Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

# Bakalářská práce

# Rozpoznávání objektů v obrazu na platformě NVidia Jetson Nano

Plzeň 2021 Michal Schwob

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2020/2021

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Michal SCHWOB** |
| Osobní číslo: | **A19B0673P** |
| Studijní program: | **B3902 Inženýrská informatika** |
| Studijní obor: | **Informatika** |
| Téma práce: | **Rozpoznávání objektů v obrazu na platformě NVidia Jetson Nano** |
| Zadávající katedra: | **Katedra informatiky a výpočetní techniky** |

## Zásady pro vypracování

1. Seznamte se s technologiemi rozpoznávání obrazu pro embedded computing hw.
2. Prozkoumejte vhodné technologie pro detekci objektů v obrazu využitím strojového učení a následně vyberte nejvhodnější.
3. Navrhněte aplikaci pro detekci a rozpoznávání objektů v obrazu založenou na vybraných technologiích.
4. Implementujte aplikaci dle návrhu.
5. Ověřte funkčnost řešení a vyhodnoťte míru úspěšnosti rozpoznávání konkrétních objektů.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Jiří Dobrý**
Aimtec a.s.
Konzultant bakalářské práce: **Ing. Martin Dostal, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **5. října 2020**
Termín odevzdání bakalářské práce: **6. května 2021**

L.S.

**Doc. Dr. Ing. Vlasta Radová**
děkanka

**Doc. Ing. Přemysl Brada, MSc., Ph.D.**
vedoucí katedry

V Plzni dne 26. října 2020

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 6. května 2021

Michal Schwob

# Abstract

**Object recognition in the image on NVidia Jetson Nano platform**
This bachelor thesis focuses on the program creation for the robot Matylda. The program processes images from a camera or a video and detects persons and objects. Based on the detection outcomes the program compiles a text dialogue. Every step is executed on a microcontroller NVIDIA Jetson Nano, which is the main part of the robot Matylda. The thesis is composed of four parts. The first part describes the theoretical foundations of image processing, machine learning and artificial neural networks. Based on these foundations the task of object and person detection in image is solved. The second part describes microcontroller NVIDIA Jetson Nano and possibilities of image recognition applications for this hardware. The third part describes the design of the program and its implementation. The last part focuses on testing the application. The result of the testing is an F-score equal to 0,908 and the performance calculated to approximately five images processed per second.

# Abstrakt

Tato bakalářská práce se zabývá vytvořením programu pro robota Matyldu. Program zpracovává obraz z kamery nebo videa a detekuje osoby a objekty. Na základě výsledků detekce sestaví textový dialog. Vše je vykonáváno přímo na mikrokontroléru NVIDIA Jetson Nano, který je základem robota Matyldy. Práce se skládá ze čtyř částí. První popisuje teoretické základy zpracování obrazu, strojového učení a neuronových sítí. Teoretické základy jsou následně využity k řešení úlohy detekce objektů a osob v obrazu. Druhá část popisuje mikropočítač NVIDIA Jetson Nano a možnosti aplikací detekce objektů a osob v obrazu pro tento hardware. Třetí část práce popisuje návrh celého programu a jeho implementaci. Poslední část se zabývá testováním programu, při kterém bylo vypočteno F-score rovno 0,908 a rychlost běhu programu určena přibližně na pět zpracovaných snímků za vteřinu.

# Obsah

# 1 Úvod

Robot Matylda je Open Source humanoidní robot navržený společností OpenTechLab [9]. Jeden exemplář byl sestaven při příležitosti Aimtec Hackathonu v Plzni. Dokáže po stisku tlačítka pořídit snímek a s využitím webových služeb rozpoznat osoby na snímku. Na základě rozpoznaných osob sestaví textový dialog, který je opět s využitím webových služeb převeden na řeč a přehrán.

Pro pro robota, který je předváděn na různých místech, je nutnost připojení k internetu občas překážkou. Na každém místě se nejdříve musí řešit připojení k internetu, aby mohla Matylda něco předvést. Jelikož se stále zvyšuje výpočetní výkon embedded počítačů, je již možné provádět zpracování obrazu přímo na malých mikrokontrolérech vhodných pro roboty. Cílem této práce je implementovat zpracování obrazu přímo pro mikrokontroléru NVIDIA Jetson Nano, který byl pro Matyldu vybrán místo původního RaspberryPi 3. Výkon Jetsonu Nano umožní běh zpracování obrazu v reálném čase, což je jedním z požadavků práce. Následně bude sestaven textový dialog, reagující na detekce v obrazu. Díky výkonu vybraného mikrokontroléru je navíc možné detekovat kromě osob i objekty, stále při splnění požadavku na běh v reálném čase. Nový HW a SW tedy Matyldě přinese nejen nezávislost na připojení k internetu, ale i schopnost reagovat i na detekované objekty. Cíleným výstupem této práce je sestavený textový dialog, jeho převedení na řeč bez připojení k internetu je vysoce nad rámec této práce a bude implementováno v budoucím SW.

V této bakalářské práci jsou popsány základy algoritmů a postupů použitých při rozpoznávání objektu a obličejů v obrazu. Kapitola 2 se věnuje teoretickým základům strojového učení a umělých neuronových sítí, speciálně i konvolučních neuronových sítí. Teoretické základy jsou využity v kapitole 3 pro rozpoznávání a detekci objektů i obličejů v obrazu za využití neuronových sítí. Kapitola 4 popisuje možnosti pro aplikace zpracování obrazu pro embedded hardware a speciálně pro mikrokontrolér NVIDIA Jetson Nano, který je základem robota Matylda. Jsou také uvedeny knihovny a frameworky dostupné na Jetsonu Nano a již existující programy, které se zabývají detekcí objektů nebo osob na embedded HW. V kapitole 5 je popsán návrh SW a proces instalace na NVIDIA Jetson Nano. Cílem je, aby navržený program zpracoval snímek z kamery nebo videa a na základě de-

tekovaných osob a objektů sestavil textový dialog. Dalším cílem je, aby bylo jednoduše možné vyměnit množinu používaných dialogů bez potřeby znovu implementovat sestavení dialogu. Kapitola 6 se věnuje přímo vlastní implementaci navrženého SW. Kapitola 7 popisuje použité metody pro testování programu a ověření funkčnosti vytvořených metod.

# 2 Strojové učení

Tématem této práce je detekce objektů a osob v obrazu. Na první pohled se může zdát, že pojem strojové učení s tématem nesouvisí, ale opak je pravdou. Neuronové sítě, které patří k algoritmům strojového učení, jsou nejpoužívanějším přístupem při řešení úloh detekce či rozpoznávání objektů a osob v obrazu. V této kapitole je stručně popsán princip strojového učení a jsou uvedeny typy učení. Dále se tato kapitola věnuje neuronovým sítím a jejich obecným základům. Poslední část popisuje speciální typ neuronových sítí, tzv. Konvoluční neuronové sítě (CNN), a konkrétní typy sítí použitých při detekci objektů v obrazu.

Strojové učení je jeden z oborů umělé inteligence, který se zaměřuje na systémy schopné se učit. Na rozdíl od tradičních přístupů k řešení problému v informatice, cílem strojového učení není přímo vyřešit problém nebo vypočítat výsledek. Algoritmus je nejdříve třeba natrénovat. Při trénování se snaží pochopit strukturu dat a na základě té provádět rozhodnutí. Program je poté schopný automatizovat rozhodování na základě zkušenosti [27].

Existují tři typy strojového učení:

- učení s učitelem,
- učení bez učitele,
- zpětnovazební učení - interaguje s prostředím, které poskytuje zpětnou vazbu na jednotlivé akce programu (již nebude dále popisováno, není pro naší práci důležité)

Učení s učitelem probíhá tak, že je programu poskytnuta člověkem označená datová sada. Každý prvek má jednoznačně přiřazen správný výstup. Učení probíhá porovnáváním aktuálního výsledku programu a požadovaného výsledku a následným upravováním modelu.

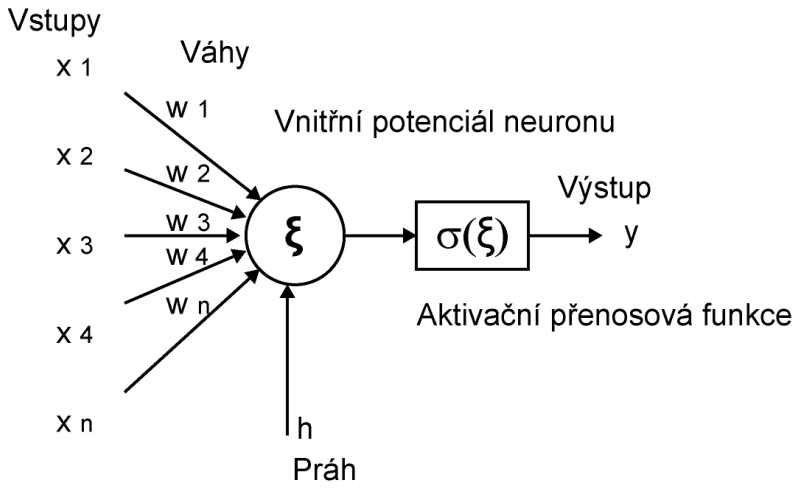
Při učení bez učitele nemají trénovací data definovaný požadovaný výstup nebo třídu, do které mají být klasifikována. Program pouze na základě podobností třídí vstupní data do tříd. Učení spočívá v upravování popisu jader jednotlivých tříd, aby co nejvíce odpovídala vstupnímu vzorku.

Pro učení neuronových sítí určených k detekci objektů v obrazu je využito učení s učitelem. Trénovací data obsahují označené objekty, které mají být programem detekovány. V následující části jsou popsány obecné základy neuronových sítí a i konkrétní příklady použitelné pro naši úlohu.

## 2.1 Umělé neuronové sítě

Již na začátku 21. století měly aplikace neuronových sítí pro úlohy detekce objektů v obraze velmi dobré výsledky. Avšak až soutěž ImageNet v roce 2012 [5] vedla k revoluci v oboru počítačového vidění. Od té doby jsou neuronové sítě, speciálně konvoluční neuronové sítě, nejpoužívanějším přístupem při řešení úloh detekce či rozpoznávání objektů v obrazu [23], a proto jsou využity i v naší úloze. V této podkapitole je popsána obecná stavba umělých neuronových sítí, konvoluční neuronové sítě a konkrétní příklady používané pro detekci objektů v obrazu.

Umělé neuronové sítě (NS) jsou systémy snažící se napodobit fungování lidského těla a procesy v mozku. Lidská nervová soustava, včetně mozku, je složena z obrovského množství neuronů, které jsou vzájemně propojeny a přenášejí mezi sebou elektrický signál (vzruch) [34].

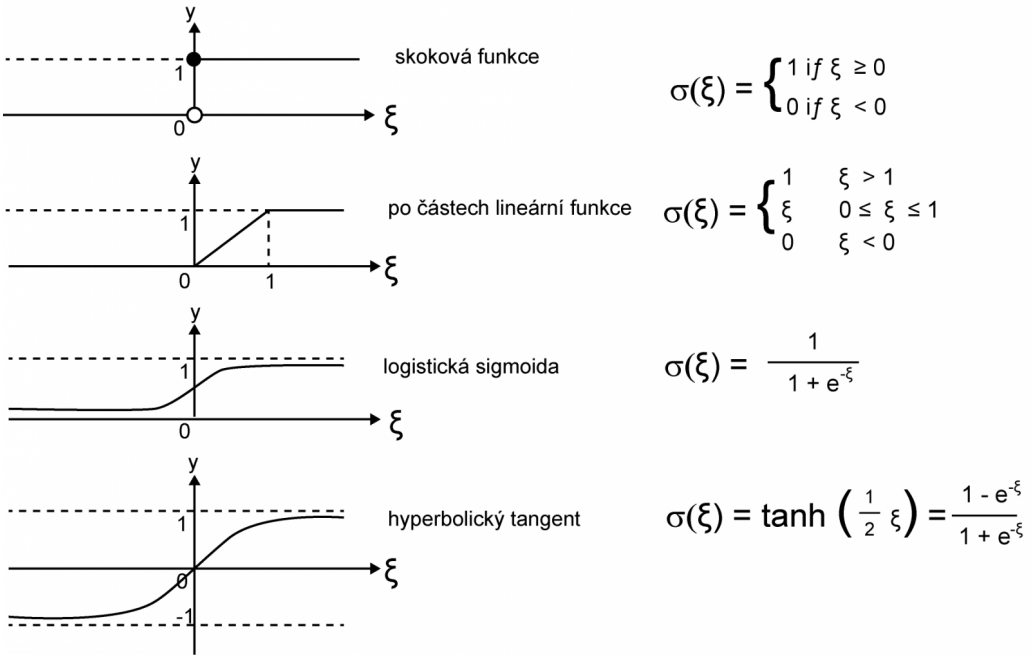



Obrázek 2.1: Schéma umělého neuronu [19]

Základní stavební jednotkou umělé neuronové sítě je neuron, navzájem propojený s mnoha dalšími. Každý jednotlivý neuron je definován n ohodnocenými vstupy, aktivační přenosovou funkcí a výstupem. Na obrázku 2.1

je uveden příklad obecného neuronu. Jednotlivé vstupy jsou označeny $x_i$ a příslušné váhy $w_i$. Součet hodnot na jednotlivých vstupech vynásobených příslušnými váhami je předán aktivační funkci [34].

Nejjednodušším příkladem aktivační je funkce skoková (na výstupu je hodnota 1, pokud vstup dosáhne definované hodnoty, jinak 0) [19]. Na obrázku 2.2 jsou uvedeny příklady nejčastějších aktivačních funkcí.




Obrázek 2.2: Aktivační funkce umělého neuronu [19]

Struktura neuronové sítě je zpravidla rozdělena na tzv. vrstvy. Neurony nejsou navzájem propojeny v rámci jedné vrstvy, spojení existují pouze mezi neurony sousedních vrstev, častým případem je úplné propojení. Každá síť je tvořena alespoň vstupní a výstupní vrstvou [19]. Vrstvy uvnitř neuronové sítě se nazývají skryté vrstvy. Jako obecný příklad se nejčastěji uvádí jedna skrytá vrstva. Pokud síť obsahuje více než jednu skrytou vrstvu, bývá označována jako hluboká neuronová síť (anglicky deep neural network).

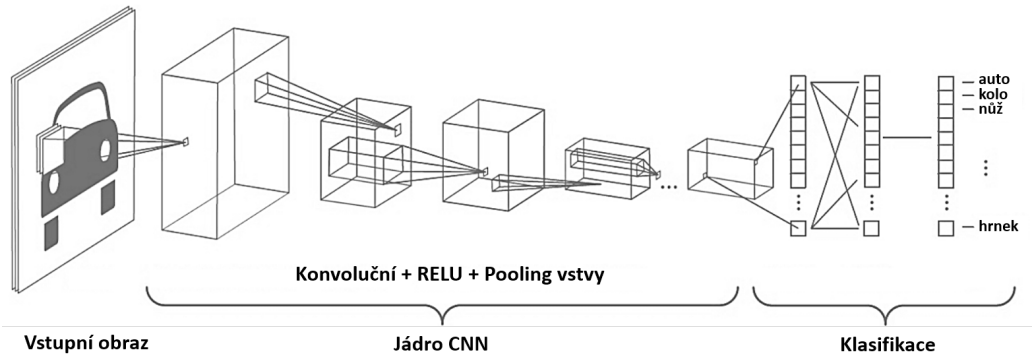
Pojem neuronová síť je obecný. Existují konkrétní typy neuronových sítí speciálně navržené pro různé aplikace. Pro zpracování obrazu jsou nejčastěji využívány konvoluční neuronové sítě, které jsou blíže popsány v následující podkapitole.

## 2.2 Konvoluční neuronové sítě

Konvoluční neuronová síť (Convolutional Neural Network, CNN) je speciálním typem hluboké neuronové sítě navržené ke zpracování dat s předem danou mřížkovou strukturou. Příkladem je reprezentace obrazu pixely uspořádanými do mřížky nebo zvukový záznam. Tyto sítě obvykle obsahují tři typy vrstev [18]:

- konvoluční vrstvy,
- pooling vrstvy,
- plně propojené vrstvy.

Na obrázku 2.3 je uveden model CNN. Na levém kraji je naznačen vstupní obraz a jeho zpracování po částech. Prostřední část je složena z konvolučních a pooling vrstev a zajišťuje rozpoznávání příznaků. Pravá část obrázku symbolizuje klasifikační část obsahující plně propojené vrstvy, která určuje rozpoznané třídy objektů. Jednotlivé vrstvy a jejich výhody CNN jsou popsány v následujících podkapitolách.

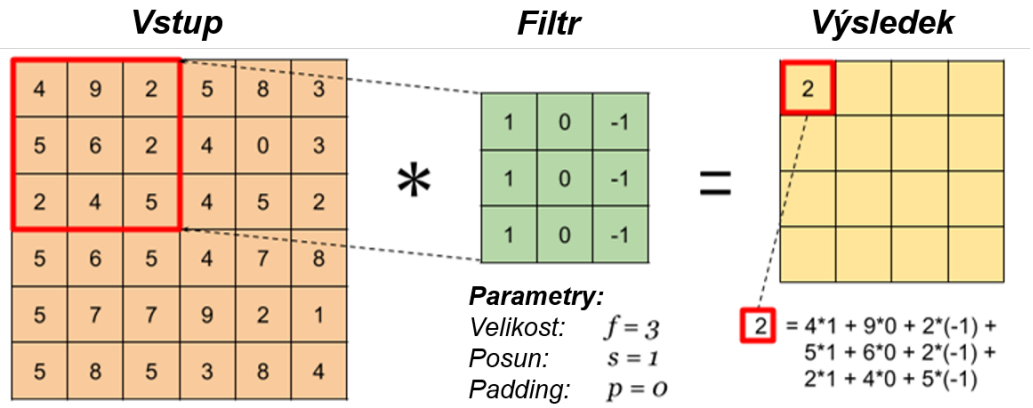



Obrázek 2.3: Model CNN se znázorněním jednotlivých vrstev

### 2.2.1 Konvoluční vrstva

V konvoluční vrstvě probíhá operace, která postupně posunuje definované konvoluční jádro po vstupních datech a na výstup vkládá součet vynásobených prvků jádra a překrytých vstupních dat. Jádro, někdy nazývané filtr, je matice mnohem menších rozměrů než vstupní data s definovaným krokem. Na obrázku 2.4 můžeme vidět příklad velikosti 3 x 3. Dále definujeme krok, se kterým se po datech posouvá, často volena hodnota 1. Výstup této

vrstvy může být také nazýván mapa vlastností (z anglického spojení feature map) [18]. Jádro obvykle obsahuje popis určitého prvku obrazu, například může obsahovat reprezentaci svislé hrany. Postupným posunem po vstupním obraze tedy vytvoří přibližnou mapu výskytu svislých hran v obrazu.




Obrázek 2.4: Operace konvoluce [8]

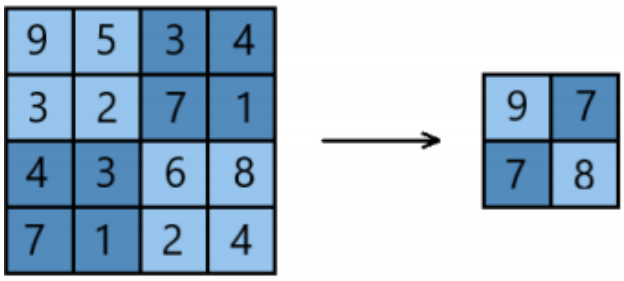
### 2.2.2 Pooling vrstva

Pooling vrstva redukuje velikost vstupu nahrazením výstupní hodnoty statisticky vypočtenou hodnotou z okolních výstupů. Podoblast matice, ze které se vypočítávají výstupní hodnoty, se nazývá jádro. To postupně posouváme po vstupní matici s definovaným horizontálním a vertikálním krokem. Množství redukovaných dat závisí na velikosti jádra a na velikosti kroku. Nejčastějším rozměrem jádra je 2 x 2, velikost kroku se často volí stejná. Mezi hlavní pooling funkce patří max pooling, viz obrázek 2.5, jejíž výstupní hodnota je rovna maximálnímu vstupu v jádře. Další velmi populární funkce využívají průměrnou hodnotu sousedů v jádře, například L2 norma, nebo vážený průměr závislý na vzdálenosti od středu [18].

Obr. 2.4

### 2.2.3 Plně propojená vrstva

Jak již název napovídá, jedná se o vrstvu, jejíž neurony jsou propojeny se všemi neurony předchozí a následující vrstvy. Výstupem konvolučních a pooling vrstev jsou rozpoznané vlastnosti rozprostřené po celém obrazu. Úkolem plně propojených vrstev je sjednotit tyto vlastnosti a tím lépe identifikovat a rozpoznat třídy objektů. Nachází se na samém konci neuronové sítě těsně před výstupní vrstvou [34].




Obrázek 2.5: Operace max pooling [35]

Výhodou je možnost výměny těchto vrstev. Při následném učení sítě již pro rozpoznání jiných tříd není potřeba učit znovu celou síť. Stačí natrénovat pouze plně propojené vrstvy výstupní vrstvu.

Další výhodou CNN jsou sdílené parametry. V běžných neuronových sítích se používá každá váha pro výpočet právě jednoho vstupu neuronu. V konvolučních neuronových sítích je použit každý prvek jádra pro všechny pozice na vstupu. Nemusíme tedy pro každou pozici trénovat speciální sadu parametrů, naučíme pouze jednu, která se poté opakovaně aplikuje [18]. Toto neplatí pouze pro plně propojené vrstvy, kde váhy fungují stejně jako u obecné neuronové sítě.

V této kapitole byly uvedeny základy strojového učení a popsány neuronové sítě. Byla vysvětlena struktura neuronových sítí, jednotlivé vrstvy konvolučních neuronových sítí a jejich výhody. V další kapitole následuje vysvětlení rozpoznávání a detekce objektů, kde jsou neuronové sítě použity ve většině aplikací [5].

# 3 Rozpoznávání obrazu a detekce objektů

V předchozí kapitole byla popsána struktura konvolučních neuronových sítí, které jsou nejpoužívanější technologií pro digitální zpracování obrazu [5]. V této kapitole je rozebrána aplikace těchto sítí na úlohu rozpoznávání a detekce objektů. Jsou popsány společné vlastnosti i rozdíly mezi pojmy rozpoznávání a detekce objektů. Nejprve je ale vysvětlena reprezentace obrazu počítačem, která je důležitá k pochopení toho, jak neuronové sítě pracují s obrazem. Závěrem kapitoly je vysvětlen princip rozpoznávání obličejů.

## 3.1 Reprezentace digitálního obrazu

Pro rozpoznání obrazu je nejdříve nutné pochopit, jak je obraz reprezentován počítačem. Digitální snímek se skládá z velkého množství elementárních jednotek, pixelů. Každý je definován svou polohou v obrazu a hodnotou, interpretovanou jako jas daného pixelu. Nejjednodušší varianta obsahuje pouze černobílou binární hodnotu pixelu, obraz je tedy složen z bílého obrazce na černém pozadí. Obvyklý pixel v dnešní době reprezentuje 8-bitové číslo, tedy 0-255 (černá - bílá). Hodnoty uvnitř intervalu jsou reprezentovány lineární interpolací od černé k bílé [22].

Jedna hodnota pro každý pixel stačí pouze k zobrazení snímku ve stupních šedi. Pro reprezentaci barevného obrazu se nejčastěji využívá tzv. RGB model. Pro každý pixel jsou uloženy tři barevné složky, červená, zelená a modrá. Zkombinováním všech tří barevných složek vznikne výsledný barevný obraz. Jinými slovy, pro uložení barevného snímku potřebujeme tři krát více úložného prostoru než pro snímek ve stupních šedi [22].

## 3.2 Rozpoznávání a detekce objektů

Obecně je rozpoznávání objektů jednou z úloh počítačového vidění. Může být také popsáno jako klasifikace obrazu. Cílem je predikovat třídu objektu nacházejícího se ve snímku na základě vstupních dat. Jedná se tedy o klasifikaci obrazu jako celku. Výstupem může být pouze jedna třída objektu, nebo

informace, že nebyl rozpoznán žádný objekt. I pokud se v obrazu nachází více různých objektů, výsledkem algoritmu je stále jen jedna třída.

Cílem detekce objektů není hodnotit obraz jako celek, ale vyhledat v něm rozpoznávané třídy a označit místo jejich výskytu. Výstupem je seznam podoblastí snímku s příslušnou rozpoznanou třídou objektu. Metody detekce mohou být rozděleny do dvou skupin, jednostupňové a dvoustupňové.

Dvoustupňové metody rozdělují proces na dvě části. Výstupem první části jsou nalezené oblasti vstupního obrazu s možným výskytem určitého objektu. Samotná klasifikace probíhá až v druhé části, kdy jsou výstupy předány natrénovanému klasifikátoru a ten již jen určuje třídu objektu, nikoliv oblast. Tímto přístupem se odfiltrují oblasti, ve kterých se nic nenachází a nemá smysl se jimi dále zaobírat [10].

Jednostupňové metody se oproti dvoustupňovým liší tím, že zároveň určují umístění ve vstupním obrazu i třídu objektu. V porovnání jsou značně rychlejší a mají téměř srovnatelnou přesnost. Mezi zástupce patří například YOLO (You Only Look Once) nebo SSD [10].

## 3.3 Metody detekce objektů

Mezi nejvíce používané jednostupňové algoritmy detekce objektů v obraze patří tyto:

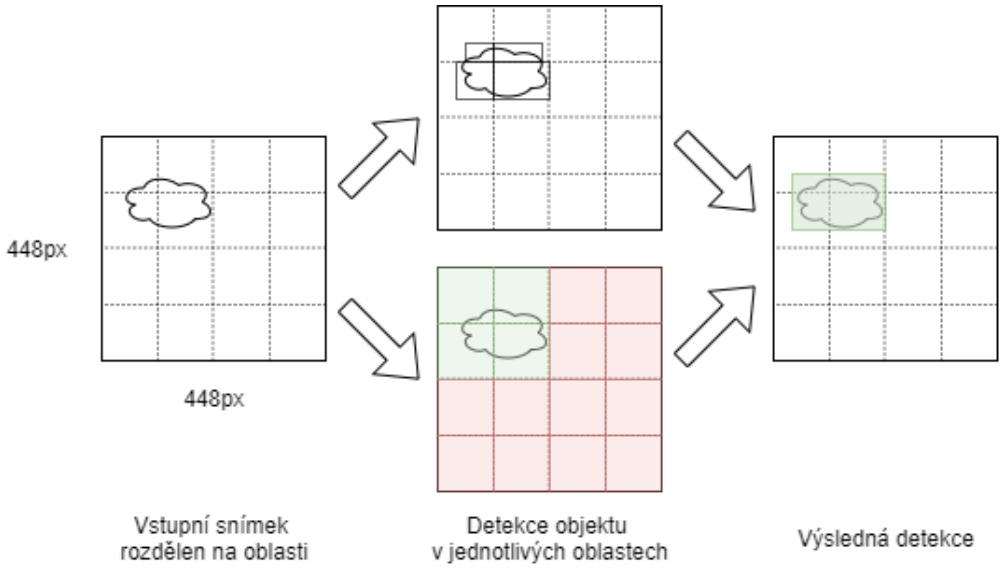
- YOLO (You Only Look Once) - rozdělí obraz na menší čtverce, ve kterých následně provádí detekci.
- SSD (The Single Shot Detector) - provádí detekci příznaků ve více různých měřítkách.
- Res-Net (Residual Network) - základní myšlenkou je duplikování dat uvnitř neuronové sítě, kopií dat přeskočit určité kroky a následně data opět spojit dohromady [14]. Více již tento typ NS nebude popisován.

### 3.3.1 YOLO (You Only Look Once).

Jedním z nejznámějších je systém YOLO (z anglického You Only Look Once). Tento systém sjednocuje více různých komponent detekce objektů do jedné neuronové sítě.

V prvním kroku se rozdělí vstupní obraz na mřížku S x S. Každá buňka předpoví B ohraničujících rámečků kolem objektů a číslo vyjadřující jistotu. Toto číslo určuje, jak si je model jistý výskytem objektu v ohraničení. Každý rámeček je popsán pěticí čísel (*x, y, w, h, a s*), kterou model předpoví. Hodnoty $x$ a $y$ popisují souřadnice středu rámečku uvnitř buňky, $w$ a $h$ udávají šířku a výšku a $s$ popisuje jistotu předpovědi ohraničujícího rámečku. Z každé buňky je ještě získána podmíněná pravděpodobnost $c$ výskytu třídy objektu, která určí pouze jednu třídu nehledě na počet ohraničujících rámečků v buňce. Výstup této první vrstvy má tedy rozměr S x S x (B * 5 + c) [20][11].

Jak je řečeno v názvu, vstupní snímek se zpracuje pouze jednou a následně se již pracuje s mnohem menším množstvím dat. Výsledná neuronová síť je složena z 24 konvolučních a dvou plně propojených vrstev, které předpovídají výstupní pravděpodobnosti a souřadnice objektů. Vstupem této sítě ovšem musí být snímek o přesném rozměru 448 x 448 [20][11].




Obrázek 3.1: Diagram detekce YOLO

Na obrázku 3.1 je uveden diagram detekce neuronové sítě architektury YOLO. Levý snímek reprezentuje vstupní obraz s jedním objektem rozdělený do mřížky. Prostřední snímky simulují ohraničující obdélníky a detekci objektu v jednotlivých oblastech. Na pravém snímku je vyobrazen výsledek celé detekce s jedním detekovaným objektem.

#### SSD

Dalším populárním modelem neuronové sítě je SSD (z anglického The Single Shot Detector). Model je založen na dopředné konvoluční neuronové síti ke které jsou přidány další příznakové (feature) vrstvy s postupně se snižujícím měřítkem.

Použitím různých konvolučních modelů síť předpovídá detekci objektů ve více různých rozměrech, narozdíl od YOLO modelu, který operuje s jedním rozměrem příznakové mapy. Speciálně je použit konvoluční filtr 3 x 3 na vygenerování nové mapy příznaků, ze které se předpovídá lokace objektu a pravděpodobnost příslušnosti ke třídě. Oproti modelu YOLO má SSD rychlejší a přesnější detekci díky využití příznaků ve více různých měřítkách [11][31].

## 3.4 Rozpoznávání obličejů

Rozpoznávání obličejů je specifickou úlohou v oboru počítačového vidění. Skládá se z detekce obličeje, získání znaků obličeje a následné klasifikace. Ještě může být přidán krok zarovnání obličeje za jeho detekování. Tento krok pomůže zvýšit přesnost rozpoznávání obličejů. Obličej je detekován stejným způsobem jako objekty, který je popsán v 3.2. Ostatní tři kroky jsou popsány v této podkapitole.

Pro zarovnání obličeje je potřeba nejdříve získat určité body z obličeje, podle kterých se bude obličej následně zarovnávat. Pro tuto úlohu se dá použít tzv. Multi-task Cascaded Convolutional Network (MTCNN). Jedná se o kaskádu tří konvolučních neuronových sítí za sebou:

- Proposal Network (P-Net) - získá kandidáty pro výskyt obličeje a jejich ohraničující obdélník.
- Refine Network (R-Net) - zpracuje všechny kandidáty a zamítne velké množství špatných kandidátů.
- Output Network (O-Net) - určí finální ohraničující obdélník obličeje a pozici pěti bodů na obličeji [21].

Tato síť detekuje na každém obličeji následující body:

- levé oko,

- pravé oko,
- nos,
- levý koutek úst,
- pravý koutek úst [21].

Podle těchto bodů na obličeji je možné zarovnat obličeje transformací obrazu tak, aby byly všechny stejně natočeny pro další zpracování. MTCNN tedy zastane krok detekce obličeje a i jej připraví pro zarovnání.

Dalším krokem je získání vektorového popisu obličeje, podle kterého je možné ho klasifikovat. Jedna z možných variant je FaceNet od společnosti Google Inc. Jedná se o hlubokou konvoluční neuronovou síť, která obličej popíše vektorem o dimenzi 128 a velikosti prvku jeden byte. Algoritmus je navržen tak, aby podobnost mezi obličeji odpovídala L2 vzdálenosti mezi vektory obou obličejů. Vektory obličejů jedné osoby mají mezi sebou malou vzdálenost a obličeje různých osob vzdálenost velkou. Díky této vlastnosti je následná klasifikace obličejů jednoduchá. Je možné použít například algoritmus K-means s vektory známých osob jako středy jednotlivých tříd [15].

V této kapitole byl popsán princip detekce a rozpoznávání objektů, konkrétní typy neuronových sítí pro detekci objektů, konkrétně YOLO a SSD. Závěrem kapitoly byl vysvětlen princip rozpoznávání obličejů, typ neuronové sítě MTCNN pro detekci obličeje a jeho pozice a FaceNet pro získání popisu obličeje vektorem. V následující kapitole budou popsány možnosti aplikací a omezení těchto algoritmů pro embedded hardware.

# 4 Aplikace pro embedded computing hardware

V předchozích kapitolách byly popsány neuronové sítě a jejich použití pro řešení úloh detekce a rozpoznávání objektů a osob v obrazu. Tato kapitola se zaměřuje na aplikace pro embedded computing hardware, což je výpočetní hardware, který může fungovat zcela samostatně jako mikrokontrolér nebo jako přídavný výpočetní modul např. pro RaspberryPi. Díky vývoji výpočetního výkonu je možné na tomto HW spustit i náročné zpracování obrazu neuronovými sítěmi. Pro běh v reálném čase je ale třeba vybrat vhodný HW a také využít optimalizaci běhu neuronových sítí pro konkrétní HW. V této kapitole je popsán mikrokontrolér NVIDIA Jetson Nano, který byl vybrán pro tuto práci z důvodu vysokého výkonu zaměřeného na běh neuronových sítí. Dále jsou uvedeny knihovny a frameworky, umožňující pracovat s obrazem, neuronovými sítěmi nebo optimalizovat běh programů pro konkrétní HW. Závěrem této kapitoly jsou sepsány některé z existujících projektů, které jsou implementované pro embedded HW nebo využívají optimalizace dostupné na desce NVIDIA Jetson Nano.

## 4.1 NVIDIA Jetson Nano

Jetson Nano je malý výkonný mikrokontrolér vyvinutý společností NVIDIA navržený pro aplikace umělé inteligence. Disponuje čtyřjádrovým 64-bitovým ARM procesorem, 128 jádrovou integrovanou grafickou kartou NVIDIA a operační pamětí o velikosti 4 GB. V porovnání množství snímků zpracovaných rozpoznáváním objektů v obrazu za sekundu je oproti podobně velkému jednodeskovému mikrokontroléru RaspberryPi 3 asi 25 krát výkonnější [16]. Hlavní rozdíl je v GPU. Jetson Nano má grafickou kartu NVIDIA Maxwell s 128 NVIDIA CUDA jádry. Pokud se při implementaci využije knihovna NVIDIA CUDA Toolkit, je možné provádět výpočty právě na jádrech grafické jednotky a tím výrazně zrychlit běh programu [16].

Výrobcem je vydáván operační systém založený na Ubuntu verzi 18.04 s podporou pro NVIDIA CUDA Toolkit. CUDA je univerzální platforma pro paralelní výpočty a programovací model, který využívá paralelní výpočetní modul v GPU k řešení mnoha složitých výpočetních problémů efektivnějším

způsobem než na CPU [13].

Pro pořízení snímku je nutné k desce připojit externí kamerový modul. Počítač NVIDIA Jetson nano má na základní desce 2 CSI porty pro kamery s podporou například modulu Raspberry Pi Camera v2. Tento modul obsahuje 8 megapixelový senzor schopný natáčet video v rozlišení 1080p s 30 snímky za vteřinu, 720p s 60 snímky za vteřinu, nebo 480p s 60 nebo i 90 snímky za vteřinu. K desce je připojen plochým kabelem [1]. Další podporovanou možností je připojení USB kamery k jednomu z dostupných USB portů.

## 4.2 Knihovny a frameworky

Pro implementaci metod zpracování obrazu bylo cílem využít již hotové technologie a případně je rozšířit podle vlastních požadavků. Existují knihovny, které nabízejí již hotové algoritmy například pro detekci objektů v obrazu, nebo umožňují export i načtení natrénovaných neuronových sítí a jejich následné spuštění. Potřebnou vlastností pro aplikace pro embedded HW je možnost optimalizovat běh programu na konkrétním HW, kterou některé knihovny nabízejí. V této kapitole jsou popsány následující nejznámější knihovny a frameworky používané v aplikacích pro zpracování obrazu a strojové učení, dostupné optimalizace a rozdíly mezi nimi.

- OpenCV - práce digitálním s obrazem, natrénované NS pro detekci objektů či osob, načtení a spuštění natrénovaného modelu, podporuje CUDA;
- TensorFlow - modelování, trénování a spouštění NS, podporuje CUDA;
- Pytorch;
- TensorRT;
- NVIDIA Cuda Toolkit.

OpenCV (Open Source Computer Vision Library) je jednou z nejpoužívanějších Open Source knihoven pro počítačové vidění a strojové učení. Obsahuje optimalizované algoritmy například pro získání obrazu z kamery, základní práci s obrazem, detekci a rozpoznání obličejů nebo objektů. Podporuje také CUDA technologii, která pomáhá urychlit výpočty díky výkonu grafických karet, což je pro naši úlohu velmi důležité. Pro integraci do programu poskytuje rozhraní pro jazyky C++, Python, Java nebo Matlab [3].

Nejčastějším využitím této knihovny je základní manipulace s digitálním obrazem. Při spouštění neuronových sítí je obvykle nutné upravovat rozměry snímku ze vstupu geometrickými transformacemi, vyjímat části obrazu nebo dokreslovat například ohraničující rámečky pro detekované objekty.

OpenCV také poskytuje možnost zpracovat uložený předtrénovaný model neuronové sítě a následně ho spustit. Ovšem nenabízí možnost model natrénovat, tedy upravit vstup nebo výstup (klasifikační třídy) neuronové sítě.

TensorFlow je open source knihovna pro strojové učení. Byla vyvinuta společností Google. Hlavní zaměření je modelování, trénování a následné využití hlubokých neuronových sítí. Umožňuje exportovat i načítat natrénované modely NS. Knihovna také podporuje CUDA technologii.

Pytorch je open source knihovna navržená pro strojové učení se zaměřením na počítačové vidění a zpracování přirozeného jazyka vyvíjená společností Facebook. Poskytuje rozhraní pro jazyky Python nebo C++.

Použitím jsou knihovny PyTorch a TensorFlow téměř shodné, proto jsou zde porovnány a zmíněny rozdíly. Tabulka 1 popisuje důležité parametry při výběru a jejich odlišnosti u obou knihoven

TensorFlow Pytorch kvalita dokumentace větší množství návodů dokumentace srovnatelná přívětivost pro uživatele více nízkoúrovňový přístup multiplatformnost (např. i pro mobily nebo servery) nabízí OOP, a tak větší možnosti abstakce, připravené moduly pro použití možnosti debugování pouze externí nástroj, který je použit odděleně od debuggovacího nástroje zbytku programu využití nástrojů pro Python k debuggování přímo za běhu programu živost projektu oba projekty jsou aktivní v posledním měsíci (kontrolováno duben 2021)

licence podobné open source licence, umožňující změny i redistribuci [30, 31]

Tabulka 1. Porovnání rozdílů mezi TensorFlow a Pytorch

NVIDIA® TensorRT™ je knihovna usnadňující běh procesů strojového učení na grafických jednotkách společnosti NVIDIA, které podporují NVIDIA CUDA® technologii. Primárně se zaměřuje na rychlé a efektivní spouštění již natrénovaných neuronových sítí na grafických jednotkách. Je navržena tak, aby doplňovala frameworky pro strojové učení, mezi které patří TensorFlow, Caffe nebo PyTorch. Dokáže zpracovat existující modely sítí

vytvořené výše zmíněnými frameworky a také poskytuje tzv. API, programovací rozhraní aplikace, pro jazyky C++ a Python, které umožňuje sestavit si vlastní model sítě [2].

TensorRT sestaví předanou síť pro konkrétní hardware a připraví ji pro nasazení do programu, který tím pádem není přenositelný mezi zařízeními, ale zbaví se režie spojené s ML frameworkem. Při sestavování se aplikuje komprimace, optimalizace na konkrétní počet jader nebo se také modifikuje přesnost vah v síti pro ušetření operační paměti a urychlení výpočtů [2].

Mikrokontrolér NVIDIA Jetson Nano má grafickou jednotku se 128 CUDA jádry. Proto je při vývoji SW užitečné používat nástroj NVIDIA CUDA Toolkit, který umožňuje vyvíjet a optimalizovat SW pro běh na GPU. Tento nástroj obsahuje optimalizované knihovny pro běh na GPU, debugovací nástroj i C a C++ překladač [13].

## 4.3 Existující projekty

V této podkapitole je uveden stručný přehled projektů, ze kterých je možné vyjít při řešení podobné úlohy. Tyto projekty jsou určeny přímo pro mikrokontrolér NVIDIA Jetson Nano, nebo používají optimalizaci běhu programu na GPU dostupnou na Jetsonu.

- Rozpoznávání obličejů s OpenCV [26],
  - Tento projekt využívá ke všem krokům pouze knihovnu OpenCV. Využívá natrénované neuronové sítě pro detekci obličeje a získání popisu obličeje vektorem, které OpenCV načte. Pro klasifikaci obličeje popsaného vektorem je natrénován vlastní klasifikátor Support Vector Machine (SVM) [26].
- Detekce objektů s TensorRT [12],
  - V tomto tutoriálu je poskytnuto několik natrénovaných modelů NS pro detekci objektů z kamery v reálném čase. Program je navržen přímo pro NVIDIA Jetson Nano a je spouštěn technologií NVIDIA TensorRT, která optimalizuje běh na GPU [12].
- Detekce objektů s TensorFlow a TensorRT [28],
  - V tomto projektu je porovnáno spuštění detekce objektů knihovnami TensorFlow a TensorRT na mikrokontroleru NVIDIA Jetson

Nano. Z výsledků je patrné, že TensorRT mnohem lépe optimalizuje běh programu na GPU, který je pro NVIDIA Jetson Nano klíčový. Díky tomu dokáže zpracovat až tři-krát více snímků za vteřinu než při použití TensorFlow pro spuštění tohoto programu [28].

- Face recognition s TensorRT [32],
  - Tento program je navržen přímo pro mikrokontrolér NVIDIA Jetson Nano s využitím technologie NVIDIA TensorRT pro optimalizaci běhu na GPU. Program detekuje obličeje, které následně klasifikuje. K detekci obličejů je využita MTCNN, na kterou navazuje neuronová síť Google FaceNet pro získání vektorového popisu obličeje.
  - Výkon uvedený autorem je přibližně 80ms pro zpracování jednoho snímku s jedním obličejem [32].

# 5 Návrh

Pro představu je v příloze D přiložen ilustrační obrázek robota Matyldy. Robot obsahuje Open Source program, který rozpoznává osoby a na základě výsledků připravuje textový dialog. Základem je mikrokontrolér RaspberryPi 3, který ale nemá dostatečně velký výkon pro běh detekčních neuronových sítí. Proto byla implementace založena na AWS (Amazon Web Services), webové službě společnosti Amazon. Po stisku tlačítka na Matyldě byl kamerou pořízen snímek a poslán do AWS, kde proběhlo rozpoznávání osoby. Výsledek byl v podobě JSON souboru poslán zpět a na základě něj sestaven dialog. Pro své fungování tedy Matylda potřebovala připojení k internetu, které je při jejím přenášení někdy obtížné zajistit. Další nevýhodou byla potřeba stisku tlačítka pro pořízení snímku následné rozpoznání osoby. Sestavení dialogu probíhá přímo na desce. Pro převedení textu na řeč je použita webová služba od KIV ZČU.

Cílem této práce je oprostit Matyldu od závislosti na připojení k internetu, implementovat program na desce NVIDIA Jetson Nano, která má dostatečný výkon pro běh neuronových sítí. Tento program bude v reálném čase detekovat objekty a osoby v obrazu z kamery. Na základě výstupů z detekce v obrazu následně sestaví dialog pro komunikaci s okolím reagující na nalezené prvky. V této práci se sestaví pouze textový dialog, jeho převod na řeč bude zajištěn dalším SW.

V této kapitole je popsán návrh a struktura programu. Závěrem kapitoly je zdokumentována instalace programu na mikrokontrolér NVIDIA Jetson Nano.

## 5.1 Struktura programu

Celý program je rozdělen do tří samostatných programů:

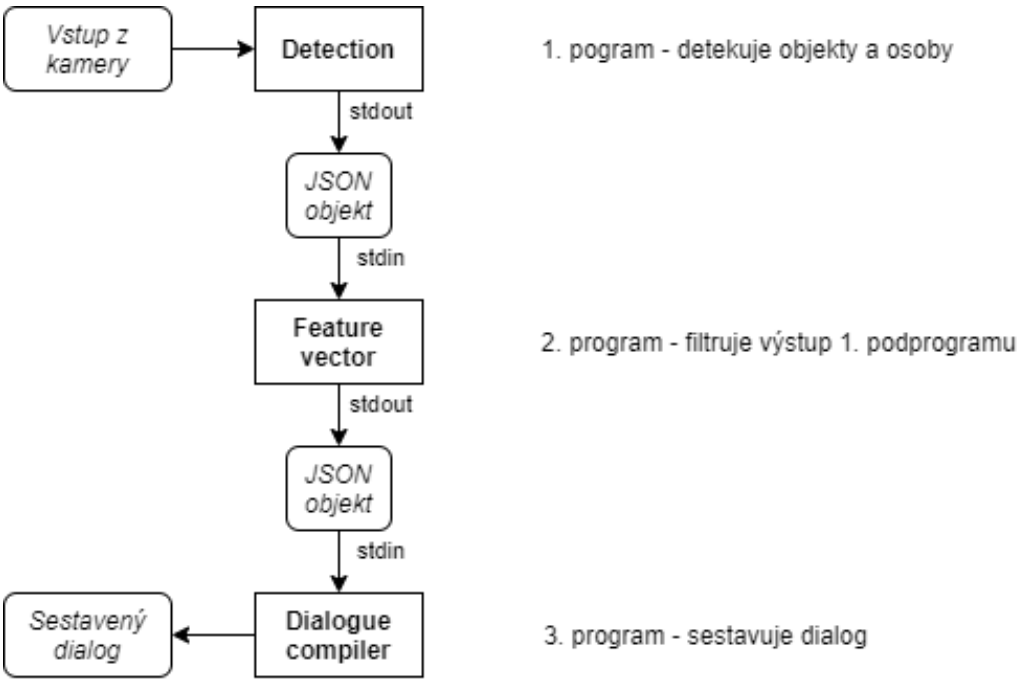
- detection - detekuje objekty a osoby v obrazu,
- feature_vector - zajišťuje filtrování výstupních dat z programu detection,
- dialogue_compiler - sestavuje dialog na základě fitrovaných dat.

Program *detection* zpracovává vstup z kamery nebo videa, detekuje objekty a osoby v obraze a vypisuje nalezené prvky v JSON formátu.

Program *feature_vector* zajišťuje filtrování výstupních dat z programu *detection* a výpis objektů a osob ze vstupu zpět na standardní výstup. Také pracuje s daty ve formátu JSON.

Poslední součástí práce je sestavení dialogu pro reakci na detekované objekty a osoby, které zajišťuje program *dialogue_compiler*.

Základním prvkem celého práce je program *detection*. Zbylé dva jsou pracují s jeho výstupem . Tok dat mezi podprogramy je znázorněn na obrázku 5.1. Propojení zajišťuje Linuxový příkaz pipe [25], který přesměruje výstup jednoho programu na vstup jiného. Jako první je spuštěn program *detection* a jeho výstup je přesměrován na vstup *feature_vector*, jehož výstup je přesměrován na vstup programu *dialogue_compiler*. Tento návrh umožňuje jednoduše obměnit jednotlivé programy nebo spustit například pouze detekci objektů a osob.




Obrázek 5.1: Flow diagram programu

## 5.2 Integrace programu do NVIDIA Jetson Nano

Pro mikrokontrolér NVIDIA Jetson Nano je výrobcem vydán operační systém založen na Ubuntu. Již po spuštění operační systém obsahuje například nainstalované knihovny pro podporu CUDA technologie, NVIDIA TensorRT, python a další knihovny a technologie. Pro běh tohoto programu je tedy potřeba nainstalovat jen cmake, openblas a TensorFlow.

Instalace cmake a openblas je zajištěna příkazem v ukázce kódu 5.1 v příkazové řádce.

```
sudo apt install cmake libopenblas-dev
```

Kód 5.1: Příkaz pro instalaci cmake a openblas

Následuje instalace TensorFlow. Příkazy pro instalaci jsou uvedeny v kódu 8.1 v příloze A. Tato knihovna je použita ke zpracování uloženého modelu neuronové sítě FaceNet z formátu protobuf (.pb) a následné uložení ve formátu UFF. Změna formátu je prováděna kvůli knihovně NVIDIA TensorRT, která neumí načíst formát protobuf. Tato akce je provedena příkazy uvedenými v kódu 5.2. V prvním příkazu je část path/to/project nahrazena cestou k umístění projektu.

```
cd path/to/project/mtcnn_facenet_cpp_tensorRT
python3 step01_pb_to_uff.py
```

Kód 5.2: Převedení modelu neuronové sítě z formátu protobuf na formát UFF

Dalším krokem je přeložení a sestavení jednotlivých programů. K sestavení programu pro detekci objektů a osob je využit nástroj *cmake*. Z kořenového adresáře programu jsou k překladu použity příkazu uvedeny v kódu 5.3.

```
cd mtcnn_facenet_cpp_tensorRT
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make -j\${nproc}
```

Kód 5.3: Překlad programu pro detekci objektů a osob

Pro přeložení a sestavení programu pro filtrování je také využit nástroj *cmake*. Kód 5.4 ukazuje příkazy k sestavení programu, které počítají se spuštěním z kořenového adresáře programu.

```
cd mtcnn_facenet_cpp_tensorRT
mkdir build && cd build
cmake ..
make
```

Kód 5.4: Překlad a sestavení programu pro filtrování detekovaných osob a objektů

Program sestavující dialog je napsán v jazyce Python, takže jej není potřeba překládat. Uživatelský návod pro spuštění je uveden v příloze B.

V této kapitole byla popsána struktura celé aplikace a její rozdělení na tři programy. Komunikace mezi programy probíhá přesměrováním výstupu jedno programu na vstup dalšího. Textová data předávaná mezi programy jsou formátována do JSON formátu. V následující kapitole je detailně popsána implementace všech tří programů, použité datové struktury a algoritmy. Jsou také uvedeny příklady výstupů jednotlivých programů.

# 6 Implementace

V předchozí kapitole byla popsána struktura celého programu a jeho rozdělení na tři samostatné programy, které jsou propojeny pomocí linuxového příkazu pipe. V této kapitole je detailně popsána implementace, použité datové struktury a algoritmy ve všech třech programech.

## 6.1 Detekce objektů a osob

V této sekci bude popsán program *detection*, který je hlavním prvkem celé práce. Zajišťuje zpracování obrazu z kamery, detekci objektů a osob a následné rozpoznání obličeje. Jedná se o kritickou část programu z hlediska požadavku na běh v reálném čase. Z toho důvodu musely být vybrány technologie umožňující vykonávání částí programů na grafické jednotce desky NVIDIA Jetson Nano, která je hlavní výhodou tohoto počítače.

Program vychází ze dvou Open Source programů, které jsou implementovány přímo pro Jetson Nano. Oba programy jsou propojeny a následně rozšířeny. Prvním programem je *mtcnn_facenet_cpp_tensorRT* [32]. Ten zajišťuje získání obrazu z kamery, detekci obličejů a jejich následné rozpoznávání. Druhý program [17] také získá obraz z kamery a poté detekuje objekty. Dále jsou popsány oba programy, jak je realizováno jejich propojení a následné další rozšíření.

Výsledný program načte snímek, detekuje obličeje, následně je rozpoznává a poté ještě ze stejného snímku detekuje objekty. Detekované informace vypíše na standardní výstup ve formátu JSON. Všechny kroky jsou detailně popsány v následujících odstavcích.

Ještě před popisem algoritmu budou popsány kroky, které probíhají před spuštěním samostatné detekce:

- Načtení modelů neuronových sítí.
- Zpracování fotek osob k rozpoznávání.

Natrénované modely neuronových sítí jsou uloženy ve speciálních souborech a musí se programem načíst. Konkrétně se jedná o model MTCNN, složený ze tří neuronových sítí, FaceNet pro získání vektorového popisu obličeje a SSD pro detekci objektů. Po načtení modelů jsou zpracovány fotky osoby, které se mají rozpoznávat. Pro každou osobu je získán a uložen do datové struktury vektorový popis obličeje. V kořenovém adresáři programu se nachází složka *images*, ve které tyto fotky musí být. Soubory s fotkami jsou pojmenovány podle jména osoby ve formátu *Jmeno_Prijmeni*, například *Jan_Novak.jpg*. Pro rozpoznávání určité osoby musí být její fotka v uvedené složce před spuštěním programu, není možné ji přidat za běhu programu. Dále následuje popis hlavní části programu, detekce objektů a osob.

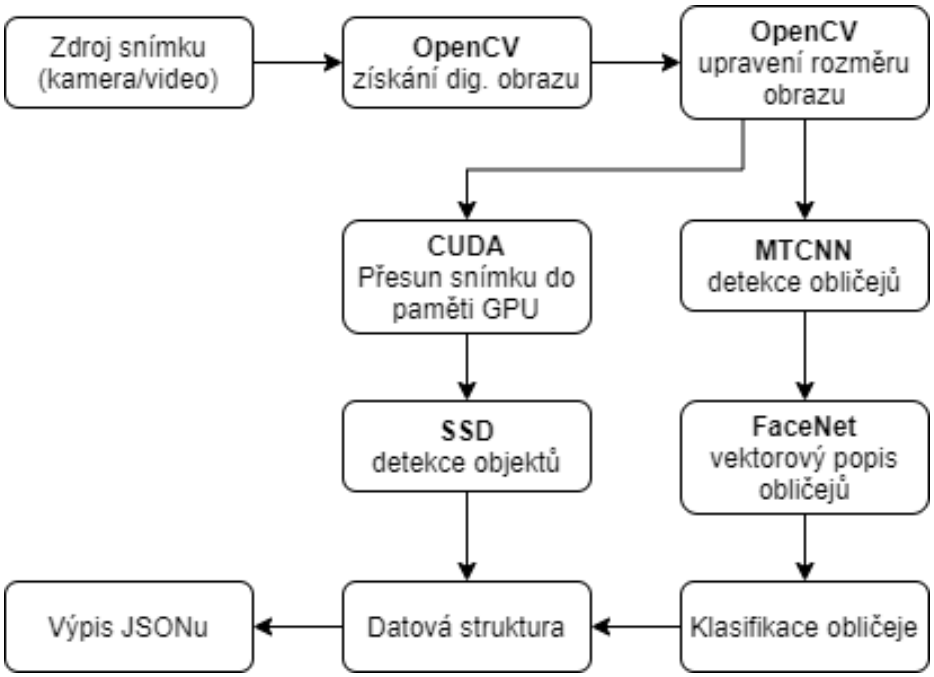
V celém programu jsou použity následující komponenty, jejich konkrétní použití je popsáno dále:

- OpenCV - získání snímku z kamery a manipulace s digitálním obrazem;
- MTCNN - detekce obličejů a jejich natočení;
- FaceNet - získání vektorového popisu obličeje;
- TensorFlow - převedení natrénovaného modelu CNN pro následné načtení,
- NVIDIA CUDA Toolkit - použito pro implementaci kódu pro grafickou jednotku,
- TensorRT.

Obrázek 6.1 znázorňuje jednotlivé kroky v algoritmu a použití výše zmíněných komponent. Všechny kroky jsou detailně popsány v následujících odstavcích.

Prvním krokem v algoritmu je zvolení zdroje snímků pro detekci. Při spouštění program očekává alespoň jeden argument, který tento zdroj určí. Možnosti jsou následující:

- –camera - argument pro nastavení vstupu z USB kamery,
  - –csi - tento argument je potřeba ještě přidat k argumentu –camera pro vstup z CSI kamery (RaspberryPi Camera),
- cesta k souboru se vstupním videem pro vstup z videa.




Obrázek 6.1: Znázornění jednotlivých kroků v programu

Pro získání snímku z vybraného zdroje je využita knihovna OpenCV, která následně i upraví jeho velikost na požadovaný rozměr. Další využití této knihovny je pro vykreslení ohraničujícího obdélníku a názvu rozpoznané třídy do výstupního obrazu, pokud je zobrazen.

Dalším krokem je detekce obličejů, pro kterou je využita neuronová síť MTCNN. Ta detekuje obličeje a zároveň i určité body na obličeji pro následné zarovnání. Detekované obličeje jsou předány k rozpoznávání. Následuje získání vektorového popisu obličeje z neuronové sítě FaceNet. Rozpoznávání obličeje je realizováno měřením L2 vzdálenosti získaného vektoru s uloženými vektory známých osob. Výsledkem je třída rozpoznané osoby (jméno z názvu souboru), nebo označení *nová osoba*. Pro podobnost vektorů je nastavena tolerance. Pokud je rozdíl s nejpodobnějším známým obličejem vyšší než tolerance, je rozpoznávaný obličej určen jako neznámá osoba.

Po dokončení detekce a rozpoznávání obličejů je spuštěna detekce objektů. V tomto kroku jsou propojeny oba výchozí programy. Každý očekává snímek uložený v jiném formátu. Pro detekci osob je použita instance z knihovny OpenCV pro reprezentaci obrazu. Použitý program pro detekci objektů očekává snímek uložený v jednorozměrném poli, kde jsou jednotlivé barevné složky pixelů seřazeny za sebou. Navíc musí být snímek uložen v paměti

grafické jednotky, nikoliv procesoru. Snímek je převeden do potřebného formátu a je pro něj alokováno místo v paměti GPU využitím knihovny NVIDIA CUDA Toolkit. Ihned po zpracování obrazu je paměť opět uvolněna.

Původní program nabízí výběr mezi různými NS pro detekci objektů. Byl vybrán typ SSD, natrénovaný na datasetu COCO [29]. Ostatní poskytnuté natrénované NS měly horší výsledky, nebo byly natrénovány na moc specifických datasetech, například jenom pro rozpoznání psů.

Všechny detekované osoby i objekty jsou uloženy do připravených datových struktur. Z výstupních dat NS je získána třída objektu nebo osoby a souřadnice a rozměry ohraničujícího obdélníku, který určuje polohu v obrazu. Pomocí knihovny pro práci s formátem JSON [29] je sestaven výstupní text.

Výstup programu je v textové podobě ve formátu JSON. Pro každý zpracovaný snímek je vypsán jeden JSON objekt na jednu řádku. Skládá se ze dvou JSON polí s klíči "objects" a "persons". Každý prvek v poli obsahuje pod klíčem "class" třídu objektu nebo osoby. Další parametry popisují výšku, šířku a souřadnice středu ohraničujícího obdélníku pod klíči "height", "width", respektive "x" a "y". Hodnoty jsou normalizovány na interval <0;1> vůči rozměru snímku. Naformátovaný příklad je uveden v ukázce kódu 6.1. Formát JSON byl vybrán z důvodu dobré čitelnosti a široké podpory knihovnami v jazycích C++ i Python.

```
{
    "objects":[
        {
            "class":"person",
            "height":0.9291666746139526,
            "width":0.9750000238418579,
            "x":0.49531251192092896,
            "y":0.5249999761581421
        }
    ],
    "persons":[
        {
            "class":"Michal",
            "height":0.4229166805744171,
            "width":0.3187499940395355,
```

```
                "x":0.23749999701976776,
                "y":0.5541666746139526
            }
        ]
}
```

Kód 6.1: Příklad formátovaného výstupu programu detection

Výstupem tohoto programu jsou pouze čistá data, obsahující formátované získané informace z NS. Pro další zpracování a využití těchto dat jsou navrženy dva další programy, které jsou popsány v následující podkapitole.

## 6.2 Zpracování dat z detekce a rozpoznávání obrazu

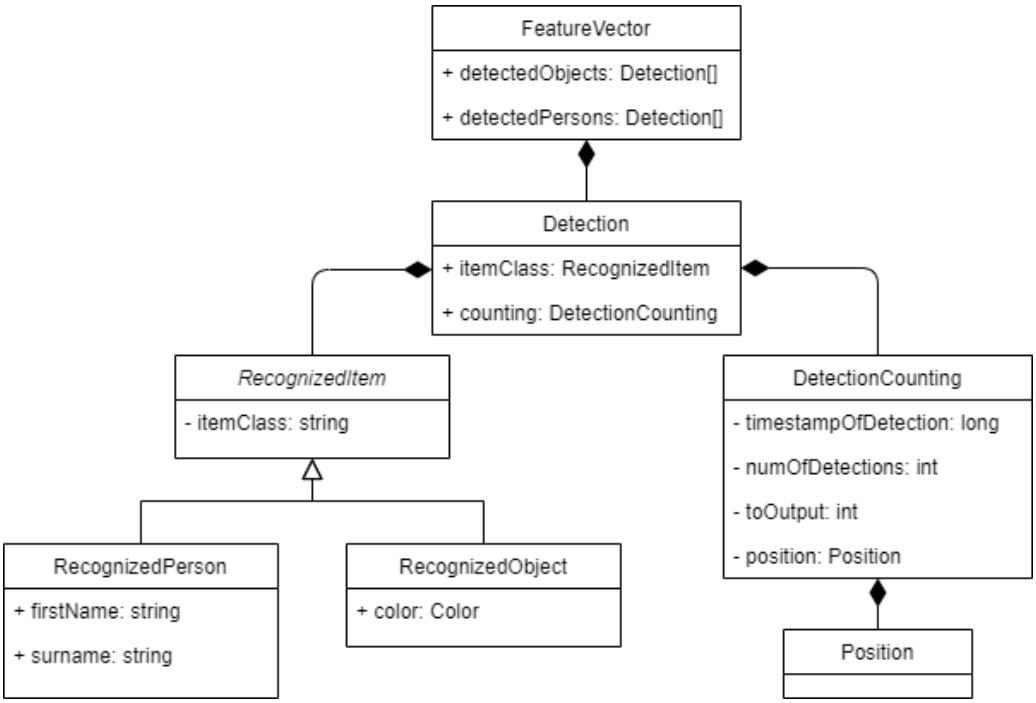
Jak již bylo popsáno v 6.1, výstupní data detekce a rozpoznání obrazu jsou zapisována v JSON formátu na standardní výstup. Tento výstup je vhodný pro testování funkčnosti nebo měření statistik a správnosti určených tříd. Program detection zpracuje několik snímků za vteřinu a tím pádem je výpis pro člověka velmi nečitelný a obsahuje velké množství často opakujících se dat. Z tohoto důvodu byly navrženy dva programy pro zpracování těchto dat:

- *feature_vector* - filtruje data obsahující detekované objekty a osoby,
- *dialuge_compiler* - na základě filtrovaných dat sestavuje dialog. Oba programy jsou detailně popsány v této kapitole.

### 6.2.1 Filtrování dat

První předzpracování zajišťuje program *feature_vector*. Jak popisuje obrázek 5.1, vstupem je výstup programu *detection* ve formátu JSON obsahující detekované objekty a osoby v obrazu, příklad dat je uveden v kódu 6.1. Získané informace ze vstupu se ukládají do připravených datových struktur, které jsou znázorněny na obrázku 6.2 a podrobněji popsány dále. Na základě dat v těchto strukturách probíhá filtrování a rozhodování, které prvky budou vypsány na výstup. Výpis je formátován stejně jako vstupní data a je doplněný o informace o celkovém počtu detekcí a o časovou značku poslední detekce daného prvku. Pro konfiguraci programu je zvolen konfigurační soubor *config.json*, ze kterého se při každém spuštění načítají uživatelem defi-

nované hodnoty. Obsah tohoto souboru je popsán později v této kapitole. Pro implementaci byl zvolen jazyk C++.




Obrázek 6.2: UML diagram datové struktury v programu *feature_vector*

Hlavní třídou datové struktury je *FeatureVector*, jejíž instance uchovávají dvě pole typu *Detection*, jedno pole pro objekty a druhé pro osoby. V obou polích je vždy maximálně jeden záznam příslušný jedné třídě objektu nebo osoby.

Třída *Detection* obsahuje instanci třídy *RecognizedItem* a *DetectionCounting* a popisuje jeden detekovaný objekt nebo osobu v obraze. Informace o třídě detekovaného prvku jsou uloženy v atributech třídy *RecognizedItem*. Jedná se o obecný popis rozpoznaného prvku s informací o názvu rozpoznané třídy. Další informace doplňují dvě oděděné třídy, *RecognizedPerson* a *RecognizedObject*. Pro objekty je přidána informace o barvě a u osoby je získáno jméno a příjmení z názvu třídy. Návrh je připraven pro budoucí rozšíření a přidání dalších informací o osobě (např. věk, pohlaví atd.).

Třída *DetectionCounting* je důležitá pro filtrování dat a obsahuje tyto informace:

- celkový počet detekcí dané třídy (*numOfDetections*),

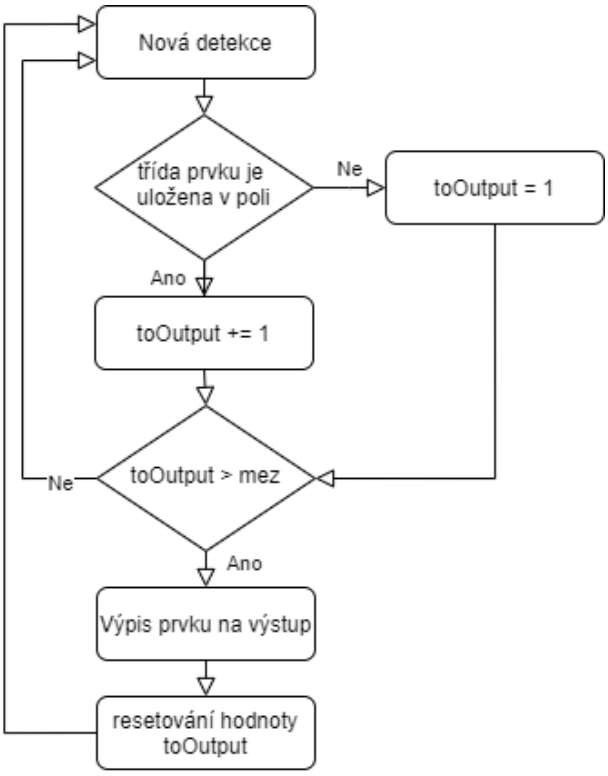
- čítač počtu detekcí od posledního vypsání prvku na výstup (*toOutput*),
- poslední pozici detekce v obraze typu *Position*,
- čas poslední detekce.

Proměnná *numOfDetections* udržuje hodnotu celkového počtu detekcí a tím i informaci o času, po který byl objekt či osoba detekována na snímcích. Celočíselná proměnná *toOutput* udává počet detekcí od posledního vypsání daného prvku na výstup a slouží k filtrování dat. V konfiguračním souboru je nastavená prahová hodnota a po jejím přesažení je prvek vypsán na výstup. Tímto je zajištěno odfiltrování náhodných detekcí a také časový rozptyl mezi jednotlivými výpisy jednoho prvku.

Po vypsání na výstup je hodnota *toOutput* nastavena na resetovací hodnotu z konfiguračního souboru. Tato hodnota je záporná z toho důvodu, aby k prvnímu výpisu mohlo dojít již po malém počtu detekcí a naopak aby byl prodloužen čas mezi dalšími výpisy daného prvku. Na obrázku 6.3 je znázorněn cyklus proměnné *toOutput*. Analogií v lidském životě je situace, kdy zahlédnete známého člověka a pozdravíte ho. Dále s ním komunikujete jen v případě, že se u vás zastaví. Pokud jen pozdraví a jde dál, žádná další komunikace neprobíhá. Podobně je nastaven i robot Matylda. Komunikuje s vámi, pokud před ní stojíte.

Dalším atributem třídy *DetectionCounting* je instance třídy *Position*, která popisuje umístění detekovaného prvku v obraze. Ve vstupních datech je pozice detekce popsána čtyřmi hodnotami, x-ovou a y-ovou souřadnicí středu, šířkou a výškou. Tyto hodnoty definují obdélník, který ohraničuje detekovaný prvek ve snímku. Posledním atributem *DetectionCounting* je časová značka poslední detekce ve tvaru timestamp, tedy počet uplynulých sekund od 1.1.1970 (UTC) [4].

Dále je detailně popsán celý algoritmus filtrování dat. V prvním kroku se přečtou data ze vstupu a jsou následně zpracována, příklad vstupních dat je uveden v kódu 6.1. K získání dat z JSON formátu je implementována vlastní funkce s využitím knihovny json [24]. Data jsou po zpracování uložena do připravených datových struktur. Informace o detekovaných osobách a objektech uchovávají instance třídy *RecognizedPerson*, respektive *RecognizedObject*. V současné chvíli se jedná pouze o název třídy prvku, ale program je připraven pro rozšíření těchto informací. Pro pozice detekovaného prvku




Obrázek 6.3: Cyklus proměnné *toOutput*

v obraze se vytvoří instance třídy *Position*. Data se následně přidávají do připravených polí oddělených pro osoby a objekty. Pro každý nově přidávaný prvek se zkontroluje, zda již neexistuje záznam v poli se stejnou třídou. Pokud takový záznam existuje, nepřidává se další, pouze se upraví hodnoty. Celkový počet detekcí i hodnota proměnné *toOutput* se zvýší o jedna, časová značka se nastaví na aktuální čas a pozice prvku se přepíše pozicí nové detekce. Je tedy uchována pouze poslední pozice prvku. Pokud záznam v poli ještě neexistuje, přidá se nový s počtem detekcí rovno jedné, aktuální časovou značkou a pozicí ze vstupních dat. Prohledávání pole při každém přidávání prvku by mohlo být velice časově náročné, ale počet záznamů v poli je nízký a nejedná se tedy o problém, který by bylo třeba optimalizovat.

V poli detekovaných objektů nebo osob může být maximálně jeden záznam každé třídy prvku. Pro osoby není tato vlastnost nijak limitující, každá osoba existuje pouze jednou. Objekty ale jsou tímto limitovány. Jelikož jsou jednotlivé objekty od sebe rozlišovány pouze na základě třídy, algoritmus nemá možnost od sebe odlišit dva prvky stejné třídy. Pokud na jednom snímku program detekuje dva objekty se stejnou třídou, filtrační program je zpracuje pouze jako jeden prvek detekovaný dvakrát. Jelikož je hlavním zaměřením

celé práce rozpoznávat osoby, tato vlastnost programu programu není limitující.

Dalším krokem v algoritmu je promazání starých detekcí z polí. Každá instance detekce prvku má informaci o času poslední detekce, podle které je možné vymazat staré záznamy a tím urychlit procházení pole. V konfiguračním souboru je možnost nastavit počet vteřin, po kterém budou záznamy z pole vymazány. Tímto algoritmem je robotovi vytvořena krátkodobá paměť. Pamatuje si pouze osoby a objekty, které viděl nedávno, v základní konfiguraci je nastavena doba 20 sekund.

Hlavní částí celého algoritmu je následující krok, výběr prvků na vypsání na výstup. Postupně jsou procházena obě pole, pro osoby i objekty, a aktuální prvek je vypsán na výstup pouze v případě, že počet jeho detekcí od posledního výpisu daného prvku překročil mezní hranici. Pokud je podmínka splněna, nastaví se tento čítač detekcí na resetovací hodnotu nastavenou v konfiguračním souboru.

Pro formátování výstupu na typ JSON je opět využita knihovna json [24]. Vytvoří se JSON objekt, každá vypisovaná třída implementuje metodu přidávající vybrané atributy do předaného JSON objektu. Následně je tento objekt vypsán na standardní výstup v jedné řádce. Obsahuje stejné informace jako výstup programu *detection* a navíc pro každý prvek informaci o časové značce poslední detekce a celkovém počtu detekcí. Pod klíčem "persons" je uloženo pole detekovaných osob a pod klíčem "object" pole detekovaných objektů. Každý prvek v poli je popsán detekovanou třídou pod klíčem "class" a pozicí, výškou a šířkou ohraničujícího obdélníku. Celkový počet detekcí je uložen s klíčem "numOfDetections" a časová značka poslední detekce s klíčem "timestampOfLastDetection". Formátovaný příklad je znázorněn v ukázce kódu 6.2.

```
{
    "persons":[
        {
            "class":"Michal",
            "x":0.239063,
            "y":0.733333,
            "width":0.20625,
            "height":0.277083,
            "numOfDetections":15,
```

```
            "timestampOfLastDetection":1612799290
        }
    ],
    "objects":[
        {
            "class":"cell phone",
            "x":0.239063,
            "y":0.733333,
            "width":0.20625,
            "height":0.277083,
            "numOfDetections":50,
            "timestampOfLastDetection":1612799290
        }
    ]
}
```

Kód 6.2: Ukázka výstupu programu *feature_vector*

Závěrem jsou shrnuty všechny vlastnosti nastavitelné v konfiguračním souboru. V kódu 6.3 je uveden příklad obsahu konfiguračního souboru. Tento soubor se musí jmenovat *config.json*. První hodnotou je hranice počtu aktuálních detekcí potřebných pro výpis na výstup s celočíselnou hodnotou pod klíčem "outputThreshold". Je vhodné zvolit nízké kladné číslo.

Další celočíselnou hodnotou je maximální stáří detekcí uložených v datové struktuře v sekundách s klíčem "maxDetectionAgeInSec". Pokud je detekce starší než uvedený počet sekund, bude z datové struktury vymazána.

Poslední celočíselná hodnota odpovídá resetovací hodnotě aktuálního počtu detekcí mezi jednotlivými výpisy prvku. Je uložena pod klíčem "toOutputResetValue". Doporučuje se zvolit zápornou hodnotu. Tím se prodlouží prodleva mezi jednotlivými výpisy daného prvku a umožní mít nízkou hranici pro první výpis, pro kterou začíná čítač na hodnotě nula.

Poslední možností konfigurace je seznam ignorovaných tříd detekovaných objektů. Pod klíčem "ignoredClasses" je pole názvů tříd objektů, které mají být ignorovány. Množina objektů se totiž nedá změnit, a tak je v filtračním programu přidána možnost určité třídy ignorovat. Seznam všech názvů tříd je uveden v souboru *object-classes.txt* v kořenovém adresáři programu.

```
{
    "outputThreshold" : 6,
        "maxDetectionAgeInSec": 15,
        "toOutputResetValue": -30,
        "ignoredClasses": [
                "person"
        ]
}
```

Kód 6.3: Ukázka konfiguračního souboru *config.json*

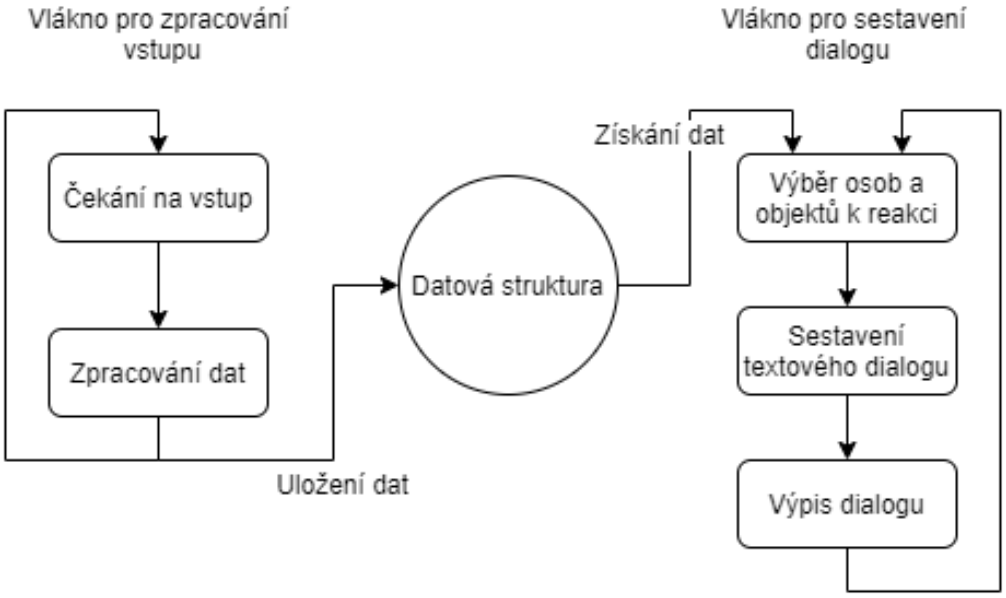
### 6.2.2 Sestavení dialogu

Program je navržen pro robota Matylda. Pokud Matylda provede pouze detekci objektů, její okolí nepozná, že vůbec něco dělá. Aby robot využil informace z detekce a prokázal tak svoji funkčnost, je navržen a implementován program, který zpracuje filtrovaná data a na základě nich sestaví smysluplný textový dialog. V dalším softwaru bude tento text převeden na řeč a přehrán jako hlas robota.

Program zpracovává výstupní data filtračního programu a průběžně sestavuje dialog na základě aktuálně detekovaných osob a objektů. K implementaci tohoto programu je zvolen jazyk Python. Jazyk byl vybrán z důvodu zjednodušení napojení dalšího softwaru. Jedná se o poslední vrstvu této práce a robot Matylda již obsahuje software v jazyce Python, který je možné napojit.

Na obrázku 6.4 uvedeno schéma programu pro sestavení dialogu, v rámci kterého jsou spuštěna dvě nová vlákna. Jedno vlákno čte data ze vstupu a ukládá je do připravených datových struktur. Druhé vlákno obstarává sestavení dialogu na základě vstupních dat a následný výpis dialogu na standardní textový výstup. Obě vlákna přistupují ke sdílené paměti, a proto je potřeba implementovat synchronizaci vláken. V této podkapitole je detailně popsána funkčnost obou vláken a také způsob jejich synchronizace.

Program celkově obsahuje tři vlákna. Hlavní vlákno pouze připraví a spustí další dvě a poté čeká na jejich dokončení. První spouštěné vlákno je určeno pro zpracování vstupu. Vstupní data jsou ve formátu JSON a obsahují filtrované detekované objekty a osoby, příklad je ukázán v kódu 6.2. Ke zpracování dat je využita Python knihovna json [6]. Získaná data se uloží do




Obrázek 6.4: Schéma programu sestavení dialogu

datové struktury, která je realizována jako mapa, klíčem pro ukládání dat je název detekované třídy. Mapa je společná pro objekty i osoby a dovoluje udržovat pouze jeden záznam od každé detekované třídy.

Program se snaží reagovat na co nejaktuálnější detekce a proto je uložen vždy pouze poslední záznam od každé třídy. Vlákno v nekonečné smyčce čte data na vstupu a zpracovává výše popsaným způsobem. Jelikož je čtení blokující operace a výpis dat po filtrování není tak častý, vlákno většinu času čeká na vstupní data. Kvůli tomu byl program rozdělen do více vláken.

Druhé vlákno zajišťuje sestavení textového dialogu na základě aktuálně detekovaných prvků. Požadavkem na tuto část programu byla jednoduchá zaměnitelnost textových odpovědí pro různé příležitosti, například pro příležitost Veletrhu pracovních příležitostí na ZČU, nebo když byla Matylda v Českém rozhlasu. Proto je navržen Python modul s jednotlivými dialogy *sentences.py*. Alternativou Python modulu byl textový soubor, který je obecnější a využitelný více různými platformami. Soubor je využit ale pouze v jazyce Python, a tak je jednodušší implementace i zpracování Python modulu. Ten má v definovaných proměnných uloženy textové dialogy s placehodery pro nahrazení za jméno osoby či název objektu. V ukázce kódu 6.4 je uveden příklad jednoho textového řetězce s placeholderem  pro oslovení.

```
"Ahoj {}, jak se ti vede?"
```

Kód 6.4: Ukázka textového řetězce s placeholderem

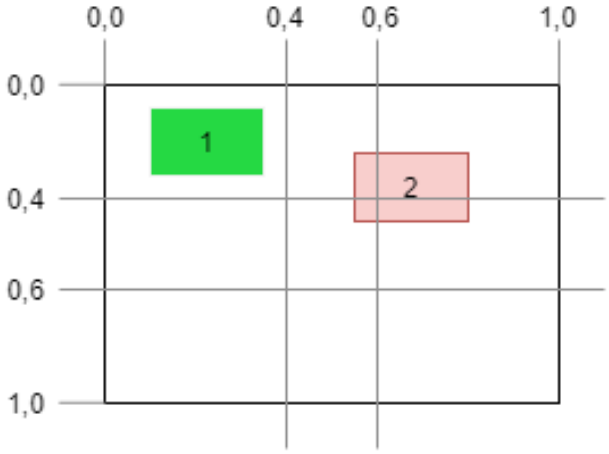
Reakce jsou rozděleny pro osoby a pro objekty. Další dělení je podle celkového počtu detekcí a tedy přibližně podle času, po který je prvek detekován. Počet detekcí je rozdělen na čtyři intervaly popsané časovým ekvivalentem. Časové intervaly se dají konfigurovat, jejich přednastavené hodnoty jsou uvedeny v následujícím seznamu:

- První reakce na daný prvek - celkový počet detekcí do 20.
- Krátká doba detekce - celkový počet detekcí do 60.
- Středně dlouhá doba detekce - celkový počet detekcí do 120.
- Dlouhá doba detekce - celkový počet detekcí nad 120.

Pro objekty se z dat dá získat i poloha a reagovat na ni. Matylda tedy může sestavit jiný dialog pro objekt, který se nachází v pravé části snímku, než pro objekt nacházející se v levé části snímku.

Pozice v obraze je rozdělena následovně. Vertikální i horizontální osa je rozdělena na tři části. Na obrázku 6.5 je vyobrazeno schéma snímku s dvěma detekovanými objekty a hranicemi pro rozdělení do oblastí. Jako hranice jsou zvoleny relativní hodnoty 0,4 a 0,6. Objekt je zařazen do okrajové oblasti pouze v případě, kdy se celý objekt nachází v dané oblasti, jinak je přiřazen do střední části. Příkladem v obrázku jsou objekty 1 a 2, kde objekt 1 patří do levé horní oblasti. Objekt 2 je ale přiřazen do středové oblasti, protože z části přesahuje hranice. Pro detekované osoby se toto určení polohy nepoužívá. Pro změnu dialogů stačí nahradit textové konstanty v souboru *sentences.py*.

Vlákno pro sestavení textového dialogu nejdříve získá data z datových struktur a na základě nich sestavuje dialogy, které následně vypisuje na standardní výstup. Po dokončení výpisu začíná znovu získáním dat. V datové struktuře jsou uložena data, která dosud nebyla zpracována pro vytvoření dialogu. Při získávání dat jsou z datové struktury vyjmuta všechna data a následně seřazena podle časové značky sestupně.




Obrázek 6.5: Rozdělení pozic objektu v obrazu

Algoritmus je navržen pro budoucí zvukovou odezvu. Počítá s tím, že přehrání hlasové odezvy trvá určitou dobu. Kvůli simulaci je výpis uměle zpomalen. Tím se omezí frekvence výpisů a přiblíží se fungování s hlasovou odezvou.

Aby měl tento algoritmus krátkou dobu reakce na detekovaný prvek, je potřeba nevytvářet dlouhé dialogy. Z toho důvodu je omezen maximální počet vět v jedné reakci na maximálně 4, reakce na dva objekty a na dvě osoby. Vždy se vyberou ty nejaktuálnější detekce osob a objektů a ostatní se nepoužijí. V budoucnu může být přidána funkcionalita zpracování starších nezpracovaných detekcí s informací, před jakou dobou byl prvek viděn, v současném návrhu ale tato funkcionalita není.

Protože je vytvořena jedna datová struktura sdílena mezi dvěma vlákny, je potřeba zajistit, aby k datům nepřistupovala obě vlákna najednou. Toto je zajištěno uvnitř datové struktury, aby byla jednoduše integrovatelná do programu bez nutnosti dalšího ošetření kritické sekce. Pro ošetření kritické sekce uvnitř datové struktury je využit zámek Lock z knihovny threading [7]. Před přístupem k datům se vždy nejprve získá zámek a po ukončení práce s daty je zámek zase uvolněn. Pouze jedno vlákno může mít v jeden okamžik přístup k zámku.

V této kapitole byla detailně popsána implementace tří programů, ze kterých se práce skládá. U každého programu byl uveden očekávaný formát vstupu i výstupu, datové struktury a použité algoritmy. V následující kapitole je vysvětlen způsob testování jednotlivých částí a dosažené výsledky. Je

také zhodnocena rychlost běhu celého programu.

# 7 Testování

V kapitole 5 byla popsána struktura celé práce a její rozdělení na tři programy. V kapitole byly uvedeny použité algoritmy a datové struktury v jednotlivých programech. Tato kapitola se zaměřuje na testování všech tří programů. Jsou popsány použité metody při testování a závěrem zhodnoceny dosažené výsledky.

Pro testování úspěšnosti detekce objektů a osob byla připravena dvě krátká testovací videa, jedno se třemi objekty a druhé s osobou a objektem. Programu *detection* byla postupně předána obě videa a výstup byl přesměrován do souboru. Zároveň byl měřen čas potřebný pro zpracování celého videa. Každé video má přesný počet snímků a program zpracuje každý snímek z videa. Z celkového času zpracovávání videa a počtu snímků se dá vypočítat průměrné doba zpracování jednoho snímku $\bar{t}$ podle vzorce 7.1:

$$\bar{t} = \frac{t}{N} \tag{7.1}$$

kde t odpovídá době zpracování celého videa v sekundách a N celkovému počtu zpracovaných snímků. Výsledek $\bar{t}$ je v sekundách. Z této hodnoty se dá následně vypočítat podle vzorce 7.2 průměrný počet snímků zpracovaných za jednu sekundu, často označovaný jako FPS (frames per second):

$$n = \frac{1}{\bar{t}} \tag{7.2}$$

kde n je počet snímků za sekundu a t doba zpracování jednoho snímku ze vzorce 7.1.

Doba zpracovávání celého videa byla vypočtena jako průměrná hodnota ze tří měření a výsledky jsou následující:

V tab. 7.1 je vidět, že se doba zpracování u obou videí příliš neliší, i když jedno video obsahuje přibližně dvakrát více snímků než to druhé. Důvodem

Tabulka 7.1: Počet snímků a celková doba zpracování testovacích videí.

| Typ videa | Počet snímků | Celková doba zpracování videa |
|---|---|---|
| pouze objekty | 201 | 17,0 sec |
| osoba a objekt | 93 | 16,1 sec |

Tabulka 7.2: Přehled doby zpracování jednoho snímku a FPS pro testovací videa.

| Typ videa | Doba zpracování snímku | FPS |
|---|---|---|
| pouze objekty | 0,085 s | 11,824 |
| osoba a objekt | 0,173 s | 5,776 |

je výskyt osoby na videu. Rozpoznávání obličeje předchází jeho detekce. Pokud není žádný obličej detekován, neprobíhá rozpoznávání, a tak je doba zpracování jednoho snímku kratší.

Po dosazení do vzorců 7.1 a 7.2 vyjdou následující hodnoty (výsledky jsou zaokrouhleny na tři desetinná místa):

Z tab. 7.2 je patrné, že se počet zpracovaných snímků za vteřinu může měnit v závislosti na výskytu osoby v obrazu.

Při zpracování videí programem pro detekci objektů a osob byl výstup přesměrován a uložen do souboru. Z dat v souboru byly následně získány jednotlivé detekované objekty a osoby na každém snímků, ze kterých byl vytvořen graf. Detekce byla u obou videí provedena také člověkem snímek po snímku pro porovnání s programem a následné výpočty přesnosti a úplnosti. Přesnost (Precision, Pr) je vypočtena podle vzorce 7.3:

$$Pr = (N - D - S - I)/N \tag{7.3}$$

Pro rovnici platí:

- N - počet všech rozpoznaných jednotek,
- D - počet vynechaných jednotek,
- S - počet zaměněných jednotek,
- I - počet vložených jednotek [30].

Další vypočtenou hodnotou je úplnost (recall). Určuje, kolik procent reálných výskytů prvku bylo algoritmem správně rozpoznáno. Vypočítá se z hodnot uvedených v tabulce 7.1 podle vzorce 7.4:

$$Rec = \frac{TP}{TP + FN} \tag{7.4}$$

Jednotlivé proměnné ve vzorci 7.4 jsou vysvětleny tabulkou 7.3.

Tabulka 7.3: Přesnost a úplnost.

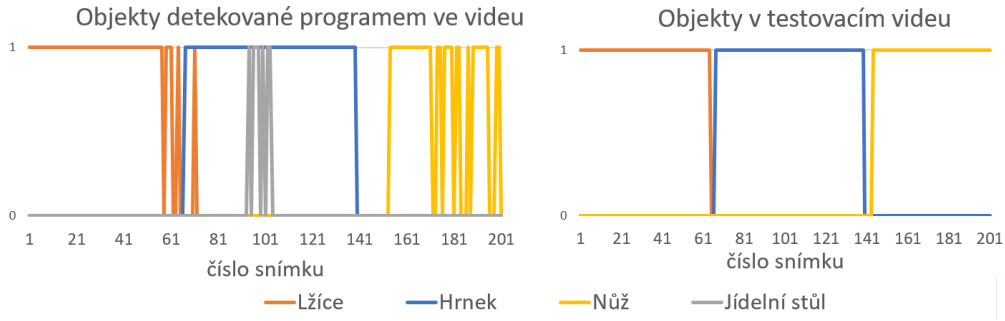
| odhad / skutečnost | 1 | 0 |
|---|---|---|
| 1 | TP | FP |
| 0 | FN | TN |

- TP - true positive;
- FP - false positive;
- TN - true negative;
- FN - false negative [30];

Kombinací přesnosti a úplnosti podle vzorce 7.5 je získáno F-score, které celkovou úspěšnost detekce. Perfektní F-score je rovno jedné.

$$F = 2 * \frac{Pr * Rec}{Pr + Rec}[33] \tag{7.5}$$

Dále jsou popsány a zhodnoceny výsledky z prvního testovacího videa. Na obrázku 7.1 jsou vyobrazeny dva grafy pro video pouze s objekty. Na vodorovné ose jsou jednotlivé snímky. Hodnota grafu určuje, zda se objekt vyskytuje v daném snímku. Pokud je hodnota rovna jedné, znamená to, že se objekt ve snímku vyskytuje. Levý graf odpovídá výstupu programu, druhý ukazuje reálný výskyt předmětů ve snímcích detekovaných člověkem.




Obrázek 7.1: Graf detekcí objektů v testovacím videu 1

Prvním rozdílem obou grafů je různý počet detekovaných objektů. Program ve videu detekoval více tříd objektů. Kromě třídy jídelní stůl jsou

detekovány ještě dvě další třídy navíc, miska a dřez. Tyto třídy nejsou vyobrazeny v grafu kvůli přehlednosti, obě jsou detekovány pouze na jednom snímku. Třída jídelní stůl je celkem detekována na sedmi snímcích. Jelikož se v pozadí videa nachází bílá deska, je záměna za jídelní stůl z určitého úhlu pohledu pochopitelná. Po dosazení naměřených dat do vzorců 7.3 a 7.4 vyjdou následující hodnoty:

$$Pr_1 = (N - D - S - I)/N = (180 - 24 - 0 - 9)/180 = 0,817 = 81,7\% \quad (7.6)$$

$$Rec_1 = \frac{TP}{TP + FN} = \frac{171}{171 + 24} = 0,877 \quad (7.7)$$

$$F_1 = 2 * \frac{Pr * Rec}{Pr + Rec} = 2 * \frac{0,817 * 0,877}{0,817 + 0,877} = 0,846 \quad (7.8)$$

Z výsledku úplnosti 7.7 vyplývá, že program nedetekoval pouze 12,3% výskytů objektů. Pro naší aplikaci je důležité, že program detekoval všechny třídy objektů, které byly na videu zachyceny. Jelikož se data následně filtrují a předávají k vytvoření dialogu, vypadnutí pár detekcí objektu není vůbec žádný problém.

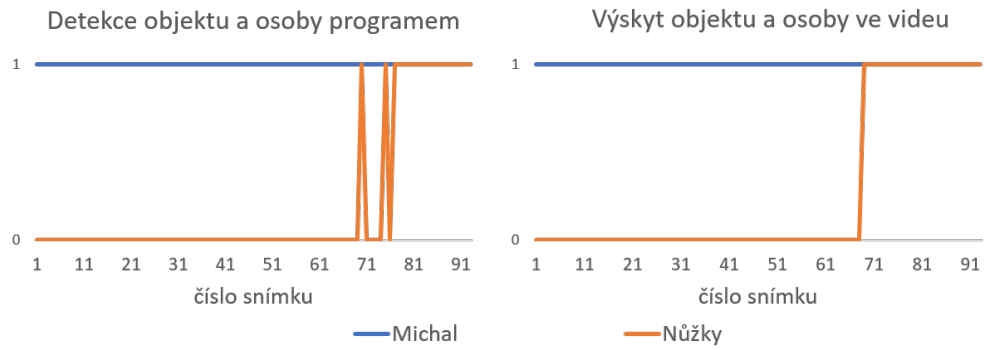
Druhé testovací video ukazuje, že je program schopný detekovat osoby i objekty najednou. Obsahuje jednu osobu, která je pro program známá, a jeden objekt. Na obrázku 7.3.2. jsou vyobrazeny dva grafy. Levý graf odpovídá výstupu detekce osob a objektů programem. Pravý graf ukazuje reálný výskyt osob a objektů v obrazu detekovaný člověkem. Pro osobu mají oba grafy stejné výsledky. Úspěšnost detekce osoby je tedy 100%. Dalším stejně úspěšným rozpoznáváním je třída osoba z detekce objektů, ta se také nachází na každém snímku a na každém snímku je detekována. Kvůli přehlednosti nebyla do grafu zahrnuta.

Pro celé video je úspěšnost a úplnost je vyhodnocena v následujících rovnicích:

$$Pr_2 = (N - D - S - I)/N = (205 - 6 - 0 - 0)/205 = 0,971 = 97,1 \quad (7.9)$$

$$Rec_2 = \frac{TP}{TP + FN} = \frac{205}{205 + 6} = 0,972 \quad (7.10)$$

$$F_2 = 2 * \frac{Pr * Rec}{Pr + Rec} = 2 * \frac{0,971 * 0,972}{0,971 + 0,972} = 0,971 \quad (7.11)$$




Obrázek 7.2: Graf detekcí objektů a osob v testovacím videu 2

Podle výsledků vzorců 7.9, 7.10 a 7.11 je ve druhém testovacím videu úspěšnost i úplnost vyšší než v prvním videu. Může to být dáno menším počtem objektů a tedy i menším prostorem pro chyby, nebo lépe rozpoznatelným objektem na videu. Obě videa jsou ale velmi krátká, a tak nemohou sloužit jako dostatečný statistický vzorek pro přesné určení přesnosti detekce. Na videích byla pouze demonstrována funkčnost a ukázána přibližná přesnost a úplnost detekce osob a objektů.

Vypočítáním aritmetického průměru F-score obou testovacích videí podle vzorce 7.12 se získá celkové F-score pro testování.

$$F = \frac{F_1 + F_2}{2} = 0,908 \tag{7.12}$$

Dále bylo otestováno rozpoznávání obličejů s brýlemi. Před kameru byla postavena osoba nejdříve bez brýlí, poté s dioptrickými brýlemi a nakonec i se slunečními brýlemi. Ani v jednom případě neměl program problém identifikovat osobu. Výsledkem tedy je, že brýle obecně neovlivní rozpoznávání obličeje.

V této kapitole byly popsány použité metody při testování. Program zpracoval dvě testovací videa. Pro porovnání správnosti byla na stejných videích provedena detekce objektů a osob člověkem, snímek po snímku. Ze získaných údajů byly sestaveny grafy a vypočteny hodnoty úspěšnosti a úplnosti detekce.

## 7.1 Shrnutí testování

Zde budou stručně zopakovány výsledky testování.

- Počet zpracovaných snímků za vteřinu je cca 5-10 a záleží na tom, jestli je detekován obličej.
- F-score pro jednotlivá testovací videa,
  - video pouze s objekty - 0,846,
  - video s objektem a osobou - 0,971,
  - průměrná hodnota pro celé testování - 0,908,
- Výsledky jsou velmi dobré, pro reakci na detekce není problém, pokud nebude nějaký výskyt detekován

# 8 Závěr

V rámci bakalářské práce byl vytvořen software pro robota Matyldu, který zpracovává obraz z kamery, detekuje osoby i objekty a vypisuje textový dialog reagující na aktuální detekce. Program běží přímo na desce NVIDIA Jetson Nano, která je základem Matyldy. Požadavek na běh v reálném čase je splněn, počet zpracovaných snímků za vteřinu se pohybuje kolem pěti, díky použitým optimalizacím. Pro Matyldu byla také implementována krátkodobá paměť, která má v základním nastavení trvání 20 vteřin. V rámci této paměti si pamatuje osoby i objekty a dokáže reagovat třemi různými dialogy podle doby, po kterou osobu či objekt vidí, příklad je uveden v příloze v kódu 9.3.

Základem jsou dva open source programy, jeden pro detekci osob a druhý pro detekci objektů. Oba programy jsou propojeny dohromady s využitím CUDA technologie pro urychlení běhu na GPU. Výsledky detekce jsou převedeny do formátu JSON a zpracovány připravenými programy. Výstupní data jsou načítána programem pro filtrování, ve kterém je implementována krátkodobá paměť. Filtrovaný výstup je zpracován programem pro sestavení textového dialogu, který je následně vypsán na standardní výstup.

Při testování detekce byla dosažena přesnost 81,7% a 97,1% u dvou testovacích videí. Pro tuto práci jsou výsledky velmi dostačující. Důležité je, že byly detekovány všechny třídy objektů, které se v obraze nacházely. Při reagování na výskyt objektu není důležité, jestli byl detekován na všech snímcích správně, hlavní je, že byl detekován.

V programu se také počítá s budoucími rozšířeními. Textový dialog bude v dalším softwaru převeden na řeč a přehrán. V tuto chvíli je možné dialog přehrát pouze s využitím cloudové služby. Dalším možným rozšířením je přidání neuronové sítě pro získání více informací o osobě, například pohlaví, přibližný věk, zda má brýle nebo vousy.

Matylda nyní dokáže v reálném čase detekovat osoby a objekty v obraze a sestaveným dialogem na detekce reagovat. Pokud má přístup k internetu, dokáže s využitím cloudové služby dialog převést i na řeč a přehrát. Pokud není připojení k dispozici, dialogy jsou pouze vypisovány do konzole v textové podobě.

Všechny požadavky na bakalářskou práci jsou splněny a práci hodnotím jako úspěšně splněnou.

# Literatura


[1] *Camera Module* [online]. RaspberryPi. [cit. 2020/11/13]. Dostupné z: https://www.raspberrypi.org/documentation/hardware/camera/README.md.

[2] *NVIDIA TensorRT Documentation* [online]. NVIDIA. [cit. 2020/11/15]. Dostupné z: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html.

[3] *OpenCV: Introduction* [online]. OpenCV. [cit. 2020/11/16]. Dostupné z: https://docs.opencv.org/master/d1/dfb/intro.html.

[4] *Epoch Unix Timestamp Conversion Tools* [online]. [cit. 2021/3/2]. Dostupné z: https://www.unixtimestamp.com/.

[5] *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* [online]. [cit. 2021/3/2]. Dostupné z: https://image-net.org/challenges/LSVRC/.

[6] *JSON encoder and decoder* [online]. 2021. [cit. 2021/3/10]. Dostupné z: https://docs.python.org/3/library/json.html.

[7] *Thread-based parallelism* [online]. 2021. [cit. 2021/3/10]. Dostupné z: https://docs.python.org/3/library/threading.html.

[8] *Student Notes: Convolutional Neural Networks (CNN) Introduction* [online]. [cit. 2021/3/10]. Dostupné z: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/.

[9] [online]. [cit. 2021/5/5]. Dostupné z: https://robot-matylda.webnode.cz/o-matylde/.

[10] Abdenour Hadid, e. a. *Deep Learning in Object Detection and Recognition*. Springer Singapore, 2019. ISBN 9811051518.

[11] Abdenour Hadid, e. a. *Deep Learning in Object Detection and Recognition*. Springer Singapore, 2019. ISBN 9811051518.

[12] Alarcon, N. *Real-Time Object Detection in 10 Lines of Python on Jetson Nano* [online]. [cit. 2021/5/5]. Dostupné z: https://developer.nvidia.com/blog/realtime-object-detection-in-10-lines-of-python-on-jetson-nano/.

[13] *CUDA C++ Programming Guide* [online]. NVIDIA, 2020. [cit. 2020/10/4]. CUDA Toolkit Documentation v11.1.0. Dostupné z: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-general-purpose-parallel-computing-architecture.

[14] Feng, V. *An Overview of ResNet and its Variants* [online]. towardsdatascience.com, 2017. [cit. 2021/3/10]. Dostupné z: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035.

[15] Florian Schroff, J. P. D. K. *FaceNet: A Unified Embedding for Face Recognition and Clustering* [online]. 2015. [cit. 2021/3/10]. Dostupné z: https://arxiv.org/pdf/1503.03832.pdf.

[16] Franklin, D. *Jetson Nano Brings AI Computing to Everyone* [online]. NVIDIA Developer Blog, 2019. [cit. 2020/10/4]. Dostupné z: https://developer.nvidia.com/blog/jetson-nano-ai-computing/.

[17] Franklin, D. *Deploying Deep Learning* [online]. [cit. 2021/3/10]. Dostupné z: https://github.com/dusty-nv/jetson-inference.

[18] Goodfellow, I. – Bengio, Y. – Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[19] HOLČÍK, K. M. e. a. k. J. *CUDA C++ Programming Guide* [online]. 2015. [cit. 2020/10/5]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotliv

[20] Joseph Redmon, S. D. R. G. A. F. *You Only Look Once: Unified, Real-Time Object Detection* [online]. [cit. 2020/12/11]. Dostupné z: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf.

[21] Kaipeng Zhang, Z. L. Y. Q. Z. Z. *Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks* [online]. [cit. 2021/3/10]. Dostupné z: https://kpzhang93.github.io/MTCNN_face_detection_alignment/paper/spl.pdf.

[22] Klette, R. *Concise Computer Vision An Introduction into Theory and Algorithms*. London : Springer London, 2014. [Online]. ISBN 1-4471-6320-6.

[23] LeCun, B. Y. . H. G. Y. Deep learning. *Nature*. May 2015, 521, 1. ISSN 1476-4687. doi: 10.1038/nature14539. Dostupné z: https://doi.org/10.1038/nature14539.

[24] Lohmann, N. *JSON* [online]. [cit. 2021/3/2]. Dostupné z: https://github.com/nlohmann/json.

[25] Modi, A. *An introduction to pipes and named pipes in Linux* [online]. opensource.com, 2018. [cit. 2021/3/2]. Dostupné z: https://opensource.com/article/18/8/introduction-pipes-linux.

[26] Rosebrock, A. *OpenCV Face Recognition* [online]. pyimagesearch.com, 2018. [cit. 2021/3/10]. Dostupné z: https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/.

[27] Tagliaferri, L. *An Introduction to Machine Learning* [online]. 2017. [cit. 2020/10/5]. Dostupné z: https://www.digitalocean.com/community/tutorials/an-introduction-to-machine-learning.

[28] Tanonwong, M. *Custom Object Detection with Tensorflow, TensorRT, and ROS on Jetson Nano from Scratch* [online]. 2020. [cit. 2021/5/5]. Dostupné z: https://medium.com/@matus.tanon/custom-object-detection-with-tensorflow-tensorrt-and-ros-on-jetson-nano-63c541bd2

[29] Tsung-Yi Lin, e. a. *Microsoft COCO: Common Objects in Context* [online]. 2015. [cit. 2021/3/10]. Dostupné z: https://arxiv.org/pdf/1405.0312.pdf.

[30] Václav Matoušek, P. K. *Klasifikace, rozpoznávání a shlukování* [online]. 2015. [cit. 2021/3/2]. Dostupné z: http://home.zcu.cz/~pkral/uir/pr5-materialy/FThema4-select.pdf.

[31] Wei Liu1, D. E. C. S. S. R. C.-Y. F. A. C. B. D. A. *SSD: Single Shot MultiBox Detector* [online]. 2016. [cit. 2020/11/13]. Dostupné z: https://arxiv.org/pdf/1512.02325.pdf.

[32] Wesemann, N. *Face Recognition on NVIDIA Jetson (Nano) using TensorRT* [online]. [cit. 2021/3/2]. Dostupné z: https://github.com/nwesem/mtcnn_facenet_cpp_tensorRT.

[33] Wood, T. *F-Score* [online]. [cit. 2021/3/10]. Dostupné z: https://deepai.org/machine-learning-glossary-and-terms/f-score.

[34] Wu, J. *Introduction to Convolutional Neural Networks* [online]. Nanjing University, China, 2017. [cit. 2020/12/3]. Dostupné z: https://cs.nju.edu.cn/wujx/paper/CNN.pdf.

[35] Zacha, J. *Konvoluční neuronové sítě pro klasifikaci objektů z LiDARových dat* [online]. 2019. [cit. 2021/5/5]. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/82351/F3-BP-2019-Zacha-Jiri-Konvolucni_neuronove_site_pro_klasifikaci_objektu_z_LiDARovych_dat.pdf.

# Seznam zkratek

**CNN** Konvoluční neuronová síť

**GPU** Graphics processing unit

**HW** Hardware

**JSON** JavaScript Object Notation

**MTCNN** MultiTask Cascaded Convolutional Neural Network

**SW** Software

# Přílohy

## Příloha A - instalace TensorFlow

```
# Install system packages required by TensorFlow:
sudo apt update
sudo apt install libhdf5-serial-dev hdf5-tools libhdf5-dev zlib1g-dev

# Install and upgrade pip3
sudo apt install python3-pip
sudo pip3 install -U pip testresources setuptools

# Install the Python package dependencies
sudo pip3 install -U numpy==1.16.1 future==0.18.2 mock==3.0.5 h5py==2

# Install TensorFlow using the pip3 command. This command will install
sudo pip3 install --pre --extra-index-url https://developer.download.n
```

Kód 8.1: Příkazy k instalaci TensorFlow na NVIDIA Jetson Nano

## Příloha B - Uživatelská příručka

Celá aplikace je rozdělena na tři samostatné programy, které jsou spuštěny najednou. Programy je možné spustit i samostatně, avšak programy pro filtrování dat a sestavení dialogu pracují se vstupními daty, která by v případě samostatného spuštění chyběla. Všechna následující příkazy počítají se spouštěním z kořenové složky programu, pokud není definováno jinak. Pro spuštění celé aplikace jsou připraveny tři možnosti se zvolením zdroje snímků.

- *run-detection-camera.sh* - spustí celou aplikaci a zvolí USB kameru jako zdroj snímků;
- *run-detection-camera-csi.sh* - spustí celou aplikaci a zvolí CSI kameru (RaspberryPi Camera) jako zdroj snímků;
- *run-detection.sh <cesta-k-videu>* - očekává jeden parametr při spouštění, který odpovídá cestě ke vstupnímu videu;

Z kořenového adresáře aplikace se všechny tři uvedené možnosti spustí stejným způsobem. Příklad je uveden v kódu 9.2.

```
./run-detection-camera.sh
```

Kód 9.2: Příklad spuštění jedné konfigurace

Stiskem kombinace CTRL + C v otevřeném terminálu se ukončí vykonávání programu.

# Příloha C - ukázka výstupu aplikace

V kódu 9.3 je uveden příklad výstupu celé aplikace. Každá řádka odpovídá jedné reakci. Jsou vypsány tři reakce na osobu Michal, ale každá je jiná. Díky implementované krátkodobé paměti dokáže robot Matylda reagovat podle příslušné doby, po kterou je osoba viděna. V dialogu zatím není implementované skloňování, které bude přidáno v dalším rozšíření SW.

```
Ahoj Michal, jak se ti daří?
Potřebuješ s něčím poradit, že tady tak postáváš, Michal?
To nemáš co jiného na práci, Michal, že už se tady na mě
koukáš tak dlouho? Vidím tady hezký/é scissors přede mnou.
```

Kód 9.3: Ukázka výstupu programu

# Příloha D - Matylda

Obrázek 8.1: Ilustrační foto Matyldy