VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

DETEKCE UDÁLOSTI V KAMEROVÉM ZÁZNAMU

EVENT DETECTION IN CAMERA RECORDS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE Mikhail Smolnikov

AUTHOR

VEDOUCÍ PRÁCE Ing. Tomáš Horváth, Ph.D.

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Mikhail Smolnikov ***ID:*** 221296

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Detekce události v kamerovém záznamu

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je nastudovat a popsat současný stav problematiky detekce a klasifikace objektů (např. osob, automobilů, apod.) ve videonahrávce z kamery. V praktické části pak navrhnout GUI aplikaci (ve vybraném programovacím jazyce), která umožní načtení videosekvence a následně provede automatizované vyhodnocení záznamu, jestli obsahuje pohyby osob/zvířat nebo jiných objektů (aut). Nalezené pohyby budou vhodnou formou prezentovány uživateli. V rámci bakalářské práce bude výsledná aplikace využívat metod zpracování obrazu a neuronových sítí jenž umožní z dlouhých nahrávek extrahovat pouze události (pohyb osoby, automobilu, či jiných objektů). Aplikace dále bude umožňovat střih videa v definovaném čase před a po detekované události.

DOPORUČENÁ LITERATURA:

[1] GONZALEZ, R. C.; WOODS R. E.: Digital Image Processing, Prentice Hall, New Jersey, 2002,

[2] OpenCV documentation. 2021 OpenCV, [cit. 2021-09-10]. Dostupné z: https://docs.opencv.org/4.5.3/

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: Ing. Tomáš Horváth, Ph.D.

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

UPOZORNĚNÍ:

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

ABSTRAKT

Tato bakalářská práce se zaměřuje na problematiku detekce a klasifikace pohybujících se objektů ve videosekvencích. Práce popisuje základní algoritmy a metody zpracování obrazových dat, včetně úvodu do použití neuronových sítí. V praktické části je znázorněna vnitřní logika desktopové aplikace umožňující uživatelům vyhodnotit vlastní videozáznamy z hlediska výskytu pohybů. Výsledná aplikace mnohonásobně urychluje proces analýzy videa na vybraném zařízení.

KLÍČOVÁ SLOVA

Detekce pohybu, klasifikace objektů, zpracování obrazu, neuronové sítě, analýza videa

ABSTRACT

This bachelor thesis focuses on the problem of detection and classification of moving objects in video sequences. The thesis describes the basic algorithms and methods of image data processing, including an introduction to the use of neural networks. The practical part shows the internal logic of a desktop application that allows users to evaluate their own video sequences for the occurrence of movements. The resulting application speeds up the process of video analysis on a selected device many times over.

KEYWORDS

Motion detection, object classification, image processing, neural networks, video analysis

Vysázeno pomocí balíčku `thesis` verze 4.07; `http://latex.feec.vutbr.cz`

SMOLNIKOV, Mikhail. *Detekce události v kamerovém záznamu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 52 s. Bakalářská práce. Vedoucí práce: Ing. Tomáš Horváth, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Mikhail Smolnikov
VUT ID autora:	221296
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Detekce události v kamerovém záznamu

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Děkuji tímto vedoucímu bakalářské práce Ing. Tomáši Horváthovi Ph.D., za jeho neúnavnou podporu a odborné vedení při tvorbě této bakalářské práce.

Obsah

Úvod **10**

1 Metody zpracování obrazu **11**
1.1 Algoritmy detekce pohybu . . . 11
1.1.1 Metoda rozdílu snímků . . . 11
1.1.2 Metoda gaussovské směsi . . . 12
1.2 Neuronové sítě . . . 13
1.2.1 Typy neuronových sítí . . . 13
1.2.2 Výhody a nevýhody použití neuronových sítí . . . 17
1.3 Algoritmy klasifikace objektu . . . 19
1.3.1 YOLO . . . 19

2 Implementace analyzátoru videí **23**
2.1 Výběr technologií . . . 23
2.1.1 Knihovna pro zpracování obrazu . . . 24
2.1.2 Knihovna pro rozpoznávání objektů . . . 24
2.1.3 Nastroj pro střih videa . . . 25
2.1.4 Knihovna pro sestavení GUI . . . 25
2.2 Algoritmus zpracování videa . . . 26
2.2.1 Detekce pohybu . . . 26
2.2.2 Ořezávání videoklipů . . . 27
2.2.3 Detekce objektů . . . 30
2.2.4 Reprezentace výsledků . . . 30
2.3 Vytvoření grafického uživatelského rozhraní . . . 33
2.3.1 Požadované funkce rozhraní . . . 33
2.3.2 Okno „Výběr videa“ . . . 33
2.3.3 Okno „Nastavení aplikace“ . . . 33
2.3.4 Okno „Výpis informací o probíhajícím procesu“ . . . 35
2.3.5 Okno „Zobrazení výsledků“ . . . 35
2.4 Konfigurace aplikace . . . 36
2.4.1 Parametry detekce pohybu . . . 37
2.4.2 Parametry detekce objektů . . . 38
2.4.3 Parametry GUI . . . 38
2.4.4 Parametry logování . . . 38

3 Měření **39**
3.1 Výkonnost modulu detekce pohybu 39
3.2 Výkonnost modulu ořezávání videa 41
3.3 Výkonnost modulu detekce objektů 42
3.4 Výkonnost aplikace 43

Závěr **45**

Literatura **46**

Seznam symbolů a zkratek **50**

A Obsah elektronické přílohy **51**

Seznam obrázků

1.1 Frame differences 12
1.2 Mixture of Gaussian 2 13
1.3 Perceptron 14
1.4 Dopředná neuronová síť 15
1.5 Konvoluční neuronová síť 15
1.6 Rekurentní neuronová síť 16
1.7 Práce neuronové sítě s neúplnými daty 18
1.8 Rozdělení obrázku mřížkou 20
1.9 Parametry ohraničujícího rámečku 21
1.10 Závislost hodnoty IoU na poloze rámečků 21
1.11 Výsledky zpracování obrazu algoritmem YOLO 22
2.1 Schéma algoritmu zpracování videa v aplikaci. 26
2.2 Vývojový diagram detekce časových intervalů událostí 28
2.3 Vývojové diagramy zpracování časových intervalů 29
2.4 Výsledky zpracování snímku pomocí modulu detekce pohybu 29
2.5 Výsledný obrázek generovaný modulem detekce objektů. 31
2.6 Obsah výsledného textového souboru 32
2.7 Uživatelské rozhraní pro výběr video souborů. 34
2.8 Uživatelské rozhraní pro volbu parametrů budoucí analýzy. 34
2.9 Uživatelské rozhraní pro sledovaní průběhu analýzy videa. 35
2.10 Uživatelské rozhraní pro zobrazování výsledků zpracování videí. . . . 36
3.1 Graf výkonu modulu detekce pohybu 40
3.2 Graf výkonu modulu ořezávání videa 41
3.3 Graf výkonu modulu detekce objektů 43

Úvod

Při neustále rostoucím množství multimediálních informací je jednou ze základních potřeb tato data zpracovat a analyzovat. Technologie počítačového vidění a strojového učení, které tyto problémy řeší, jsou nyní široce používané v různých průmyslových odvětvích a sférách lidského života.

Díky rychlému rozvoji neuronových sítí se v poslední době zvýšila i rychlost zpracování obrazů. V současné době lze některé operace provádět během několika milisekund a někdy i v reálném čase. To umožňuje analyzovat dlouhá videa v poměrně krátkém časovém období.

Cílem této práce je přinést ucelený pohled na současný stav problematiky detekce a klasifikace pohybujících se objektů ve videonahrávce a poskytnout podrobný popis metod a algoritmů, které jsou k tomuto účelu využívány. Praktická část práce se zaměřuje na vývoj desktopové aplikace, která s využitím těchto technik umožní uživatelům analyzovat vlastní videozáznamy z hlediska výskytu pohybu.

1 Metody zpracování obrazu

Digitální zpracování obrazu lze kategorizovat do dvou širokých oblastí:

- metody, jejichž vstupem a výstupem jsou obrázky. Názorným příkladem je aplikace filtru pro zvýšení kontrastu;
- metody, jejichž vstupy mohou být obrázky, ale jejichž výstupy jsou atributy extrahované z těchto obrázků, jako je detekce výskytu určitého objektu.

Hlavní pozornost je v této práci věnována spíše druhé části – algoritmům počítačového vidění.

1.1 Algoritmy detekce pohybu

Pro detekci pohybu ve videonahrávce se používají algoritmy, v jejichž výstupu jsou oblasti snímku, kde došlo ke značné změně polohy objektu. Pokud se zjistí, že skutečný rozdíl překračuje přípustnou prahovou hodnotu (Threshold), dá se předpokládat, že byl zachycen pohyb. Výběr prahové hodnoty je klíčový. Pokud je příliš nízká, nebude to stačit pro potlačení šumu v obraze. Pokud je příliš vysoká, budou zanedbány užitečné změny v obraze.

1.1.1 Metoda rozdílu snímků

Metoda rozdílu snímků je jednou z nejčastěji používaných metod pro detekci a segmentaci pohybujících se objektů. Základním principem je extrahovat pohybující se oblast v sekvenci obrazů hodnocením rozdílu hodnot pixelů mezi dvěma snímky. Pokud se odpovídající hodnota pixelu změní méně než předem určená prahová hodnota, to může být považováno za část pozadí. Pokud se hodnota pixelů oblasti obrázku výrazně změní, lze to považovat za výsledek pohybujícího se objektu.

Protože časový interval mezi dvěma sousedními snímky je velmi krátký, použití předchozího obrázku jako modelu pozadí aktuálního snímku má dobrý výkon v reálném čase. Pozadí se nehromadí a rychlost aktualizace je vysoká, algoritmus je jednoduchý a je vhodný pro výpočet malého množství dat. Nevýhodou algoritmu je, že je citlivý na okolní šum. Také u velkých pohybujících se objektů může dojít k ztrátě pixelů uvnitř [1].

Obr. 1.1: Výsledky zpracování metodou rozdílu snímků. Nahoře: Dva po sobě jdoucí snímky ve videosekvenci. Dole: Zleva, rozdíl hodnot pixelů těchto dvou snímků. Zprava, výsledek po filtrování na základě prahové hodnoty a zesílení. Zdroj videa, ze kterého je snímek pořízen: [2].

1.1.2 Metoda gaussovské směsi

Metoda „Mixture of Gaussian“ používá model gaussovské směsi (The Gaussian Mixture Model) – kategorie pravděpodobnostního modelu, který říká, že všechny generované datové body jsou odvozeny ze směsi konečných Gaussových distribucí. Každý pixel se modeluje pomocí k směsí Gaussovy distribuce, přičemž k je v rozmezí 3 až 5. Hodnota každé z těchto distribucí použitých na modelu představuje dobu, po kterou byly tyto barvy ve scéně přítomny. Pravděpodobné barvy pozadí jsou ty, které zůstávají déle a jsou více statické. Pixely popředí mění svou barvy častěji a tím mají nižší hmotnost.

Druhá verze této metody byla vyvinuta s cílem vyřešit jedno z omezení, které MOG měla – pevné množství použitých distribucí. Použitím proměnlivého množství Gaussových distribucí dosahuje MOG2 lepší reprezentace složitosti barev v každém snímku, tím zvyšuje pravděpodobnost správného extrahování pohybujících se objektů [3].

Obr. 1.2: Výsledky zpracování algoritmu Mixture of Gaussian 2. Zleva: Snímek videosekvence. Zprava: Výsledek po aplikaci algoritmu a zesílení. Zdroj videa, ze kterého je snímek pořízen: [4].

1.2 Neuronové sítě

Umělé neuronové sítě ANN (Artificial Neural Networks) byly původně vyvinuty ve snaze získat lepší výsledky pro takové problémy, jako je například rozpoznávání řeči a obrazu. Nejdůležitějším aspektem umělé neuronové sítě je její schopnost učit se, a tedy adaptivně zlepšovat svůj výkon.

Neuronová síť se v podstatě skládá z jednotek nazývaných neurony. Tyto prvky jsou mezi sebou propojeny a transformují sadu vstupů na sadu požadovaných výstupů. Výsledek převodu je určen typy prvků a váhovým propojením mezi nimi, takže úpravou vah lze ovládat výstup. Proces aktualizace vah a prahových hodnot se nazývá trénování.

Trénování má obvykle formu prezentace sítě se sadou typických vstupních vektorů x_1, x_2,..., x_n. Při řízeném trénování je každý vstup x_i doprovázen hodnotou očekávaného výstupního vektoru t_i. Poté se cílový výstup t_i porovná se skutečným výstupem y_i, který síť poskytuje, a poté upravuje váhy a prahové hodnoty podle schématu trénování. V režimu „bez dohledu„ jsou poskytnuta pouze vstupní data; nastavení se provádí tak, aby síť mohla rozpoznat vstupní data jako náležející k určitému profilu [5].

1.2.1 Typy neuronových sítí

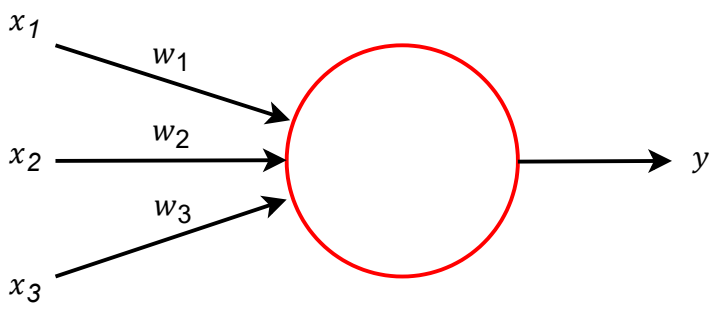
Perceptron

Perceptron také známý jako TLU (Threshold Logic Unit) je jedním z nejjednodušších a nejstarších modelů neuronu. Jedná se o nejmenší jednotku neuronové sítě,

která provádí určité výpočty k získání výstupu jako konečného výsledku. Obrázek 1.3 znázorňuje jednoduchý neuronový model perceptronu. Výstup neuronu nabývá hodnoty $y \in \{-1, 0, 1\}$ a je dán následujícím vztahem [6]:

$$\begin{aligned} net &= \sum_{i=1}^{J_1} w_i x_i - \theta, \\ y &= \phi(net). \end{aligned} \tag{1.1}$$

kde x_i je i-tá vstupní hodnota, w_i je váha spoje z i-tého vstupu, θ je prahová hodnota nebo odchylka a J_1 je počet vstupu. Aktivační funkce $\phi(\cdot)$ je obvykle spojitá nebo nespojitá funkce, mapující reálná čísla do intervalu $(1, 1)$ nebo $(0, 1)$ [7].



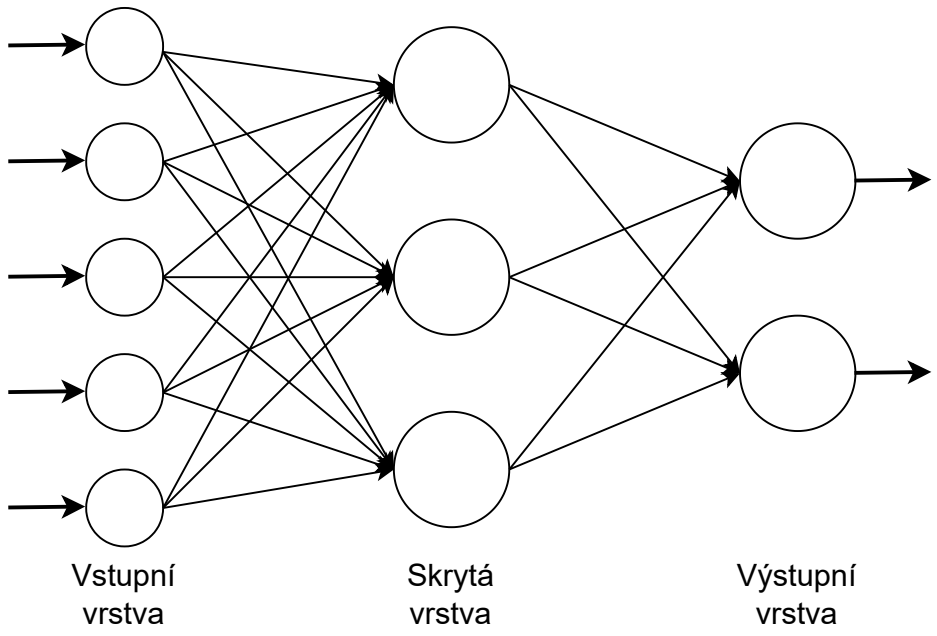
Obr. 1.3: Matematický model perceptronu.

Dopředná neuronová síť

Dopředná neuronová síť FNN (Feedforward Neural Network) je umělá neuronová síť, ve které nejsou smyčky ani zpětnovazební smyčky. V této síti se informace šíří pouze jedním směrem dopředu od vstupních uzlů přes skryté uzly (pokud existují) až do výstupních uzlů. Váhy jsou zde statické. Součet součinů vah a vstupů se vypočítá v každém uzlu, a pokud hodnota překročí určitou prahovou hodnotu (obvykle 0), neuron se aktivuje a převezme aktivovanou hodnotu (obvykle 1); jinak nabude deaktivované hodnoty (obvykle 1) [8].

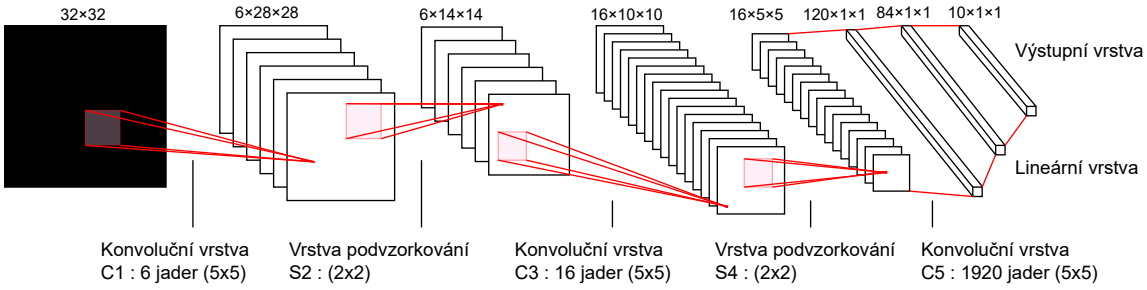
Konvoluční neuronová síť

Konvoluční neuronové sítě nebo CNN (Convolutional Neural Network) se od ostatních neuronových sítí odlišují svým vynikajícím výkonem při aplikaci na obrazu, řeči nebo audio signálu. Základní myšlenka konvolučních neuronových sítí tedy spočívá ve střídání konvolučních vrstev (Convolutional layers) a vrstev podvzorkování



Obr. 1.4: Model vícevrstvé dopředné neuronové sítě.

(Subsampling layers). S každou vrstvou zvyšuje síť výpočetní náročnost a přesněji identifikuje části obrazu. První vrstvy se zaměřují na jednoduché prvky, jako jsou barvy a okraje. Čím dále postupují data, tím detailněji síť rozpoznává tvary objektů, dokud nakonec neidentifikuje zamýšlený objekt [9, 10].

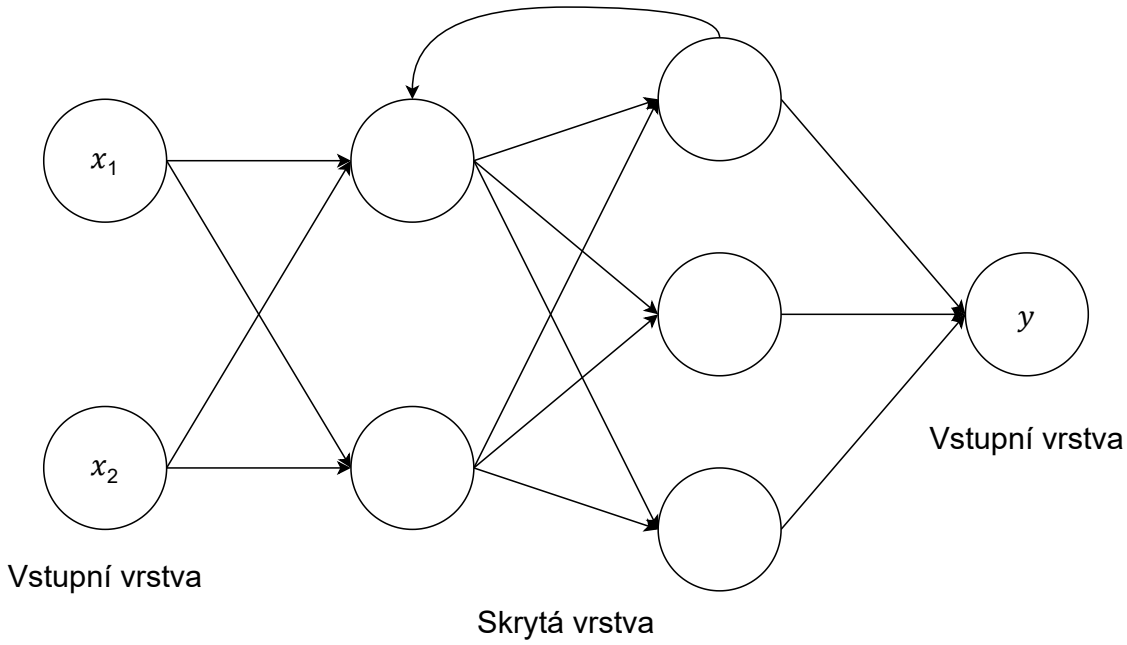


Obr. 1.5: Model konvoluční neuronové sítě LeNet-5 [9].

Recurrent Neural Network

Rekurentní neuronová síť nebo RNN (Recurrent Neural Network) je typem neuronové sítě, kde spojení mezi prvky tvoří řízenou sekvenci. To umožňuje zpracovat řadu událostí v časových nebo sekvenčních prostorových řetězcích. Rekurentní sítě mohou používat svou vnitřní paměť ke zpracování sekvencí libovolné délky. Proto

jsou RNN užitečné v úkolech, kde lze nějaký celek rozdělit na části, například rozpoznávání rukopisu nebo řeči. Pro opakující se sítě bylo navrženo mnoho různých architektur od jednoduchých po složité. V poslední době jsou nejrozšířenější Long short-term memory (LSTM) – sítě s dlouhodobou a krátkodobou pamětí a Gated Recurrent Unit (GRU) – sítě s řízenou rekurentní jednotkou [11, 12].



Obr. 1.6: Ilustrace rekurentní neuronové sítě.

1.2.2 Výhody a nevýhody použití neuronových sítí

Neuronové sítě reprezentují významný přechod v oblasti umělé inteligence a nabízejí slibný potenciál v mnoha oblastech. Nicméně, jak je běžné u nových technologií, i přestože přinášejí mnoho výhod, nejsou zcela bez nedostatků. Pro plné pochopení a efektivní využití neuronových sítí je nezbytné pečlivě zhodnotit jak jejich přínosy, tak i omezení.

Výhody

1. **Výkon**
   Nejvýraznější výhodou využití neuronových sítí je rychlost zpracování vstupních dat. Neuronové sítě mají v některých procesech potenciál pro ultra vysoký výkon díky použití masivního paralelismu – schopnosti spouštět velké počty paralelních úkolů při zpracování informací najednou [13].
2. **Imunita vůči šumu ve vstupních datech**
   Schopnost pracovat v přítomnosti velkého množství neinformativních vstupních signálů, jež obsahují šum. Není potřeba je předem filtrovat, neuronová síť sama určí jejich nevhodnost pro řešení problému a zahodí je. Názorným příkladem je systém schopný pracovat se zašuměnými daty ze skenovací sondy v mikroskopii, popsaný v [14].
3. **Schopnost pracovat s neúplnými daty**
   Po zaškolení mohou ANN produkovat výstup i s neúplnými vstupními daty, například při doplňování chybějících částí obrázku 1.7 metodou popsanou v [15]. Časová náročnost zde závisí na důležitosti chybějících informací.
4. **Adaptace na změny prostředí**
   Neuronové sítě mají schopnost přizpůsobovat se změnám prostředí. Zejména neuronové sítě natrénované pro provoz ve specifickém prostředí lze snadno přeškolit na provoz v podmínkách malých výkyvů parametrů prostředí. Čím vyšší jsou adaptivní schopnosti systému, tím stabilnější bude jeho práce v nestacionárním prostředí.
   Je však třeba poznamenat, že adaptabilita ne vždy vede ke stabilitě; někdy to vede k přesně opačnému výsledku. Například adaptivní systém s parametry, které se v čase rychle mění, může také reagovat na vnější události, které měly být vynechané, a způsobit tím ztrátu výkonu. Aby bylo možné využít všechny výhody adaptability, musí být základní parametry systému dostatečně stabilní, aby bylo možné ignorovat vnější rušení, a dostatečně flexibilní, aby bylo možné reagovat na významné změny prostředí [16].

Obr. 1.7: Práce neuronové sítě s neúplnými daty. Zleva: původní obrázek. Uprostřed: obrázek po odstranění částí pixelů. Zprava: výsledek po zpracování algoritmu [15].

Nevýhody

1. **Náročnost a délka trénovaní**
   Aby neuronová síť správně řešila zadané úlohy, je potřeba ji natrénovat na velkém datasetu – desítkách milionů vstupních dat. To je způsobeno fenoménem přeškolení, když při změně podmínek nebo úkolu je potřeba se rychle přizpůsobit nové situaci a vrátit správný výsledek. Proto proces trénování je časově poměrně náročný.
2. **Interpretovatelnost**
   Umělé neuronové sítě jsou ve většině případů „černé skříňky“ – systém, ve kterém jsou známé parametry vstupu a výstupu, ale vnitřní vybavení není možné prozkoumat. To znamená, že kromě výsledku nelze z ANN získat žádné informace, dokonce ani statistické údaje. Kvůli tomu je velmi obtížné pochopit, jak ANN rozhoduje. To je typické pro většinu typů ANN, ale existují výjimky, například CNN v problémech s rozpoznáváním. V tomto případě mají některé mezivrstvy představu o aktuálním seznamu parametrů funkcí (dá se zjistit, jestli na původním obrázku byl nalezen nějaký jednoduchý vzor), takže lze sledovat mezistavy jednotlivých neuronů. Na základě této informace je možné následně měnit váhy a metody tréninku.
3. **Náklady na technické vybavení**
   Proces trénování umělých neuronových sítí je v podstatě velmi velké množství matematických operací včetně sčítání a násobení matic a vektorů u jednotli-

vých vrstev. V takových úkolech grafické procesory (Graphics Processing Unit) mají větší potencial než běžné procesory (Central Processing Unit). Je to dáno vyšší šířkou pásma paměti, která umožňuje přenášet velké množství informací současně. Pokud je CPU vhodné pro práci s malým množstvím dat, GPU vítězí v úlohách, kde je potřeba zpracovávat informace mnohonásobně více – například při trénování neuronových sítí. Vzhledem k vysokým nárokům na výpočetní výkon je vývoj a provoz neuronových sítí značně složitý a nákladný [13].

1.3 Algoritmy klasifikace objektu

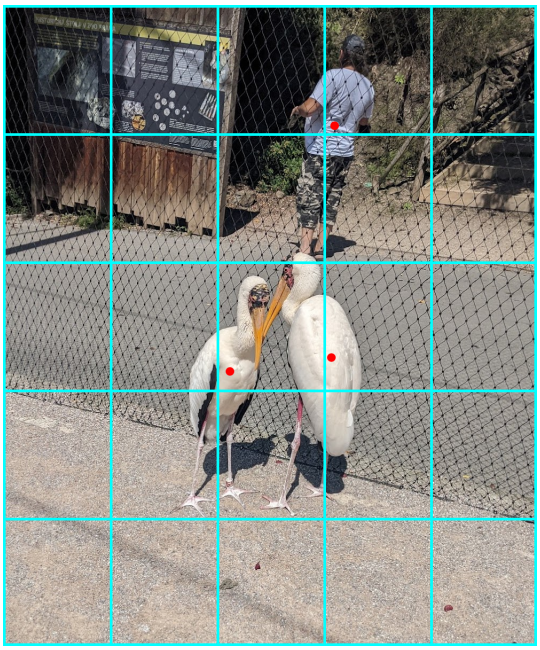
Další úlohou v počítačovém vidění je stanovit, zda je na obrázku nějaký objekt, a pokud ano, zjistit, o jaký typ se jedná. V závislosti na algoritmu může být poloha objektu v pixelovém souřadnicovém systému původního obrázku charakterizována souřadnicemi ohraničujícího rámečku, klíčovými body nebo obrysem objektu. Samotnou identifikací objektu se většinou zabývají umělé neuronové sítě. Po zpracování lze objekt zařadit do jedné z konečných množin tříd, které seskupují jednotlivé typy objektů, jako jsou auto, pes, strom atd.

V současné době existují především dva typy detektorů objektů. Na jednu stranu jsou dvoustupňové detektory (two-stage detectors), které v první fázi vyhodnocují pravděpodobnost výskytu objektů, poté je podmíněně klasifikuje. Na druhou stranu jsou jednostupňové detektory (single-stage detectors), které zpracují obrázek jako celek bez ohledu na falešně pozitivní výsledky. Takové modely dosahují nižší přesnosti, ale jsou mnohem rychlejší než dvoustupňové detektory objektů [18].

1.3.1 YOLO

Jedním z nejmodernějších jednostupňových systémů pro detekování objektu na snímku je YOLO (You Only Look Once). Algoritmus, jak již název napovídá, vyžaduje pouze jediné dopředné šíření neuronovou sítí k detekci objektů. Není potřeba vícekrát procházet obrázkem, jak to například dělá algoritmus Sliding Window [19].

Nejprve se obrázek rozděluje mřížkou S×S (Obr. 1.8). Každá její buňka bude detekovat objekty, které se v ní objeví. Pokud se například v určité buňce mřížky objeví střed objektu, bude tato buňka zodpovědná za jeho detekci.



Obr. 1.8: Rozdělení obrázku 1000×1200 mřížkou 5×5. Střed objektů je označen červeně.

Dále se určí ohraničující rámečky, které zvýrazňují pozice objektů v jednotlivé buňce (Obr. 1.9). Obecně je stanoveno, že takovým rámečkem je obdélník. Každá buňka teď má následující parametry:

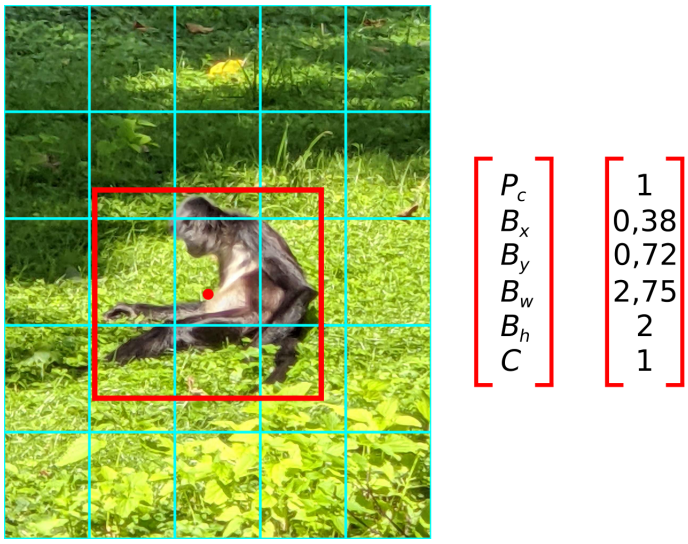
- P_c – pravděpodobnost přítomnosti objektu v buňce,
- B_w – šířka ohraničujícího rámečku,
- B_h – výška ohraničujícího rámečku,
- B_x, B_y – souřadnice středu ohraničujícího rámečku,
- C – třída objektu (např. člověk, pes, strom atd.).

Tedy na obrázku 1.9 zleva je vidět fotografie velikosti $160 \times 200px$. Jediný objekt tady je opice, jejíž ohraničující rámeček má šířku a výšku přesahující rozměry buňky 2,75× a 2×. Střed objektu je umístěn v souřadnicích (0,38; 0,72) vzhledem k levému hornímu rohu. Předpokládá se, že parametr `C` odpovídá za třídu objektů typu „opice“.

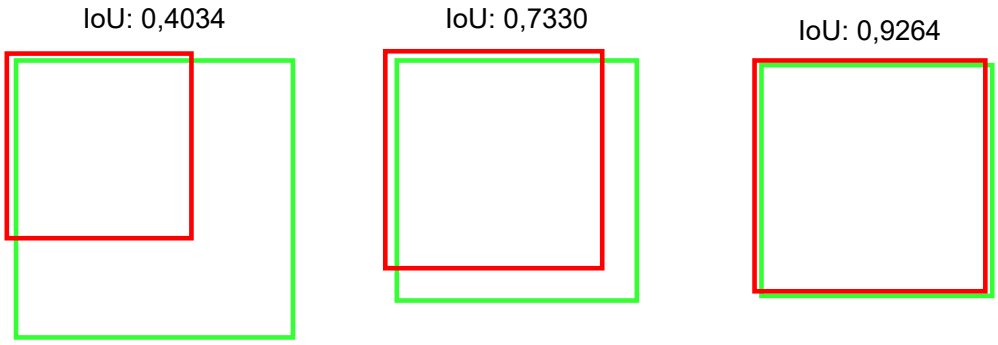
Potom YOLO algoritmus pomocí metriky Intersection over Union (IoU) stanoví, jak přesný byl předpovězený ohraničující rámeček ve srovnání se skutečným. Čím vyšší je hodnota IoU, tím přesnější je ohraničovací rámeček (Obr. 1.10).

Výpočet hodnoty IoU se provádí následovně:

$$IoU = \frac{\text{Oblast překrytí mezi ohraničujícími rámečky}}{\text{Oblast spojení mezi ohraničujícími rámečky}} \tag{1.2}$$



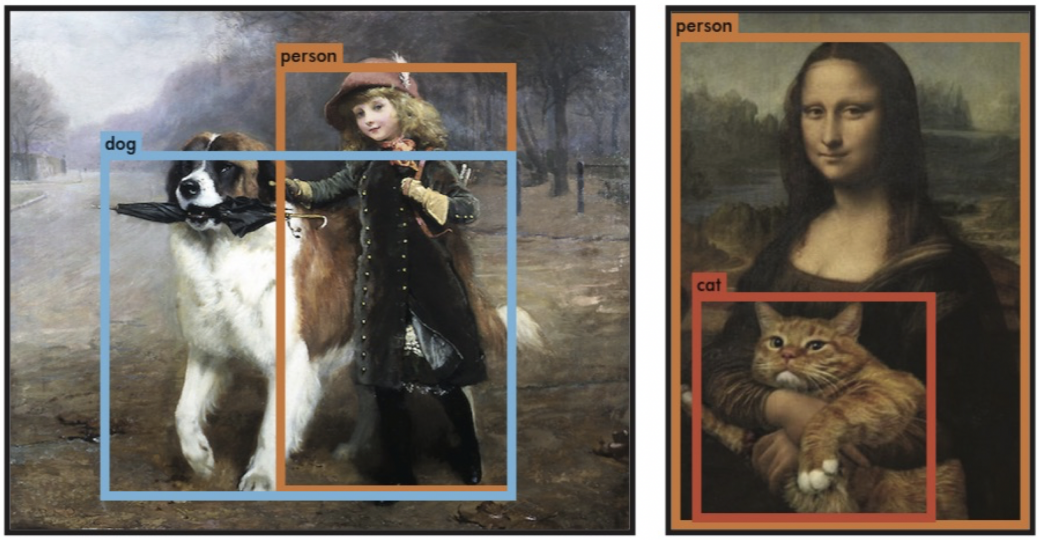
Obr. 1.9: Ukázka vypočtených parametrů ohraničujícího rámečku.



Obr. 1.10: Závislost hodnoty Intersection over Union na poloze rámečků. Seřazeno vzestupně.

Všechny ohraničující rámečky, jejichž hodnota IoU je menší než prahová hodnota (nejčastěji 0,5), se neberou v úvahu [20].

Výsledkem zpracování obrázku algoritmem YOLO je seznam ohraničujících rámečků s parametry popisujícími, kde se objekty nachází, jaká je jejich velikost a o jaké typy objektů se jedná (Obr. 1.11) [21].



Obr. 1.11: Výsledky zpracování obrazu algoritmem YOLO [21].

2 Implementace analyzátoru videí

S ohledem na široké nasazení operačního systému Windows na trhu bylo rozhodnuto se primárně zaměřit na implementaci desktopové aplikace pro uživatele operačního systému Windows. Nicméně většina vybraných technologií je podporována i operačními systémy typu UNIX, jako je například macOS nebo Linux, takže využívání aplikace na jakémkoliv z nich je možné.

Vzhledem k povaze úlohy, byl pro implementaci aplikace zvolen programovací jazyk Python. Python disponuje velkým množstvím knihoven, které jsou specializované na zpracování obrazu a práci s neuronovými sítěmi. Tyto knihovny jsou vysoce výkonné a nabízejí rozmanité funkcionality, které umožňují dosáhnout vysoké přesnosti a kvality výsledků. Výběr Python a jeho knihoven zároveň snižuje časovou náročnost na implementaci.

Videa, která byla použita k testování a ověření kvality navrženého řešení, byla poskytnuta vedoucím práce. Tato videa byla uložena v populárním formátu MPEG-4 Part 14, známém také jako MP4, a dosahovala vysokého rozlišení 2560×1920 pixelů. Soubory měly bitrate 6390 kb/s a framerate 30,05 fps, což zaručovalo plynulé a kvalitní záznamy. Pro zakódování videa byly použity moderní kodeky H.264 pro video a AAC(Advanced Audio Coding) pro zvuk, což zajišťovalo efektivní kompresi a zároveň zachovávalo vysokou kvalitu hlavně obrazového signálu [22].

Vzhledem k tomu, že byla tato videa pořízena z venkovního prostoru, poskytují cenný materiál k pozorování pohybů parkujících aut a osob. Tyto prvky jsou pro testování zpracování videa významné, protože představují reálné situace, kterým by mělo navržené řešení čelit. Díky těmto videím bylo možné vyzkoušet různé funkce a vlastnosti navržené aplikace a ověřit, zda dokáže úspěšně detekovat pohyby objektů a zpracovávat videa různých formátů.

2.1 Výběr technologií

Při výběru technologií pro implementaci projektu byly zohledněny specifické požadavky na funkcionalitu, efektivitu, výkon a udržitelnost aplikace. Výběr vhodných technologií byl klíčový pro úspěšnou realizaci projektu a dosažení požadovaných výsledků. Je třeba mít na paměti, že některé technologie mohou být nekompatibilní s konkrétním operačním systémem nebo hardwarem, což může vést k problémům s integrací a interoperabilitou. Například použití hardwarové a softwarové architektury CUDA (Compute Unified Device Architecture), která umožňuje na vybraných grafických výpočetních jednotkách s vysokou efektivitou spouštět programy pracující s obrazovými daty, je omezeno pouze na vybrané grafické karty od firmy NVIDIA

[23]. Proto je důležité pečlivě zvažovat výběr technologií a zvolit takové, které budou nejvhodnější pro danou aplikaci a budou zajišťovat optimální výkon a spolehlivost.

2.1.1 Knihovna pro zpracování obrazu

Pro přesné a efektivní řízení zpracování obrazu se používá balíček `opencv-python`, což je vazební knihovna pro celosvětově známou knihovnu `OpenCV` (Open Source Computer Vision) původně napsanou v jazyce C++ [24, 25]. Vazební knihovny jsou speciálním typem knihoven, které umožňují používat funkce a metody z jiného programovacího jazyka, než který je použit pro samotnou knihovnu. Díky této „binding“ knihovně mohou vývojáři používat funkce a metody z `OpenCV` přímo v Python, aniž by museli sami implementovat veškerou funkcionalitu v jazyce. Tato knihovna obsahuje mnoho zabudovaných modulů a metod, které implementují populární algoritmy pro transformaci a analýzu obrazových dat. Příklady takových metod mohou být modul `cv2.createBackgroundSubtractorMOG2`, který je implementací algoritmu „Mixture of Gaussian 2“ a metoda pro zmenšení rozlišení obrázku - `cv2.resize`.

Balíček `opencv-python` nemá téměř žádný ekvivalent. Většina dalších knihoven je založena na této a pouze přidává další funkce. Například knihovna `imutils`, která je přímo závislá na `opencv-python`, může nabídnout sofistikovanější verzi metody `cv2.resize` - `imutils.resize`, která místo dvou hodnot - výšky a šířky - může přijímat pouze jednu z nich a bude se snažit zachovat poměr stran [26]. Funkcionalita původního balíčku je zcela dostačující, a v případě nutnosti je možné dopočítat hodnotu druhé strany samostatně. Knihovna `imutils` byla proto považována za zbytečnou závislost projektu.

2.1.2 Knihovna pro rozpoznávání objektů

Při rozpoznávání objektů byla použita knihovna `ultralytics`. Tato knihovna umožňuje využívat klíčový produkt firmy Ultralytics - model konvoluční neuronové sítě `YOLOv8`, který představuje nejnovější verzi modelu „You Look Only Once“ [27]. Výsledkem zpracování obrázků pomocí modelu `YOLOv8` je seznam objektů, včetně informací o jejich typech, pravděpodobnostech detekce a poloze na obrázku. Balíček je také schopen na základě výsledků automaticky generovat obrázek, na kterém jsou popsány všechny nalezené objekty.

Knihovna podporuje zpracování mediálních vizuálních dat pomocí výpočetních schopností grafické karty. Toto nastavení lze použít k urychlení procesu rozpoznávání a klasifikace objektů a ke zefektivnění využití zdrojů počítače.

Jedinou nevýhodou je sběr anonymních dat o průběhu zpracování, který je výchozím nastavením knihovny. Tuto funkci lze jednoduše vypnout změnou jednoho parametru v konfiguračním souboru, který knihovna vytvoří při prvním spuštění.

V závislosti na použitém operačním systému se cesta k konfiguračnímu souboru může lišit:

- `C:\Users{username}\AppData\Roaming\Ultralytics\settings.yaml` – pro Windows;
- `/Users/{username}/Library/ApplicationSupport/Ultralytics/settings.yaml` – pro macOS;
- `/home/{username}/.config/Ultralytics/settings.yaml` – pro Linux;

Po úspěšném načtení daného souboru ve formátu YAML je nutné provést úpravu, která zahrnuje nastavení parametru `sync` na hodnotu `false`. Tento specifický parametr slouží k deaktivaci odesílání anonymních informací o událostech modulu `ultralytics.hub.utils.Events` na servery firmy Ultralytics [28]. Z tohoto důvodu bude aplikace chráněna před jakýmkoli neoprávněným sběrem dat, které by jinak mohly být shromážděny a využity bez vědomí uživatele.

2.1.3 Nastroj pro střih videa

Za účelem úpravy videa v definovaném časovém rámci byl využit nástroj „ffmpeg“. Jedná se o aplikaci s otevřeným zdrojovým kódem sloužící k zpracování multimediálních souborů, včetně videí a audia. Jeho široká použitelnost spočívá v možnosti provádět základní editační úpravy videí, jako je střih, spojování, změna formátu (transkódování) a skalování videí. Všechny tyto funkce lze ovládat prostřednictvím příkazové řádky operačního systému [29].

2.1.4 Knihovna pro sestavení GUI

Pro sestavení GUI (Graphical User Interface) bylo potřeba najít knihovnu, která umožní dynamicky měnit obsah okna a flexibilně nastavovat reakce aplikace na různé typy událostí, jako jsou například kliknutí na tlačítko nebo pokus o zavření okna. Dalším kritickým faktorem je schopnost nástroje pracovat paralelně v několika vláknech. Nejčastějším příkladem použití této funkce je aktualizace indikátoru průběhu probíhajícího na pozadí procesu. Všechny tyto požadavky splňuje knihovna `PyQt6`, což je šestá verze vazební knihovny pro populární multiplatformní framework `Qt` [30, 31]. V kontextu tohoto řešení se slovo „multiplatformní“ vztahuje k tomu, že aplikace, bez ohledu na operační systém, kde je spuštěna, bude vypadat podobně. Vývojáři tedy mají díky této knihovně přístup k široké paletě widgetů a mohou snadno přizpůsobit vzhled své aplikace. `PyQt6` je navíc stabilní a spolehlivá knihovna, která se v komunitě Python hojně používá již mnoho let [32].

2.2 Algoritmus zpracování videa

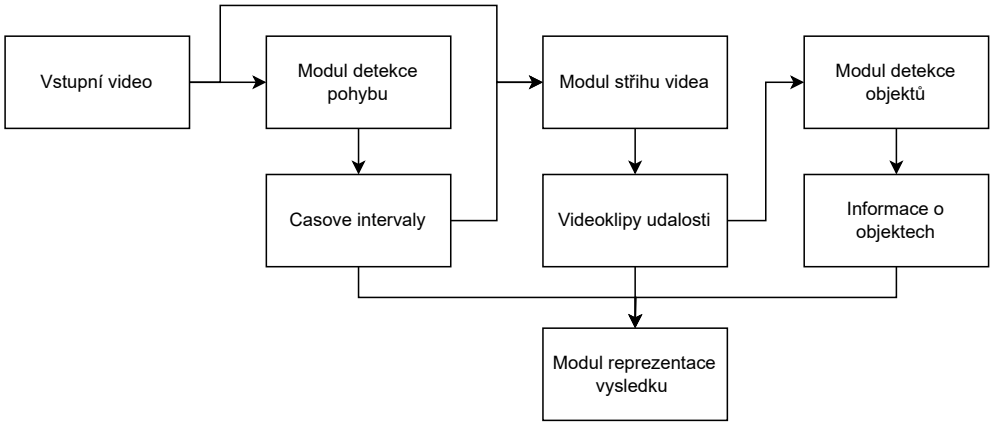
Video se skládá ze sekvence snímků, což jsou statické obrazy složené z pixelů. Každý pixel má matematickou hodnotu popisující jeho barvu, kterou lze reprezentovat jako hodnotu RGB (červená, zelená, modrá). Tyto hodnoty se používají k manipulaci s obrazem pro účely analýzy nebo úpravy pomocí různých technik zpracování obrazu.

Hlavním cílem projektu bylo dosáhnout vyšší rychlosti zpracování ve srovnání s manuální analýzou videa. K dosažení tohoto cíle byla zvolena metoda, při níž jsou snímky zpracovány pouze jednou a poté je výsledek sestaven z získaných dat.

Nejprve proběhne analýza vstupního videa s cílem získat časové intervaly jednotlivých událostí v něm. Tyto intervaly slouží k rozdělení původního videa na videoklipy, z nichž každý reprezentuje jednotlivou akci ve videozáznamu. Následně jsou vybrané snímky z jednotlivých videoklipů vyhodnoceny pro identifikaci objektů, které pohyb způsobily. Nakonec jsou všechny mezivýsledky zpracovány a vytvořena zpráva o zpracování celého videa.

Celý proces zpracování videa lze tedy rozdělit na následující fáze:

1. Detekce pohybu.
2. Ořezávání videoklipů podle časových intervalů pohybu.
3. Detekce objektů ve videoklipech.
4. Reprezentace výsledků zpracování.



Obr. 2.1: Schéma algoritmu zpracování videa v aplikaci.

2.2.1 Detekce pohybu

Každý snímek videa musí projít několika kroky předzpracování, aby byl připraven pro detekci pohybu. Prvním krokem je snížení rozlišení každého snímku. Tato technika zjednodušuje obrazová data a umožňuje rychlejší zpracování, což v konečném

důsledku vede ke zlepšení celkového výkonu. Dalším krokem v procesu je převod barevné mapy snímku na stupnici šedé pomocí techniky zvané „greyscale conversion“. Tento krok je velmi důležitý, protože umožňuje plynulejší zpracování obrazových dat. Kdyby byla zachována barevná mapa, výsledné zpracování by bylo mnohem pomalejší a náročnější na výpočetní výkon. Poté se použije technika „Gaussian blur“, aby byly odstraněny případné šumy na snímku. Celkově tyto kroky zajišťují, že každý snímek videa je připravený pro vyhodnocení pohybu s maximální výkonností [33].

Po dokončení předzpracování a vyčištění snímků se používá algoritmus „Mixture of Gaussian 2“ ke detekci pohybu. Tento algoritmus slouží k identifikaci oblastí, kde se pravděpodobně nachází objekty, které nejsou součástí pozadí snímku. Rozhodnutí o tom, zda se v dané oblasti objevuje pohyb, je založeno na výpočtu čtvercové Mahalanobisovy vzdálenosti mezi pixelem a modelem pozadí a prahové hodnotě této vzdálenosti [34, 35]. Výsledkem je binární maska, kde jedničkami jsou označeny oblasti popředí a nulami jsou označeny oblasti pozadí. Díky této masce je aplikace schopna detekovat, zda v obraze došlo k pohybu. Tato informace umožňuje stanovit časové úseky videa, ve kterých k události došlo. K tomu se využívá algoritmus vývojový diagram, kterého je zobrazen na obrázku 2.2.

Následuje proces slučování a filtrace časových intervalů, kdy jsou intervaly, které jsou navzájem blíže než N_1 sekund, sloučeny. Pokud výsledný interval je kratší než N_2 sekund, bude odstraněn ze seznamu. Tento proces zajistí, že jsou výsledky detekce pohybu co nejpreciznější a relevantní. N_1 a N_2 jsou celočíselné hodnoty, které lze měnit v rámci konfigurace aplikace. Na obrázku 2.3 jsou zobrazeny vývojové diagramy těchto dvou procesů.

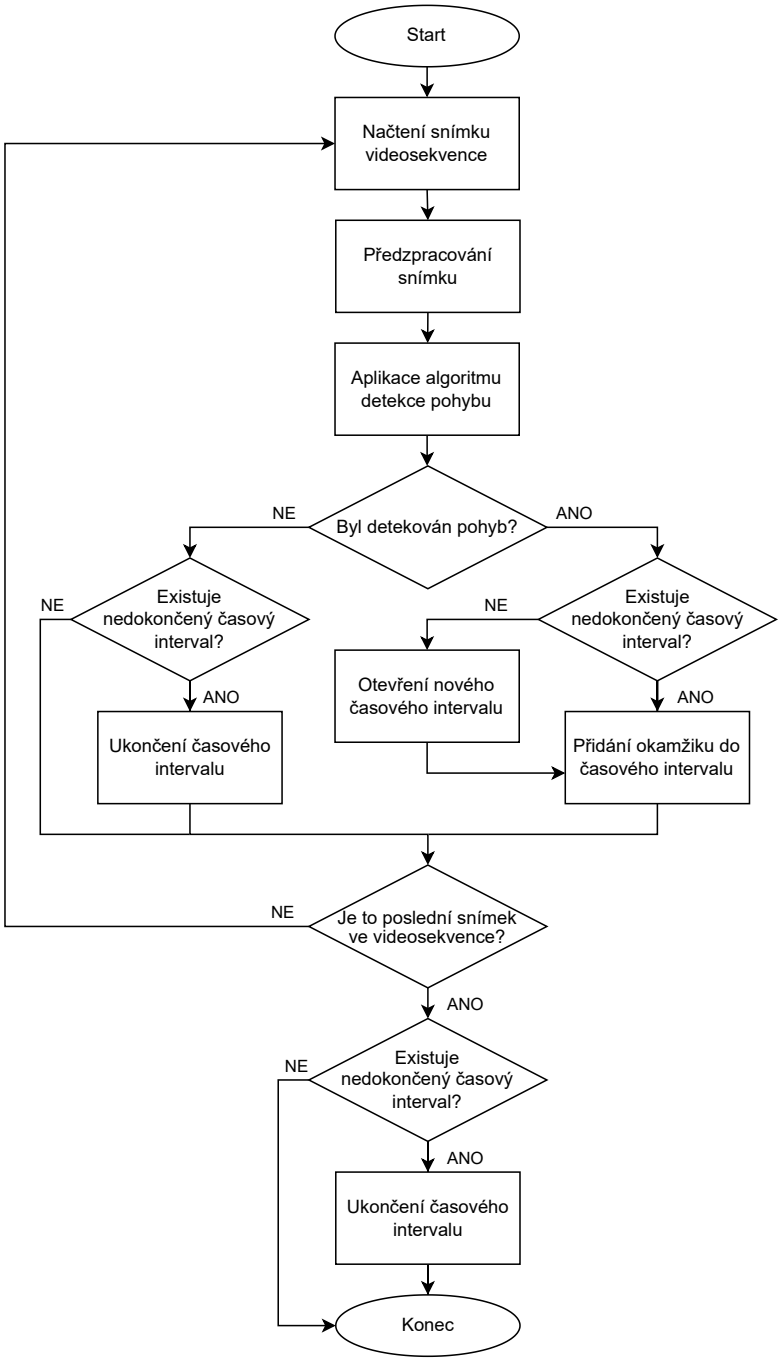
2.2.2 Ořezávání videoklipů

V modulu „Střih videa“ se na základě stanovených časových intervalů vygenerují videoklipy. Každý videoklip představuje jednu specifickou událost, která se vyskytla v původní videonahrávce. Proces vytváření těchto videoklipů probíhá pomocí specializovaného nástroje `ffmpeg`, který je integrován do algoritmu. Pro ovládání nástroje `ffmpeg` jsou použity příkazy v příkazové řádce, které jsou vyvolány prostřednictvím modulu `Popen` z Python knihovny `subprocess`.

Konstruktor třídy `Popen` přijímá jako vstupní argument seznam částí příkazu, který je potřeba spustit. Definice funkce `ffmpeg_extract_subclip`, která připravuje a spouští příkaz pro ořezávání videa, se nachází v souboru `app/utils/ffmpeg.py`.

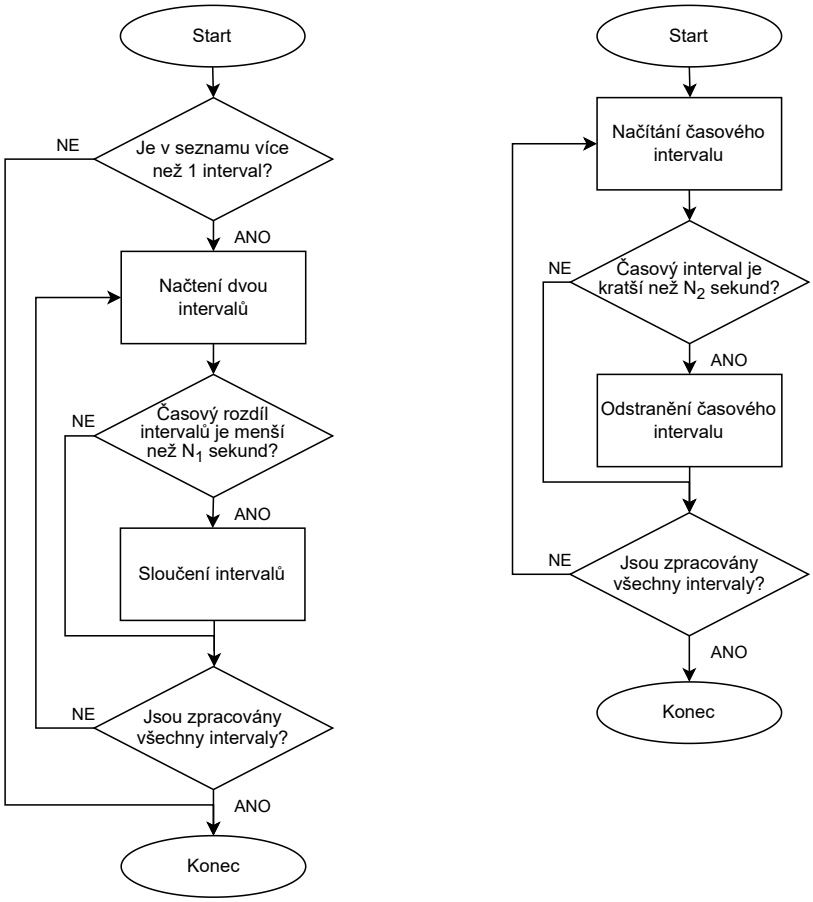
Na vstup této funkce se předávají čtyři parametry:

- `file_path` – cesta k původnímu videu, které je třeba oříznout.
- `t1` – Desetinné číslo představující počet milisekund, na kterých se má začít ořezávání videa.



Obr. 2.2: Vývojový diagram detekce časových intervalů, ve kterých došlo k pohybu

- `t1` – Desetinné číslo představující počet milisekund, na kterých se má ukončit ořezávání videa.
- `target_path` – cesta k souboru, kam bude uložen videoklip.



Obr. 2.3: Vývojové diagramy slučování (zleva) a filtrování (zprava) časových intervalů

Obr. 2.4: Výsledky zpracování snímku pomocí modulu detekce pohybu. Zleva: původní obrázek. Uprostřed: obrázek po předzpracování. Vpravo: binární maska reprezentující pohybující se objekt.

Na základě dokumentace `ffmpeg` [25] se z těchto argumentů sestaví příkaz pro ořezávání videa, který se následně spustí.

Výpis 2.1: Defenice funkce pro stříhání videa.

```
def ffmpeg_extract_subclip(
    file_path: str,
    t1: float,
    t2: float,
    target_path: str,
) -> None:
    ffmpeg_binary = get_executable()
    cmd = [
        ffmpeg_binary, "-y",
        "-ss", "%0.2f" % t1,
        "-i", file_path,
        "-t", "%0.2f" % (t2 - t1),
        "-map", "0",
        "-vcodec", "copy",
        "-acodec", "copy",
        target_path,
    ]
    subprocess_call(cmd)
```

2.2.3 Detekce objektů

Vzhledem k tomu, že detekce objektů je časově náročný proces, je nutné zvolit vhodnou strategii pro optimalizaci výpočetního času. Proto je přístup s výběrem klíčových snímků, které nejpravděpodobněji obsahují důležité informace o pohybujících se objektech, efektivním řešením. Z každého videoklipu, vytvořeného v předchozí fázi, se vybírá okamžik uprostřed události. Následně se z videa získávají snímky, které odpovídají zvoleným časovým značkům.

Tato fáze zahrnuje využití předtrénovaného modelu konvoluční neuronové sítě `YOLOv8` s cílem detekovat objekty na snímcích. Po předání klíčových snímků modelu se provede analýza obrazových dat a následně jsou identifikovány objekty na snímcích. Výstupem tohoto modulu je seznam typů objektů s pravděpodobností predikce a přiložené obrázky s vykreslenými a podepsanými hranicemi těchto objektů.

2.2.4 Reprezentace výsledků

Na závěr celého procesu detekce pohybu a objektů následuje krok reprezentace všech výsledků, výsledky, které byly získány během zpracování. Celkový výstup programu se skládá ze tří částí: videoklipů událostí, obrázků pohybujících se objektů pro každé ze zvolených videonahrávek a textové zprávy shrnující všechny výsledky v uživatelsky přívětivém formátu.

Obr. 2.5: Výsledný obrázek generovaný modulem detekce objektů.

Videoklipy událostí jsou vytvořeny na základě časových značek začátku a konce pohybů, které byly získány v modulu detekce pohybu. Pokud uživatel zvolil přidání času před a po události, toto nastavení se projeví při stříhání videa v daném kroku. Tyto videoklipy představují vizuální reprezentaci událostí, což může být užitečné pro pochopení kontextu a posloupnosti akcí, které se odehrály. Tyto soubory jsou ukládány do složky „clips“ a jsou pojmenovány tak, aby bylo možné rozlišit, ke kterému originálnímu videu patří a o jaký časový interval se jedná.

Obrázky pohybujících se objektů byly vygenerovány při detekování objektů a jsou uloženy do složky „images“. Tyto snímky jsou pojmenovány podobným způsobem jako videoklipy a zobrazují ohraničující rámečky objektů spolu s odpovídající třídou a pravděpodobností předpovědi.

Textová zpráva je generována v formátu HTML (HyperText Markup Language) a poskytuje uživateli přehled o všech zpracovaných videích a jejich výsledcích. Obsahuje časové značky začátku, konce a vypočtenou délku trvání procesu zpracování. Následně jsou uvedeny informace pro každé ze zvolených videonahrávek, jako je počet detekovaných pohybů, které jsou níže vypsány v podobě tabulky. Z této tabulky lze snadno vyčíst informace o jednotlivých objektech a jejich pravděpodobnostech. Navíc jsou zde poskytnuty odkazy na videoklipy a obrázky vztahující se k danému intervalu. Tento soubor má název „report.html“ a ukládá se do kořenové složky pojmenované časem, kdy zpracování bylo spuštěno a lze ho otevírat pomocí jakéhokoliv prohlížeče.

Zprava o detekci pohybu

Začátek zpracování: *08.05.2023 18:36:47*

Konec zpracování: *08.05.2023 18:41:32*

Delka zpracování: *0:04:44*

video720p.mp4

Počet detekovaných pohybů: *7*

Čas začátku	Čas konce	Počet detekovaných objektů	Objekty	Video události	Fotografie události
0:00:00	0:00:06	1	person (66.90%)	Otevřít	Otevřít
0:00:39	0:01:04	1	truck (67.39%)	Otevřít	Otevřít
0:23:59	0:24:06	0	nerozpoznano	Otevřít	Otevřít
0:28:19	0:28:26	1	truck (65.85%)	Otevřít	Otevřít
0:28:37	0:28:52	0	nerozpoznano	Otevřít	Otevřít
0:29:27	0:29:32	1	person (78.91%)	Otevřít	Otevřít
0:29:46	0:29:56	1	person (77.87%)	Otevřít	Otevřít

Obr. 2.6: Obsah souboru „report.html“ po provedení analýzy videa „video720p.mp4“.

Příklad struktury adresáře „results“, kde jsou uloženy výsledky po zpracování dvou souborů „video1.mp4“ a „video2.mp4“, které bylo spuštěno 26.4.2023 v 19:27:18, je následující:

```
results_2023-04-26_19-27-18
├── video1
│   ├── clips
│   │   ├── video1_0_00_00-0_00_15.mp4
│   │   └── video1_0_01_27-0_01_32.mp4
│   └── images
│       ├── video1_0_00_00-0_00_15.png
│       └── video1_0_01_27-0_01_32.png
├── video2
│   ├── clips
│   │   └── video2_0_00_07-0_00_12.mp4
│   └── images
│       └── video2_0_00_07-0_00_12.png
└── report.html
```

2.3 Vytvoření grafického uživatelského rozhraní

Hlavním cílem grafického uživatelského rozhraní desktopové aplikace je poskytnout uživateli jednoduchý a intuitivní způsob, jak ovládat aplikaci, nastavit parametry zpracování a pohodlně zobrazit výsledná data. Při návrhu GUI byly zohledněny tyto tři aspekty s cílem vytvořit uživatelsky přívětivé prostředí, které by připomínalo běžné grafické instalátory, které jsou široce využívány na desktopových počítačích.

2.3.1 Požadované funkce rozhraní

V rámci návrhu grafického uživatelského rozhraní pro výslednou aplikaci byly stanoveny následující cíle:

- Volba videosouborů uživatelem, které je potřeba zpracovat;
- Nabídnout základní možnosti nastavení formátu výstupních dat;
- Vyhodnocení záznamu z hlediska výskytu pohybujících se objektů;
- Uživatelsky přívětivá reprezentace výsledků.

Na základě těchto požadavků byla implementována následující čtyři uživatelská okna:

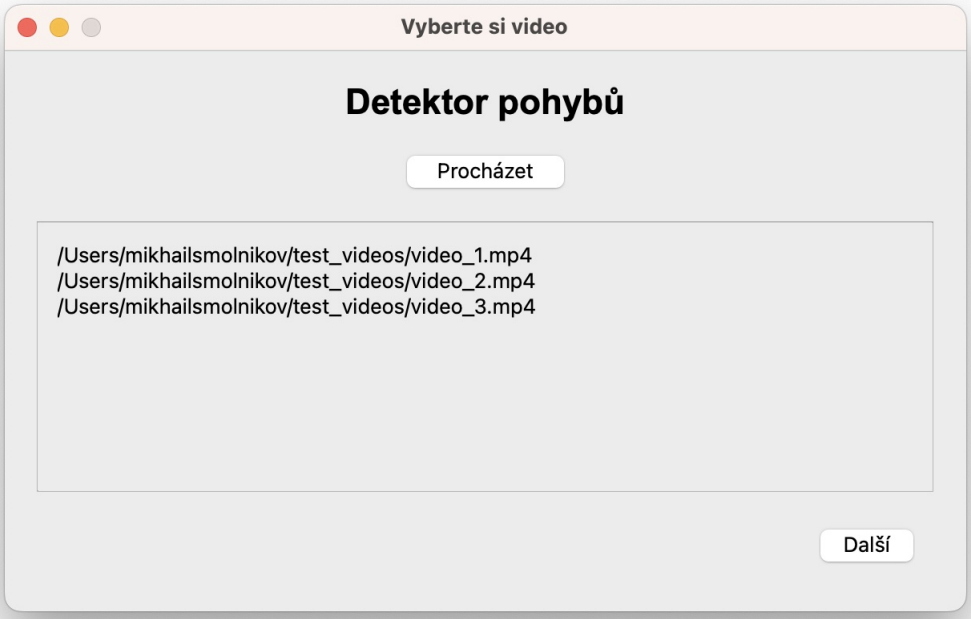
1. Výběr videa;
2. Nastavení aplikace;
3. Výpis informací o probíhajícím procesu;
4. Zobrazení výsledků.

2.3.2 Okno „Výběr videa“

Toto okno 2.7 vyzývá uživatele k výběru videonahrávek, které jsou nutné pro zpracování. Kliknutím na tlačítko „Procházet“ se otevře dialogové okno pro výběr videosouborů, přičemž budou zobrazeny pouze soubory s podporovanými formáty videosekvencí (`.mkv` a `.mp4`). Po zvolení cesty k souboru jsou informace zobrazeny v dolní části obrazovky. Po ověření souboru stiskem tlačítka „Další“ se proces pokračuje dále.

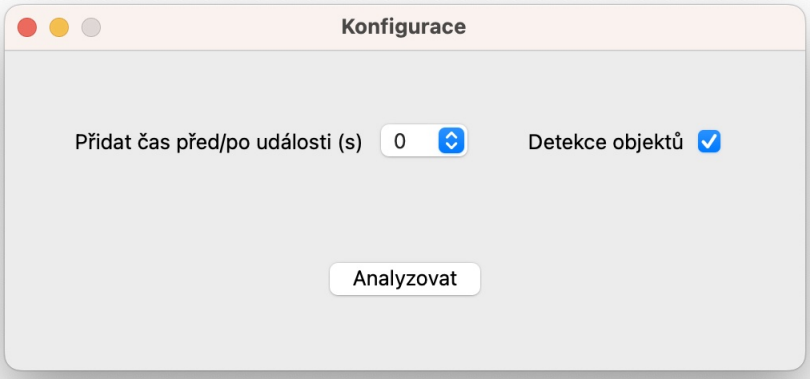
2.3.3 Okno „Nastavení aplikace“

Po zvolení videonahrávek následuje konfigurační okno 2.8, kde uživatel může specifikovat další detaily zpracování. Mezi nimi patří například možnost přidání časového úseku před a po samotné události, který se zahrne do výstupního videoklipu, a tím uživatel získá více informací o události. Důležitou volbou je také zaškrtávací políčko „Detekce objektů“, které umožňuje řídit, zda se modul detekce objektů použije během procesu zpracování. Ve výchozím nastavení jsou události stříhány bez přidání



Obr. 2.7: Uživatelské rozhraní pro výběr video souborů.

času před a po, a detekce objektů je vypnuta. Po dokončení konfigurace kliknutím na tlačítko „Další“ se uživatel přesune na další obrazovku.



Obr. 2.8: Uživatelské rozhraní pro volbu parametrů budoucí analýzy.

2.3.4 Okno „Výpis informací o probíhajícím procesu“

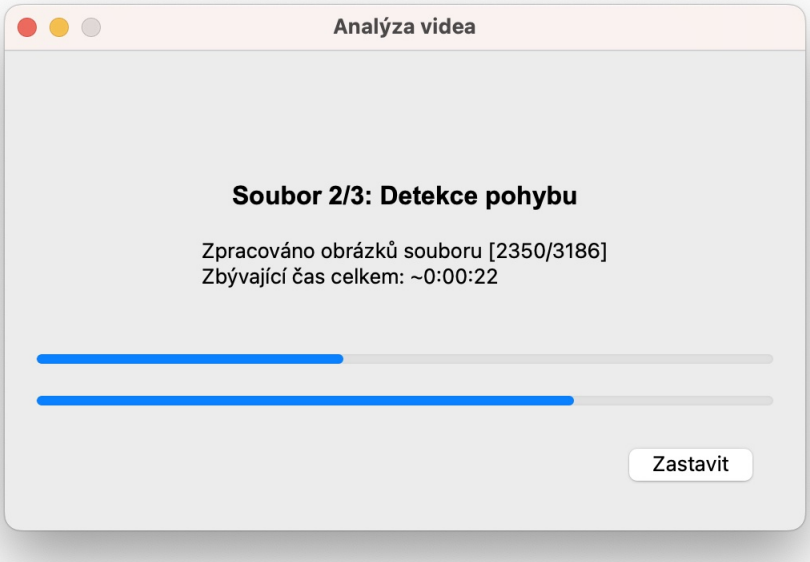
Hned po otevření tohoto okna (Obr. 2.9) začne algoritmus zpracovávat video. Nahoře se zobrazuje aktuální fáze analýzy:

- Detekce pohybu – určení časových okamžiků, kdy došlo k pohybu.
- Detekce objektů – klasifikace pohybujících se objektů.
- Střih videa – oříznutí videa na základě vypočtených časových úseků.
- Dokončeno – označuje stav, kdy bylo zpracování videa dokončeno.

Pro každou fázi v jednotlivých souborech je zobrazen počet již zpracovaných prvků ve srovnání s celkovým počtem prvků v dané fázi. Prvky mohou být snímky nebo časové úseky. Níže je uveden vypočtený odhadovaný čas dokončení celé analýzy.

V dolní části obrazovky jsou k dispozici dva indikátory průběhu pro snadné sledování procesu. První indikátor ukazuje celkový postup zpracování všech souborů. Druhý indikátor zobrazuje průběh jednotlivých fází.

Pro přerušení procesu zpracování lze použít tlačítko „Zastavit“. Potvrzením bude proces ihned zastaven bez ohledu na aktuální fázi zpracování.



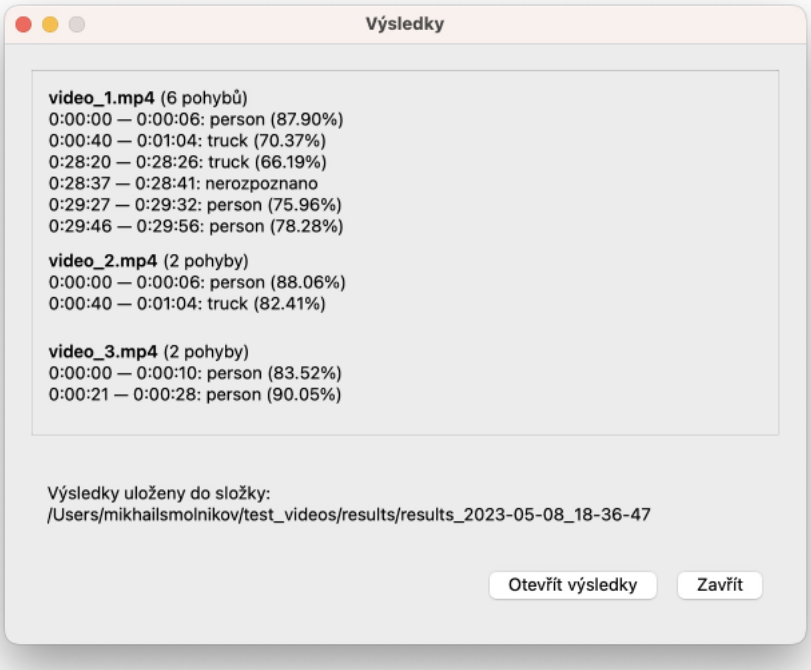
Obr. 2.9: Uživatelské rozhraní pro sledovaní průběhu analýzy videa.

2.3.5 Okno „Zobrazení výsledků“

Po dokončení zpracování všech videonahrávek se uživatel dostane na finální obrazovku aplikace, která poskytuje přehledný souhrn zpracovaných dat. Zde může uži-

vatel vidět počet detekovaných pohybů v každé videonahrávce a podrobné informace o detekovaných událostech a objektech, včetně časových úseků, kdy se vyskytly.

Dolní část obrazovky obsahuje informaci o umístění složky s výsledky zpracování a tlačítko „Otevřít výsledky“, které umožňuje uživateli rychle otevřít složku s výsledky a prohlédnout si zpracovaná data podrobněji. Pro ukončení aplikace slouží tlačítko „Zavřít“, které umožňuje uživateli bezproblémově opustit aplikaci po dokončení práce.



Obr. 2.10: Uživatelské rozhraní pro zobrazování výsledků zpracování videí.

2.4 Konfigurace aplikace

V této části práce se nacházejí informace o konfiguraci aplikace, které ovlivňují chování programu. Tyto hodnoty jsou definovány v souboru `app/constants.py` a jsou přístupné z jakéhokoliv místa v kódu po přidání importu modulu `app.constants`:

```
from app import constants as c
print(c.RESIZE_IMAGE_WIDTH)
```

Konstanty jsou zodpovědné za používání metod knihoven, jako je například `RESIZE_IMAGE_WIDTH`, jejichž hodnota je vkládána do metody `cv2.resize`, a interní

logiku aplikace, jako je například `UPDATE_MOTION_DETECTION_STATUS_INTERVAL`, jejíž hodnota ovlivňuje frekvenci posílání stavů implementovanou v rámci aplikace.

Je důležité mít na paměti, že jakákoliv změna hodnoty vyžaduje restartování aplikace, aby se projevila v jejím chování. Správné nastavení konstant je důležité pro optimalizaci výkonu aplikace a její správnou funkčnost. Proto je nutné provádět změny s velkou opatrností a zvažovat dopady na celkovou funkčnost aplikace.

Konstanty jsou rozděleny do jednotlivých sekcí podle oblastí, ve kterých působí. Vždy jsou uvedeny popis, název proměnné a její hodnota.

2.4.1 Parametry detekce pohybu

Šířka obrázku pro detekci pohybu v pixelech (nižší hodnoty jsou rychlejší).

```
RESIZE_IMAGE_WIDTH = 640
```

Výška obrázku pro detekci pohybu v pixelech (nižší hodnoty jsou rychlejší).

```
RESIZE_IMAGE_HEIGHT = 640
```

Počet pixelů v oblasti, která je považována za pohyb (nižší hodnoty jsou citlivější na změny).

```
CONTOUR_SIZE_THRESHOLD = 100
```

Tolerance pro časový rozdíl mezi dvěma událostmi v sekundách.

```
JOIN_INTERVALS_THRESHOLD = 10
```

Minimální délka časového intervalu v sekundách.

```
MIN_MOTION_INTERVAL_LENGTH = 3
```

Počet snímků, které se použijí pro model pozadí (nižší hodnoty jsou citlivější na změny).

```
BACKGROUND_SUBTRACTION_HISTORY = 500
```

Detekce stínů (pokud je `True`, stíny jsou detekovány a označeny jako pohyb, zpomaluje proces).

```
BACKGROUND_SUBTRACTION_DETECT_SHADOWS = False
```

Prahová hodnota pro korelaci modelu pozadí s aktuálním snímkem (nižší hodnoty jsou citlivější na změny).

```
BACKGROUND_SUBTRACTION_THRESHOLD = 100
```

2.4.2 Parametry detekce objektů

Prahová hodnota důvěryhodnosti detekce objektů.

```
OBJECT_DETECTION_CONFIDENCE_THRESHOLD = 0.5
```

Seznam tříd pro detekci (prázdný seznam znamená vypnutí filtru). Aktuální seznam tříd je v souboru `app/CLASSES.txt`.

```
OBJECT_DETECTION_DETECT_ONLY_CLASSES = (
    "person", "car", "truck", "bus", "bicycle", "motorbike"
)
```

2.4.3 Parametry GUI

Počet snímků mezi každou aktualizací stavu procesu detekce pohybu.

```
UPDATE_MOTION_DETECTION_STATUS_INTERVAL = 50
```

Povolené formáty videa.

```
VIDEO_FILE_EXTENSIONS = (".mp4", ".mkv")
```

2.4.4 Parametry logování

Úroveň logování (jedno z `"DEBUG"`, `"INFO"`, `"WARNING"`, `"ERROR"`, `"CRITICAL"`).

```
LOGGING_LEVEL = "INFO"
```

3 Měření

V rámci projektu, jehož hlavním cílem je zrychlení analýzy videonahrávek, byla provedena důkladná měření jednotlivých částí procesu zpracování. Tato měření jsou klíčová pro ověření, zda výsledná desktopová aplikace splňuje stanovené požadavky na rychlost a efektivitu. Výsledky měření poskytují důležité informace pro další vývoj aplikace a optimalizaci jednotlivých procesů.

Je nutno zdůraznit, že měření proběhlo na počítači Apple MacBook Air s procesorem M1, bez použití výhod platformy CUDA. Je tedy třeba vzít v úvahu, že výsledky se mohou výrazně lišit v závislosti na specifikacích použitého zařízení a konkrétní konfiguraci aplikace.

Bylo rozhodnuto otestovat časovou náročnost jednotlivých modulu aplikace na čtyřech různých rozlišeních videí: 2560×1920, 1920×1080, 1280×720 a 640×360. Tyto rozlišení jsou mezi uživateli nejpopulárnější a často se používají v kamerových systémech.

3.1 Výkonnost modulu detekce pohybu

Během testování výkonu modulu pro detekci pohybu pro každé ze čtyř testovaných videi bylo vybráno prvních 100 snímků, na nichž byl zřejmý pohyb. Tyto snímky byly pak systematicky načítány a zpracovávány metodou `MotionDetector.process_frame`, která provádí veškeré výpočty, včetně předzpracování a detekce pohybu.

Testování se provádělo pomocí následujícího kódu:

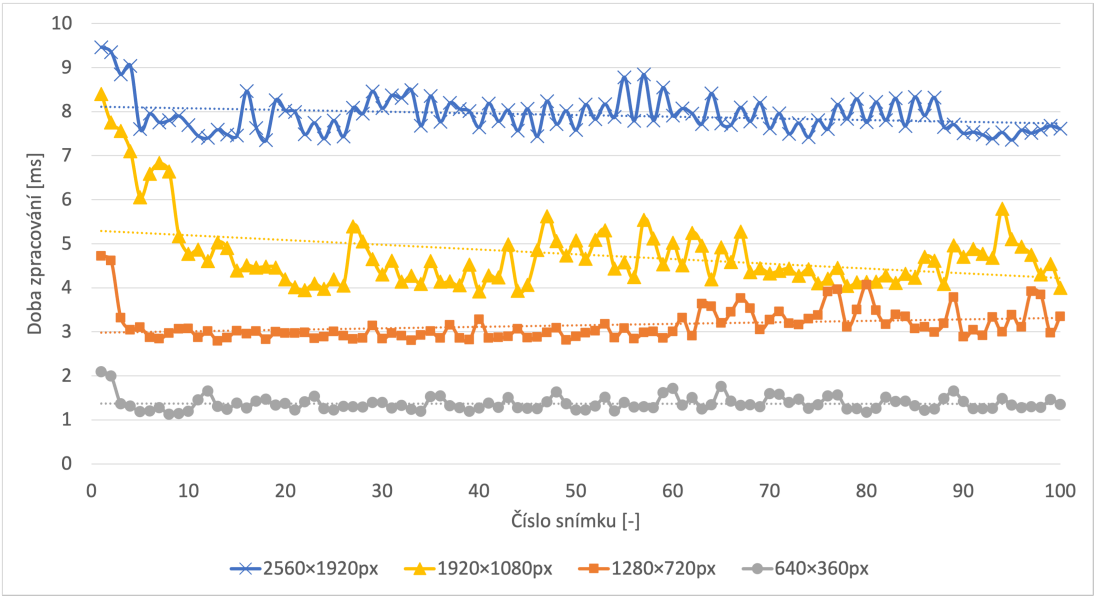
```
import time
import cv2
from app.services.motion_detector import MotionDetector

detector = MotionDetector()
cap = cv2.VideoCapture("test.mp4")

for _ in range(100):
    start_time = time.perf_counter()
    _, frame = cap.read()
    contours = detector.process_frame(frame)
    end_time = time.perf_counter()
    processing_time = end_time - start_time
```

Na základě vypočtených časových hodnot byl sestaven graf (Obr. 3.1) a vypočten průměr a medián časové náročnosti pro každé z rozlišení (Tab. 3.1). Z těchto statistických dat lze vyčíst, že čas potřebný na zpracování snímku přímo závisí na

počtu pixelů v něm. Větší časové hodnoty v první části grafu lze vysvětlit specifikou algoritmu `Mixture of Gaussian`. Tento algoritmus začne objekty správně vyhodnocovat až po zpracování několika prvních snímků.



Obr. 3.1: Závislost doby zpracování snímků modulem detekce pohybu na rozlišení videa.

Tab. 3.1: Doba zpracování snímků modulem detekce pohybu.

Rozlišení videa	**Průměrná doba**	**Medián doby**
640×360 px	1,366 ms	1,323 ms
1280×720 px	3,146 ms	3,016 ms
1920×1080 px	4,755 ms	4,528 ms
2560×1920 px	7,921 ms	7,803 ms

3.2 Výkonnost modulu ořezávání videa

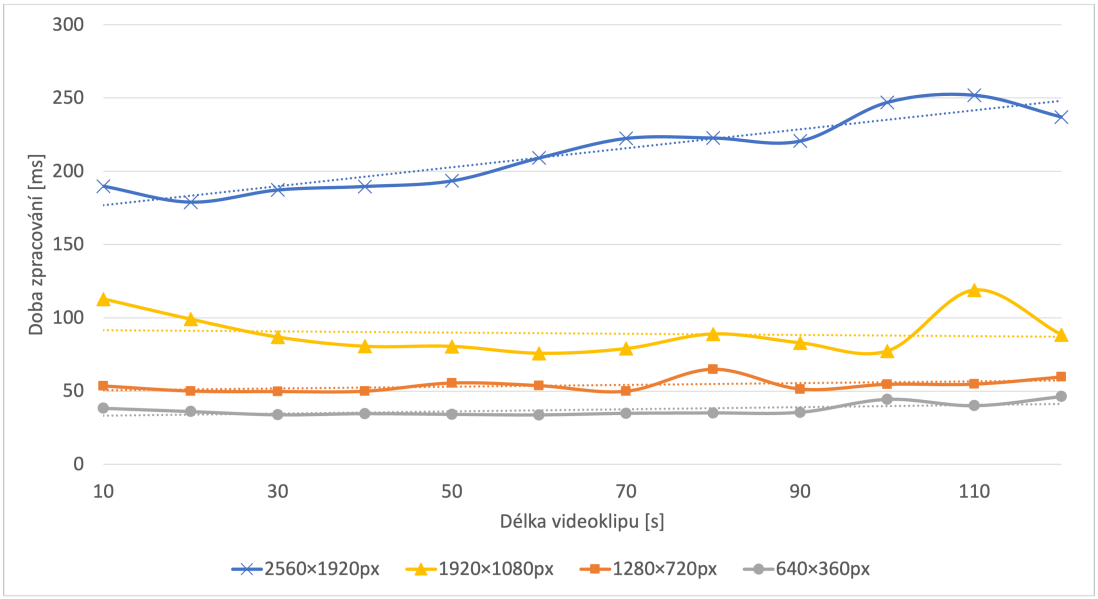
Výkon modulu pro ořezávání videa přímo souvisí s rychlostí zpracování příkazů nástrojem `ffmpeg`. Při testování tohoto modulu byla funkce `ffmpeg_extract_subclip` použita k výřezu videoklipů o délce 10 až 120 sekund. Poté byl spočítán čas, který zabrala generace těchto videí.

Následující kód definuje testovací proces:

```
import time
from app.utils.ffmpeg import ffmpeg_extract_subclip

video = "test.mp4"
for t2 in range(10, 121, 10):
    start_t = time.perf_counter()
    ffmpeg_extract_subclip(video, 0, t2, f"test_{t2}.mp4")
    end_t = time.perf_counter()
    processing_time = end_t - start_t
```

Výsledné hodnoty byly použity k sestavení grafu (Obr. 3.2) a výpočtu průměru a mediánu doby trvání procesu ořezávání videí (Tab. 3.2). Je možné pozorovat, že délka videoklipu téměř nehraje roli. Hlavním faktorem, který výrazně ovlivňuje výkonnost modulu, je rozlišení zpracovávaného videa.



Obr. 3.2: Závislost doby ořezávání videa na délce videoklipu.

Tab. 3.2: Doba střihu jednotlivých videoklipů.

Rozlišení videa	Průměrná doba	Medián doby
640×360 px	37,628 ms	35,316 ms
1280×720 px	54,918 ms	53,599 ms
1920×1280 px	85,834 ms	84,873 ms
2560×1920 px	221,545 ms	214,834 ms

3.3 Výkonnost modulu detekce objektů

Modul detekce objektů v rámci aplikace pracuje s jednotlivými snímky v videoklipech událostí. Výkonnost tohoto modulu hlavně ovlivňuje výpočetní rychlost modelu neuronové sítě `YOLOv8`. Testovaná metoda `ObjectDetector.process_frame` má za účel provést zpracování snímku tímto modelem a extrahovat z výstupu potřebná data o nalezených objektech.

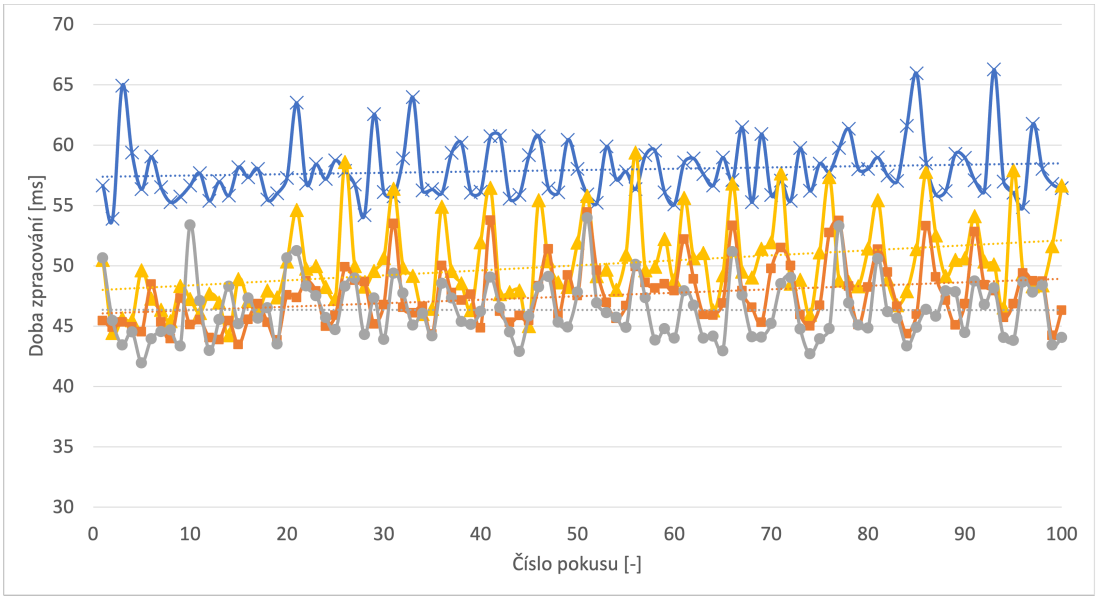
Testování se provádělo za využitím vybraných snímků z původních videí různých rozlišení. Každý snímek pro účely kompletní analýzy chování algoritmu byl zpracován 100krát.

Následující kód umožňuje spustit testování:

```
import time
import cv2
from app.services.object_detector import ObjectDetector

detector = ObjectDetector(model_path=Path("yolov8n.pt"))
for _ in range(100):
    start_time = time.perf_counter()
    frame = cv2.imread("test.jpg")
    detector.process_frame(frame)
    end_time = time.perf_counter()
    processing_time = end_time - start_time
```

Výsledkem testu je seznam časových hodnot trvání procesu zpracování pro různá rozlišení. Na základě těchto hodnot byl sestaven graf (Obr. 3.3). Pro lepší přehlednost výsledků byly také vypočítány průměr a medián časové náročnosti modulu detekce objektů (Tab. 3.3). Rozptyl hodnot v dolní části grafu není sice přehledný, ale je znovu zřejmá přímá závislost výkonnosti modulu na rozlišení zpracovávaného snímku.



Obr. 3.3: Závislost doby zpracování snímku modulem detekce objektů na rozlišení snímku.

Tab. 3.3: Doba zpracování snímku modulem detekce objektů.

Rozlišení videa	**Průměrná doba**	**Medián doby**
640×360 px	46,351 ms	45,769 ms
1280×720 px	47,489 ms	46,865 ms
1920×1080 px	50,046 ms	49,309 ms
2560×1920 px	57,935 ms	57,235 ms

3.4 Výkonnost aplikace

Celkový výkon aplikace byl vyhodnocen na základě časové náročnosti zpracování celého videa všemi moduly: detekce pohybu, ořezávání videa a detekce objektů. Zdrojem dat pro toto testování sloužila videa různých rozlišení. Délka a počet snímků těchto videí v každém testovacím případě byly stejné - 10 minut a 18030 snímků. Protože se jedná o videa stejného obsahu, počet zachycených pohybů a nalezených objektů je také stejný.

Výsledné časové hodnoty t lze použít pro výpočet rychlosti zpracování v a hodnoty přínosu k ve srovnání s délkou původního videa T.

$$v = \frac{N}{t} \tag{3.1}$$

$$k = \frac{T}{t} \tag{3.2}$$

Tab. 3.4: Celková doba zpracování videa v aplikaci.

Rozlišení videa	**Doba zpracování**	**Rychlost**	**Urychlení**
640×360 px	54,735 s	329,403 fps	10,961×
1280×720 px	79,741 s	226,104 fps	7,524×
1920×1080 px	114,56 s	157,372 fps	5,237×
2560×1920 px	169,403 s	106,432 fps	3,541×

Lze pozorovat, že doba analýzy videa pomocí aplikací se značně liší v závislosti na rozlišení vstupního videa. Čím vyšší je rozlišení, tím více času je potřeba na jeho zpracování. To znamená, že použití aplikace může být až 11krát výhodnější ve srovnání se sledováním videa uživatelem.

Závěr

Detekce a klasifikace objektů ve videosekvencích je rozsáhlé téma pokrývající řadu technologií od detektorů po neuronové sítě. V první části této práce jsou shrnuté základní algoritmy, které je možné využít při implementaci vlastního systému. Byly znázorněny dva typy detektorů: Frame Differences a MOG2, jejichž úkolem je odhalit změnu polohy objektu ve videosekvenci. Na druhou stranu je zde popsán základní princip algoritmu YOLO, který je v současné době častou volbou pro klasifikace objektů na obrázku.

V praktické části je na základě zadání navržena a vytvořena desktopová aplikace, která umožňuje uživateli zjistit časové úseky událostí ve vlastním videu. V aplikaci je implementována řada funkcí pro snadné nastavení potřebných parametrů, sledování zpracování a zjišťování výsledků analýzy. Na základě měření bylo zjištěno, že použití aplikace může 11× urychlit proces analýzy videa o rozlišení 640×360 px.

Dílčí výsledky praktické části této práce byly prezentovány na konferenci Student EEICT 2023 v kategorii „Systems Modeling, Sensors, and Signal Processing“.

Literatura

[1] SHI, Guiming, Jidong SUO, Chang LIU, Kang WAN a Xiaoying LV: *Moving target detection algorithm in image sequences based on edge detection and frame difference*, 740-744, 2017. DOI: 10.1109/ITOEC.2017.8122449.

[2] Vibrant Films: *Cruise Self Driving Car* [online]. Vimeo, 12. 11. 2015 [cit. 18. 5. 2023]. Dostupné z URL: <`https://vimeo.com/145559428`>

[3] Song, Young-Min and Noh, SeungJong and Yu, Jongmin and Park, Cheonwi and Lee, Byung-geun: *Background subtraction based on Gaussian mixture models using color and depth information*, 132-135, 2015. DOI: 10.1109/ICCAIS.2014.7020544.

[4] Caleo: // *Going Fat in Livigno* // [online]. Vimeo, 24. 11. 2014 [cit. 18. 5. 2023]. Dostupné z URL: <`https://vimeo.com/112674400`>

[5] Sani, Yusuf and Mohamedou, Ahmed and Ali, Khalid and Farjamfar, Anahita and Azman, Mohamed and Shamsuddin, Solahuddin: *An overview of neural networks use in anomaly Intrusion Detection Systems*, 89-92, 2009. DOI: 10.1109/SCORED.2009.5443289.

[6] I. Morishita: *Analysis of an Adaptive Threshold Logic Unit*. DOI: 10.1109/TC.1970.222857.

[7] Du, K.-L and Swamy, M.N.s: *Perceptrons*, 67-81, 2014. DOI: 10.1007/978-1-4471-5571-3_3.

[8] Sazli, Murat: *A brief review of feed-forward neural networks*, 11-17, 2006. DOI: 10.1501/0003168.

[9] Jiuxiang Gu and Zhenhua Wang and Jason Kuen and Lianyang Ma and Amir Shahroudy and Bing Shuai and Ting Liu and Xingxing Wang and Li Wang and Gang Wang and Jianfei Cai and Tsuhan Chen: *Recent Advances in Convolutional Neural Networks*, 11-17, 2017. DOI: 10.48550/arXiv.1512.07108.

[10] IBM: *Convolutional Neural Networks* [online]. 20.10.2020 [cit. 18. 5. 2023]. Dostupné z URL: <`https://www.ibm.com/cloud/learn/convolutional-neural-networks`>

[11] Sherstinsky, Alex: *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network*, 2020. DOI: 10.1016/j.physd.2019.132306.

[12] SRay Antonius and Hendra Tjahyadi: *Electromyography Gesture Identification Using CNN-RNN Neural Network for Controlling Quadcopters*, 12-75, 2021. DOI: 10.1088/1742-6596/1858/1/012075.

[13] Kayid, Amr and Khaled, Yasmeen and Elmahdy, Mohamed: *Performance of CPUs/GPUs for Deep Learning workloads*, 2018. DOI: 10.13140/RG.2.2.22603.54563.

[14] Nikolay Borodinov, Sabine Neumayer, Sergei V. Kalinin, Olga S. Ovchinnikova, Rama K. Vasudevan Stephen Jesse: *Deep neural networks for understanding noisy data applied to physical property extraction in scanning probe microscopy*, 2020. DOI: 10.1038/s41524-019-0148-5.

[15] Wu, Xian and Li, Rui-Long and Zhang, Fang-Lue and Liu, Jian-Cheng and Wang, Jue and Shamir, Ariel and Hu, Shi-Min: *Deep Portrait Image Completion and Extrapolation*, 2020. DOI: 10.1109/tip.2019.2945866.

[16] Nolfi, Stefano and Parisi, Domenico: *Learning to Adapt to Changing Environments in Evolving Neural Networks*, 2001. DOI: 10.1177/105971239600500104.

[17] Huang, Guang-Bin and Zhu, Qin-Yu and Siew, Chee: *Real-Time Learning Capability of Neural Networks*, 2006. DOI: 10.1109/TNN.2006.875974.

[18] Lohia, Aditya Kadam, Kalyani Joshi, Rahul Bongale, Arunkumar: *Bibliometric Analysis of One-stage and Two-stage Object Detection* [online], 2021, [cit. 18. 5. 2023]. Dostupné z URL: <`https://www.researchgate.net/publication/349297260_Bibliometric_Analysis_of_One-stage_and_Two-stage_Object_Detection`>

[19] Fumarola, Fabio and Ciampi, Anna and Appice, Annalisa and Malerba, Donato: *A Sliding Window Algorithm for Relational Frequent Patterns Mining from Data Streams*, 2009, pp. 385-392. DOI: 10.1007/978-3-642-04747-3_30.

[20] Cowton, Jake and Kyriazakis, Ilias and Bacardit, Jaume: *Automated Individual Pig Localisation, Tracking and Behaviour Metric Extraction Using Deep Learning*, 2019. DOI: 10.1109/ACCESS.2019.2933060.

[21] Joseph Redmon and Santosh Divvala and Ross Girshick and Ali Farhadi: *You Only Look Once: Unified, Real-Time Object Detection*, 2016. DOI: 10.48550/arXiv.1506.02640.

[22] Chen, Jian-Wen Kao, Chao-Yang Lin, Youn-Long: *Introduction to H.264 advanced video coding*, 2006. DOI: 10.1109/ASPDAC.2006.1594774.

[23] R. S. Dehal, C. Munjal, A. A. Ansari and A. S. Kushwaha: *GPU Computing Revolution: CUDA*, 2018, pp. 197-201. DOI: 10.1109/ICACCCN.2018.8748495.

[24] *OpenCV: OpenCV-Python Tutorials* [online]. OpenCV, 21.4.2023 [cit. 22.4.2023]. Dostupné z URL: <https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html>

[25] *About - OpenCV* [online]. OpenCV, 21.4.2023 [cit. 22.4.2023]. Dostupné z URL: <https://opencv.org/about>

[26] *PyImageSearch/imutils: A series of convenience functions to make basic image processing operations such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and Python* [online]. GitHub, 27.1.2022 [cit. 22.4.2023]. Dostupné z URL: <https://github.com/PyImageSearch/imutils>

[27] *YOLOv5 - Ultralytics | Revolutionizing the World of Vision AI* [online]. Ultralytics, 22.4.2023 [cit. 22.4.2023]. Dostupné z URL: <https://ultralytics.com/yolov5>

[28] *utils - Ultralytics YOLOv8 Docs* [online]. Ultralytics, 22.4.2023 [cit. 22.4.2023]. Dostupné z URL: <https://docs.ultralytics.com/reference/hub/utils/>

[29] *ffmpeg Documentation* [online]. FFmpeg, 22.4.2023 [cit. 22.4.2023]. Dostupné z URL: <https://ffmpeg.org/ffmpeg.html>

[30] *Qt for Python* [online]. Qt, 22.4.2023 [cit. 22.4.2023] Dostupné z URL: <https://doc.qt.io/qtforpython>

[31] *Qt | Development Framework for Cross-platform Applications* [online]. Qt, 22.4.2023 [cit. 22.4.2023] Dostupné z URL: <https://www.qt.io/product/framework>

[32] W.Joshua: *Overview of PyQt5* v *Modern PyQt: Create GUI Applications for Project Management, Computer Vision, and Data Analysis*, 2021, pp. 1-42, DOI: 10.1007/978-1-4842-6603-8_1.

[33] P. Benický and J. Ladislav: *Real Time Motion Data Preprocessing*, 2011, pp.247-251. DOI: 10.2478/v10187-010-0035-2.

[34] G. Mclachlan: *Mahalanobis Distance*, 1999, pp.20-26. DOI: 10.1007/BF02834632.

[35] *OpenCV: cv::BackgroundSubtractorMOG2 Class Reference* [online]. OpenCV, 22.4.2023 [cit. 22.4.2023] Dostupné z URL: <https://docs.opencv.org/4.x/d7/d7b/classcv_1_1BackgroundSubtractorMOG2.html>

Seznam symbolů a zkratek

MOG Mixture of Gaussian – směs Gaussových distribucí, algoritmus detekce pohybu

MOG2 Mixture of Gaussian 2 – směs Gaussových distribucí 2, druhá verze algoritmu detekce pohybu

ANN Artificial Neural Network – umělá neuronová síť

FNN Feedforward Neural Network – dopředná neuronová síť

CNN Convolutional Neural Network – konvoluční neuronová síť

RNN Recurrent Neural Network – rekurentní neuronová síť

LSTM Long Short-term Memory – síť s dlouhodobou a krátkodobou pamětí

GRU Gated Recurrent Unit – síť s řízenou rekurentní jednotkou

GPU Graphics Processing Unit – grafická procesorová jednotka

CPU Central Processing Unit – centrální procesorová jednotka

YOLO You Only Look Once – algoritmus detekce objektů

IoU Intersection over Union – metrika přesnosti detektoru objektů

AAC Advanced Audio Coding – formát komprese zvuku

CUDA Compute Unified Device Architecture – architektura paralelních výpočtů, pro zvýšení výpočetního výkonu pomocí grafických procesorů NVIDIA

HTML HyperText Markup Language – značkovací jazyk hypertextových dokumentů pro prohlížení webových stránek

A Obsah elektronické přílohy

Kromě elektronické verze práce je přiložena složka se zdrojovým kódem aplikace, která má následující obsah:

```
app ..................................................... kořenová složka aplikace
├── commons ............................... složka obsahující pomocné soubory
│   ├── ffmpeg
│   │   └── ffmpeg.exe
│   ├── templates
│   │   ├── full_report.html
│   │   └── video_report.html
│   └── yolo
│       └── yolov8n.pt
├── data_structures ......................... definice datových struktur aplikace
│   ├── color.py
│   ├── motion_detection.py
│   ├── object_detection.py
│   ├── processing.py
│   ├── time_interval.py
│   └── video_cutting.py
├── gui ........................... implementace grafického uživatelského rozhraní
│   ├── boxes ............................... definice použitých vyskakovacích oken
│   │   ├── close.py
│   │   └── error.py
│   ├── windows ............................... definice jednotlivých oken aplikace
│   │   ├── analysis.py
│   │   ├── base.py
│   │   ├── browse_file.py
│   │   ├── configuration.py
│   │   └── results.py
│   └── controller.py
├── services ................. implementace modulů pro jednotlivé fáze zpracování
│   ├── detection_processor.py
│   ├── image_saver.py
│   ├── motion_detector.py
│   ├── object_detector.py
│   ├── processing_manager.py
│   ├── report_generator.py
│   ├── report_saver.py
│   └── video_cutter.py
└── utils ..................................... modul obsahující pomocné funkce
    ├── ffmpeg.py
    ├── format.py
    ├── subprocessing.py
    ├── time.py
    └── video.py
```

```
│   │   │   │
```