VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁSTROJ PRO SBĚR A ZPRACOVÁNÍ SPORTOVNÍCH VIDEÍ

TOOL FOR COLLECTION AND PROCESSING OF SPORTS VIDEOS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MAREK BAHNÍK**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

143240

Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Student: **Bahník Marek**
Program: Informační technologie
Specializace: Informační technologie

Název: **Nástroj pro sběr a zpracování sportovních videí**
Kategorie: Počítačové vidění
Akademický rok: 2022/23

Zadání:

1. Seznamte se s problematikou rozpoznávání lidské postavy a s problematikou analýzy sportovních videí s použitím počítačového vidění.
2. Vyhledejte dostupné datové sady videí z oblasti sportu.
3. Navrhněte a vyviňte sadu nástrojů pro sběr datové sady sportovních videí vhodných pro strojové učení.
4. Iterativně získávejte vhodná videa a rozvíjejte vytvořené nástroje pro dosažení co nejlepších datových sad pro počítačové vidění ve sportu. Pokryjte vhodně zvolené sporty – na základě konzultace s vedoucím.
5. Vyhodnoťte vlastnosti pořízených datových sad a sady dobře zdokumentujte.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Manisha Verma et al.: Yoga-82: A New Dataset for Fine-grained Classification of Human Poses, CoRR / arXiv, https://arxiv.org/abs/2004.10362
- Goodfellow, Bengio, Courville: Deep Learning, MIT Press, 2016
- Bharath Ramsundar, Reza Bosagh Zadeh: TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning, O'Reily Media, 2018
- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011

Při obhajobě semestrální části projektu je požadováno:
body 1. a 2., značné rozpracování bodů 3. a 4.

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

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 31.10.2022

Abstrakt

Cílem této práce je vytvořit nástroj v jazyce Python pro získávání videí na určitou sportovní tematiku z YouTube a jejich úpravu do formy potřebné např. pro učení neuronových sítí. Videa jsou rozdělena na jednotlivé klipy pomocí proprietární metody porovnávání dvou snímků okolo podezřelého snímku. Tyto klipy jsou dále klasifikovány na škále relevance s využitím optického toku. Pro stažení kvalitní sady celých videí je třeba zadat správné klíčové slovo a vytvořit filtry pro YouTube vyhledávání. Tento nástroj není plně automatizovaný a optimalizovaný, ale je navržen tak, aby co nejvíce zrychlil a ulehčil proces vytváření datové sady. Obecně je v průběhu volání jednotlivých částí potřeba uživatelsky manuálně zasahovat a kontrolovat výstupní data.

Abstract

The aim of this thesis is to create a tool in Python for downloading videos from YouTube and parsing them into usable form for possible neural network training. Videos are split into separate clips using proprietary method of comparing two frames around suspect frame. These clips are furthermore classified on a scale of relevance using optical flow. To download quality videos from YouTube, user has to input proper search word and custom filters. This tool is not fully automated and optimized, but it is designet to make dataset creation much faster and simpler. In general user has to intervene manually and check output data.

Klíčová slova

Python, stahování videí, detekce postury, detekce střihů, datová sada videí, optický tok, klasifikace videí

Keywords

Python, video downloading, pose detection, cut detection, video data set, optical flow, video classification

Citace

BAHNÍK, Marek. *Nástroj pro sběr a zpracování sportovních videí.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Nástroj pro sběr a zpracování sportovních videí

Prohlášení

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

.....
Marek Bahník
7. května 2023

Poděkování

Rád bych zde poděkoval svému vedoucímu prof. Ing. Heroutovi, Ph.D., za jeho ochotu, trpělivost a rady a nápady, které mi pomohly během vytváření a testování nástroje.

Obsah

1 Úvod **2**

2 Existence předchozích řešení a užitečné znalosti **4**
2.1 Existující řešení 4
2.2 Využité znalosti 5

3 Návrh a rozdělení nástroje **14**
3.1 Scraper – vyhledání videí na **YouTube** 15
3.2 Downloader – stažení nalezených videí z **YouTube** 15
3.3 Splitter – rozdělení nalezených videí na klipy 16
3.4 Classifier – klasifikace rozdělených klipů podle relevance 18

4 Implementace nástroje **22**
4.1 Společné konstrukce 22
4.2 Scraper – vyhledání videí na **YouTube** 23
4.3 Downloader – stažení nalezených videí z **YouTube** 24
4.4 Splitter – rozdělení nalezených videí na klipy 26
4.5 Classifier – klasifikace rozdělených klipů podle relevance 31
4.6 Pomocné části nástroje 35

5 Závěr **38**
5.1 Nástroj pro vytvoření datové sady 38
5.2 Datová sada 38
5.3 Navazující práce 39

Literatura **40**

Kapitola 1

Úvod

V posledních letech se umělá inteligence a neuronové sítě staly stále více populárními technologiemi v mnoha oblastech informačních technologií. Technologie neuronových sítí se stala stěžejní pro řešení mnoha problému v oblasti rozpoznávání obrazu, překladu textu, ve finančním sektoru a mnoho dalších. v oblasti rozpoznávání obrazu se využívá k rozpoznání objektů ve fotkách a videích, k sledování jejich pohybu v rámci videa a jejich segmentaci, stejně tak jako rozpoznání lidské postavy z video záznamu [26]. Ve finančním sektoru se neuronové sítě využívají například k detekci podezřelých pohybů na bankovních účtech anebo k predikci pohybu kurzu měn, nebo komodit [7]. Díky své schopnosti adaptovat se na nové situace jsou neuronové sítě využívány k řešení mnoha složitých problémů, které jsou v běžném životě člověka přítomny [26]. Datová sada je klíčovým faktorem, který dělí neuronové sítě na úspěšné a neúspěšné. Bez kvalitní datové sady nemohou neuronové sítě produkovat přesné a spolehlivé výsledky. Datové sady jsou základem pro trénování neuronových sítí a umožňují modelům identifikovat vzorce v datech a předpovídat výsledky pro nová vstupní data. Datová sada by měla mít několik vlastností, které povedou k tomu, že bude natrénovaná neuronová síť co nejpřesnější [17].

1. Datová sada by měla být reprezentativní problému, pro který je trénovaná. Sada by měla obsahovat pouze data relevantní k problému, se kterým má síť pracovat.

2. Sada by také měla být dostatečně velká a rozmanitá tak, aby obsahovala co největší množství různých variací problému. Obecně se dá říct, že čím více rozmanitých dat, tím lépe, avšak pořád by měla dodržovat první bod.

3. Další vlastností by také měla být standardizace. v našem případě by např. měly hodnoty pixelů být ve stejném rozsahu, nejlépe [0,1], nebo [-1,1] [17].

4. v neposlední řadě by měla být datová sada správně anotovaná a popsaná. Tím je umožněno co nejefektivnější učení sítě a co nejpřesnější výstup [15].

Na internetu existuje mnoho stránek, na kterých může uživatel získat přístup k již vytříděným, popsaným a vyzkoušeným datovým sadám. Příkladem těchto stránek je například **Kaggle**[1] nebo sběrnice datových sad od společnosti **Google**[2]. Problém nastává tehdy, když je potřeba datová sada na nějakou velmi konkrétní problematiku. Tyto webové archivy naplněné daty obsahují miliony datových sad, ale data pro některé specifické problematiky, které nemají tak velkou uživatelskou základnu, nebo ekonomický potenciál, tam

[1] https://www.kaggle.com/
[2] https://datasetsearch.research.google.com/

bohužel mohou chybět. Pokud je potřeba vytvořit neuronovou síť zabývající se takovýmto problémem bez veřejně dostupné datové sady, je potřeba vytvořit datovou sadu vlastní.

A právě v této situaci by mohla osoba, která chce vytvořit vlastní datovou sadu, využít tuto sadu skriptů, nebo případně jen její část. Cílem této práce bylo vytvořit právě takový nástroj, který umožní získat základní a neanotovaný datový set co možná nejvíce automatizovaně. Tato data jsou získávána z **YouTube**[3], který obsahuje ohromná množství videí s různou tematikou, délkou i kvalitou. Pro nás zajímavá jsou právě sportovní videa. na **YouTube** lze totiž nalézt hodiny a hodiny lekcí jógy, záznamy celých fotbalových zápasů, sestříhaných videí z Olympijských her a mnoho dalších videí ze všech různých sportů. a přesně tato data bude program pomocí tohoto nástroje filtrovat, stahovat, dělit a třídit tak, aby se dala případně využít pro trénovací účely.

[3] https://www.youtube.com

Kapitola 2

Existence předchozích řešení a užitečné znalosti

V této první kapitole bych rád rozebral dříve implementovaná řešení podobných problémů. Předchozí řešení mohou být důležitá z mnoha důvodů. Je dobré se inspirovat tím, co předchozí nástroje udělaly správně a kvalitně, je ale také dobré se poučit z jejich chyb a nedostatků. Také jsem si ujasnil, zda je opravdu potřeba takový nástroj vytvářet – nemá smysl vytvářet něco, co už někdo vytvořil před námi, a případně i lépe.

Také v kapitole 2.2 rozeberu znalosti, které jsem potřeboval získat nebo již znal k tomu, abych mohl tento nástroj vytvořit. Nejedná se pouze o očekávané znalosti informatické a programovací, ale také sportovní, jak je popsáno v kapitole 2.2.5.

2.1 Existující řešení

V této kapitole bych rád odpověděl na dvě otázky: „Existují nějaké podobné nástroje?“ a případně „Jsou tyto nástroje dostačující?“. v úvodu této kapitoly jsem řekl, proč je důležité tyto otázky zodpovědět. Nyní se na ně pokusím odpovědět.

2.1.1 Existují nějaké podobné nástroje? a jsou dostačující?

Na první otázku je jednoduché odpovědět krátce: neexistují. Delší odpověď je ale už trošku méně přímočará. Tento nástroj je velmi specifický, a to je pravděpodobně ten primární důvod, proč neexistuje nic podobného. Jeden vzdáleně podobný nástroj ale je zde[1]. Tento nástroj stahuje videa z **YouTube** a z nich poté získává obrázky, ze kterých vytváří datovou sadu. Je to však velmi základní nástroj bez větších požadavků a absolutně nedostačující naším požadavkům. Druhým možná ještě vzdáleněji podobným nástrojem je *Youtube Speech Data Generator*[2]. Ten podobně jako nástroj první získává videa z **YouTube**, ale tento nástroj vytváří zvukovou sadu dat – tím pádem je to pro nás opět nevhodný. Toto jsou však jediné nástroje, které jsem našel, co se zabývají podobnou tématikou.

[1] https://towardsdatascience.com/making-an-image-dataset-from-youtube-videos-5116252d20a3
[2] https://github.com/hetpandya/youtube_tts_data_generator

2.2 Využité znalosti

Pro vytvoření funkčního a kvalitního nástroje bylo potřeba mít nebo získat znalosti z poměrně velkého množství odvětví. Samozřejmě nejdůležitější jsou znalosti programování a skriptování v jazyce Python, který jsem použil pro vytvoření tohoto nástroje. Zadruhé bylo nutné porozumět tomu, jak funguje vyhledávání a filtrování na **YouTube**. v návaznosti na to bylo také důležité vybrat správné kodeky ideální pro navazující práci. v doméně samotného zpracovávání videí toho bylo na porozumění více. Některá videa z **YouTube** nejsou dobře zpracovatelná, některá mohla být stažena špatně. Proto muselo být toto dobře ošetřeno tak, aby nenastaly problémy. Nejdůležitější však bylo porozumět detekčním algoritmům. na těch je totiž založen téměř celý princip tohoto nástroje.

2.2.1 Python

Python je v dnešní době jeden z nejpopulárnějších programovacích jazyků. Jedním z důvodů je právě modularita. Pro Python existuje nepřeberné množství knihoven, z nichž pro každý skript jich bylo využito hned několik. Další výhodou Pythonu je jeho jednoduchost. Programovací konstrukce nejsou nijak složité, cykly jsou jednoduché na implementaci a datové struktury jsou jednoduše uchopitelné. Zároveň je to jazyk, se kterým mám nejvíce zkušeností, a tak jsem pro implementaci tohoto nástroje vybral právě ten.

Samozřejmě má i svoje nevýhody. Jednou z nich, a pravděpodobně tou nejvýznamnější, je jeho rychlost, respektive pomalost. Python se neřadí mezi nejrychlejší programovací jazyky, a to například díky dynamickému typování proměnných anebo internímu "garbage collectoru"- systému, který se stará o uvolňování paměti například pomocí mazání nepoužívaných struktur. Nejen díky tomu také zpracovávání videí trvá poměrně dlouhou dobu. Jelikož se jedná spíše o tzv. "proof of concept", česky volně přeložitelné jako "důkaz o proveditelnosti konceptu", tak ale pro mě nebyla rychlost prioritní.

Výhody však z mého pohledu převyšují nevýhody, a tak jsem se pro Python nakonec rozhodl. Jiné možnosti však mohou být například C, C++ nebo dokonce C# i Java – možností je taky více.

2.2.2 Práce s YouTube

Mým prvotním plánem bylo provádět vyhledávání na **YouTube** manuálně a výsledky získávat ze samotného HTML[3] pomocí knihovny **Beautiful soup**[4]. Naštěstí jsem však našel dvě knihovny pro Python, které toto výrazně zjednodušily a tím pádem nebylo potřeba manuálně prohledávat HTML soubory a získávat z nich pracně data. i tak však bylo velmi důležité porozumět různým informacím obsaženým ve videích na **YouTube**, podle kterých je třídím a získávám. Těchto informací je více, ale primárně se jedná o tzv. kodek, pomocí kterého bylo dané video zakódováno a může být použit pro jeho rozkódování. Pro naše účely jsou vybíral kodeky **avc1**, **av01** a **vp9** prioritně v tomto pořadí. Více informací o formátu videí je v kapitole 4.3.1.

2.2.3 Práce s videi

Jelikož se jedná o nástroj, který pracuje nějakým způsobem s videi, je třeba videa často upravovat, číst, ukládat nebo s nimi nějak jinak nakládat. Proto je nutné rozumět tomu, jak

[3] https://www.w3.org/TR/html401/struct/global.html#h-7.5.4
[4] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

funguje kódování videí. Během opakovaného zakódovávání a dekódovávání se ztrácí kvalita videa, a proto je dobré toto dělat co nejméně [25]. Pro dekódování, kódování a obecně pracování s video byl velmi užitečný nástroj FFMPEG[5]. Jedná se o konzolový nástroj, který právě tyto možnosti poskytuje. Zároveň ho lze zavolat ve skriptech, a dokonce existuje i knihovna pro Python. Oboje jsem využil během vytváření téměř všech skriptů.

2.2.4 Detekční metody

Během zpracovávání a analýzy videí bylo potřeba otestovat několik různých detekčních metod, které pracují s postavami, a vybrat tu, která bude pro nástroj nejvhodnější. v nabídce jich je na internetu spousta, ale já jsem testoval tři z nich: **YOLO** [16], **Mediapipe** [2] a **OpenCV** [10]. Zároveň jsem pracoval s knihovnou **PySceneDetect** [11], která poskytuje základ na detekci střihů.

Detekce postav pomocí knihovny YOLO

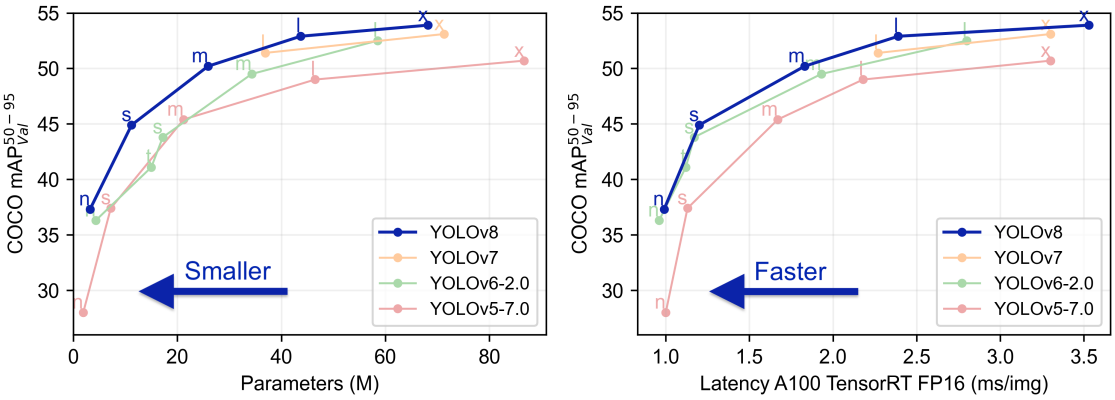
YOLO, zkratka „You Only Look Once“, je jedním z nejznámějších modelů na počítačové vidění, který se používá pro detekci objektů v reálném čase. Když jsem začínal s vývojem, tak byla aktuální verze **YOLOv7** [30] vydaná v polovině roku 2022, tu jsem chtěl využít a otestovat, ale po zprovoznění nástroje se mi nepovedlo zprovoznit část modelu pro detekci postav a tak jsem se rozhodl **YOLOv7** nepoužít. Bohužel (i když obecně samozřejmě bohudík) byla začátkem roku vydaná pro **YOLO** verze 8, nazývaná Ultralytics [28], která by měla hodnoty **YOLOv7** překonat téměř ve všech bodech [16]. Bohužel už jsem měl v době vydání hotovou část nástroje, která by tento model mohla využít. Tím pádem jsem jej jen otestoval, zda by se nedal využít na jinou část, která ještě hotová nebyla, ale bohužel. Zde je určitě prostor pro zlepšení a další iterace tohoto nástroje by mohla být rychlejší a možná přesnější. To však je pouze hypotéza na základě obecných hodnot **YOLO** verze 8, které jsou na obr. 2.1. Bohužel studie popisující YOLO verzi 8 ještě nebyla vydána, i když je tak slibováno už od vydání knihovny, takže přesnější informace o implementaci modelu nejsou ještě k dispozici[6]

Detekce klíčových bodů na postavě pomocí knihovny Mediapipe

MediaPipe je knihovna vytvořená společností Google pro zpracování multimediálních dat, jako jsou videa a fotky. v rámci této knihovny existuje několik předtrénovaných detekčních modelů pro detekci různých objektů v obraze, například obličeje, rukou, gest, a právě pro nás využitelné postavy [2]. Tato knihovna byla velmi jednoduchá na zprovoznění a společně se správcem balíčků PiP[7] pro Python bylo vložení knihovny do mého nástroje jednoduché.

Tato knihovna má detekční model nazvaný **BlazePose**, který slouží k právě zmíněné detekci postav v obraze. Tento model využívá techniku **pose estimation**, která se zaměřuje na odhad pozice klíčových bodů na lidském těle. **BlazePose** využívá neuronové sítě, které jsou trénovány na velkém množství dat obsahujících různé pozice těla v obraze. Samotná detekce probíhá ve dvou krocích: nejprve se použije tzv. „upper body detector“ (česky „detektor vrchní části těla“) který detekuje horní polovinu těla v obraze, a poté se použije „full body detector“ (česky „detektor celého těla“), který určuje pozice klíčových bodů na celém těle [19].

[5] https://ffmpeg.org/
[6] https://github.com/ultralytics/ultralytics/issues/204
[7] https://pypi.org/project/pip/



Obrázek 2.1: Obrázek popisuje na osách y naměřené hodnoty střední průměrné přesnosti COCO mAP (*Common Objects in Context mean Average Precision*), což je metrika pro vyhodnocování úspěšnosti detekčních modelů v oblasti počítačového vidění [27]. na levém obrázku tvoří osu x počet parametrů modelu a na pravém tvoří osu x odezvu grafické karty A100 s použitím knihovny TensorRT v režimu poloviční přesnosti. Lze poznat, že YOLOv8 překonává předchozí iterace této knihovny téměř ve všech bodech.

BlazePose vypočítává těchto klíčových bodů na lidském těle 33, včetně klíčových bodů na hlavě, ramenou, loktech, zápěstích, břiše, kolenou a kotníků. Informace o těchto bodech na těle jsou na obrázku 2.2 [19]. Právě porovnávání počtu a viditelnosti těchto bodů na dvou různých snímcích z videa nám umožní vytvořit přesnější algoritmus pro detekci střihů. Přesný popis algoritmu, který má toto na starost je popsán v kapitole 4.4. Algoritmus porovnává počet klíčových bodů na dvou různých snímcích v určitě vzdálenosti od podezřelého snímku a podle určitého klíče porovná tyto obrázky. v případě, že podobnost je dostatečně malá, tak označí onen podezřelý snímek jako bod střihu ve videu.

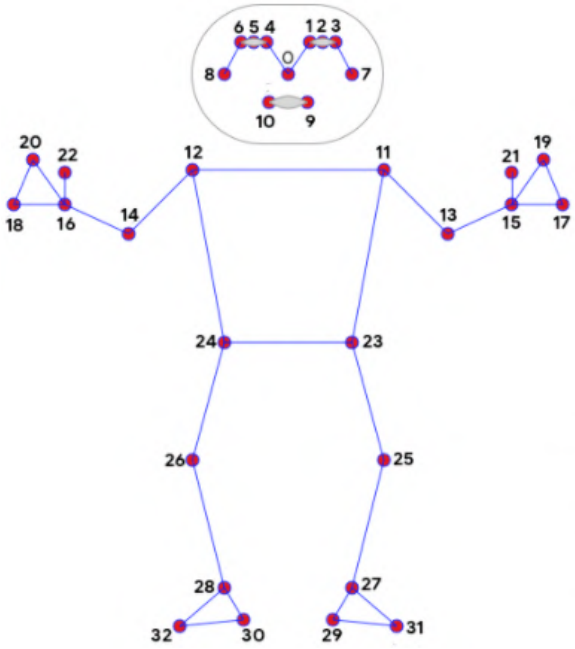
Čtení videí a detekce postav pomocí knihovny OpenCV

OpenCV (Open Source Computer Vision) je open-source knihovna programů a algoritmů pro počítačové vidění a zpracování obrazu. **OpenCV** obsahuje stovky funkcí, které umožňují aplikacím zpracovávat, analyzovat a interpretovat obrazová data. Mezi funkce OpenCV patří detekce objektů, sledování pohybu, rozpoznávání obličejů, segmentace obrazu a mnoho dalších [9].

Jedna z hlavních funkcí, které OpenCV poskytuje, je otevření a čtení videa snímek po snímku. Tohoto je využito v **Splitteru** a **Classifieru**. Jelikož je cílem pracovat s jednotlivými snímky ve videu, tak se nám tento přístup hodí.

Druhý typ funkcí jsou konverzní metody mezi barevnými spektry. Nejvíce je využita konverze z RGB do stupňů šedi anebo převod z RGB do HSV. Důvody k této konverzi jsou v kapitole 3.3.

Další velmi důležitou funkcí, kterou využiji je implementace optického toku. Optický tok je technika, která se používá k určování rychlosti a směru pohybu objektů v obraze. Tato technika je založena na sledování pohybu bodů v obraze mezi dvěma po sobě následujícími snímky [29]. Pro výpočet optického toku se využívají různé metody, já jsem se zabýval dvěma: **Lucas-Kanade** [18] metoda a **Farneback** [14] metoda.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

Obrázek 2.2: Na obrázku jsou zobrazeny klíčové body BlazePose na lidském těle. Obrázek pochází z MediaPipe stránky na jejich Git repositáři [2].

Obě metody pracují se stejnými dvěma předpoklady: Intensita pixelů objektu se nemění mezi jednotlivými snímky a sousední pixely mají stejný směr pohybu. Níže jsou rovnice užívané pro výpočty optického toku.

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{2.1}$$

Při aplikování aproximace pomocí Taylorovy řady na pravou stranu rovnice (2.1) a redukce společných prvků v rovnici a vydělením**dt** je získána rovnice (2.2) [4]:

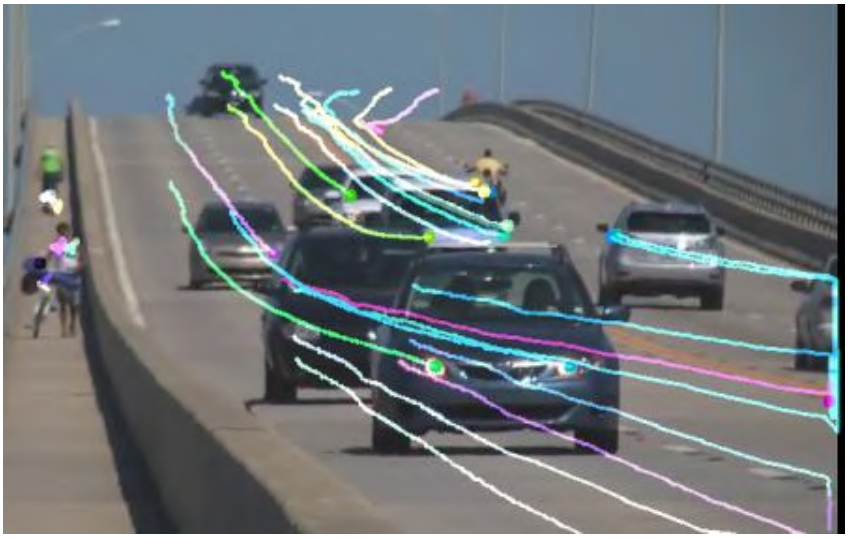
$$f_x u + f_y v \ + f_t = 0 \tag{2.2}$$

kde:

$$\begin{aligned} f_x = \frac{\partial f}{\partial x}; f_y \frac{\partial f}{\partial y} \\ u = \frac{dx}{dt}; v \ = \frac{dy}{dt} \end{aligned} \tag{2.3}$$

Těmto rovnicím se říká rovnice optického toku a pracují s nimi obě dříve zmíněné metody. Metoda **Lucas-Kanade** pracuje tak, že se vezme 9 pixelů v mřížce 3×3 a na nich je vypočítána funkce optického toku, která je popsána ve funkcích 2.2 a 2.3. Pokud je však pohyb pixelu větší, tak by ho tato metoda nezachytila, proto pracuje **Lucas-Kanade** pyramidově, tzn. tyto 3×3 bloky zhustí do jednoho celku a s těmito celky opět pracuje v 3×3 mřížce [18].

Metoda **Farneback** funguje trochu na jiném principu, než **Lucas-Kanade**. Tato metoda totiž počítá optický tok pro celý snímek najednou, na rozdíl od **Lucas-Kanade**, která počítá blokově. Zde je již matematický princip více komplikovaný, pro více informací je zde článek [14] zabírající se touto problematikou. Naštěstí jsou tyto metody implementovány



Obrázek 2.3: Na obrázku je grafickou reprezentaci metody Lucas-Kanade. Obrázek pochází z oficiálních stránek OpenCV [23].

ve funkcích **OpenCV** knihovny, a tak jsem se touto matematikou nemusel dlouho zabývat, tento problém byl již vyřešen.

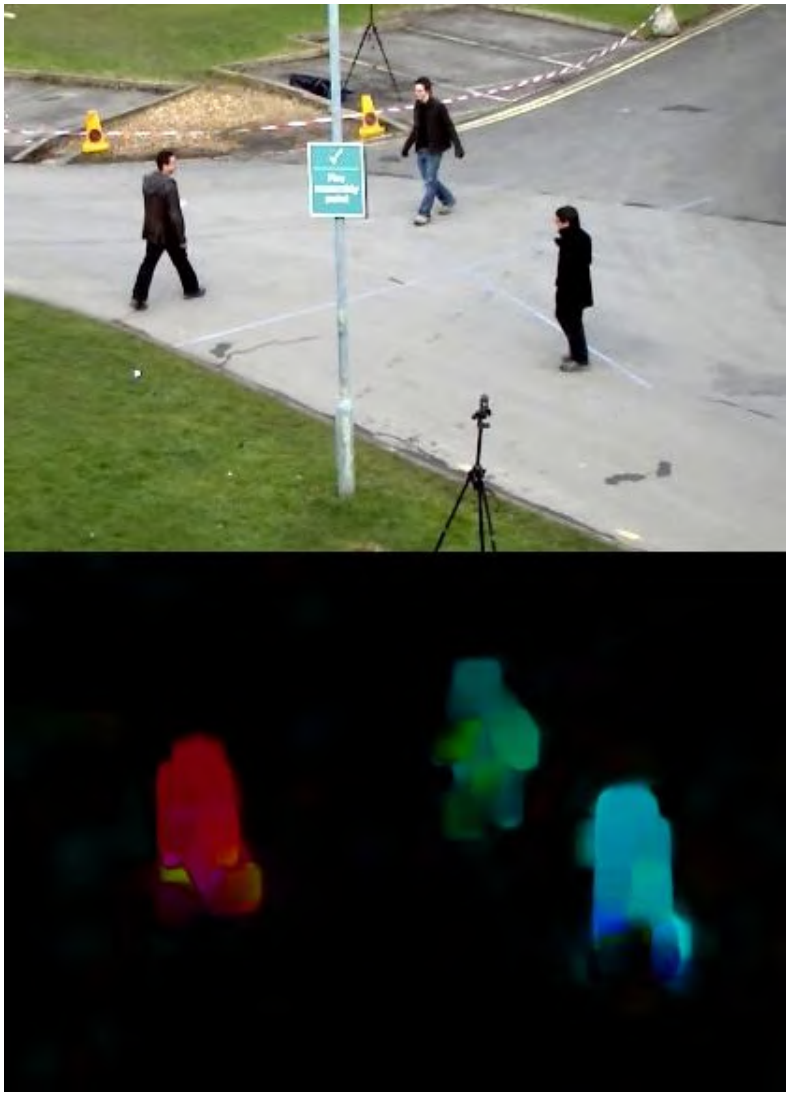
Detekce střihů a PySceneDetect

Velmi důležitým bodem nástroje je i detekce změny scény neboli střihu. Jelikož se může stát, a pravděpodobně se také stane, že nástroj stáhne z **YouTube** až třeba několik hodin dlouhé video, tak je ho třeba rozdělit právě na jednotlivé scény. Tyto scény mohou mít různé délky, ale většinou se v našem případě pohybují v rámci vteřin. Já jsem pracoval hlavně se třemi typy střihů: **přímý střih**, **postupný střih** a **animovaný střih**. Příklady těchto střihů jsou na obrázcích 3.4, 3.5 a 3.6.

Přímý střih je z těchto tří typů nejjednodušší. Změna scény zde nastává v rámci dvou snímků. Je to poslední snímek předchozí scény a první snímek následující scény. Při rozdělení videa v bodě změny scény zůstanou dvě kratší videa, ve kterých nebudou žádné pozůstatky druhého videa – nelze rozeznat, že nějaký střih nastal.

U **postupného střihu** již je několik různých metod, jak ke střihu dochází. Příkladem je prolnutí, vytlačení, rozptýlení, nebo třeba notorický přechod hvězdou. Co mají tyto střihy společné je to, že existuje v průběhu videa několik snímků, na kterých jsou části obou scén najednou, buď jako celky anebo například při distorzi jako jednotlivé pixely [8]. Toto pro náš případ není úplně ideální, ale metody, jak se s tím dá vypořádat jsou v kapitole 3.3.

Animovaný střih je nejsložitější z těchto třech typů. Základ je velmi podobný tomu, jak funguje postupný střih. Akorát je ještě navíc přes tento běžný přechod vložena nějaká animace. Příklad je na obrázku 3.6. Možná bude překvapením, že na zpracování je tento druh střihu možná mírně jednodušší než ten postupný. Důvod je ten, že tím, že je přes samotný střih ještě navíc vložena animace, se zakryje pozadí a tím jsou skryté některé rušivé elementy, které můžou mást můj detekční software. Samozřejmě má i svoje nevýhody oproti postupnému střihu, v moment, kdy je animace podobně zbarvená jako samotné video, tak může detekční software považovat animaci jako součást videa a střih nedetekovat.



Obrázek 2.4: Obrázek graficky zobrazuje funkcionalitu metody Farneback. Muž nalevo se pohybuje rychleji než muž uprostřed, tudíž jeho pohybová stopa na spodním obrázku má více červený odstín. Muž napravo stojí na místě a pohybuje se minimálně, proto je reprezentován jako odstín modré. Pozadí je statické, a proto je černé. Obrázek pochází z oficiálních stránek OpenCV [22].

Pro Python existuje knihovna **PySceneDetect**. Jedná se o knihovnu, která umožňuje právě detekci střihu ve videích. Nabízí několik tzv. detektorů, které mají každý trochu jiné vlastnosti a fungují na jiný typ střihu. Bohužel tyto implementované detektory nefungují velmi dobře na animované střihy a postupné střihy, avšak knihovna umožňuje vytváření vlastních detektorů pomocí dědičnosti na základě jejich základních detektorů, což jsem také udělal. K samotnému provedení více v kapitole 4.4.

2.2.5 Skok do výšky

Tato kapitola se pravděpodobně vymyká obsahu v této práci, avšak je podle mě nutné tyto informace zmínit. i když je část programu poměrně univerzální v aplikovatelnosti na různé

sporty, některé pozdější části jsou specifické právě pro skok vysoký. Právě pro tuto část nástroje je třeba mírné porozumění tomu, jak takový skok do výšky vypadá a co jsou jeho výrazné znaky, které lze použít na rozpoznání relevantních videí. Co je primárním prvkem, který je využit při klasifikaci je to, že u velké většiny videí skoku vysokého je velké množství pohybu (primárně horizontálního) ve videu. Toho využívám s pomocí optického toku, více detailů k této metodě je v kapitole 2.2.4. Dalším faktem, kterých by se dalo využít v některých bodech nástroje je například to, že ve většině videí skoku do výšky je jen jedna hlavní osoba, tudíž se mohou detektory omezit na jednu detekovanou postavu. Zároveň se také jedná o plynulý pohyb – v kamerovém záběru nedochází k nějakým rapidním nečekaným pohybům. Tyto informace mohou být použitelné v případných dalších iteracích nástroje.

2.2.6 Datový výstup a MS Excel

Jendou z velmi důležitých součástí vývoje byla analýza dodatečných výstupních dat detekčních algoritmů. Tyto detekční algoritmy implementované v různých knihovnách, jako např. **PySceneDetect**, mají často samy jako jedním z možných výstupů datovou sadu ve formátu **.csv** (ang. „comma separated values“). Do těchto dat jsem si sám přidával navíc některé hodnoty, které knihovna do dat sama neukládala. v případě, že to samotná knihovna nepodporovala a bylo potřeba mít nějaké hodnoty na ladění funkčnosti algoritmu, tak jsem si sám vytvářel podobné datové sady a ukládal je do vlastních **.csv** souborů pomocí knihovny **openpyxl**, konkrétně v kapitole 4.5. Ukázku části výstupního souboru vygenerovaného automaticky knihovnou **PySceneDetect** s několika vlastními přidanými sloupci je na obrázku 2.5.

Frame Number	Timecode	conf	content_val	delta_edges	delta_rgb	det_cut	diff	is_cut	z_diff
1618	00:01:04.680		5.175585364	11.66342781	0.0443091				
1619	00:01:04.720		5.42189623	15.36188264	0.57920182				
1620	00:01:04.760		6.851384017	18.84882453	0.10381711				
1621	00:01:04.800		6.604472778	14.7213006	0.44201662				
1622	00:01:04.840		6.720494834	11.60903877	0.03044522				
1623	00:01:04.880		6.81040857	12.34631245	0.26814548				
1624	00:01:04.920		6.130557399	11.76616267	0.45155939				
1625	00:01:04.960		6.743222106	9.741681676	0.39137201				
1626	00:01:05.000		7.288613455	12.09249692	0.21936518				
1627	00:01:05.040		7.270191487	14.72734382	0.00293867				
1628	00:01:05.080		7.599156318	16.2985828	0.64025816				
1629	00:01:05.120		7.273959617	13.59121718	0.12928556				
1630	00:01:05.160		4.378582488	10.70859797	0.30655354				
1631	00:01:05.200	1	67.48584384	141.4719405	19.5423658	1			
1632	00:01:05.240		14.50348374	33.89645938	0.94038929			1	
1633	00:01:05.280		15.31517995	34.84524599	0.7455999				
1634	00:01:05.320		14.3960565	34.31948526	0.72612728				

Obrázek 2.5: Na obrázku je část výstupu detekčního algoritmu pro detekci střihu ve formátu **.csv**. Tato část popisuje oblast videa, ve které nastává přímý střih (přímo se jedná o řádek obsahující ve sloupci **Frame Number** hodnotu 1631). Zde moment střihu není dostatečně očividný. Data nejsou nijak upravena, jen přečtena pomocí MS Excel.

Na obrázku 2.5 je několik datových sloupců, které bych rád krátce popsal – detailnější popis a jejich význam je vysvětlen v kapitole 4.4. v první skupině jsou datové sloupce vytvořené samotnou knihovnou.

- **Frame Number** – číslo snímku ve videu
- **Timecode** – odhadnutý čas ve videu pomocí čísla snímku
- **content_val** – hodnota váženého průměru HSV a počtu hran, viz kapitola 3.3
- **delta_edges** – rozdíl počtu hran několika sousedních snímků
- **delta_rgb** – rozdíl barevnosti několika sousedních snímků

v této druhé skupině jsou hodnoty, které jsem přidal já do této datové sady pro ulehčení analýzy funkčnosti algoritmu pro detekci střihu.

- **conf** – hodnota důvěry (angl. „confidence“) v detekci na škále $< 0, 1 >$, prázdné pole znamená, že nebyla provedena detekce postav
- **det_cut** – booleovská hodnota popisující výsledek detekce střihu
- **diff** – rozdíl hodnot detekovaných postav ve dvou snímcích okolo podezřelého snímku
- **z_diff** – rozdíl odhadované hloubky (osa **z**) klíčových bodů na detekovaných postavách ve dvou snímcích okolo podezřelého snímku
- **is_cut** – ručně vložené hodnoty (**1** nebo prázdné pole) signalizující reálná místa ve videu, kde střih nastal; pro porovnání funkce algoritmu

Během ladění jsem tedy tyto hodnoty četl a analyzoval v nástroji **MS Excel**[8], který mi přišel na toto velmi vhodný. Užitečnou funkcí bylo zvýrazňování buněk podle barevné škály – v **MS Excel** jsem na to použil nastroj **podmíněné formátování**, který právě toto umožňuje pro velké množství buněk najednou. Tím jsem mohl z až několika tisíc řádků dat velmi jednoduše okem poznat, kde nastává nějaká změna ve videu a analyzovat, jak se v ten moment detekční software chová. Ukázka barevně odlišených hodnot je na obr. 2.6.

Na obrázku 2.6 je sice je lidským okem krásně viditelné, kde ke střihu dojde, ale tak to nebylo úplně vždy. To proto, že na zmíněném obrázku je příklad výstupu v oblasti přímého střihu. To je nejjednodušeji detekovatelný střih a tím pádem dobře rozpoznatelný i právě lidským okem v datech. Často ty hodnoty vypadaly daleko více uniformě, a i tak se jednalo o bod střihu. Ovšem i tak byl Excel extrémně užitečným nástrojem, bez kterého by pravděpodobně byla kvalita nástroje daleko menší.

[8] https://www.microsoft.com/en/microsoft-365/excel

Frame Number	Timecode	conf	content_val	delta_edges	delta_rgb	det_cut	diff	is_cut	z_diff
1618	00:01:04.680		5.175585364	11.66342781	0.0443091				
1619	00:01:04.720		5.42189623	15.36188264	0.57920182				
1620	00:01:04.760		6.851384017	18.84882453	0.10381711				
1621	00:01:04.800		6.604472778	14.7213006	0.44201662				
1622	00:01:04.840		6.720494834	11.60903877	0.03044522				
1623	00:01:04.880		6.81040857	12.34631245	0.26814548				
1624	00:01:04.920		6.130557399	11.76616267	0.45155939				
1625	00:01:04.960		6.743222106	9.741681676	0.39137201				
1626	00:01:05.000		7.288613455	12.09249692	0.21936518				
1627	00:01:05.040		7.270191487	14.72734382	0.00293867				
1628	00:01:05.080		7.599156318	16.2985828	0.64025816				
1629	00:01:05.120		7.273959617	13.59121718	0.12928556				
1630	00:01:05.160		4.378582488	10.70859797	0.30655354				
1631	00:01:05.200	1	67.48584384	141.4719405	19.5423658	1			
1632	00:01:05.240		14.50348374	33.89645938	0.94038929			1	
1633	00:01:05.280		15.31517995	34.84524599	0.7455999				
1634	00:01:05.320		14.3960565	34.31948526	0.72612728				

Obrázek 2.6: Na tomto obrázku je stejná část dat jako na obr. 2.5, avšak data jsou barevně zvýrazněna pomocí podmíněného formátování poskytnutého programem MS Excel. na řádku 1631 je po podmíněném formátování očividné, že nastává nějaká změna v obrazu. Změna je viditelná v téměř všech datových sloupcích. Data také nejsou nijak upravena, jen barevně zvýrazněna.

Kapitola 3

Návrh a rozdělení nástroje

V dnešní době se velmi hojně využívá modulární programování a není tomu jinak ani v mém případě. Téměř na každý menší nebo větší úkol existuje nějaká knihovna, která se postará o základní strukturu řešení problému a programátor si poté pouze volá knihovní metody tak, jak potřebuje a uzná za vhodné. Od zpracovávání argumentů příkazové řádky přes procházení souborů v počítači až po zpracovávání videí, knihovna se dá dnes najít už opravdu na hodně případů. a nejenom proto jsem vybral programovací jazyk **Python**.

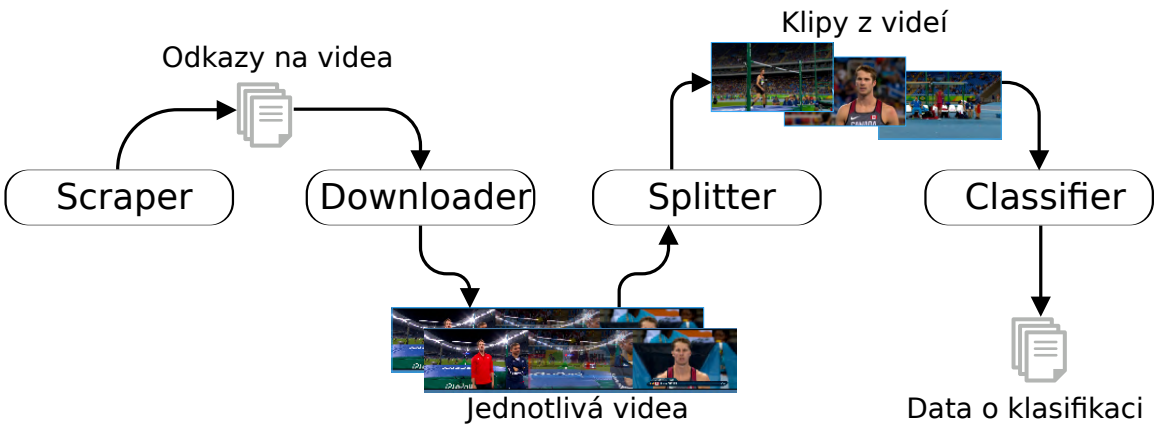
Program jsem tedy navrhoval s vědomím toho, že budu pracovat v jazyce, který dodržuje principy objektově orientovaného programování. a nejen proto jsem nástroj rozdělil do čtyřech logických jednotek. Z těchto jednotek má každá na starost jinou část procesu vytváření datové sady. Důvodů pro to je několik. Cílem vytvoření tohoto nástroje je jeho jednoduchá a univerzální využitelnost. Tím pádem je jen na uživateli, jak se rozhodne nástroj využít, na jaká data je aplikuje a kterou část nástroje na tato data využije. Zároveň byl tento přístup ideální během vývoje, kdy na sebe tyto jednotky přímo navazují, a tak bylo zřejmé, co je třeba implementovat, co chybí, kde jsou rezervy a co už je připravené.

Nástroj jsem tedy rozdělil na čtyři samostatné jednotky – dále je budu nazývat **skripty** (i když se jedná o sady skriptů) – které na sebe logicky navazují. Není nutné využít všechny, i když je to vřele doporučované. Datové struktury výstupu jednoho skriptu jsou většinou vybrány tak, aby byly přímo použitelné jako vstupy navazujícího skriptu. Názvy jednotlivých částí jsou vybrány anglicky – to proto, že se jedná o programové jednotky a v programování jsou běžné, až téměř konvenční, názvy anglické. Dělení je následovné:

- **Scraper**
- **Downloader**
- **Splitter**
- **Classifier**

První dva skripty jsou funkční na obecná videa v případě, že uživatel zadá správné vstupní parametry. Třetí skript – **Splitter** – už je trochu více specifický, je testován primárně na videích skoku do výšky a je vytvořen se záměrem dělení právě takových videí. Myslím si, ale že by měl poměrně slušně fungovat i na videa z jiných sportů nebo obecně jiná videa – ovšem nemohu zaručit to, jak se bude skript chovat v takových podmínkách. Poslední skript – **Classifier** – je již navržen pro specifický sport a to právě skok do výšky. Zde už nebylo na výběr a jelikož je mi skok do výšky velmi blízký, tak bylo rozhodnutí

jednoduché. Pro zjednodušení představy toho, jak nástroj funguje a jak na sebe jednotlivé skripty navazují přikládám grafiku na obrázku 3.1 popisující celý systém.



Obrázek 3.1: Obrázek popisuje strukturu systému. Šipky vedoucí dovnitř buňky značí vstupy daného skriptu a šipky vedoucí ven značí výstupy.

3.1 Scraper – vyhledání videí na YouTube

Tento první skript je pravděpodobně ze všech nejjednodušší. Cílem skriptu je získat odkazy na videa z **YouTube** a uložit je do textového souboru. K vyhledávání na **YouTube** je potřeba několik vstupů od uživatele, aby bylo co nejefektivnější.

Jako první a nejdůležitější je potřeba vložit klíčové slovo, pomocí kterého se najde prvotní sada videí. Výběrem správného klíčového slova může uživatel nejvíce ovlivnit nalezené výsledky. Výchozí hodnotou v případě, že uživatel hodnotu nezadá, je **high jump** (česky „skok vysoký“.

Druhým vstupem od uživatele je seznam zakázaných slov v titulku, názvu kanálu a klíčových slovech u videa. Tento seznam je ale také extrémně důležitý. Například při vyhledávání pomocí klíčového slova **high jump** je seznam výsledků naplněn videi o skocích do vody, různými herními videi anebo dokumenty o operaci *High Jump*[1]. Pro skok vysoký jsem vytvořil sadu filtrů, která právě tyto, pro nás nezajímavá, videa vytřídí a ponechá pouze ta zajímavá. Porovnání použití skriptu bez použití filtrů a s nimi je na obrázcích 3.2 a 3.2. Lze si všimnout, že s použitím filtrů je relevantních videí devět z desíti a bez použití filtrů pouze pět.

Třetím vstupem je celé číslo stanovující počet odkazů, které má program vyhledat. Posledním vstupem od uživatele je cesta k textovému souboru, do kterého se mají nalezená videa uložit.

3.2 Downloader – stažení nalezených videí z YouTube

Druhý skript velmi blízce navazuje na ten předchozí. Jako vstup zpracovává seznam odkazů, který vytvořil právě **Scraper**. Cílem je z těchto odkazů videa stáhnout a uložit do lokální složky. Kromě běžných uživatelských vstupů používá ještě omezení na maximální velikost všech videí ke stažení. na **YouTube** má téměř každé video několik možných variant, které

[1] https://en.wikipedia.org/wiki/Operation_Highjump

1. HUGE HIGH JUMP vs Subscribers!, Nick Symmonds
2. 😲 4 Feet High Jump #training || #shorts , #viral , #ytshorts , #ssc_cpo , #trending ., MARCOS PHYSICAL ACADEMY
3. WOMEN'S HIGH JUMP QUALIFICATION ISTANBUL 2023 EUROPEAN INDOOR ATHLETICS CHAMPIONSHIPS ALL ATHLETICS, EVOLUTION SPORTS
4. Most Beautiful Moments Women's High Jump Beskydska Latka Trinec 2023 Athletics, CM4sports
5. Long jump, Mauryan Empire2.0
6. 2.24M High jump of Sarvesh Kushare at National jumps championship 2023, Indian Athletes
7. CARIFTA50: High Jump Heptathlon Girls - Event 2 - Part 1 | SportsMax TV, SportsMax TV
8. High jump, Track Pain
9. UNBELIEVABLE Long Jump vs Subscribers! #NSTC, Nick Symmonds
10. HOW HIGH can my HORSE JUMP? // Thoroughbred High Jump, Holly Lenahan

Obrázek 3.2: Na obrázku je seznam prvních deseti videí bez použití filtrů. Zde je zhruba polovina videí nerelevantních k našem vyhledanému slovu.

1. Men's High Jump Final | Rio 2016 Replay, Olympics
2. WOMEN'S HIGH JUMP QUALIFICATION ISTANBUL 2023 EUROPEAN INDOOR ATHLETICS CHAMPIONSHIPS ALL ATHLETICS, EVOLUTION SPORTS
3. WOMEN'S HIGH JUMP FINAL 2023 EUROPEAN INDOOR ATHLETICS CHAMPIONSHIPS FULL REPLAY HIGH JUMP 2023, EVOLUTION SPORTS
4. The Highest Ever Olympic High Jumps! | Top Moments, Olympics
5. Men's High Jump final | Tokyo Replays, Olympics
6. Yaroslava Mahuchikh claims GOLD in the Women's High Jump Final | Munich 2022, Athletics Fan
7. High Jump WOMEN. Australian championship. Highlights, gbukatin
8. Most Beautiful Moments Women's High Jump Beskydska Latka Trinec 2023 Athletics, CM4sports
9. High Jump - 3FT to 4'5FT Only One Day 😠 ख़ुद देखो कैसे | Best High Jump Technique in 2023, Sports Basti
10. Most Beautiful Moments Women's High Jump Catalunya World Indoor Tour 2023 Athletics, CM4sports

Obrázek 3.3: Na obrázku je seznam prvních deseti videí s použitím filtrů. v tomto případě je relevantních 9 z 10 výsledků.

se dají stáhnout. Různí se například v kodeku, který byl použit. **YouTube** jich nabízí ke stažení rovnou několik, pro zjednodušení navazující práce budou vybrány v pořadí podle priority tyto kodeky: **avc1**, **av01** a **vp9**. Zároveň mě samozřejmě zajímá kvalita videa. Pro některé části budu sice u videa snižovat kvalitu pro zrychlení operací, ale někde se bude hodit co nejvyšší kvalita. Videa se různí hodně v jejich rozlišení, kvalitě videa i stáří, ale pro naše účely vytváření datové sady je to téměř vhodné, přinese to do datové sady trošku variace.

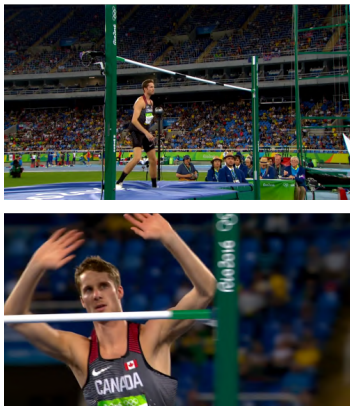
3.3 Splitter – rozdělení nalezených videí na klipy

Splitter má na starost automatizované dělení videí na jednotlivé klipy pomocí detekce střihů. Tyto střihy jsou detekovány pomocí knihovny **PySceneDetect** upravenou pomocí vlastního detekčního algoritmu. Jedním ze vstupních argumentů je cesta k videu, které je třeba rozdělit. Hlavním výstupem je složka obsahující množství videí získaným pomocí dělení originálního videa na části.

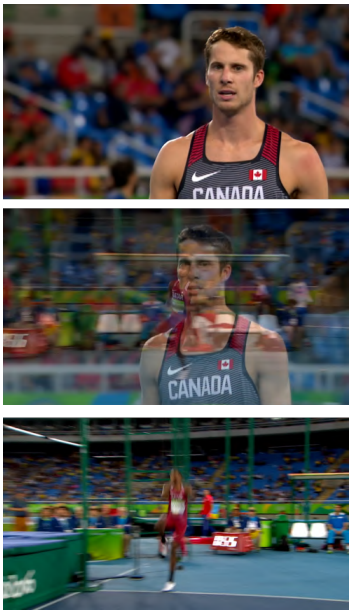
Typy střihů, na které program může narazit jsou tři: přímý střih, postupný střih a střih s animací. Jejich příklady jsou na obr. 3.4, 3.5 a 3.6.

Každý z těchto typů střihu vyžaduje jiný přístup k řešení. Pro první typ, přímý střih, se jedná o poměrně jednoduchý princip, ale u ostatních dvou se jedná o poměrně komplexní systém.

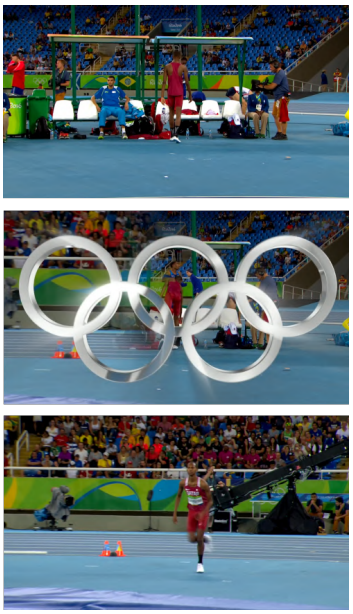
Princip, na základě kterého bude prováděna detekce přímého střihu, je porovnávání určitých hodnot dvou sousedních snímků. Těchto hodnot je několik. Základní hodnoty jsou hodnoty HSV. Těmi jsou odstín, saturace a světlost (z angl. „hue, saturation and brightness value“). Z hodnoty světlosti se ještě dále vypočítává hodnota počtu hran. Tyto čtyři hodnoty se získávají pro každý pixel a z nich se pro celý obrázek vytvoří vážený průměr [11]. Tato finální hodnota v kombinaci s hodnotami jednotlivých částí rozhoduje o tom, zda se jedná



Obrázek 3.4: Dva sousední snímky přímého střihu



Obrázek 3.5: Tři snímky v rámci 10 snímků popisující postupný střih



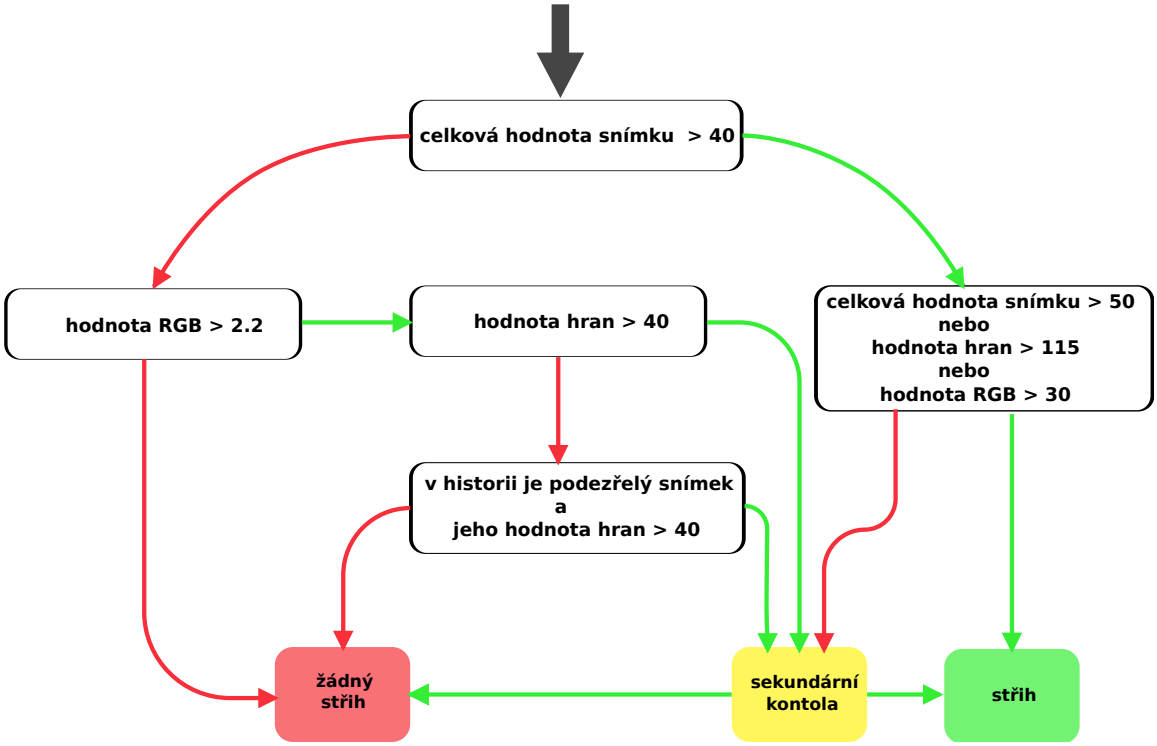
Obrázek 3.6: Tři snímky v rámci 10 snímků popisující animovaný střih

o přímý střih, nebo ne. Příklad těchto hodnot je v kapitole 4.4. Tento způsob porovnávání dvou snímků je velmi efektivní a ideální pro tento první případ.

Princip pro detekci zbylých dvou případů leží v podobné rovině, i když tento nápad je trochu více rozvinut. Stejně jako u přímého střihu je pro každý snímek vytvořena hodnota složena z váženého průměru hodnot HSV a počtu hran, navíc je využita průměrná hodnota RGB ve snímku. Přesný postup výpočtu je v kapitole 4.4. Pokud tedy aktuální snímek překoná určitou hodnotu RGB a zároveň určitý počet hran, anebo je v historii posledních osmi snímků snímek, který překročil hodnotu hran, ale ne hodnotu RGB, tak je označen jak podezřelý. v tento moment totiž nejsou snímky, které překročí určité hodnotu, označovány definitivně jako místa střihu, ale pouze jsou označena jako podezřelá místa nachystaná na sekundární zpracování. Tento proces je náročný na vysvětlení, samotnou implementaci popisuji kapitole v 4.4. Obrázek 3.7 však obsahuje graf popisující první část této metody, která má na starost nalezení podezřelých snímků.

Poslední chybějící dílek do této skládačky je metoda popisující blok označen jako **sekundární kontrola** v obrázku 3.7. Jak jsem zmínil výše, snímky, které nemají dostatečně vysoké hodnoty na to, aby prošly detekcí na přímo, ale mají je aspoň dostatečně vysoké na to, aby bylo považovány za podezřelé, tak tyto snímky projdou ještě druhotnou kontrolou. Tato kontrola finálně rozhodne o tom, zda se jedná o střih nebo ne.

Principem této metody je porovnávání detekovaných postojů na dvou okolních snímcích okolo podezřelého snímku. Můj nápad byl totiž následovný: v případě, že nastane změna scény ve videu, tak na prvním a druhém videu bude pravděpodobně vidět jiná část lidského těla, nebo třeba také žádná. Cílem je tedy vytvořit metodu, která porovná tyto dvě případné postavy a při dostatečném rozdílu vyhodnotí původní snímek jako střih. Pro lepší vizualizaci opět situaci popisuji na obrázku 3.8



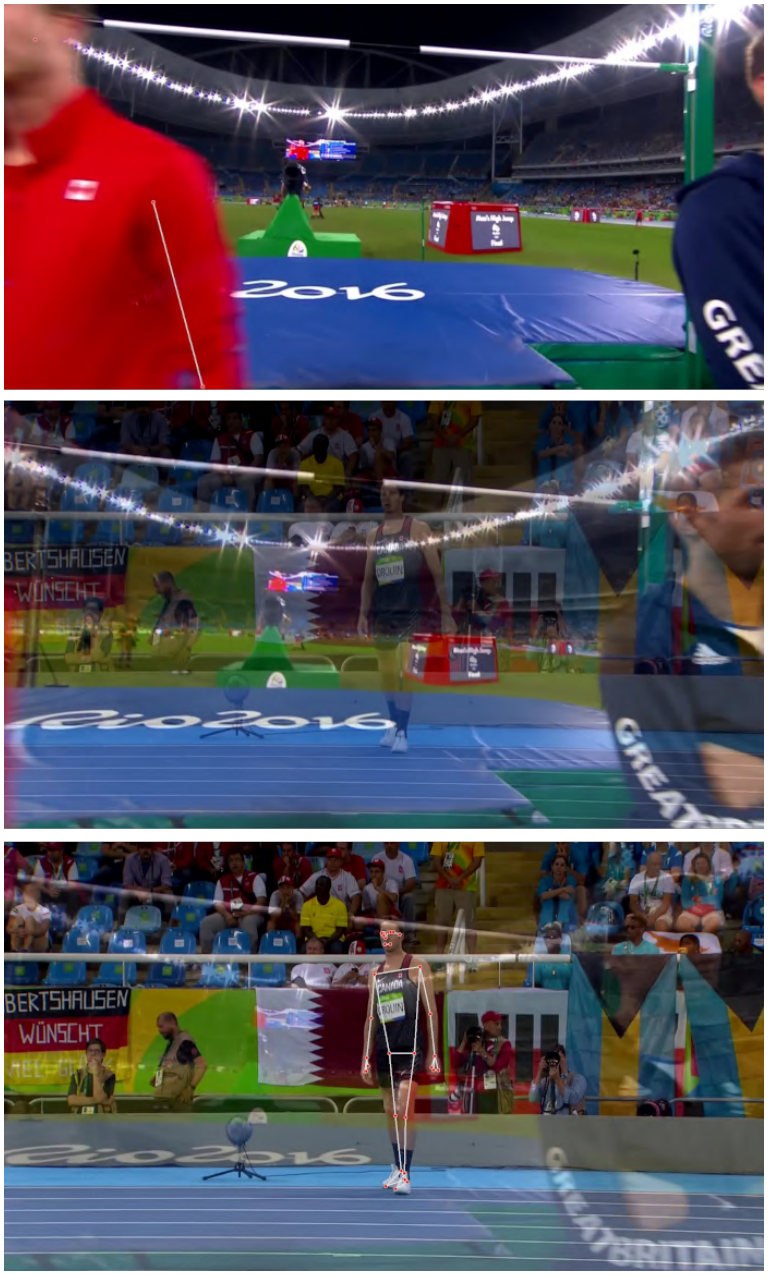
Obrázek 3.7: Na obrázku je graf popisující zpracovávání a vyhodnocení snímku. Vstupem do grafu je černá šipka na horní straně. Zelené šipky značí úspěch (pravdu), červené neúspěch (nepravdu).

3.4 Classifier – klasifikace rozdělených klipů podle relevance

Klasifikátor je skript, který tvoří poslední krok tohoto nástroje. Má na starost určení, v jaké míře se jedná o video skoku do výšky či nikoliv. v ideálním případě by k tomu byla využita neuronová síť, která by právě tato videa byla schopná rozpoznat, ale jelikož je cílem této práce vytvořit datovou sadu právě na případné trénování oné neuronové sítě, tak to z logického hlediska nelze. Proto je třeba najít nějaké konstantní vlastnosti videí skoku do výšky, které nám umožní je rozpoznat od videí ostatních. Videa stahovaná z **YouTube** mají totiž dost částí, které nejsou pro účel této práce relevantní – záběr na diváky, záběr na trenéra nebo na závodníky čekající na svůj pokus. Nás ovšem zajímají pouze videa čistě skoku do výšky.

Mým prvním plánem bylo opět využít detekci postav na videu a pomocí algoritmu opět porovnávat průměrné počty osob na videu, viditelnost celého člověka na videu nebo jiné metriky. To se ovšem ukázalo jako poměrně náročný úkol. Ve skoku do výšky totiž nastane v jednu chvíli situace, kdy se postava velmi špatně detekuje. Od momentu odrazu od země, přes překonávání skokanské laťky, až po dopad do matrace se dostává skokan do pozic, které nejsou konzistentně detekovány pomocí detekčních nástrojů, které jsem používal. Takovou pozici popisuje obr. 3.9. Proto jsem se nakonec rozhodl tuto metodu nepoužít.

Druhým nápadem bylo detekovat celou osobu na videu, ne jeho postavu s klíčovými body. Předchozí nápad cílil na samotný skok přes laťku, v této metodě jsem chtěl využít rozběh skokana. Chtěl jsem detekovat horizontální pohyb skokana na videu a podle toho poznat, zda se jedná o skok do výšky či nikoliv. Bohužel ani to nebylo velmi konzistentní.



Obrázek 3.8: První snímek je několik snímků před podezřelým, detekována je pouze ruka. Prostřední snímek je podezřelý. Spodní snímek je několik snímků po podezřelým, detekovaná je celý postoj.

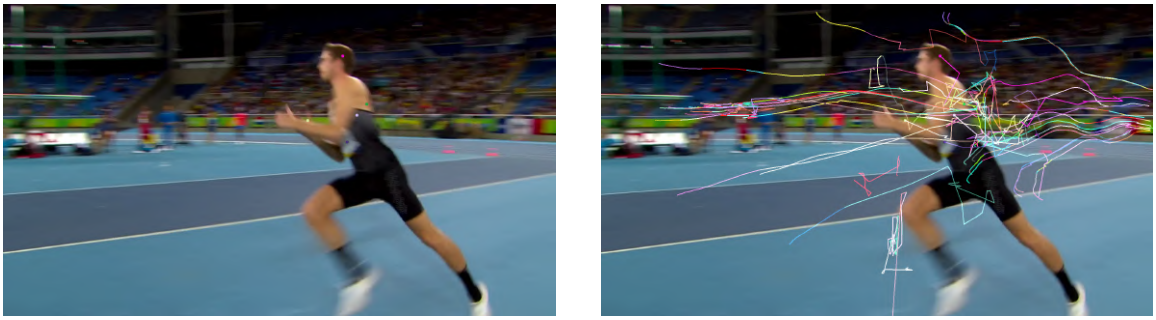
Jelikož se jedná o opravdu rychlý pohyb, tak se často stává, že kameraman nezvládá udržet skokana ve stejné pozici během celého skoku a tím pádem se horizontální vektor osoby měnil natolik, že jsem s tím nezvládl pracovat.

Poslední a finálně použitý nápad byl mírně inspirován tím předchozím. na rozdíl od druhého nápadu, kde byl plán detekovat horizontální pohyb postavy, jsem rozhodl využít tento nápad trochu jinak. Zkusil jsem detekovat také pohyb, ale ne osoby, ale pozadí. K tomu jsem využil právě optického toku, který jsem popsal v kapitole 2.2.4. Pro optický tok

Obrázek 3.9: Obrázek zobrazuje olympijského vítěze z Ria Dereka Drouina během skoku do výšky. Při pokusu o provedení detekce postavy na tomto obrázku je výsledkem neúspěch. Autorem fotky je Lucas Oleniuk [20].

.

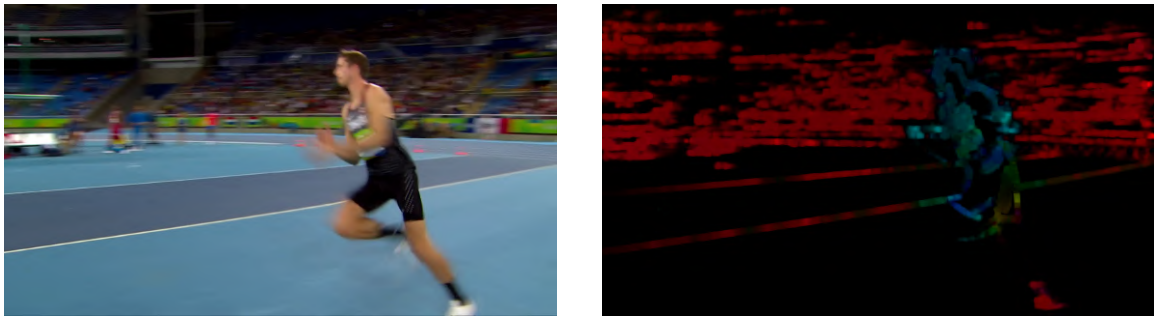
jsem testoval jeho dvě hlavní metody a to **Lucas-Kanade** a **Farneback dense**. První z nich funguje bodově – to znamená že vybere nějaké body ve videu (nejlépe rohy nebo hrany) a ty sleduje a případně vykresluje jejich pohyb obrazem. To se ale pro naši implementaci nehodí, protože výběr bodu na sledování nefungoval nejlépe, jak je vidno na obrázku 3.10.



Obrázek 3.10: Na levém obrázku je snímek z finále Olympijských her v Rio de Janeiro [21]. na pravém obrázku je vstup metody **Lucas-Kanade** vložen přes původní obrázek. Lze poznat, že i když některé zachycené body mají detekovaný pohyb, není to nic konzistentního a pro naši metodu tudíž nepoužitelné.

Proto jsem se rozhodl využít metodu **Farneback dense**, která vypočítává optický tok pro celý obraz [4]. Příklad toho, jak metoda **Farneback dense** funguje je na obrázku 3.11.

Zde je určitě hodně prostoru pro zlepšení, jeden z nápadů pana prof. Ing. Herouta, Ph.D., byl pomocí segmentace detekované postavy odstranit ze snímků postavu člověka, aby byl optický tok detekován pouze pro pozadí. Tomu jsem se také chvilku věnoval, ale problém byl opět podobně jako u druhého nápadu takový, že při samotném skoku mimo rozběh není často detekovaná postava a tím pádem ani není co segmentovat. Proto jsem se rozhodl zůstat u tohoto jednoduššího, nicméně stále dobře funkčního řešení.



Obrázek 3.11: Na levém obrázku je snímek z finále Olympijských her v Rio de Janeiro [21]. na pravém obrázku je výstup po aplikaci metody **Farneback dense** na toto video. Jelikož kameraman drží sportovce ve středu záběru, tak má pozadí hodnotu optického toku větší a sportovec má hodnotu optického toku minimální.

Kapitola 4

Implementace nástroje

V této kapitole popíšu samotnou implementaci toho, co jsem navrhl v kapitole předchozí. Jak jsem již zmínil v kapitole 2.2.1, vybraným programovacím jazykem, který jsem využil ve vytváření tohoto nástroje, byl Python 3. Konkrétní verze Pythonu 3 by neměla ovlivnit funkci programu, ale silně doporučuji verzi Python 3.8 nebo novější. Během vytváření kódu jsem se snažil dodržovat pravidla psaní čistého kódu, jak nám bylo vštípeno v několika předmětech během našeho studia. Kód by měl být tedy samonosný a samopopisný, ovšem v některých složitějších konstrukcích se může programátor snažit sebevíce, ale bez okomentování nikdo nepozná, o co se jedná. Proto jsem radši vložil komentářů více nežli méně. Snažil jsem se dodržovat zásady popsané v oficiálním dokumentu o psaní čistého kódu v Pythonu [6]. Většina částí nástroje je také dělena do tříd, což usnadní práci mně – programátorovi – ale i lidem, kteří budou po mně kód studovat nebo číst. Zároveň je to opět dobrá praktika, kterou dodržovat při užívání jazyka, který podporuje objektově orientované programování [13].

Každá část nástroje, ať hlavní, či vedlejší, obsahuje soubor *requirements.txt*, který obsahuje informace o verzích a knihovnách potřebných pro správnou a testovanou funkčnost nástroje. Pro instalaci knihoven pomocí nástroje PyPI [3] lze využít tento příkaz ve stejném adresáři, kde je uložen soubor *requirements.txt*:

```
python3 -m pip install -r requirements.txt
```

Výpis 4.1: Příkaz pro instalaci požadovaných knihoven

V případě, že by byla požadovaná verze nedostupná, lze nainstalovat verzi nejnovější. Pokud by ani to nefungovalo, neváhejte mě kontaktovat.

4.1 Společné konstrukce

V této další kapitole bych rád zmínil dvě programovací konstrukce, které se velmi pravidelně opakují ve všech skriptech. Tyto konstrukce nejsou nijak komplexní, ale je důležité je vysvětlit zde, aby zbytečně nezahltil jejich popis důležitější části. Jedná se o téměř duplicitní sekce kódu s velmi malou variabilitou, které se ale bohužel nedají dobře sjednotit. Jedná se za prvé o konstrukci umožňující odlišení vstupu do programu z příkazové řádky nebo použití souboru jako modulu a za druhé metodu pro zpracování vstupních argumentů. Daly by se také nazvat jako tzv. „boilerplate“[1].

[1] https://en.wikipedia.org/wiki/Boilerplate_code

První z těchto konstrukcí je vstupním bodem programů. Tato konstrukce vypadá následovně:

```
if __name__ == '__main__':
    # some code
```

Výpis 4.2: Konstrukce pro vstup do programu z příkazové řádky

Tato část kódu proběhne pouze tehdy, když je skript zavolán z příkazové řádky. Pokud by uživatel využil soubory jako moduly, tak tato část kódu neproběhne. Proto jsou běžně v této konstrukci volány ostatní metody, které už obsluhují samotnou funkci programu.

Druhou konstrukcí je zpracovávání vstupních argumentů z příkazové řádky. Tuto práci jsem delegoval na knihovnu **argparse**, která je přesně k těmto účelům vytvořená. Jak příklad vezmu metodu ze skriptu pro spojování videí.

```
parser = argparse.ArgumentParser(description='Merge two videos.')
parser.add_argument('first', type=str, help='The first video path')
parser.add_argument('second', type=str, help='The second video path')
```

Výpis 4.3: Ukázka zpracování vstupních argumentů

Toto je příklad jednoho z jednodušších využití konstrukce, avšak mohu na něm dobře ukázat, jak tento styl konstrukce funguje. Nejprve je vytvořen samotný objekt, který má na starost zpracovávání argumentů. K němu jsou poté přidány samotné argumenty. Tyto argumenty mají mnoho parametrů a umožňují hodně svobody ve zpracovávání vstupních argumentů. Ty, které jsem využil, jsou:

- **name** – název argumentu, může být v krátkém (**-h**) nebo dlouhém (**--help**) tvaru nebo jako poziční argument
- **action** – specifikuje, jak naložit se získanou hodnotou, v našem případě nejčastěji uložit (**store**) nebo uložit booleovskou hodnotu (**store-true**).
- **default** – hodnota, kterou argument získá v případě, že není poskytnuta uživatelem
- **type** – očekávaný typ vstupní hodnoty
- **choices** – možnosti, které jsou uznávány jako validní
- **required** – přepínač označující argument za vyžadovaný

Více informací je v dokumentaci knihovny [1].

4.2 Scraper – vyhledání videí na YouTube

Tento skript je ze všech čtyřech hlavních skriptů pravděpodobně ten nejjednodušší. Jeho funkce je pomocí klíčového slova najít množství videí na **YouTube** a ten přetřídit pomocí seznamu filtrových klíčů tak, aby videa co nejvíce odpovídala klíčovému slovu. K těmto videím je získáno URL a je uloženo do textového souboru.

Hlavní funkcí, která má na starost prohledávání **YouTube** je `get_urls()`. Knihovnou, která je využita právě pro prohledávání **YouTube** je knihovna **youtubesearchpython**. Tato metoda má jeden parametr, který je výstupem funkce, která zpracovává vstupní argumenty. Objekt, který je takto předán obsahuje všechny informace získané od uživatele.

`get_urls()`

Po inicializaci proměnných je zkontrolováno, zda již existuje výstupní soubor, a pokud ano je nahrazen novým prázdným a ten je otevřen v režimu zápisu. Jednou z inicializovaných proměnných je instance objektu `VideoSearch` z knihovny **youtubesearchpython**.

```
search_daemon = VideosSearch(searchword, limit=(args.required * 10))
```

Výpis 4.4: Ukázka vytvoření instance VideoSearch třídy

Tento objekt při inicializace vyžaduje jako parametr právě klíčové slovo, pomocí kterého bude vyhledávat, a volitelný parametr pro omezení počtu výsledků. Počet výsledků zadávám jako desetinásobek toho, co zadal uživatel, aby se vykompenzovala vyfiltrovaná videa. Následně ve smyčce program iteruje skrze výsledky vyhledávání. Pro každý výsledek je provedena kontrola filtry, které jsou zadané v externím souboru. v souboru **blacklists.py** jsou přiloženy filtry pro skok vysoký, avšak každý uživatel může vytvořit po vzoru již hotových filtrů svoje vlastní filtry pro jiná klíčová slova. Následující kód je příklad aplikace filtru – nejprve je iterováno přes všechny zakázaná slova ve filtru a ta jsou porovnávána se všemi klíčovými slovy u videa. v případě, že je nalezena shoda je změněn přepínač pro úspěch a smyčka ukončena.

```
for keyword in keyword_blacklist_en:
    if keyword in [x.lower() for x in result["keywords"]]:
        fail = True
        break
```

Výpis 4.5: Příklad filtrů při vyhledávání v YouTube

Název videa, klíčová slova a název kanálu jsou všechny získány ze zmíněného objektu. v případě, že projde video skrz filtry je zkontrolována minimální velikost videa, aby se odstranila **YouTube Shorts** videa. Při úspěšném průchodu i tímto filtrem je získána z objektu i cesta URL a zapsána do otevřeného textového souboru. v případě jakéhokoliv neúspěchu během smyčky vyhledávání je využit příkaz **continue** pro pokračování dalším výsledkem. Při splnění počtu nalezených videí je smyčka ukončena a uzavřen textový soubor s URL cestami.

4.3 Downloader – stažení nalezených videí z YouTube

Druhým skriptem přímo navazuji na **scraper**. Prakticky by se daly tyto dva skripty spojit dohromady, avšak kvůli širší využitelnosti jsem se rozhodl skripty rozdělit. **Downloader** má na starost stahování videí, na které ukazují cesty z textového souboru, který je výstupem **scraperu**. K tomu je využita knihovna **PyTube**.

Hlavní metodou tohoto skriptu je metoda `download_links()`. Tato metoda má dva parametry, první je pole obsahující řetězce získané z textového souboru s URL adresami. Druhý parametr jsou zpracované vstupní argumenty příkazové řádky.

4.3.1 `download_links()`

Po inicializaci základních proměnných pro chod programu začíná hlavní smyčka. v této smyčce je iterováno přes všechny řetězce v poli řetězců adres URL. Pro každý řetězec je poté vytvořen objekt **YouTube** z knihovny **PyTube**. Instance tohoto objektu obsahuje veškeré informace potřebné ke stažení videa. Jedním z atributů této instance je pole s jednotlivými

proudy (angl. „streams“). Tyto proudy se různí v několika vlastnostech, pro nás je důležitá kvalita videa a kodek. Kvalitu videa program vybírá nejlepší možnou. s kodeky je to mírně komplikovanější. Kodeky jsem vybíral pro nejlepší kompatibilitu s navazujícími skripty, proto jsem je vybíral v tomto pořadí:

1. **avc1** – Také znám jako **h.264** je nejvíce používaný kodek [5], tudíž i pro nás je prioritním – je kompatibilní se všemi dále využitými nástroji
2. **av1** – Tento kodek na rozdíl od **avc1** podporuje open-source implementace a údajně dosahuje lepší datové komprese [12].
3. **vp9** – Jedná se o předchůdce**av1** [12], některá videa bohužel existují jen s tímto kodekem a proto se jedná o poslední možnost.

Tohoto prioritního výběru jsem v kódu dosáhl pomocí kaskády ošetřování výjimek. Níže je zkrácená verze této sekce v kódu.

```
try:
    stream = yt.streams.filter(
        adaptive=False, video_codec="avc1.640028"
    ).order_by("resolution")[-1]
except IndexError:
    try:
        stream = yt.streams.filter(
            adaptive=False, video_codec="av01.0.00M.08"
        ).order_by("resolution")[-1]
    except IndexError:
        try:
            stream = yt.streams.filter(
                adaptive=False, video_codec="vp9"
            ).order_by("resolution")[-1]
        except IndexError:
            continue
```

Výpis 4.6: Ukázka procesu výběru kodeku u videa

V případě, že není nalezen proud obsahující video s požadovaným kodekem, metoda `order_by()` vyvolá výjimku, která je zachycena. Jako ošetření této výjimky je znovu volána metoda `filter()`, jen vybrán jiný kodek. Při neúspěchu je toto opakováno znovu, přičemž pokud není nalezen ani jeden vhodný proud, tak je tento odkaz přeskočen a program pokračuje dalším v poli odkazů.

Při úspěšném nalezení vyhovujícího proudu nastávají další kontroly. První zkontroluje, zda je video již staženo. Pokud ano, opět přeskakuje stažení a pokračuje dalším. Druhá kontrola hlídá, aby celkový počet videí nepřesáhl velikost zadanou uživatelem. v případě, že by video stažením přesáhlo celkovou maximální velikost, je přeskočeno a pokus se opakuje do té doby, než nejsou otestována všechna videa. Pokud obě kontroly projdou úspěšně je video staženo do specifikované složky odpovídající kodeku. Celý proces stažení videa je časován a v případě zapnutého přepínače pro tisk dodatečných informací jsou tyto informace o rychlosti a době trvání stažení vypsány na standardní výstup.

Bohužel **YouTube** v nedávné době pravděpodobně změnil API, přes kterou se dalo přistupovat k videím a jejich datům a tím pádem bohužel tento nástroj již nefunguje tak

jak by měl. Změna nenastala ani ve verzi použité knihovny, ani v mém kódu – bohužel se nyní nedá nic dělat. Naštěstí jsou nyní k dispozici aktualizované knihovny, které přístup opět umožňují – příkladem jsou **pytube3** nebo **pytube4**.

4.4 Splitter – rozdělení nalezených videí na klipy

Skript, který má na starost rozdělení videí pomocí detekce střihů na kratší klipy je pravděpodobně nejvíce komplikovaný ze všech čtyř hlavních skriptů. Pro připomenutí, detekce střihu probíhá tak, že je pomocí rozdílů v HSV anebo počtu hran nalezen podezřelý snímek. Okolo tohoto snímku se v určitém rozpětí provede detekce postav na jiných snímcích. Tyto detekované postavy se porovnají a rozhodne se, zda se jedná o střih, nebo ne.

Tento samotný skript je rozdělen do čtyř souborů tak, aby byly oddělené jejich funkce od sebe. Každý program má na starost jinou část procesu dělení.

- **splitter.py** – Tento soubor je jádrem celého skriptu. Má na starost zpracování vstupních argumentů, přípravu výstupů, inicializaci hlavních instancí objektů a volání funkcí z ostatních souborů.
- **cut_detector.py** – Druhý soubor obsahuje třídu reprezentující potomka detektorů z knihovny **PySceneDetect**. Má na starost zpracovávání jednotlivých snímků, aktualizaci statistik, nebo volání detekčních metod na postavu z detektoru postav.
- **pose_detector.py** – Předposlední soubor obsahuje třídu, která má na starost detekci postav. Její metody jsou např. detekce osob na obrázku nebo porovnávání klíčových bodů těchto osob na dvou obrázcích.
- **values.py** – Poslední soubor pouze obsahuje různé hodnoty, které jsou nahrávány ostatními soubory. Těmito hodnotami jsou třeba hodnoty důvěry v detekci nebo minimální rozestup mezi detekcemi.

Nyní tyto jednotlivé soubory jeden po druhém popíši z pohledu implementace v kódu. Jediný soubor, který používá obecné konstrukce popsané v sekci 4.1, je **splitter.py**, který je zároveň vstupním bodem tohoto skriptu.

4.4.1 Jádro programu – splitter.py

Tento soubor slouží jako vstupní bod tohoto skriptu, a tudíž jako jediný obsahuje již zmíněné konstrukce pro volání všech metod a zpracování vstupních argumentů. Důležitý úkol je kontrola existence adresářů, do kterých se průběžně budou ukládat detekované obrázky, výstupy a podobně. Po zpracování vstupů a nachystání adresářů je zavolána metoda `find_scenes()`, která má jako návratovou hodnotu pole obsahující pozice detekovaných střihů ve videu.

Na začátku funkce `find_scenes()` je provedena kontrola úspěchu otevření videa. K tomu je využita konstrukce **try** – **except** s několika různými výjimkami k zachycení podle toho, zda např. video neexistuje, nejde přečíst nebo má špatnou snímkovou frekvenci.

Po kontrole videa je vytvořena instance objektu `SceneManager` z knihovny **PySceneDetect**. Jako parametr `stats_manager` je vložena nově vytvořená instance objektu umožňujícího zápis ladících hodnot do **.csv** souboru s názvem `StatsManager`. Objekt `SceneManager` umožňuje napojení vlastního detektoru do systému detekce střihu, který knihovna poskytuje. To je pro chod programu klíčové. Do tohoto objektu je tedy přidán vlastní detektor

(`CutDetector`). Nyní konečně lze zavolat knihovní funkci `detect_scenes()`, která v zadaném videu najde střihy pomocí algoritmu specifikovaného v programu `cut_detector()` v kapitole 4.4.2. Část programu pro vytvoření instancí a volání metody je v kódu níže.

```
scene_manager = SceneManager(
    stats_manager=StatsManager())
scene_manager.add_detector(
    CutDetector(
        video_path=args.path
    )
)
scene_manager.detect_scenes(
    video_file, show_progress=True
)
```

Výpis 4.7: Příklad inicializace instancí pro detekci střihů

V moment, kdy skončí běh metody na detekci střihů se zaplní instanční atribut v instanci objektu `SceneManager` polem s místy ve videu, kde byl detekován střih. Toto pole je využito jako návratová hodnota této metody.

Po dokončení běhu metody `find_scenes()` má program k dispozici pole obsahující odhadovaná místa střihů. Pokud je pomocí argumentů příkazové řádky uživatelem zapnutý přepínač `split`, je opět zavolána metoda knihovny **PySceneDetect** `split_video_ffmpeg()`. Tato metoda pomocí nástroje **ffmpeg** rozdělí celé vstupní video na jednotlivé klipy pomocí hodnot ze získaného pole, jak je vidno na kódu níže.

```
split_video_ffmpeg(
    args.path,
    scene_list,
    show_progress=True,
    output_file_template="clips/$VIDEO_NAME/$SCENE_NUMBER.mp4"
)
```

Výpis 4.8: Příklad volání metody pro rozdělení videa podle seznamu střihů

Výsledné klipy jsou uloženy do adresáře relativního k zdrojovému souboru. Cesta od zdroje je `/clips/název-videa/číslo-scény.mp4`.

4.4.2 Detektor střihu – cut_detector.py

Detektor střihu je reprezentován jako třída s názvem `CutDetector`. Tato třída je potomkem dvou jiných tříd: `ThresholdDetector` a `ContentDetector`. Tyto dvě třídy jsou knihovní třídy, jejichž některé metody a atributy bude nástroj potřebovat. v konstruktoru třídy `CutDetector` jsou inicializovány i oba rodičovské detektory. Také jsou inicializovány instanční atributy, hodnoty jsou často získávány ze souboru **values.py**. Zároveň je jako instanční atribut vytvořena nová instance třídy `PoseDetector`, která je popsána v kapitole 4.4.3. Během inicializace instance ještě nejsou volány žádné instanční metody.

Detektory knihovny PySceneDetect

Nyní bych rád popsal, jak fungují základní detektory, ze kterých dědí můj vlastní `CutDetector`. Tyto dva detektory mají mimo jiné dvě důležité metody na detekci střihů. První z nich je

`process_frame()`. Tato metoda je volána pro každý snímek videa během jeho čtení a je provedena detekce pomocí vestavěného algoritmu, který však pro naše účely není dostatečný. Druhá metoda se jmenuje `post_process()`. Tato metoda je zavolána na konci čtení videa a umožňuje dodatečné úpravy nebo přídavné detekce. v mém detektoru je využita pouze metoda `process_frame()`, avšak jediné společné je název, jinak je kompletně změněna.

Získaní hodnot ze snímku – `calculate_frame_score()`

Toto je metoda, která je z velké části převzatá z knihovny, avšak bylo potřeba provést drobné změny pro umožnění zapisování statistik. Tato funkce uloží potřebné hodnoty do instanční proměnné `frame_score`. Tato proměnná je instancí třídy `FrameScore`, která je ukázána níže.

```
class FrameScore(NamedTuple):
    content_val: float = 0.0
    delta_edges: float = 0.0
    delta_rgb: float = 0.0
```

Výpis 4.9: Třída s hodnotami potřebnými pro detekci střihu

Tyto třídní atributy mohou být povědomé, jedná se totiž o hodnoty potřebné pro rozlišení podezřelých snímků.

- **content_val** – získána pomocí váženého průměru hodnot HSV a počtu hran
- **delta_edges** – rozdíl počtu hran aktuálního a předchozího snímku
- **delta_rgb**– rozdíl průměrných hodnot RGB aktuálního a předchozího snímku

Postup k výpočtu těchto hodnot je získán z knihovny **PySceneDetect**. Přišlo mi ale důležité zmínit, jak se k těmto hodnotám došlo.

Sekundární kontrola snímku – `pose_detection_check()`

Sekundární kontrola označená na diagramu 4.1 žlutě je provedena právě touto metodou. Nejdříve pomocí pomocné metody `screenshot()` vytvoří dva snímky z videa v určité vzdálenosti na obě strany od podezřelého snímku. Z těchto dvou snímků jsou získány hodnoty klíčových bodů osob metodou `get_image_landmarks()` popsané v kapitole 4.4.3. v případě, že na jednom obrázku nebyla osoba detekována osoba, ale na druhém ano, je vytvořena nová dvojice snímků posunutá o několik snímku dále. Toto by mělo mít za efekt mírné snížení špatné detekce v momentu, kdy je osoba v poloze, na které je špatně detekovatelná postava. Vhodnější dvojice snímků je porovnána pomocí metody `get_landmark_diff()` popsané v kapitole 4.4.3. Výsledné hodnoty porovnání **diff** a **z-diff** jsou zkontrolovány následující podmínkou:

```
if (
    (diff >= 0.3 and z_diff >= 0.5)
    or (diff >= 0.5 and z_diff >= 0.3)
    or (diff >= 0.6 and z_diff >= 0.25)
    or (diff >= 0.7 and z_diff >= 0.2)
    or (diff >= 0.8 and z_diff >= 0.1)
    or diff >= 1
```

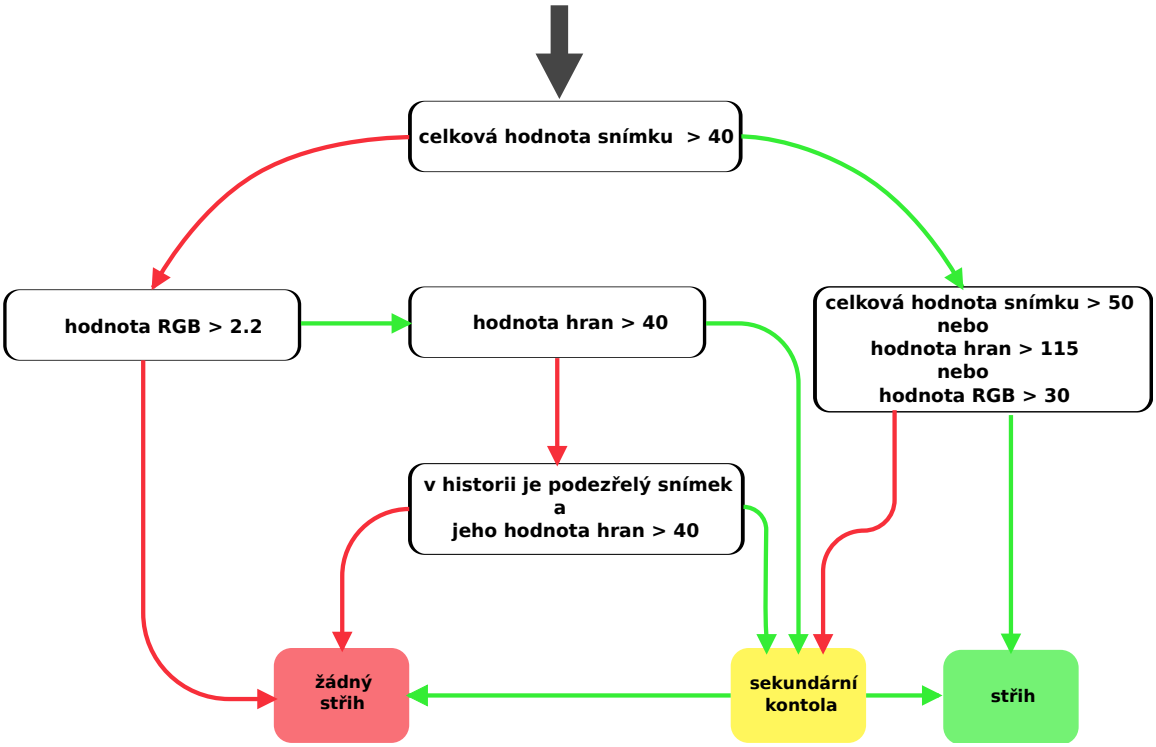
```
):
```

Výpis 4.10: Podmínky pro kontrolu bodu střihu

Tato podmínka byla vytvořena pomocí opakovaného testování algoritmu a ladění hodnot. v případě, že jsou hodnoty **diff** a **z-diff** v zadaném rozsahu, jsou jejich hodnoty a přibližná důvěra v detekci zapsány do statistik a metoda končí úspěchem. v opačném případě je navrácen neúspěch.

Vlastní metoda pro zpracování snímku – `process_frame()`

Tato metoda je nejdůležitějším prvkem celé detekce. Spojuje dohromady všechny v této kapitole popsané metody do jednoho funkčního celku. Zde je důležité si připomenout obrázek 4.1, který jsem ukazoval dříve. Je na něm popsán právě princip, pomocí kterého jsem vytvářel tuto metodu.



Obrázek 4.1: Na obrázku je zobrazen graf popisující detekci střihu pomocí metody `process_frame()`. Cílem metody bylo simulovat právě průchod tímto grafem pomocí podmínek a kontrol.

Jak jsem již zmínil, tato metoda je volána pro každý snímek ve videu. Nejprve je snímek ohodnocen pomocí metody `calculate_frame_score()` popsané v kapitole 4.4.2. Návratové hodnoty musí mít nějakou minimální hodnotu, jinak je detekce ukončena neúspěchem. Dále je již hlavní tělo této metody. Skládá se z mnoha vnořených podmínek **if**. Tyto podmínky rozvětví kód pomocí detekovaných hodnot, jak je popsáno na diagramu 4.1. Dohromady jsou zde tři možné body, do kterých se program může dostat.

První výsledek je nejjednodušší na zpracování. Pokud žádné získané hodnoty nepřekročí požadované hranice, je snímek vyhodnocen jako nezajímavý, metoda je ukončena a je navráceno prázdné pole.

Druhý výsledek je přesným opakem. v případě, že získané hodnoty překročí určité maximální hranice stanovené v souboru **values.py**, tak je snímek vyhodnocen jako jistý střih, je přidán do pole obsahující místa střihu a toto pole je předáno jako návratová hodnota.

Poslední varianta zpracovává hodnoty mezi předchozími dvěma variantami. U těchto snímků je nutné provést sekundární kontrolu. Tato kontrola je provedena metodou `pose_detection_check()` popsanou v kapitole 4.4.2. v případě, že volaná metoda navrátí úspěch, je tento snímek považován za bod střihu. Je přidán do pole, které bude později navráceno a proměnná značící poslední místo střihu je aktualizována. Před tím, než je metoda ukončena je ještě potřeba aktualizovat historii počtu hran ve snímcích. Objekt udržuje historii počtu hran posledních osmi snímků. Toto je využito v moment, kdy je vyhledáván podezřelý snímek v historii. na diagramu 4.1 se jedná o buňku „*v historii je podezřelý snímek a jeho hodnota hran je* > 40“.

4.4.3 Detektor postav – pose_detector.py

Detektor postav je opět reprezentován jako třída, tentokrát s názvem `PoseDetector`. Tato třída má dvě metody: `get_img_landmarks()` a `get_landmark_diff()`. Při inicializaci instance jsou ze souboru **values.py** načteny potřebné hodnoty pro nastavení detektoru. Tyto hodnoty jsou následující:

- **detection_confidence** – stanovuje minimální hodnotu důvěry detektoru, že je na obrázku osoba
- **landmark_vis_threshold** – spodní hranice, kdy se považuje klíčový bod jak viditelný
- **diff** – hodnota, stanovující minimální rozdíl dvou klíčových bodů tak, aby se považovaly za rozdílné

Získání klíčových bodů na obrázku – `get_img_landmarks(image)`

Tato metoda slouží k získání klíčových bodů na obrázku. Tento obrázek je upřesněn pomocí parametru `image`, který obsahuje řetězec s cestou k obrázku. Tento obrázek je přečten pomocí metody `cv.imread()` z knihovny **OpenCV**. Poté jsou inicializovány proměnné použité k detekci postavy. K tomu byla využita knihovna **MediaPipe** popsána v kapitole 2.2.4.

```
mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils
pose = mp_pose.Pose(
    static_image_mode=True,
    model_complexity=1,
    min_detection_confidence=self.detection_confidence
)
```

Výpis 4.11: Příklad inicializace proměnných potřebných pro funkci **MediaPipe** detekční knihovny

Přepínač `static_image_model` nastavuji jako pravdu, protože je prováděna detekci na obrázku, a ne na videu. Složitost modelu je nastavena na **1**, při zvýšení komplexity se zvedá i trvání detekce a zlepšení kvality detekce toto nevykompenzuje. Parametr

`min_detection_confidence` určuje, kde je hranice uznání klíčového bodu. Pokud je důvěra v detekci bodu menší, bod není zaznamenán.

Po této inicializaci je potřeba převést obrázek z barevného modelu BGR, ve kterém je obrázek po přečtení metodou `cv.imread()`, na běžný model RGB. Nyní již může být provedena samotná detekce. Výsledek detekce je uložen do proměnné **result** a detektor je ukončen. Tento výsledek je poté vrácen jako návratová hodnota.

Porovnání dvou sad klíčových bodů – `get_landmark_diff()`

Tato metoda slouží k porovnání dvou sad klíčových bodů na dvou různých obrázcích. Návratovou hodnotou je dvojice desetinných hodnot. První z nich je hodnota v rozmezí $\langle 0; 1 \rangle$, kde čím větší hodnota je, tím více rozdílné jsou klíčové body. Druhou hodnotou je hodnota ve stejném rozmezí, akorát popisuje jejich rozdílnost v odhadované hloubce klíčových bodů.

Klíčové body jsou získány metodou `get_img_landmarks()`. Pokud ani na jednom obrázku není detekovaná postava, metoda vrací dvojici $(0.0, 0.0)$. Pokud je jen jedna sada prázdná, je nastavena její hodnota 0.

Dále je již samotné jádro metody. Porovnává se zde všech 33 klíčových bodů popsaných na obrázku 2.2. Pokud je rozdíl důvěry detekce dvou stejných klíčových bodů na obou obrázcích větší než hodnota stanovená v souboru **values.py**, je inkrementováno počítadlo rozdílných bodů. To samé je provedeno pro hloubku každého klíčového bodu. na konci je vypočten poměr rozdílných klíčových bodů na každém obrázku pro normální body a body popisující hloubku. Tyto dvě hodnoty jsou vráceny a metoda je ukončena. Ukázka je na obrázku 4.2.

V tomto algoritmu pro porovnávání vidím osobně určité rezervy. Jednou z nich je přidání vah ke každému klíčovému bodu. na hlavě je totiž bodů 13, ale nezabírá tak velkou plochu a může to poměry dělat nepřesnějšími. Zároveň určitě lze více doladit nastavení detektoru, případně využít knihovnu **YOLOv8**, která by mohla být rychlejší i přesnější.

4.4.4 Soubor s hodnotami – values.py

Soubor pojmenovaný **values.py** neobsahuje žádné metody ani funkce. Do tohoto souboru jsem vložil proměnné, které byly potřeba při ladění detekčních algoritmů. Příkladem jsou hraniční hodnoty důvěry pro detekci, minimální délka videa před opětovnou detekcí střihu nebo šíře záběru snímků okolo podezřelého snímku. Pro otestovaný a podle mého nejlepší výsledek doporučuji hodnoty neměnit, avšak pokud by uživatel chtěl testovat a případně zlepšit kvalitu detekce, tak může s hodnotami manipulovat.

4.5 Classifier – klasifikace rozdělených klipů podle relevance

Posledním z hlavních skriptů je klasifikátor klipů. Jeho cílem je ohodnotit každý klip hodnotou od **0** do **1** podle toho, jak blízko ho algoritmus považuje ke skoku vysokému. Toho je dosaženo pomocí metody zvané optický tok popsané v kapitole 2.2.4. Princip je takový, že je na celý klip aplikována metoda `cv.calcOpticalFlowFarneback()` a podle výsledných hodnot je rozhodováno o klasifikaci a její důvěře.

4.5.1 Přípravné části skriptu

Nejprve je potřeba provést určité přípravy před prováděním hlavní smyčky. Po zpracování vstupních argumentů jsou inicializovány proměnné pro práci s tabulkou ve formátu **.xlsx**

Obrázek 4.2: Obrázky ukazují snímek před a snímek po podezřelém snímku. Vrchní obrázek je několik snímků před podezřelým, celkem je viditelných 13 bodů (hlava obsahuje 11, ramena každé jeden) Spodní snímek je několik snímků po podezřelém. Obsahuje celé lidské tělo, tedy 32 klíčových bodů. Poměr těchto hodnot značí, že jsou snímky ze zhruba 60 % rozdílné. Při porovnání hloubky (hodnoty **z**) společných bodů je zjištěno, že jsou rozdílné ze zhruba 50 %.

pomocí knihovny **openpyxl**. Do této tabulky jsou vloženy hlavičky sloupců, do kterých se později budou zapisovat statistická data.

Další přípravnou částí je vytvoření adresářů, do kterých se budou následně ukládat klasifikovaná videa. Tyto složky jsou vytvořeny na základě názvu složky, ze které mají být brány videa na klasifikaci nebo názvu videa předaného jako argument.

4.5.2 Hlavní tělo skriptu

Po provedení přípravných procesů již přichází na řadu samotné jádro skriptu. Toto tělo je prováděno pro každé video ve složce, nebo pro samostatné video – to záleží na zadaných argumentech příkazové řádky.

Nejprve je provedena kontrola, zda je video vůbec ve správném formátu a zda jej lze otevřít. Dále je smyčka, která prochází skrz všechny snímky ve videu jeden po druhém. Pro každý snímek je provedeno zmenšení na uživatelem požadovaný poměr. Doporučovaná a výchozí hodnota je 50 % pro video ve **Full HD** rozlišení. Nižší hodnoty nemají dostatečnou

přesnost a větší hodnoty výrazně zpomalí zpracovávání. Této změny rozlišení je dosaženo metodou `cv.resize()` knihovny **OpenCV**. Tento zmenšený snímek je dále převeden z barevného modelu **BGR**, která je výchozí pro knihovnu **OpenCV** do stupňů šedi metodou `cv.cvtColor(frame, cv.COLOR_BGR2GRAY)`. v tomto barevném modelu totiž pracuje metoda `cv.calcOpticalFlowFarneback()`, která je využita pro výpočet optického toku.

Výpočet optického toku – `cv.calcOpticalFlowFarneback()`

Metoda pochází z knihovny **OpenCV**. Jedná se o implementaci metody **Farneback dense** popsané v kapitole 2.4. Tato metoda má několik parametrů. v závorce za názvem parametru jsou napsány hodnoty, které jsem během testování vyhodnotil jako nejvhodnější pro tento program. Popisy parametrů jsem získal z dokumentace knihovny **OpenCV** [4].

- **prev** a **next** – Dva po sobě jdoucí snímky stejné velikosti, na které bude aplikována metoda **Farneback dense**
- **pyr_scale (0.5)** – Měřítko, ve kterém bude vytvořena pyramida pro obrázek
- **levels (1)** – Počet iterací (vrstev) pyramidy včetně prvotní vrstvy celého obrázku
- **winsize (10)** – Průměrná velikost okna, čím větší, tím robustnější, ale pomalejší
- **iterations (5)** – Počet iterací na každé vrstvě pyramidy
- **poly_n (5)** – Počet pixelů v okolí použitých k výpočtu
- **poly_sigma (1.2)** – Hodnota využita k vyhlazení derivátů využitých jako základy polynomiální expanze

Tato metoda vrací pole o stejné velikosti, jako je rozlišení vstupního snímku, jen má o jednu dimenzi navíc. Tomuto poli se říká tenzor, tento je tenzor třetího řádu [24]. To znamená, že má rozměry $1920 \times 1080 \times 2$ pro video v rozlišení **Full HD**. Velikosti první a druhé dimenze jsou měněny podle změny rozlišení snímku na začátku zpracovávání snímku, avšak počet dimenzí zůstává stejný. Tento tenzor obsahuje pro každý pixel hodnotu optického toku.

Zpracování dat v tenzoru

Po získání tenzoru `flow` je z tohoto tenzoru vypočítán průměr metodou `np.mean()` knihovny **NumPy**. Dále jsou pomocí příkazu $flow = flow[:, :, 1]$ vybrány z výsledků pouze horizontální hodnoty. Z těchto hodnot je také vypočítán průměr. Tyto dvě hodnoty celkového a horizontálního průměru jsou sčítány pro každý snímek. na konci metody jsou tyto sumy vyděleny počtem snímků a je zjištěn celkový průměr pohybu a průměr horizontálního pohybu v celém videu. Všechna tato data jsou zapisována do listu, který byl vytvořen v přípravné části skriptu 4.5.1. Do tohoto listu jsou data průběžně zapisována a soubor jako takový je vytvořen až po skončení zpracovávání všech videí.

Tyto dvě průměrné hodnoty jsem dlouho testoval na různých videích a ukázalo se, že opravdu nejpřesnější pro klasifikaci je obecné množství pohybu, nejen horizontální pohyb. Korelaci mezi celkovým optickým tokem, horizontálním optickým tokem a hodnotou snímku můžete vidět na tabulce 4.3.

Jelikož nebylo možné stanovit jasnou hranici, která by stanovovala, zda je jedná nebo nejedná o video skoku do výšky, tak jsem se rozhodl přiřazovat hodnotu jistoty na škále

Scene	Avg delta	Avg horizontal delta	Accepted	Confidence
103	4.24399019	2.626930007	0	0.9
017	4.11423059	3.327825491	1	0.9
079_080	3.75308817	1.534280306	1	0.85
086_087	3.60014612	2.963739773	1	0.85
020_021	3.55289763	1.434902502	1	0.85
031_032	3.52048675	1.821516483	1	0.85
060	3.43999996	1.706296251	1	0.85
084	3.4378181	3.151302535	0	0.85
044_045	3.4119249	1.372323817	1	0.85
059_1	3.3884042	2.531193697	1	0.85
054	3.18560815	3.640920937	0	0.85
010_4	3.12441053	1.650218203	1	0.85
100	3.10589576	1.897747313	1	0.85
059_2	3.02176328	2.214748972	1	0.85
029_1	2.9946635	1.38135051	1	0.5
077_4_078_1	2.92843785	1.187284788	1	0.5
053	2.85663077	1.77148997	1	0.5
023	2.84452024	1.758154131	1	0.5
042_1	2.84106228	1.912362724	1	0.5
109	2.79271138	1.162270551	0	0.5
094	2.79057288	1.26315245	1	0.5
072	2.78525645	2.120364721	0	0.5
068	2.7668632	1.323232739	1	0.5
083	2.76118936	2.619270058	0	0.5
082	2.74077407	2.41851436	0	0.5

Obrázek 4.3: Na obrázku je tabulka statistik po zpracování klipů. Řádky jsou seřazeny sestupně podle množství celkového průměrného pohybu na jednotlivých klipech. Sloupec **Accepted** značí videa, která byla předem označena jako skok do výšky. Lze si všimnout, že není nijak velká korelace mezi průměrným horizontálním pohybem a relevancí videa. Ovšem celkové průměrné množství optického toku na videu určitou spojitost s relevancí má.

$\langle 0, 1\rangle$ (technicky vzato **1** nikdy nenastane, ale v rámci principu je tato škála v pořádku). Tyto hodnoty jistoty jsem přiřazoval na základě celkového průměrného pohybu v klipu. Tyto hodnoty jsem vytvořil pomocí testování několika videí a analýzy výstupních dat v tabulce 4.3. Rozsahy hodnot jsem vložil do slovníku v jazyce **Python** a následně pomocí něj přiřazoval hodnotu jistoty pro každý klip.

```
delta_confidence_ranges = {
    (0, 1.7)    : 0,
    (1.7, 2.5)  : 0.2,
    (2.5, 3)    : 0.5,
    (3, 4)      : 0.85,
    (4, 100)    : 0.9
}
```

Výpis 4.12: Slovník s rozsahy hodnot důvěry

Tato hodnotící funkce by určitě šla také upravit pro lepší výsledek, případně zkombinovat s průměrem horizontálního pohybu. Ovšem i tak metoda dosahuje poměrně slušných výsledků klasifikace videí.

Finálním krokem klasifikace je přesun jednotlivých klipů do odpovídajících složek podle důvěry v klasifikaci. Existence těchto složek je kontrolována a pokud neexistují, tak jsou vytvořeny.

Celý proces klasifikace videa je časován a jeho trvání je vypsáno na standardní výstup. Toto jsem využil při porovnávání trvání klasifikace s různými jinými hodnotami pro zjištění ideálního poměru rychlosti a kvality. Tyto hodnoty mohou být zajímavé a důležité i pro uživatele, tak jsem je v kódu nechal.

4.6 Pomocné části nástroje

Mimo čtyři hlavní části nástroje, které tvoří tělo nástroje, jsem vytvořil ještě dva menší nástroje. Tyto nástroje by v ideálním světě nebyly potřeba a v případné další iteraci tohoto nástroje už tomu tak snad bude. Bohužel nyní má nástroj určité procento neúspěchu, a to procento je dostatečně velké na to, aby se s tím mělo nějak vypořádat. Dalo by se říct, že tyto pomocné nástroje zjednoduší manuální práci, kterou musí uživatel z důvodu nedostatků v hlavním nástroji dělat.

4.6.1 Manuální dělení videí

Prvním z pomocných programů, který jsem vytvořil, je program na manuální dělení těch videí, ve kterých detektor střihu nezvládl střih detekovat. Tento nástroj umožní uživateli pomocí grafického rozhraní manuálně vybrat snímek ve videu, ve kterém dochází ke změně scény a program poté automaticky video rozdělí na dvě části ve vybraném bodě.

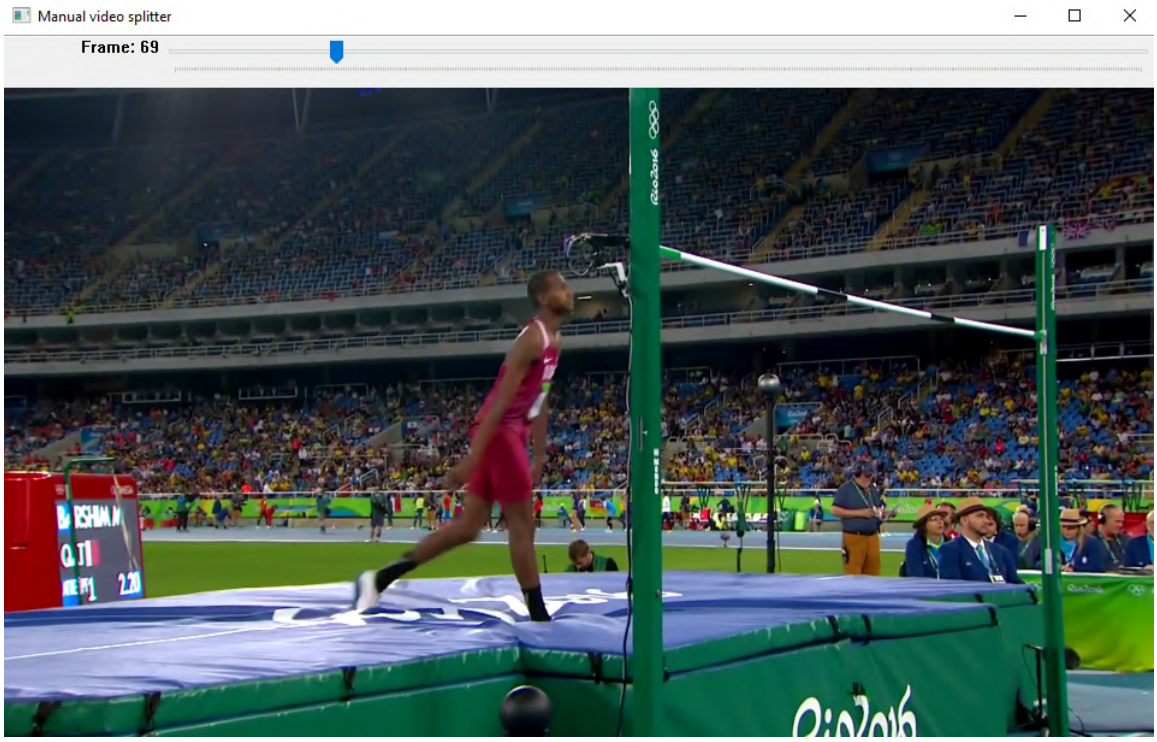
Mimo knihovny využité ve všech ostatních skriptech jsou zde navíc tři další knihovny. První z nich je opět knihovna OpenCV [10]. Tato knihovna má mnoho různých nástrojů a jedním z nich umožňuje přehrávat video v reálném čase a dále s ním manipulovat. Druhou knihovnou je **subprocess**. Tato knihovna umožňuje spouštět a řídit externí procesy pomocí Pythonu. Toho je využito při volání nástroje **ffmpeg**. Tento nástroj jsem částečně popisoval v kapitole 2.2.3. Díky tomuto nástroji lze číst upravovat, překódovat a jinak pracovat s videi. Python má vlastní knihovnu **ffmpeg-python**, ale já osobně radši pracuji s **ffmpeg** právě pomocí knihovny **subprocess**, protože jsem zvyklý používat příkazovou řádku k těmto účelům.

Jako vstupní bod programu je opět běžná *main* konstrukce, kterou jsem popisoval v úvodu do kapitoly 4. v této konstrukci jsou nejdříve zpracovány argumenty příkazové řádky a uloženy do lokální proměnné. Nutnou hodnotou předanou z příkazové řádky je cesta k videu, které chce uživatel rozdělit. Tato cesta je předána funkci `find_cut_loc()`, která má na starost smyčku umožňující vyhledání momentu střihu.

`find_cut_loc()`

V této funkci jsou nejdříve inicializovány proměnné, které budou využity během interakce s grafickým rozhraním. Těmito proměnnými je načtené video pomocí knihovny OpenCV, počet snímků ve videu a jeho snímková frekvence, které jsou získány z objektu, který vytvořilo právě OpenCV na základě cesty k videu. Také je zde inicializována booleovská proměnná, která slouží jako přepínač mezi pozastaveným a spuštěným stavem videa.

Následně je vytvořeno okno, do kterého bude zobrazeno video. Do tohoto okna jsem vložil navíc tzv. „trackbar“ – lištu, pomocí které lze vyhledávat ve videu dopředu a dozadu. Vzhled grafického rozhraní je na obrázku 4.4



Obrázek 4.4: Obrázek obsahuje snímek obrazovky, na kterém je vzhled grafického rozhraní nástroje na manuální dělení videí. na vrchní hraně videa je vidět zmíněný „trackbar“, pomocí kterého může uživatel ručně navigovat v pozastaveném videu

.

Po vytvoření proměnných pro okno a trackbar začíná hlavní smyčka. v té je čteno snímek po snímku video. v případě, že další snímek nenásleduje, smyčka je ukončena. v jakýkoliv moment promítání může uživatel zasahovat do průběhu pomocí klávesnice.

- „**p**“ pozastaví/obnoví přehrávání videa a umožní uživateli při pozastavení přesouvat pozici ve videu pomocí trackbaru.
- „**q**“ okamžitě ukončí čtení videa a zavře okno bez provádění střihu.
- „**s**“ označí aktuální snímek ve videu jako moment střihu, vypne grafické rozhraní a přesune program do sekce střihu.

Na konci funkce je ukončeno čtení videa a zavřeno okno s grafickým rozhraním a jsou vráceny hodnoty označeného snímku a frekvence snímků čteného videa. na tuto funkci navazuje funkce `cut_by_frame()`

`cut_by_frame()`

Tato funkce nejprve zkontroluje, zda uživatel během čtení videa vybral nějak snímek k provedení střihu. v případě existence označeného snímku je proveden střih pomocí nástroje **ffmpeg**. Ekvivalenty těchto příkazů pro příkazovou řádku jsou:

```
ffmepg -i video_path -ss 00:00:00 -t cut_time output_name_1
ffmepg -i video_path -ss cut_time output_name_2
```

Výpis 4.13: Příkaz FFMPEG pro rozodělení videa na dvě kratší videa

Proměnná **time** je vypočítána jako podíl vybraného snímku a snímkové frekvence videa. Výsledná videa jsou pojmenována jako video původní, jen je k nim přidán na konec index videa v páru. Původní video je zachováno beze změny pro případ lidské chyby nebo opětovného dělení.

Nedostatky

Do budoucna by určitě tento skript využil několik změn, které by práce ještě více zjednodušily, avšak jelikož má sloužit jako záplata na chyby vytvořené detektorem, tak mi přijde tento stav dostačující. Některé další změny by mohly být např.:

- Umožnění uživateli vybrat ve videu více snímků, ve kterých provést střih
- Umožnění využití trackbaru i během přehrávání videa, nejen při pozastavení
- Jednodušší přístup k instrukcím a klávesovým zkratkám

4.6.2 Manuální spojování videí

Druhý pomocný skript je značně jednodušší než ten první. Má na starost pravý opak, a to spojování videí. Spojení funguje v té nejjednodušší podobě, druhé video je připojeno na konec prvního videa. To má na starost funkce `merge_videos()`. Tato funkce z cest k videím získá jejich názvy a jejich kořenovou složku. Videa poté pomocí nástroje **ffmpeg** spojí dohromady zavoláním příkazu knihovnou **subprocess**. Ekvivalentem zavolaného příkazu je:

```
ffmpeg -i first -i second -filter_complex
       concat=n=2:v=1:a=0 -f mp4 -y output_path.mp4
```

Výpis 4.14: Příkaz pro spojení dvou videí dohromady

Proměnné **first** a **second** jsou cesty k oběma videím, `output_path.mp4` je cesta k výsledku spojení těchto videí.

Kapitola 5

Závěr

Tato práce měla dva primární cíle. Prvním cílem bylo vytvořit nástroj, pomocí kterého může uživatel poměrně jednoduše vytvořit datovou sadu sportovních videí, která může být využita k různým účelům, očekává se však využití v učení neuronové sítě. Nástroj nikdy neměl být plně automatický, avšak měl práci výrazně zrychlit a zjednodušit pro uživatele.

5.1 Nástroj pro vytvoření datové sady

První dvě části nástroje, **Scraper** a **Downloader**, jsou podle mého v dobrém stavu a zásah uživatele je pouze v zadání filtrů a klíčových slov. Samozřejmě je zde určitě mnoho možností, jak jejich chod zlepšit nebo zrychlit, ale na celkové funkcionalitě to mnoho nezmění.

Skript **Splitter** má určitě rezervy v rychlosti i přesnosti – na vině může být špatný výběr detekčního nástroje i špatná implementace. Zde se dá určitě do příští iterace nástroje značně pokročit a zlepšit přesnost detekce střihu. Zároveň využité knihovny byly stále ve fáze vývoje a jsou také stále zlepšovány, hlavně knihovna **PySceneDetect**. Velkým problémem je také, že knihovna **MediaPipe** neumožňuje přesun výpočtů na GPU. To velmi zpomaluje celý proces detekce u velkého množství videí.

Poslední skript **Classifier** funguje na jeho jednoduchost poměrně slušně. Uživatel sice musí ručně zkontrolovat klasifikovaná videa, ale pro zlepšení detekce by byla potřeba např. neuronová síť, ke které je potřeba právě datová sada ke trénování – jedná se o začarovaný kruh, ve kterém se bohužel manuální práci a chybovosti nedá dobře vyhnout.

5.2 Datová sada

Druhým cílem této práce bylo vytvořit datovou sadu videí na konkrétní sportovní tématiku. Tato datová sada je uložena na serveru `sophie1.fit.vutbr.cz` po domluvě s vedoucím práce. V kapitole 1 jsem popsal jaké vlastnosti by měla datová sada splňovat. Nyní vysvětlím, zda je vytvořená sada splňuje či nikoliv.

- **Reprezentativita problému** – Vytvořená datová sada obsahuje po ruční kontrole pouze videa skoku do výšky, tudíž bych jí označil jako reprezentativní problému
- **Velikost a rozmanitost** – Jelikož nástroj není optimalizován a jeho rychlost nebyla úplně cílem, výsledná datová sada nemá největší objem. Druhým bodem je nečekaná změna v **YouTube API**, která neumožnila stažení dostatečného množství videí. Zde

je ovšem možnost pomocí určitých metod v kapitole 12.2.1.2 v knize „Deep learning“ [17] datovou sadu uměle zvětšit.

- **Standardizace pixelů** – Pixely na videích sice nejsou v rozsahu [0,1], nebo [-1,1], jak je doporučováno v této knize [17], avšak jsou uniformě v rozsahu [0,255], což by mělo být dostačující.
- **Anotovanost** – Tento krok je velmi časově náročný a cílem této práce nebylo se jím zabývat, tudíž tento bod sada nesplňuje.

5.3 Navazující práce

Z mého pohledu se dá pokračovat více směry. První možnost je pokusit se dále pracovat na tomto nástroji a optimalizovat jeho fungování. Toho bych dosáhl využitím jiné knihovny na detekci postav, nejlépe takové, která umožňuje výpočty na GPU. Také bych pravděpodobně zkusil ještě více nápadů na klasifikaci videí. Druhým směrem, kam by se mohlo dále pokračovat je vytvořit nástroj v jazyce, který již sám o sobě umožňuje rychlejší práci s větším množstvím dat, jako například jazyk **C++**. Poslední možností je začít pracovat s vytvořenou datovou sadou v případě, že je množství dostačující pro účely uživatele.

Literatura

[1] *Argparse – parser for command-line options, arguments and sub-commands.* Python Software Foundation. Dostupné z: https://docs.python.org/3/library/argparse.html.

[2] *Google/mediapipe: Cross-platform, customizable ML solutions for live and streaming media.* Google. Dostupné z: https://github.com/google/mediapipe.

[3] *Python Package Index – PyPI.* Python Software Foundation. Dostupné z: https://pypi.org/.

[4] *Optical flow.* Dec 2022. Dostupné z: https://docs.opencv.org/3.4/d4/dee/tutorial%5Foptical%5Fflow.html.

[5] Bitmovin, 2023. Dostupné z: https://bitmovin.com/wp-content/uploads/2022/12/bitmovin-6th-video-developer-report-2022-2023.pdf.

[6] Alchin, M., Goodger, D. a Rossum, G. van. Docstring Conventions. *Pro Python.* Springer. 2010, s. 303–307.

[7] Anand, C. Comparison of stock price prediction models using pre-trained neural networks. *March 2021.* 2021, sv. 3, č. 2, s. 122–134.

[8] Beaver, F. E. a Beaver, F. *Dictionary of film terms: the aesthetic companion to film art.* Peter Lang, 2006.

[9] Bradski, G. a Kaehler, A. *Learning OpenCV: Computer vision with the OpenCV library.* "O'Reilly Media, Inc.", 2008.

[10] Bradski, G., Kaehler, A. et al. OpenCV. *Dr. Dobb's journal of software tools.* 2000, sv. 3, č. 2.

[11] Castellano, B. Pyscenedetect. *Last accessed.* 2020.

[12] Conway, A. *What is AV1 and why does it matter for the Google Chromecast HD?* XDA Developers, Mar 2023. Dostupné z: https://www.xda-developers.com/av1/.

[13] Cox, B. J. *Object oriented programming: an evolutionary approach.* Addison-Wesley Longman Publishing Co., Inc., 1986.

[14] Farnebäck, G. Two-frame motion estimation based on polynomial expansion. *Image Analysis.* 2003, s. 363–370. DOI: 10.1007/3-540-45103-x_50.

[15] Francois, C. *Deep learning with python.* Manning Publications Co., 2018.

[16] Jocher, G., Chaurasia, A. a Qiu, J. *YOLO by Ultralytics*. Leden 2023. Dostupné z: https://github.com/ultralytics/ultralytics.

[17] LeCun, Y., Bengio, Y. a Hinton, G. Deep learning. *Nature*. Nature Publishing Group UK London. 2015, sv. 521, č. 7553, s. 436–444.

[18] Lucas, B. D. a Kanade, T. An Iterative Image Registration Technique with an Application to Stereo Vision. *Proceedings of Imaging Understanding Workshop, pp 121-130*. 1981.

[19] Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E. et al. Mediapipe: A framework for building perception pipelines. *ArXiv preprint arXiv:1906.08172*. 2019.

[20] Oleniuk, L. *Canada's Derek Drouin completes his fourth jump on his route to a...* 2016. Dostupné z: https://www.gettyimages.ae/detail/news-photo/canadas-derek-drouin-completes-his-fourth-jump-on-his-route-news-photo/590504314.

[21] Olympics. Olympics, Aug 2016. Dostupné z: https://youtu.be/zW87tVnDKIU.

[22] OpenCV. *Optical Flow: Farneback dense*. OpenCV, Apr 2023. Dostupné z: https://docs.opencv.org/3.4/opticalfb.jpg.

[23] OpenCV. *Optical Flow: Lucas-Kanade*. OpenCV, Apr 2023. Dostupné z: https://docs.opencv.org/3.4/opticalflow%5Flk.jpg.

[24] Ramsundar, B. a Zadeh, R. B. *TensorFlow for deep learning: from linear regression to reinforcement learning*. "O'Reilly Media, Inc.", 2018.

[25] Richardson, I. E. *Video codec design: developing image and video compression systems*. John Wiley & Sons, 2002.

[26] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks*. 2015, sv. 61, s. 85–117.

[27] Sharma, A. *Mean average precision (MAP) using the Coco Evaluator*. Jul 2022. Dostupné z: https://pyimagesearch.com/2022/05/02/mean-average-precision-map-using-the-coco-evaluator/.

[28] Solawetz, J. *What is Yolov8? the ultimate guide*. Roboflow Blog, Jan 2023. Dostupné z: https://blog.roboflow.com/whats-new-in-yolov8/.

[29] Szeliski, R. *Computer Vision Algorithms and Applications*. Springer, 2011.

[30] Wang, C.-Y., Bochkovskiy, A. a Liao, H.-Y. M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *ArXiv preprint arXiv:2207.02696*. 2022.