mají stejné receptivní pole a hledají stejný vzor, ale na různých místech. Tím vzniká prostorová invariance sítě *ConvNet*.

[21] Obrázek 3.14 je převzat a přeložen z [4].

RetinaNet

Je jedním z nejlepších jednostupňových modelů detekce objektů, který se osvědčil při práci s hustými a malými objekty. Z tohoto důvodu se stal oblíbeným modelem detekce objektů, který se používá s leteckými a satelitními snímky. Vznikl na základě dvou vylepšení oproti stávajícím jednostupňovým modelům detekce objektů - *Feature Pyramid Networks* (FPN) a *Focal Loss*.

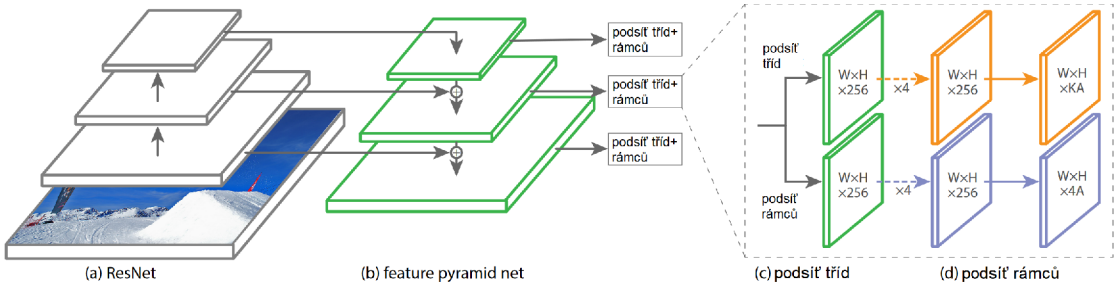
Feature Pyramid Networks jsou pyramidy prvků postavené na obrazových pyramidách. To znamená, že se vezme obraz a rozdělí se na snímky s nižším rozlišením a menší velikostí (čímž se vytvoří pyramida). Z každé vrstvy pyramidy se pak extrahují ručně vytvořené rysy, které slouží k detekci objektů. Díky tomu je pyramida měřítkově invariantní. Tento proces je však náročný na výpočetní výkon a paměť. S příchodem hlubokého učení byly tyto ručně konstruované rysy nahrazeny *CNN*. Později byla samotná pyramida odvozena z vlastní pyramidové hierarchické struktury CNN. V architektuře *CNN* se velikost výstupních map příznaků po každém následujícím bloku konvolučních operací zmenšuje a vytváří pyramidální strukturu. [3]



Obrázek 3.15: Na tomto obrázku jsou mapy příznaků označeny modrými obrysy a silnějšími obrysy označují sémanticky silnější rysy. [22]

Focal Loss (FL) je vylepšením *Cross-Entropy Loss* (CE) a je zaveden pro řešení problému nevyváženosti tříd u jednostupňových modelů detekce objektů. Ty trpí problémem extrémní nevyváženosti tříd popředí a pozadí v důsledku hustého vzorkování ohraničujících rámců. V síti *RetinaNet* mohou být v každé pyramidové vrstvě tisíce ohraničujících rámců, avšak pouze několik z nich bude přiřazeno pravdivému objektu, zatímco naprostá většina bude patřit do třídy pozadí. Tyto snadné příklady (detekce s vysokou pravděpodobností), ačkoli vedou k malým hodnotám ztrát, mohou model dohromady zahltit. Funkce *Focal Loss* snižuje vliv ztrát ze snadných příkladů a zvyšuje důležitost opravy chybně klasifikovaných příkladů.

[22] Obrázek 3.15 i s popisem převzat z [18]



Obrázek 3.16: RetinaNet využívá páteřní síť Feature Pyramid Network (FPN) na vrcholu architektury ResNet (a), která vytváří bohatou, více škálovou konvoluční pyramidu (b). K této páteři RetinaNet připojuje dvě podsítě, jednu pro klasifikaci ohraničujících rámců (c) a druhou pro regresi z ohraničujících rámců do rámců pravdivých objektů (d). [23]

[23] Obrázek 3.16 i s popisem převzat z [3]

Kapitola 4

Návrh

4.1 Technologie

4.1.1 OpenCV - knihovna pro zpracování obrazu

Open source Computer Vision (*OpenCV*)[11] je knihovna pro analýzu obrázků a videí. Původně představena před více než 20 lety firmou Intel. Dnes se jedná o jednu z nejpoužívanějších knihoven na zpracování obrazu a to i díky její jednoduché dostupnosti v jazycích C++ a Python. Volba padla na tuto knihovnu kvůli její snadné dostupnosti, optimalizaci a široké nabídce funkcí. Pro mě spočívá její hlavní využití v modulu *DNN* (Deep Neural Networks), který mi umožní snadnou a rychlou implementaci *YOLOv4* algoritmu a Darknetu. Implicitně běží tento modul na procesoru, což je ovšem silně výpočetně náročné a může až zastavit běh systému. Takže je značně nespolehlivý. Backend *DNN* modulu se dá ovšem přepnout na jiná běhová prostředí. Já využiji konkrétního a to *CUDA*, jenž je dostupný pouze pro grafické karty NVIDIA. Přepnutím na tento backend se rychlost výpočtů zvětší násobně, pro některé operace až 20x. A navíc nehrozí riziko pádu systému.

4.1.2 YOLOv4

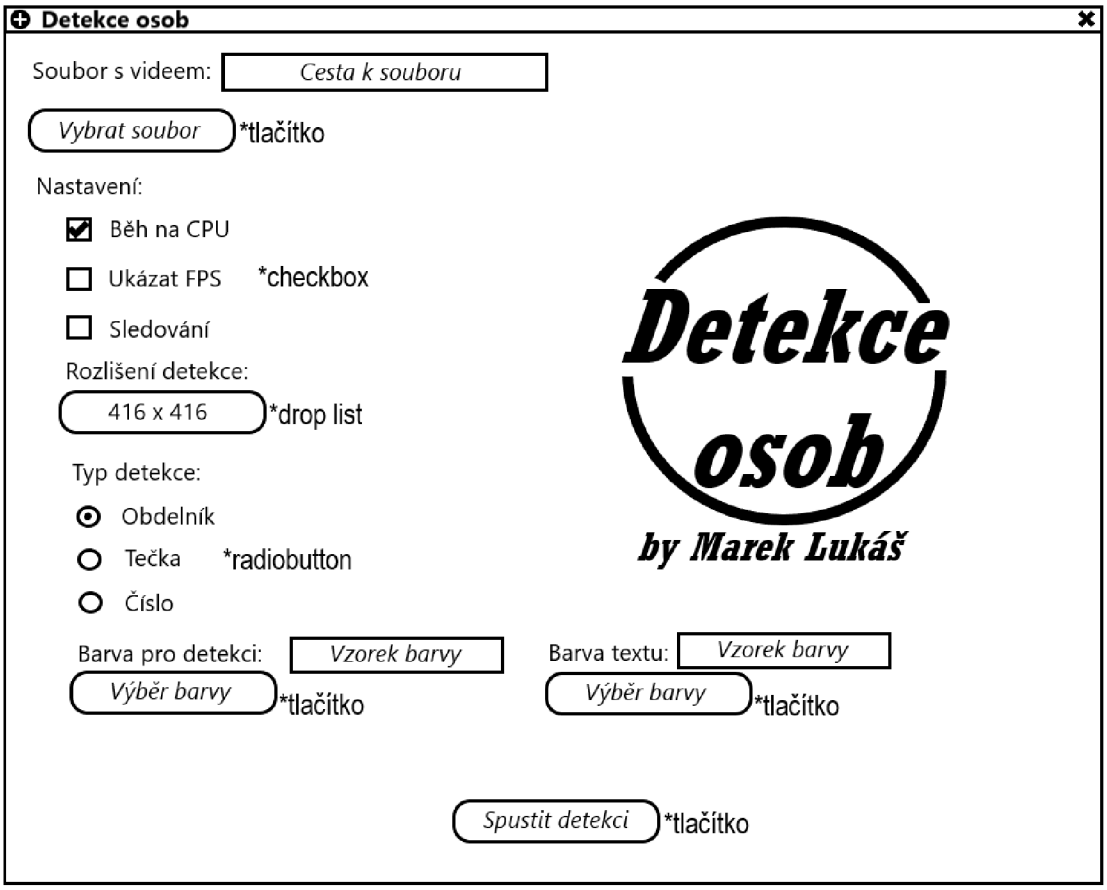
Jakožto detekční algoritmus jsem si vybral jeden z nejmodernějších, *YOLOv4* [8]. Jedná se o rychlejší a přesnější *YOLOv3*, takže popsaná teorie spadá i na tento model. Volba padla na tento model taky díky jednoduché implementaci v rámci *OpenCV*. Tato verze je taky v poměru výkon/přesnot jeden z nejlepších. *YOLOv4* obsahuje před trénované váhy pro detekci až 80 tříd, jedna z nich je i osoba, což je cíl mé práce. Takže implementace je více než přímočará. Samozřejmě v základu pro rozlišení 416x416 je docela nepřesný, ovšem s narůstajícím rozlišením, narůstá i přesnost, ale na oplátku klesá rychlost. Více v sekci testování.

4.2 Hlavní program

Hlavní program bych rozdělil na dvě části a zároveň do dvou Python skriptů. Samostatný skript na aplikaci a samostatný skript na detekci spolu se sledováním.

4.2.1 První část - UI/vstup

Aby se skript nespouštěl přes terminál, se složitým zadáváním parametrů, rozhodl jsem se vytvořit menší aplikaci s jednoduchým uživatelským rozhraním.



Obrázek 4.1: Finální návrh rozložení uživatelského rozhraní

Aplikace bude obsahovat širokou možnost konfigurace detekce. Samotné části se dají rozdělit do 6 sekcí:

- Vložení souboru
- Možnosti detekce
- Rozlišení detekce
- Typ detekce
- Barva
- Spuštění detekce

Vložení souboru

První bude pole pro vložení souboru. Pomocí tlačítka *Vybrat soubor* by se mělo otevřít okno a nechat uživatele vybrat soubor s videem. Po vybrání videa se cesta k němu zobrazí v okně *Cesta k souboru* pro kontrolu.

Možnosti detekce

- Běh na CPU - implicitně bude detekce běžet přes *CUDA* rozhraní na grafické kartě NVIDIA. Aby toto bylo možné musí být nainstalován *CUDA toolkit* s *cuDNN* rozšířením. Taky je potřeba sestavit OpenCV s podporou *CUDA*. Což je značně omezující, proto zde bude možnost, přepnutí na CPU. To je ovšem mnohem pomalejší a může to i vést k nestabilitě systému, ale dá se to tak spustit v podstatě na jakémkoliv systému.
- Ukazatel FPS - V levém horním rohu se zobrazí ukazatel snímků za sekundu, aby uživatel viděl, jak rychle daná detekce probíhá.
- Sledování - Zapíná sledování osob. Sledování bude implementováno pomocí identifikátoru. Každé osobě je přiděleno identifikační číslo.

Rozlišení detekce

Kliknutím na drop list by se mělo zobrazit na výběr několik rozlišení. Jelikož jsou připuštěny pouze hodnoty dělitelné 32, tak nebude možné zadávat vlastní hodnoty.

Typ detekce

Detekční model vrátí rámcový box a zbývá na mě jak s tím naložit. Vybral jsem tři možnosti

- Obdélník - osoba bude ohraničena obdélníkem a pokud je sledování aktivní, číslo je umístěno v levém spodním rohu.
- Tečka - místo obdélníku je osoba označena tečkou, která je vykreslena uprostřed boxu. V případě sledování je číslo umístěno v této tečce.
- Číslo - bude dostupné pouze pro sledování (bez sledování nebude číslo/ID dostupné). Číslo bude opět umístěno doprostřed boxu.

Barva

Dalším jednoduchým přizpůsobením detekce je nastavení vlastní barvy. To se provede kliknutím na tlačítko *Výběr barvy* a měla by se zobrazit barevná paleta pro výběr barvy. Po vybrání se pole *Vzorek barvy* obarví danou barvu, jakožto indikace vybrání barvy. Uživatel může nastavit jak barvu ohraničení, tak barvu textu.

Spuštění detekce

A na konec tlačítko pro samotné spuštění detekce. Na pozadí proběhne kontrola správnosti vstupu a spustí se detekce.

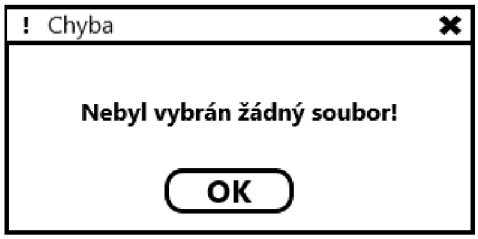
Zpětná vazba

Zpětná vazba v podobě vyskakovacích oken jako u klasických systémů *Windows* a interaktivita.

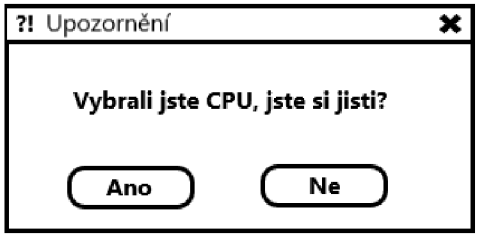
- Výstražné okno - při pokusu o spuštění detekce bez vybraného souboru by mělo vyskočit okno s výstrahou o nevybrání souboru, viz obrázek 4.2.

[24] box označuje oblast, kde se podle detekčního modelu nachází osoba.

- Informační okno - při přepnutí na CPU by mělo vyskočit okno upozorňující uživatele na možné problémy se stabilitou, viz obrázek 4.3. Také mu dá na výběr, zda si je s volbou jistý. Pokud klikne na *Ne* tak se volba CPU, smaže.
- Sledování - při zakliknutí možnosti sledování se zpřístupní možnost čísla v typu detekce. Jinak je možnost vypnuta (nedává smysl, když se nikdo nesleduje)



Obrázek 4.2: Návrh chybového vyskakovacího okna



Obrázek 4.3: Návrh informačního vyskakovacího boxu

Návrh vyskakovacích oken je pouze ilustrační a bude nejspíš odpovídat systémovému nastavení.

4.2.2 Druhá část - detekce a trasování

Obě části budou umístěny v jednom skriptu a nejspíše se bude jednat o samostatné funkce.

Detekce

Jak jsem popsal výše, k detekci použiji nejmodernější detekční algoritmus a to *YOLOv4*. Je více způsobů jak se chopit implementace (cloud, PyTorch, CPU, GPU). Jelikož jsem vlastníkem výkonného počítače, tak jsem se rozhodl provést lokální implementaci s tím, že veškeré výpočty budou probíhat na grafické kartě. Rozhodl jsem se pro použití *DNN* modulu knihovny *OpenCV* se kterým lze *YOLOv4* algoritmus lehce sestavit.

Sledování

Sledování lze provést více způsoby, ale společný prvek mají všechny stejný - odlišit jednotlivé osoby od sebe. Já si vybral identifikátory. Po dobu působení osoby na videu, bude mít daná osoba u sebe číslo, které by se nemělo měnit.

Ve zkratce bude sledování implementováno porovnáváním pozic osob mezi jednotlivými snímky. Podrobnější postup sledování:

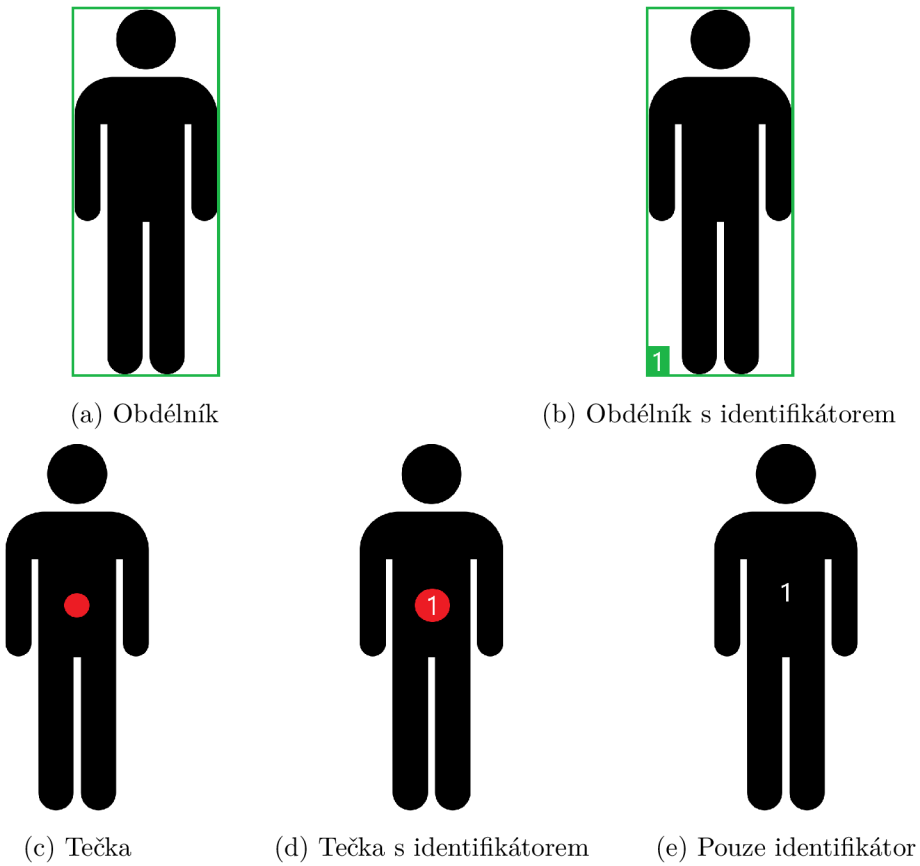
1. Získání pozic osob ze snímku

2. Pro každou osobu zjišťuji, zda už na snímku byla či nikoliv
   (a) Vypočítání vzdálenosti pozice osoby od možné pozice jejího předchozího pobytu
   (b) Získání nejmenší vzdálenosti
   (c) Podle nejmenší vzdálenosti získání pozice a příslušného identifikátoru
3. Osobě je přidělen identifikátor
4. Uložení aktuálních pozic a opakování od kroku 1

Vizualizace

Výsledkem detekce jsou pozice všech detekovaných objektů a na mně je pak jejich vizualizace. Zvolil jsem tři typy vizualizace (viz obrázek 4.4) osob a to:

- Obdélník - osoba bude ohraničena obdélníkem a pokud je sledování aktivní, číslo je umístěno v levém spodním rohu.
- Tečka - místo obdélníku je osoba označena tečkou, která je vykreslena uprostřed jejího ohraničujícího rámce. V případě sledování je číslo umístěno v této tečce.
- Číslo - identifikátor bude opět umístěno doprostřed rámce.



(a) Obdélník (b) Obdélník s identifikátorem

(c) Tečka (d) Tečka s identifikátorem (e) Pouze identifikátor

Obrázek 4.4: Vizualizace osob

Kapitola 5

Implementace

Implementace se ve většinově podobá návrhu a nic důležitého nebylo změněno či odstraněno. Rychlý přehled použitých technologií:

- *Windows* - operační systém stroje na kterém byl program testován
- *Python 3.10* - použitý programovací jazyk
- *OpenCV 4.5.5* - použitá knihovna pro detekci a zobrazení výsledků, obzvláště pak její DNN modul
- *TKinter* - vestavěná knihovna Pythonu pro tvorbu aplikačního rozhraní
- *YOLOv4* - použitý detekční model
- *CUDA* - použitý backend pro *DNN* modul

5.1 Instalace prerekvizit

Předtím, než mohu začít psát samotný program, musím ovšem nainstalovat několik programů, které mi zpřístupní moji volbu detekce. Seznam prerekvizit a odkazy na jejich stáhnutí:

- *CUDA toolkit* - link
- *NVIDIA CuDNN* - link (pro stáhnutí je potřeba přihlášení)
- *Python 3.10* - link
- *OpenCV 4.5.5* - link
- *OpenCV Contrib* - link
- *CMake* - link
- *Visual Studio 2019* - link

CUDA toolkit

Jedná se o vývojové rozhraní od společnosti NVIDIA pro tvorbu vysoce výkonných aplikací akcelerovaných grafickým procesorem. Stačí stáhnout a nainstalovat.

NVIDIA CuDNN

Jedná se o GPU akcelerovanou knihovnu primitiv pro hluboké neuronové sítě. *cuDNN* poskytuje vysoce vyladěné implementace standardních postupů, jako je dopředná a zpětná konvoluce, sdružování, normalizace a aktivační vrstvy. V mém případě ji potřebuji, abych s její pomocí mohl sestavit *OpenCV DNN* modul. Je důležité stáhnout verzi, která je kompatibilní s verzí *CUDA toolkit*u. Instalace je přímočará, stačí přetáhnout soubory ze složek v adresáři *CuDNN* do stejných složek v adresáři *CUDA toolkit*u.

Python 3.10

Jeden z nejrozšířenějších programovacích jazyků na světě. Před instalací je lepší se zbavit jakýchkoliv jiných nainstalovaných Pythonu.

OpenCV 4.5.5

OpenCV (Open Source Computer Vision Library) je softwarová knihovna s otevřeným zdrojovým kódem pro počítačové vidění a strojové učení. Je potřeba stáhnout zip soubor se zdrojovými kódy, aby bylo možné sestavení této knihovny.

OpenCV Contrib

Je potřeba stáhnout také repositář s rozšiřujícími moduly pro *OpenCV*.

CMake

CMake je multiplatformní rodina nástrojů s otevřeným zdrojovým kódem určená k sestavování, testování a balení softwaru. *CMake* slouží k řízení procesu kompilace softwaru pomocí jednoduchých konfiguračních souborů nezávislých na platformě a kompilátoru a ke generování nativních souborů makefile a pracovních prostorů, které lze použít v prostředí kompilátoru dle vlastního výběru. Tento program je potřebný pro vlastní sestavení *OpenCV*.

Visual Studio 2019

Program pro úpravu kódu od společnosti Microsoft. Je potřebný pro instalaci sestavené knihovny *OpenCV*.

5.2 Sestavení

Sestavení je dosti přímočaré a jednoduché. Prvně musím přidat složku se zdrojovými soubory knihovny *OpenCV*. Dále složku *build* do které se daná knihovna sestaví (tu si vytvořím kde chci). Započnu konfiguraci a preferovanou platformu si vyberu x64. Kliknu na *finish* a počkám na dokončení prvotní konfigurace. Jakmile je dokončena zobrazí se mnoho možností, které můžu nastavit. Já potřebuji označit následující:

- *ENABLE_FAST_MATH*
- *WITH_CUDA* - povolím sestavení s *CUDA* rozhraním a při opětovné konfiguraci se nám zpřístupní následující možnosti.

- *CUDA_ARCH_BIN* - jedná se o nastavení výpočetní kompatibility. Je možné ji nechat v původním nastavení, ale specifikace konkrétní třídy zrychlí samotné sestavení. V mém případě se jedná o verzi 8.6
- *CUDA_FAST_MATH*
- *OPENCV_DNN_CUDA* - povolím sestavení *DNN* modulu zapomoci *CUDA* rozhraní, klíčová možnost pro celkovou funkčnost
- *OPENCV_EXTRA_MODULES_PATH* - je také potřeba propojit základní knihovnu *OpenCV* s rozšiřujícími moduly. Zde dám cestu ke složce *modules* nacházející se uvnitř složky *OpenCV Contrib*.

V nastavení je možnost i sestavit testy, ty se však ukázali plné chyb a zabraňovali dokončení instalace a navíc prodlužují dobu instalace, proto tyto možnosti odznačím:

- *BUILD_PERF_TESTS*
- *BUILD_TESTS*
- *BUILD_opencv_python_tests*
- *INSTALL_TESTS*
- *OPENCV_GENERATE_SETUPVARS* - tuto možnost není třeba odstraňovat, ale vyhazuje zbytečnou chybu na výstup.

Po vyplnění předešlých možností opět spustím konfiguraci. Při správném vyplnění by neměla být žádná možnost červená. Pokud není, můžu bezpečně vygenerovat soubory pro instalaci stiskem tlačítka *Generate*. Nyní mám generaci souborů hotovou a můžu zavřít *CMake* a přesunout se do *Visual Studia*.

V něm otevřu soubor *OpenCV.sln*, který je umístěn ve složce *build*, do které jsem v předchozím kroku tento soubor vygeneroval. Po načtení otevřu složku *CMakeTargets* a kliknu na *ALL_BUILD*. Dále změním typ sestavení z Debug na Release a nechám to sestavit. Sestavení trvá několik desítek minut. Jakmile je dokončeno zkontroluji, že se vše sestavilo úspěšně a pokud ano, pokračuji. Nyní kliknu na *INSTALL* a spustím sestavení. Tentokrát sestavení trvá chvilku. Při úspěšném sestavení mám knihovnu *OpenCV* nainstalovanou s podporou *CUDA* rozhraní a můžu se pustit do cíle této práce – detekce osob.

5.3 Aplikace a UI

Uživatelské rozhraní jsem na-implementoval téměř přesně podle návrhu. Využil jsem k tomu vestavěnou knihovnu Pythonu *TKinter*. Implementace je provedena ve skriptu *app.py*. Skript je složen ze třech hlavních částí:

- `main` funkce
- Pomocné funkce:
  - `select_file`
  - `choose_color`

  - `choose_text_color`
  - `cpu_warning`
  - `tracking_check`
  - `check_settings`
- Třída `ToolTip`

Obrázek 5.1: Výsledná podoba aplikace

5.3.1 Main funkce

V `main` funkci se nachází veškeré rozmístění jednotlivých prvků aplikace. Aplikace má pevnou velikost 700x750 pixelů a nelze ji, za běhu, měnit velikost. Použité prvky z knihovny TKinter:

- `Label` - veškerý samostatný text je zobrazen přes tento prvek
- `Button` - prvek pro vytvoření tlačítka. Funkce, kterou tlačítko při stisknutí zavolá, se předá pomocí parametru `command`.
- `Checkbutton` - prvek pro vytvoření zaškrtávacích polí. Přes parametr `variable` je potřeba předat speciální *TKinter* proměnnou `IntVar()` do které se ukládá vybraná hodnota (on = 1 / off = 0).
- `OptionMenu` - prvek pro vytvoření vyskakovacího menu rozlišení.
- `RadioButton` - prvek pro vytvoření výběrových tlačítek.
- `Frame` - tento prvek slouží k zobrazení vybrané barvy. Má pevnou délku, šířku a mění se pozadí dle výběru barvy.

Všechny prvky jsou pevně umístěny v okně pomocí vestavěné funkce `place(x=X, y=Y)`. Pro větší pohodlí uživatele, obsahuje funkce ještě kontolu přítomnosti *CUDA toolkit*u. Pokud není nalezen, je možnost běhu na CPU zaškrtnuta a její vypnutí je zakázáno.

5.3.2 Pomocné funkce

`select_file`

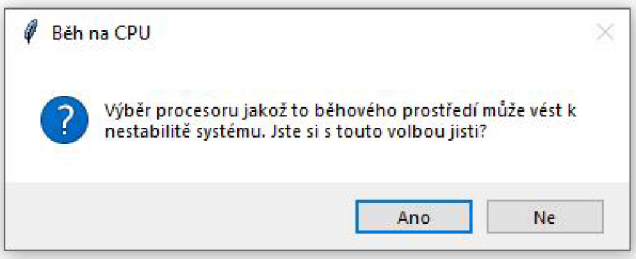
Funkce slouží ke vložení souboru. Pomocí vestavěné funkci `askopenfilename` zobrazí dialogové okno pro výběr souboru. Nevyberu-li žádný soubor nic se nestane, jinak se vypíše cesta k vybranému souboru.

`choose_color a choose_text_color`

Funkce slouží k vybrání barev. Pomocí vestavěné funkci `askcolor` zobrazí paletu barev s možností výběru. Při vybrání barvy se přebarví daný `Frame` na vybranou barvu.

`cpu_warning`

Při zakliknutí možnosti běhu na CPU se zobrazí vyskakovací okno 5.2 pomocí vestavěné funkce `messagebox.askyesno()`. Jestliže kliknu na *ne* tak se volba zruší.



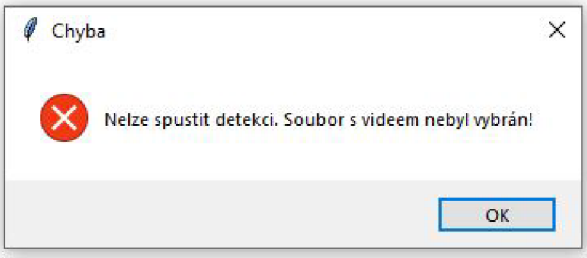
Obrázek 5.2: Výsledná podoba informačního vyskakovacího okna

`tracking_check`

Funkce kontroluje zda je zvolena možnost sledování a podle toho zpřístupňuje volbu čísla (co by typu detekce) a barvu textu (bez sledování není text => není potřeba barva). To je provedeno vestavěnou funkcí `configure(state=ACTIVE/DISABLED)`.

`check_settings`

Funkce je zavolána při spuštění detekce. První je kontrola vybrání souboru. Pokud soubor není vybrán vyskočí chybové okno 5.3 pomocí vestavěné funkce `messagebox.showerror()`. Když je soubor vybrán dojde ke složení příkazu. Příkaz se skládá ze samostatných řetězců podle vybraných parametrů detekce. Složený příkaz se pak spustí pomocí funkce `system("příkaz")` z knihovny `os`.



Obrázek 5.3: Výsledná podoba chybového vyskakovacího okna

5.3.3 Třída `ToolTip`

Třída slouží k tomu, když najedu ukazatelem na nějaký `Label` tak se zobrazí nápověda, co dané nastavení dělá. Kód pro tuto třídu byl převzat z [29].

5.4 Detekce a sledování

Detekce a sledování jsem obě na-implementoval do skriptu `detect.py`.

5.4.1 Detekce

Jak jsem napsal výše pro detekci využívám knihovny `OpenCV` a algoritmu `YOLOv4`. Samotná detekce se dá rozdělit do čtyř kroků.

První krok

Prvním krokem je složení detekčního modelu a k tomu slouží moje funkce `setup_detection`. Nejprve musím příkazem `cv.dnn.readNetFromDarknet("cfg", "weight")` načíst konfigurační soubory sítě v `Darknet` formátu. Dále musím nastavit backend pomocí příkazu `cv.dnn.setPreferableBackend()` a cíl pomocí příkazu `cv.dnn.setPreferableTarget()`. Implicitně se backend nastaví na `CUDA` následovně:

- `cv.dnn.setPreferableBackend(cv.dnn.DNN_BACKEND_CUDA)`

- `cv.dnn.setPreferableTarget(cv.dnn.DNN_TARGET_CUDA_FP16)`

V případě vlastní volby nebo nedostupnosti grafické karty, ovladačů apod. se nastaví na procesor následovně:

- `cv.dnn.setPreferableBackend(cv.dnn.DNN_BACKEND_DEFAULT)`
- `cv.dnn.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)`

Na závěr složení pak získám detekční model příkazem `cv.dnn_DetectionModel(síť)` a pomocí `setInputParams(scale=1/255, size="rozlišení detekce", ...)` nastavím parametry tohoto modelu. Jediné co já nastavuji je rozlišení detekce podle vstupu. Funkce nakonec vrátí připravený detekční model.

Druhý krok

Nyní mám detekční model a mohu tak začít se samotnou detekcí voláním své funkce `detect_persons(detekční_model)`. Jako první je otevření vybraného videa pro čtení pomocí funkce `cv.VideoCapture(soubor)`, dále nastavení pár pomocných proměnných a následuje cyklus `while` s `True` podmínkou takže se sám nikdy neukončí.

V tomto cyklu prvně přečtu snímek pomocí funkce `otevřený_soubor.read()`. Ta mi vrátí jednak návratovou hodnotu (jestli byl snímek přečten či bylo dosaženo konce videa) a druhak jeden snímek videa. Po kontrole zda neskončilo video dojde k samotné detekci pomocí funkce `model.detect(snímek, confThreshol=0.3, nmsThreshold=0.4)`. Tato funkce vrací tři hodnoty:

- Identifikátor třídy - číslo reprezentující třídu detekovaného objektu (v mém případě nulu = člověka)
- Důvěryhodnost - to je hodnota s jakou jistotou si je model jistý, že objekt je daná třída
- Rámce - ten se skládá ze čtyř hodnot takto (x, y, šířka, délka), kde:
  1. x - je pozice pixelu na ose x
  2. y - je pozice pixelu na ose y
  3. šířka - je délka objektu na ose x
  4. délka - je délka objektu na ose y

Následuje jednoduchý `for` cyklus ve kterém si vyberu pouze lidi a jejich rámce uložím do nového pole.

Třetí krok

Třetí krokem je další `for` cyklus, který prochází všechny osoby, které jsem si v předchozím kroku vyfiltroval. V něm probíhá i sledování, ale o tom více v další sekci. Pokud osoby sledujeme nebo ne, stejně je musíme na snímku vyznačit. To provedu jednoduše několika funkcemi podle toho, co chci vykreslit:

- `cv.rectangle(snímek, (bod1), (bod2), barva, hranice)` - nakreslí obdélník jakožto rámec od bodu 1 po bod 2

- `cv.circle(snímek, (bod), poloměr, barva, výplň)` - nakreslí kruh se středem v bodu
- `cv.putText(snímek, text, (bod), font, velikost, barva)` - nakreslí text s počátkem na zadaném bodu

Čtvrtý krok

Nyní mám všechny detekované osoby vyznačeny a zbývá jen snímek ukázat na výstupu. To provedu pomocí funkce `cv.imshow("Název okna", snímek)`. Každý snímek je pomocí funkce `cv.waitKey(1)` ukázán na 1ms (okno se samo nezavře) a následně započne vytváření dalšího snímku, takže detekce probíhá tak rychle, jak dokáže grafická karta stíhat. Kromě toho lze kdykoliv během zobrazování stisknout q a detekci tak ukončit. Po zobrazení celého videa už zbývá jen otevřený soubor uzavřít funkcí `soubor.release()`, pomocí funkce `cv.destroyAllWindows()` zavřít okno a detekce skončila.

5.4.2 Sledování

Sledování je implementováno převážně ve funkci `track_person()` se dvěma pomocnými funkcemi `get_closest_point_from_active_positions` a `create_new_index_position`.

`track_person`

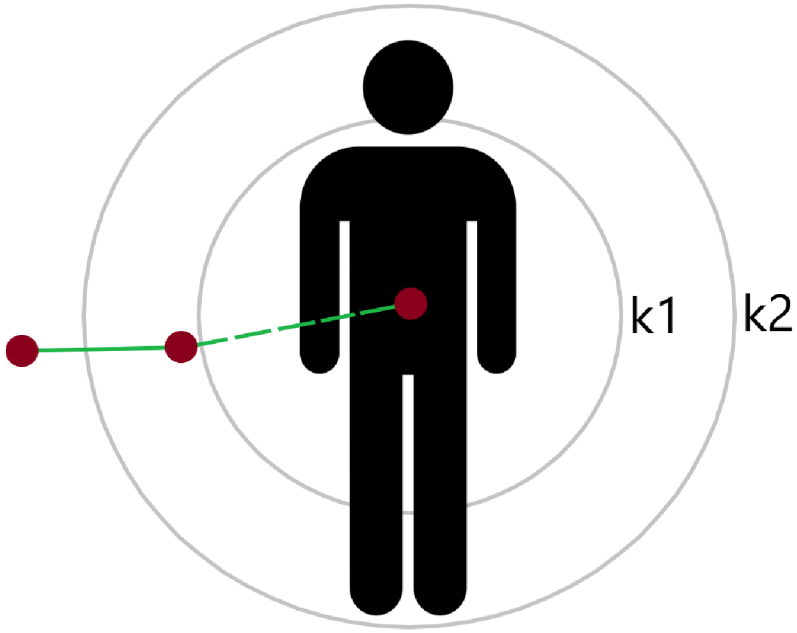
Funkce je volána na počátku třetího kroku detekce a má dva vstupní parametry:

- `coordinates` - souřadnice aktuálně detekované osoby
- `last_frame_positions` - pole s pozicemi osob v předešlém snímku, do tohoto pole se osoby uloží hned po zobrazení snímku

Na počátku si souřadnice rozložím do čtyř proměnných pro lehčí práci s nimi. Dále vytvořím proměnnou `point` což je bod odpovídající středu ohraničujícího rámce. Důležitým krokem je nastavení vzdálenostních limitů (viz obrázek 5.4):

- `distance_limit` - nastavuji na třetinu šířky ohraničujícího rámce. Tento limit označuje jakou délku by člověk na videu neměl být schopen překročit mezi snímky. Pokud je osoba detekována po celou dobu jejího pohybu na videu, její identifikátor by se měnit neměl, protože je stále v dosahu tohoto limitu.
- `shortest_distance_limit` - nastavuji na polovinu šířky rámce. Jedná de-facto o záložní limit. Detekce není perfektní a nejspíš dlouho nebude. Může se stát, že člověk za něco zajde, je zakryt jiným člověkem nebo ho detekce na pár snímků prostě nevezme. V tuto chvíli přichází na řadu záložní vzdálenostní limit, ten určuje jak daleko mám hledat aktivní bod pro daného člověka.

První je `for` cyklus, který projde všechny pozice z minulého snímku. Pro každou tuto pozici se vypočítá vzdálenost od aktuálně detekované osoby (pozice předána jako první argument) přičemž se ukládá nejkratší vzdálenost mezi těmito body a bod samotný. Po získání těchto údajů následuje rozhodování. Porovnám jestli je nejmenší vzdálenost větší než `distance_limit` pokud je tak volám funkci `get_closest_point_from_active_positions` a pokud vzdálenost, kterou mi tato funkce vrátí, bude menší než `shortest_distance_limit`



Obrázek 5.4: Grafické vyznačení limitů (jen pro ilustraci, hodnoty se liší). k1 značí `distance_limit` a k2 značí `shortest_distance_limit`, červené body jsou známé pozice a zelená čára je cesta osoby

tak aktualizuji bod daného indexu a vrátím tento index. Jinak se s největší pravděpodobností jedná o novou postavu a vytvořím ji nový index.

Pokud je nejmenší vzdálenost menší než `distance_limit` tak `for` cyklem projdu všechny osoby v `persons_list`, porovnávám body a pokud jsou body totožné tak jsem našel předchozí pozici osoby a tím i jeho index. Aktualizuji tedy jeho pozici a vracím jeho index. V případě, že se žádný z uložených bodů nerovná, mohlo dojít ke krátkodobé ztrátě pozice této osoby. Proto zavolám opět funkci `get_closest_point_from_active_positions` a pokud vzdálenost, kterou mi tato funkce vrátí, bude menší než `shortest_distance_limit` tak aktualizuji bod daného indexu a vrátím tento index.

`get_closest_point_from_active_positions`

Funkce má jeden vstupní argument a to bod. Jak už název funkce napovídá, jejím cílem je získat nejbližší bod z aktivních pozic. Funkce se sestává z jednoho `for` cyklu, který prochází všechny zaznamenané pozice a počítá vzdálenost mezi body. Pokud je vzdálenost menší než dosud získaná nejkratší vzdálenost tak ji nahradím. Funkce vrací jak nejbližší index, tak nejkratší získanou vzdálenost.

`create_new_index_position`

Funkce má jednoduchý účel a to přidat do pole `persons_list` nový záznam osoby. Vstupem je bod pozice osoby a po zjištění posledního zapsaného indexu je do zmíněného pole zapsána osoba s pozicí v bodě a indexem o 1 větším. Návratovou hodnotou je pak tento index.

Kapitola 6

Testování

6.1 Výsledky

Testování probíhalo nad mnou vytvořenými videi a na mém vlastním stolním počítači, který disponuje procesorem *AMD Ryzen 5 5600X* (s frekvencí na 4.5GHz) a grafickou kartou *NVIDIA GeForce RTX 3080*. Testování jsem prováděl s před-trénovanými váhami. Následuje tabulka s výsledky testování nad stejným videem pro různá rozlišení, se zapnutým sledováním, ukazatelem FPS a obdélníkem jako typem detekce. Hranice důvěry je nastavena na 50%.

Rozlišení	**Verze modelu**	FPS		**ID**
		CPU	**GPU**	
320x320	YOLOv4	8.6	66.3	90
	Vlastní	8.7	57.4	127
416x416	YOLOv4	5.9	53.3	108
	Vlastní	5.8	49.2	111
512x512	YOLOv4	3.9	42.8	107
	Vlastní	3.7	42.2	121
640x640	Scaled YOLOv4	1.7	39.2	68
	Vlastní	1.5	34.7	111
896x896	Scaled YOLOv4	<1.0	20.8	99
	Vlastní	<1.0	26.7	110
1280x1280	Scaled YOLOv4	<1.0	11.8	93
	Vlastní	<1.0	17.4	111

Tabulka 6.1: Porovnání rychlostí vykonávání detekce

V rámci testování jsem si vytvořil několik jednoduchých videí pro snazší testování a pro zahrnutí krajních případů. Například co se stane když jedna osoba zakryje jinou nebo když osoba instantně zmizí a zase se objeví apod. Tyto nahrávky mi pomohli zlepšit a zpřesnit sledovaní.

Co se týče výkonu grafické karty. Ta se při rozlišení 1280x1280 sice nataktuje na 2GHz, ale její využití je pouhých 40%. Na tom jde vidět, že herní grafické karty, i když velmi výkonné, nejsou na tuto zátěž příliš uzpůsobené.

Verze	Rozlišení	Přesnost	
		mAp@0.5	**Ap@0.5:0.95**
YOLOv4	320x320	60.0%	38.0%
	416x416	62.8%	41.2%
	512x512	64.9%	43.0%
Scaled YOLOv4	640x640	68.7%	50.0%
	896x896	70.0%	51.6%
	1280x1280	72.1%	54.0%

Tabulka 6.2: Tabulka přesností před-trénovaných modelů pro MS COCO data set

Provedl jsem i kalkulaci přesnosti na svém data setu a pro všechny před-trénované modely činila přesnost přibližně 97%. Můj model měl přesnost přibližně 99%. Tyto minimální rozdíly a vysoká přesnost jsou dost pravděpodobně způsobeny velmi jednoduchým a malým data setem.

6.2 Vlastní model

Abych dosáhl větší přesnosti a rychlosti rozhodl jsem se natrénovat vlastní váhy pro detekci.

6.2.1 Data set

K tomu jsem si vytvořil data set se 400 obrázky. Většina je ve vysokém rozlišení (více jak 4K rozlišení) obsahující jednu osobou a dále několik skupinových fotek v menším rozlišení. Následně jsem ručně na každém z nich označil osoby do ohraničujících rámců pomocí volně dostupného nástroje. Pro vylepšení data setu jsem využil bezplatných služeb společnosti Roboflow pro vytvoření data setů specifických pro trénování. Při vytváření data setu lze určit rozlišení obrázku (to jsem dal na 416x416) a lze přidat různé augmentace pro rozšíření data setu. Já zvolil dvě a to:

- Jas - vytvoří další dva obrázky, jeden s -10% jasem a druhý s +10% jasem
- Rozostření - vytvoří obrázek rozostřený o 1.25 pixelů

Po těchto augmentacích mám celkem 981 obrázků rozdělených do dvou skupin. 905 pro trénování a 76 pro validaci. Pro testování nemám žádné, protože si výsledný model testuji rovnou na videích.

6.2.2 Trénování

Využil jsem volně dostupného Python notebooku pro trénování na platformě Google Colab.

1. pokus

Trénoval jsem po dobu 1200 iterací na rozlišení 416x416 a dle výpočtu, které skript poskytuje, by měla být přesnost 94.95%. Na vysokém rozlišení jsou výsledky výborné. Jednak je model rychlejší, ale taky přesnější. U nižšího rozlišení ovšem nastává problém. Ohraničující rámce jsou celkově méně přesné, tj. neohraničují osobu úplně tak jak by měly (rámce jsou menší nebo větší). Při malém rozlišení také nastává problém, že je osoba označena dvojitě 6.1a nebo se dvě osoby spojí do jedné 6.1b.

(a) Chybné dvojité ohraničení

(b) Dvě osoby ohraničeny dohromady i přes to, že každá má svůj rámec

Obrázek 6.1: Chyby v detekci s vlastními váhami

2. pokus

Trénoval jsem po dobu 2000 iterací opět na rozlišení 416x416 a dle výpočtu, které skript poskytuje, by měla být přesnost 95.79% viz obrázek 6.2. Oproti prvnímu pokusu se může rozdíl 1% zdát zanedbatelný, ovšem detekce, při nízkém rozlišení, se prokazatelně zlepšila. Ohraničující rámce jsou dosti přesnější a už téměř nedochází k chybným označením jak lze vidět na obrázku 6.6.

```
 calculation mAP (mean average precision)...
76
 detections_count = 237, unique_truth_count = 176
class_id = 0, name = person, ap = 95.79%         (TP = 168, FP = 8)

 for conf_thresh = 0.25, precision = 0.95, recall = 0.95, F1-score = 0.95
 for conf_thresh = 0.25, TP = 168, FP = 8, FN = 8, average IoU = 82.41 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.957923, or 95.79 %
Total Detection Time: 2 Seconds
```

Obrázek 6.2: Výpis výpočtu přesnosti

Shrnutí

Jelikož je model natrénován pouze na jednu třídu, tak jeho vykonávání je pro menší rozlišení stejné, ne-li lehce pomalejší. Se zvyšujícím se rozlišení se ale i zvyšuje rychlost oproti standartnímu modelu. Rychlost vykonávání je pak až o 47.45% rychlejší. Model je celkově více přesnější, jak jde vidět na obrázku 6.4, dokáže zachytit i velmi malé kousky osoby, nohy a podobně.

V rámci testování jsem přirozeně chtěl zkusit natrénovat model i nad větším rozlišením (896x896 nebo až 1280x1280), ale na Google Colabu je to prakticky nemožné. Jednak Google limituje délku připojení a druhak i kdyby ji nelimitoval, tak trénink na rozlišení 896x896 trvá přes 20 hodin a to za podmínky připojení k výpočetní kartě s *Turing* nebo *Volta* architekturou. V opačném případě se doba násobí. Například na rozlišení 416x416 po dobu

2000 iterací. To s kartou *NVIDIA T4* trvalo cca 4 hodiny, s kartou *NVIDIA Tesla K80* ten samý trénink trval cca 10 hodin. To je 2.5 násobný rozdíl času mezi *Kepler* a *Volta* architekturou.

Jakožto vlastník grafické karty s *Ampere* architekturou bych se mohl vyhnout oběma těmto problémům. Proto jsem zkusil zprovoznit trénování lokálně. Bohužel se instalace *Darknet* prostředí nezdařila, Takže tento směr zůstává neprobádaný.

6.3 Grafické porovnání detekce

Obrázek 6.3: Test detekce při rozlišení 1280x1280 se zapnutým sledováním

Obrázek 6.4: Test detekce při rozlišení 1280x1280 se zapnutým sledováním s vlastními váhami

Obrázek 6.5: Test mnou natrénovaného modelu při rozlišení 416x416 (1200 iterací)

Obrázek 6.6: Test mnou natrénovaného modelu při rozlišení 416x416 (2000 iterací)

Obrázek 6.7: Porovnání přesností detekce vlastních vah pro rozlišení 1280x1280, červená - 1200 iterací, zelená - 2000 iterací

Kapitola 7

Závěr

Cílem této bakalářské práce bylo implementovat program pro detekci osob ve videozáznamu. K dosažení tohoto cíle bylo potřeba prostudovat nejmodernější detekční algoritmy, zpracování obrazu a potřebné knihovny programovacího jazyka Python.

Nad rámec zadání jsem implementoval i sledování osob a jednoduchou aplikaci pro mnohem snazší nastavení a zapnutí detekce. Detekci jsem otestoval na několika mnou vytvořených videích a na základě poznamenaných výsledků ji upravoval až do finální podoby. Samotné sledování osob by se samozřejmě dalo přesnějším a rozsáhlejším testováním ještě více vyladit. Detekci jsem se pokusil úspěšně zpřesnit vytvořením vlastního modelu. Možným pokračováním je vytvoření mnohem většího data setu a natrénování vlastního modelu na vyšším rozlišení, což ovšem i na výkonném počítači trvá až desítky hodin.

Literatura

[1] *Canny Edge Detection* [online]. [cit. 2021-12-27]. Dostupné z: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html.

[2] *How Mask R-CNN Works?* [online]. [cit. 2022-01-07]. Dostupné z: https://developers.arcgis.com/python/guide/how-maskrcnn-works/.

[3] *How RetinaNet works?* [online]. [cit. 2021-12-30]. Dostupné z: https://developers.arcgis.com/python/guide/how-retinanet-works/.

[4] *How single-shot detector (SSD) works?* [online]. [cit. 2022-01-07]. Dostupné z: https://developers.arcgis.com/python/guide/how-ssd-works/.

[5] *Morphological Transformations* [online]. [cit. 2021-12-27]. Dostupné z: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html.

[6] Benz, G. *8, 12, 14 vs 16-Bit Depth: What Do You Really Need?!* [online]. 2018 [cit. 2021-12-27]. Dostupné z: https://petapixel.com/2018/09/19/8-12-14-vs-16-bit-depth-what-do-you-really-need/.

[7] Blackledge, J. M. Chapter 16 - Segmentation and Edge Detection. In: Blackledge, J. M., ed. *Digital Image Processing*. Woodhead Publishing, 2005, s. 487–511. Woodhead Publishing Series in Electronic and Optical Materials. DOI: https://doi.org/10.1533/9780857099464.4.486. ISBN 978-1-898563-49-5. Dostupné z: https://www.sciencedirect.com/science/article/pii/B9781898563495500161.

[8] Bochkovskiy, A., Wang, C. a Liao, H. M. YOLOv4: Optimal Speed and Accuracy of Object Detection. *CoRR*. 2020, abs/2004.10934. Dostupné z: https://arxiv.org/abs/2004.10934.

[9] Boesch, G. *Object Detection in 2021: The Definitive Guide* [online]. 2021 [cit. 2021-12-29]. Dostupné z: https://viso.ai/deep-learning/object-detection/.

[10] Canny, J. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1986, PAMI-8, č. 6, s. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[11] Culjak, I., Abram, D., Pribanic, T., Dzapo, H. a Cifrek, M. A brief introduction to OpenCV. In: *2012 Proceedings of the 35th International Convention MIPRO*. 2012, s. 1725–1730.

[12] Gonda, F., Wei, D., Parag, T. a Pfister, H. Parallel Separable 3D Convolution for Video and Volumetric Data Understanding. In: *British Machine Vision Conference (BMVC)*. 2018.

[13] Han, W., Khorrami, P., Paine, T. L., Ramachandran, P., Babaeizadeh, M. et al. Seq-NMS for Video Object Detection. *CoRR*. 2016, abs/1602.08465. Dostupné z: http://arxiv.org/abs/1602.08465.

[14] He, K., Gkioxari, G., Dollár, P. a Girshick, R. B. Mask R-CNN. *CoRR*. 2017, abs/1703.06870. Dostupné z: http://arxiv.org/abs/1703.06870.

[15] Hou, R., Chen, C. a Shah, M. An End-to-end 3D Convolutional Neural Network for Action Detection and Segmentation in Videos. *CoRR*. 2017, abs/1712.01111. Dostupné z: http://arxiv.org/abs/1712.01111.

[16] K, S. *Non-maximum Suppression (NMS)* [online]. 2019 [cit. 2021-12-27]. Dostupné z: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c.

[17] Kacmajor, T. *Hough Lines Transform Explained* [online]. medium.com, 2017 [cit. 2021-12-26]. Dostupné z: https://medium.com/@tomasz.kacmajor/hough-lines-transform-explained-645feda072ab.

[18] Lin, T., Dollár, P., Girshick, R. B., He, K., Hariharan, B. et al. Feature Pyramid Networks for Object Detection. *CoRR*. 2016, abs/1612.03144. Dostupné z: http://arxiv.org/abs/1612.03144.

[19] Liu, M. a Zhu, M. Mobile Video Object Detection with Temporally-Aware Feature Maps. *CoRR*. 2017, abs/1711.06368. Dostupné z: http://arxiv.org/abs/1711.06368.

[20] Liu, M., Zhu, M., White, M., Li, Y. a Kalenichenko, D. Looking Fast and Slow: Memory-Guided Mobile Video Object Detection. *CoRR*. 2019, abs/1903.10172. Dostupné z: http://arxiv.org/abs/1903.10172.

[21] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E. et al. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, abs/1512.02325. Dostupné z: http://arxiv.org/abs/1512.02325.

[22] Meel, V. *YOLOv3: Real-Time Object Detection Algorithm (What's New?)* [online]. 2021 [cit. 2021-12-29]. Dostupné z: https://viso.ai/deep-learning/yolov3-overview/.

[23] Odemakinde, E. *Mask R-CNN: A Beginner's Guide* [online]. 2021 [cit. 2021-12-29]. Dostupné z: https://viso.ai/deep-learning/mask-r-cnn/.

[24] Parker, J. *Algorithms for Image Processing and Computer Vision*. Leden 1997. 42 s. ISBN 978-0-471-14056-6.

[25] PhD., G. A. *Digital Image Processing* [online]. University of Tartu, 2014 [cit. 2021-12-26]. Dostupné z: https://sisu.ut.ee/imageprocessing/avaleht.

[26] Rafael C. Gonzales, R. E. W. *Digital image processing*. 3. vyd. Academia, 2008. 760-791 s. ISBN 978-0131687288.

[27] Redmon, J. a Farhadi, A. YOLOv3: An Incremental Improvement. *CoRR*. 2018, abs/1804.02767. Dostupné z: http://arxiv.org/abs/1804.02767.

[28] Saravanan, C. Color Image to Grayscale Image Conversion. In: *2010 Second International Conference on Computer Engineering and Applications*. 2010, sv. 2, s. 196–199. DOI: 10.1109/ICCEA.2010.192.

[29] Sharma, G. *Python 3 Tkinter Display Popup Tooltip* [online]. [cit. 2022-4-21]. Dostupné z: https://codingshiksha.com/python/python-3-tkinter-display-popup-tooltip-message-on-mouse-hover-over-entrybox-widget-gui-desktop-app-full-project-for-beginners/.

[30] Zhu, X., Wang, Y., Dai, J., Yuan, L. a Wei, Y. Flow-Guided Feature Aggregation for Video Object Detection. *CoRR*. 2017, abs/1703.10025. Dostupné z: http://arxiv.org/abs/1703.10025.

[31] Zhu, X., Xiong, Y., Dai, J., Yuan, L. a Wei, Y. Deep Feature Flow for Video Recognition. *CoRR*. 2016, abs/1611.07715. Dostupné z: http://arxiv.org/abs/1611.07715.

Příloha A

Obsah přiloženého paměťového média

Přiložená SD karta obsahuje veškeré soubory potřebné pro spuštění programu. Jmenovitě:

- Zdrojové soubory app.py a detect.py
- Testovací videa
- Všechny potřebné váhy (soubory .weights s příslušnými .cfg soubory)
- README.txt – textový soubor s popisem programu
- ZIP složka s LaTeXprojektem