TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Systém pro poloautomatické zpracování videonahrávek pořadů

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Martin Snětivý**
Vedoucí práce: doc. Ing. Josef Chaloupka, Ph.D.

Liberec 2019

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

System for semi-automatic processing of video recordings

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information technology

Author: **Martin Snětivý**
Supervisor: doc. Ing. Josef Chaloupka, Ph.D.

Liberec 2019

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Systém pro poloautomatické zpracování videonahrávek pořadů

Jméno a příjmení: **Martin Snětivý**
Osobní číslo: M16000053
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav informačních technologií a elektroniky
Akademický rok: **2018/2019**

Zásady pro vypracování:

1. Seznamte se s problematikou zpracování a rozpoznávání obrazu a problematikou vytváření desktop aplikací v C# WPF.
2. Navrhněte a realizujte systém pro (polo)automatické zpracování video nahrávek televizních a jiných pořadů.
3. Systém by měl obsahovat algoritmy pro automatickou segmentaci video signálu a sadu algoritmů pro rozpoznávání zajímavých objektů v jednotlivých video snímcích.
4. Výsledný systém by měl být dostatečně uživatelsky příjemný pro poloautomatické zpracování a expertizu video obsahu v rozsáhlých video archívech.

Rozsah grafických prací:	Dle potřeby dokumentace
Rozsah pracovní zprávy:	cca 30-40 stran
Forma zpracování práce:	tištěná/elektronická

Seznam odborné literatury:

[1] Davies, E., R.: Machine Vision – Theory, Algorithms, Practicalities. Morgan Kaufmann Press. UK, ISBN 0-12-206093-8, 2005

[2] Šonka, M., Hlaváč V., Boyle. R.: Image processing, analysis, and machine vision. 3rd ed. Toronto: Thomson, 829 s. ISBN 978-0-495-08252-1, 2008

[3] Hlaváč, V., Sedláček, M.: Zpracování signálů a obrazů. 2. přeprac. vyd. Praha: ČVUT, 255 s. ISBN 978-80-01-03110-0, 2007

[4] Weil, A.: Learn WPF MVVM – XAML, C# and the MVVM pattern, In Lulu Press, USA, ISBN 978-1326847999, 2017

Vedoucí práce:	doc. Ing. Josef Chaloupka, Ph.D. Ústav informačních technologií a elektroniky
Konzultant práce:	Ing. Karel Paleček, Ph.D. Ústav informačních technologií a elektroniky
Datum zadání práce:	18. října 2018
Předpokládaný termín odevzdání:	30. dubna 2019

L. S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci 18. října 2018

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že texty tištěné verze práce a elektronické verze práce vložené do IS STAG se shodují.

17. 4. 2019 Martin Snětivý

Abstrakt

Tato bakalářská práce se zabývá vytvořením uživatelský přívětivého softwaru pro analýzu a zpracování videonahrávek pořadů. Dále obsahuje algoritmy pro detekci tváří, textů a identifikaci osob. Systém je psán v jazyce C# a Python. Jedná se o WPF aplikaci. Jazyk C# zde převážně funguje jako prostředník mezi vzhledem a zprostředkovává komunikaci s Pythonem, ve kterém jsou za pomocí knihoven DLIB a OpenCV implementovány algoritmy pro detekci a identifikaci tváře. Optické rozeznávání znaků je implementováno pomocí knihovny Tesseract OCR a jedná se o Wrapper pro C#. Vytvořený systém pak tedy umožňuje rozsáhlou video expertizu.

Klíčová slova: video abstrakt, detekce tváří, identifikace osob, optické rozpoznávání znaků, C#, Python, Dlib, OpenCV, Tesseract OCR

Abstract

This thesis is about creation of user-friendly software for analysis and processing of video recordings. It includes face, text and person identification algorithms. The system is written in C# WPF and Python. C# language mainly acts as a middleman between user interface and mediates communication with Python, using algorithms for detection and identification of faces using DLIB and OpenCV libraries. Optical Character Recognition is implemented using the Wrapper Tesseract OCR library for C#. The created system then allows extensive video analysis.

Keywords: video abstract, face detection, face recognition, optical character recognition, C#, Python, Dlib, OpenCV, Tesseract OCR

Obsah

Seznam zkratek

MVVM	Model-View-ViewModel
API	Application Programming Interface
XML	Extensible Markup Language
CSV	Comma-separated values
OCR	Optical character recognition
WPF	Windows Presentation Foundation
MSDN	Microsoft Developer Network
GC	Garbage collector
UI	User interface
HP	Hewlett-Packard
GDI	Graphics Device Interface
HoG	Histogram of oriented gradients
PCA	Principal component analysis
CPU	Central processing unit
SVM	Support Vector Machine
EXE	executable
CNN	Convolutional neural network
SVD	Singular-value decomposition

Seznam obrázků

1 Úvod

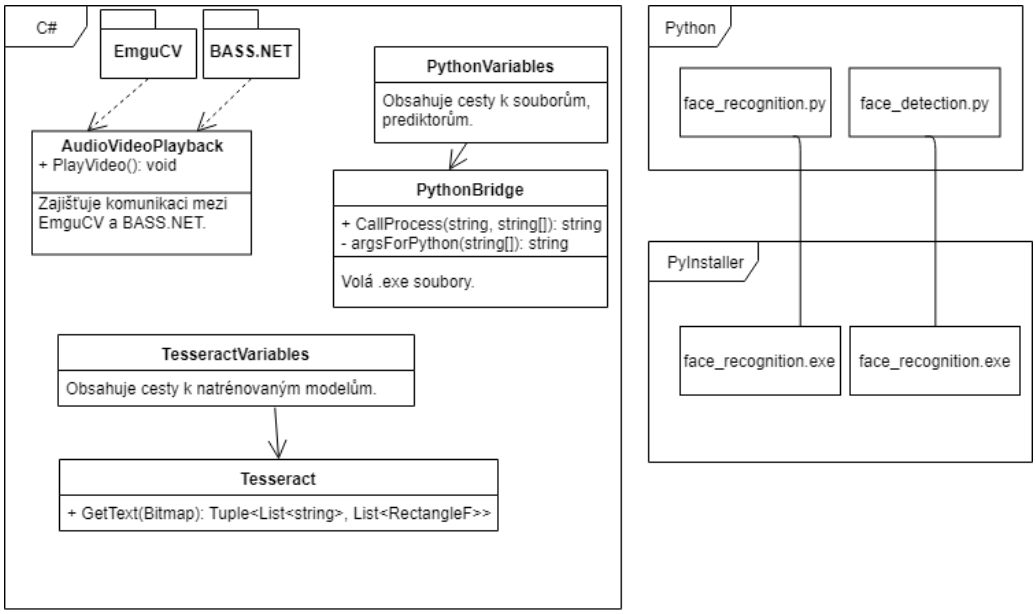
Segmentace a indexace videonahrávky je nezbytně nutná pro efektivní výběr relevantních dat, která jsou uložená ve velkých multimediálních databázích. K vytvoření účinné indexace je třeba z videonahrávky vybrat množinu hlavních snímků, jež reprezentují celý obsah souboru. Nejdříve se musí videonahrávka rozdělit na jednotlivé snímky a poté vybrat optimální počet klíčových snímků mezi přechody tak, aby byla zachována informace o nahrávce. Vytvořený video abstrakt není poté nic jiného, než množina jdoucích snímků po sobě. Tato sekvence je kratší než původní nahrávka, avšak zachovává původní myšlenku. Dalším plusem video abstraktu je to, že je snazší na procházení než kompletní nahrávka.

Mnohem rozsáhlejší analýzu videonahrávek používají giganti jako Amazon a jeho Amazon Rekognition nebo Google a Video AI případně Facebook. Tato analýza může sloužit například pro okamžitou reakci na veřejnou bezpečnost a ochranu, nebo k najití pohřešované osoby v obsahu sociálních médií. Dále k vytvoření video knihovny s metadaty z nahraných videí, takže lze vytvořit vyhledávací rejstřík jmen osob a času jejich zobrazení. V neposlední řadě k okamžité filtraci explicitního nebo sugestivního obsahu ve videích a vytváření vlastních pravidel, jež jsou vhodná pro kulturu a demografii uživatelů. Poté k zjednodušení doporučení obsahu pro jednotlivé uživatele na základě jejich preferencí a historie procházení. S tím souvisí i reklamy a jejich umístění, aby byly pro obsah videa kontextově relevantní.

Celý systém je postaven od základů s myšlenkou modularity, a proto by například nebyl problém implementovat další jazykovou kulturu případně nový typ souboru pro uložení abstraktu. Systém obsahuje sadu algoritmů pro rozpoznávání zajímavých objektů v jednotlivých video snímcích. Výsledný systém by měl být dostatečně uživatelsky příjemný pro poloautomatické zpracování a expertizu video obsahu v rozsáhlých video archívech.

2 Představení systému a jeho modulů

Klíčové moduly a jejich vzájemná komunikace bez jednotlivých částí MVVM (Model–view–viewmodel) je zobrazena na zjednodušeném UML diagramu 2.1. Z diagramu je zřejmé, že zpracování videonahrávky je zprostředkováno v C# pomocí knihoven EmguCV a BASS.NET v rámci třídy AudioVideoPlayback. Detekce textu je prováděna třídou Tesseract, která používá pomocnou statickou datovou třídu TesseractVariables, jež obsahuje proměnné k nacvičeným modelům pro OCR. Detekce tváří a jejich identifikace je poté samostatná kapitola z důvodu problémů popsaných v kapitole 7.8. V Pythonu je napsán funkční parametrický skript, jež vypisuje výsledky do konzole. Tento skript je poté transformován knihovnou PyInstaller na EXE (executable) spustitelný soubor. Spuštění takového souboru zajišťuje třída PythonBridge a její pomocná třída PythonVariables. PythonBridge má připravenou veřejnou metodu CallProcess, jež spustí nový proces na základě cesty k takovému souboru a jeho parametrům. Dále musí tato metoda obsahovat přemostění z konzolového výstupu procesu zpět do C#. Tvorbu parametrů k procesu zajišťuje doprovodná privátní metoda argsForPython.



Obrázek 2.1: Krátké představení jednotlivých modulů projektu pomocí UML

3 Segmentace videonahrávky

3.1 Video indexace

Automatizované procházení videonahrávek je v dnešní době stále problém a nejspolehlivějším způsobem zůstává manuální procházení snímků jeden po druhém a jeho následná analýza. Nevýhoda tohoto přístupu je zřejmá. Je časově náročná, pracná a navíc je mnohdy vedená pouze textově, a to vede k eventuální ztrátě dat při procházení a indexování velké množiny dat. Dotyčný musí neustále procházet jednotlivé snímky, hledat jejich přechody a následně je označovat. Navíc je problematické určit, jak detailní má být úroveň popisu. Částečným řešením je vytvoření uceleného systému k ulehčení procházení videonahrávky. Nicméně kvůli složitosti to znamená, že valná většina archivovaných dat stále zůstává nezpracována.

Pro odborníky, zabývající se touto problematikou, představovali online katalogy velký posun vpřed. Dříve byli nuceni fyzicky navštěvovat dostupné zdroje k provedení analýzy, případně poslat jiného člena jako zástup. S online katalogem se tento přístup razantně změnil a tato činnost se dala provádět vzdáleně. Tím se ušetřil čas a peníze. S přehledem lze tedy říci, že výhody online katalogu platily několikanásobně jednalo-li se o zahraniční katalogy. Online katalogy se ovšem vyplatí pouze tehdy, pokud jsou alespoň stejně sofistikované jako staré systémy.

Aby byl systém vhodným zdrojem dat, měl by:

- obsahovat detaily celé databáze, neboť specificky zaměřené databáze nemusí být vhodné pro všestranné použití
- nabízet efektivní nástroje pro indexaci a dotazování. Starší fyzické databáze s oporou schopného personálu, který zná svoji kolekci občas může poskytnout rychlejší přístup a další reference, jež mohou snadno překonat většinu vyhledávačů

Příkladem takového online katalogu může být třeba footage.net[1]. Poskytuje přístup ke stovkám hlavním *stock footage*[2] zdrojům po celém světě bez nutnosti je fyzicky navštívit. Obsahuje prostředky pro prohledávání nespočetného množství snímků v kombinovaných databázích na základě porovnávání textového vstupu od uživatele vůči popisku záběru.

Stock footage je kolekce filmových snímků, které byly natočeny v minulosti a jsou znovupoužívány v reklamách, seriálech apod. Například pokud by potřebovala cestovní společnost propagační materiály ze zahraničí se záběry na pláž, tak namísto najímání lidí, půjčování vybavení, cestování do destinace a následného natočení záběru a jeho zpracování je dost velká šance, že tento záběr již existuje ve *stock footage*, který se dá použít místo natáčení celého nového filmu. Jediným problémem se znovupoužitím vhodného existujícího materiálu je složitost jeho hledání v tak rozsáhlých kolekcích.

Současný stav dosavadních online katalogů je hledáním podle klíčových slov na základě popisku u snímku. Například hledáním slova "noční pláž"na footage.net vrací 26 759 výsledků v 26 databázích. Samozřejmě se jedná o poměrně abstraktní dotaz a pakliže by bylo potřeba dále specifikovat určitý snímek, lze použít více klíčových slov. Použitím delšího klíčového řetězce označení můžeme dostat přesnější výsledky a zároveň přivést i nežádoucí záběry. V tuto chvíli by musel odborník projít všechny záběry a případně zaplatit ty, o které má zájem. Stále se mnohdy jedná o levnější cestu než natočení nového filmu. Pakliže by se odborník nacházel přímo ve fyzické knihovně, mohl by vidět individuální záběry, ale mnoho knihoven si za tuto službu nechává platit. Nehledě na to, že se jedná o časově náročný proces, v kterém se lokalizuje každý jeden snímek. Z těchto důvodů je použití online katalogů nebo knihoven znemožněno neschopností efektivně procházet videozáznamy.

První cestou, ke zautomatizování indexace záznamu, je vytvoření jeho abstraktu. Video abstrakt je definován jako posloupnost obrázků vytažených z videa. Tato sekvence je kratší, než původní snímek, avšak zachovává kontext celé nahrávky. Také je časově méně zatěžující na vytvoření než textová anotace. Dalším plusem je vizuální shrnutí, které je pro lidstvo bohatší než textové shrnutí. Video abstrakt se poté dá použít k indexaci nahrávky, která ještě nebyla zařazena do katalogu, neboť procházet kratší sekvenci snímků je rychlejší než sledovat celý film. Lze tuto techniku kombinovat s textovou anotací k vylepšení hledání. V příkladu s noční pláží by tedy video abstrakt vrátil vizuální informaci obsahu o každém jeho snímku, což by usnadnilo výběr vhodných záběrů.

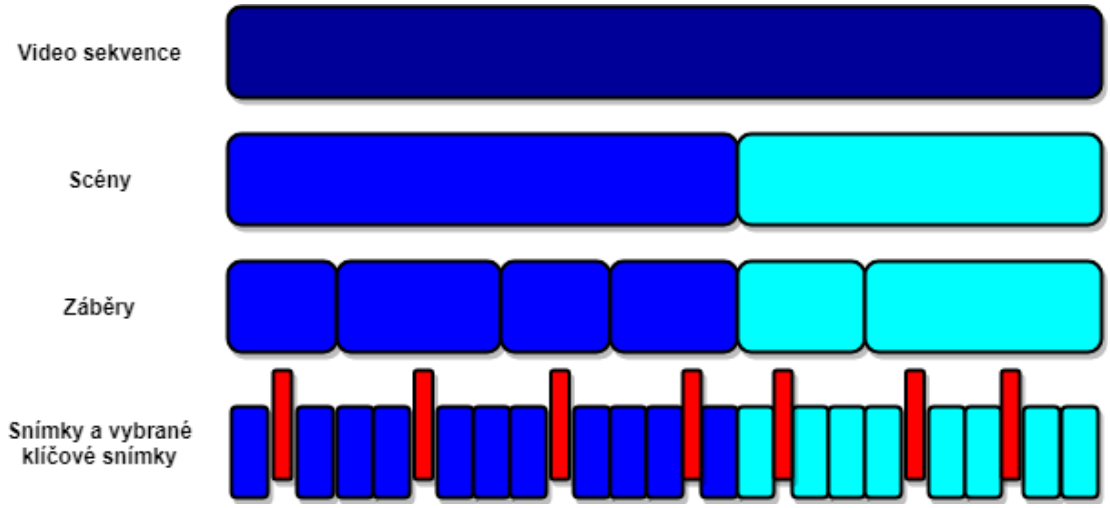
Komplexnost vytvořením abstraktu spočívá ve vybrání správných snímků, které reprezentují nahrávku. Jaké snímky vybrat, záleží také na aplikaci a kde se budou používat. Abstrakt zaměřený na ukázky by měl být krátký a měl by upoutávat pozorovatele bez přílišného odhalování obsahu. Naopak abstrakt pro dokumentární katalog by se měl snažit o reprezentování celého obsahu.

Efektivní video abstrakt pro katalog se pozná tak, že každý jeho klíčový snímek reprezentuje určitý segment videa, kde se neodehrává žádná změna, nebo nevýznamná změna ve scéně. Z toho vyplývá, že bude zachován obsah sekvence při

[1] footage.net

[2] stock footage - jedná se o snímky, které jsou znovupoužitelné, jež byly dříve natočeny.

odstranění nadbytečných snímků. Ve video sekvenci je zachována hierarchická struktura, jak je zobrazeno na obr. 3.1, která může být využita k extrakci takového klíčového snímku. Na nejnižší úrovni se jedná o řadu snímků. Na další úrovni jsou snímky spojeny dohromady a tvoří záběry definované jako sekvence snímků, jež byly pořízeny nepřetržitě z jedné kamery. Záběry jež jsou spojeny prostředím nebo událostí jsou seskupeny dohromady a tvoří scénu. Tyto scény poté skládají dohromady video sekvenci. Jakmile je video sekvence rozdělena do logických záběrů, dá se udělat jednotlivá charakteristika komponent pro indexaci a anotaci. Tudíž dočasná segmentace video sekvence je typicky prvním krokem k automatické anotaci nahrávek.



Obrázek 3.1: Hierarchická struktura uvnitř sekvence

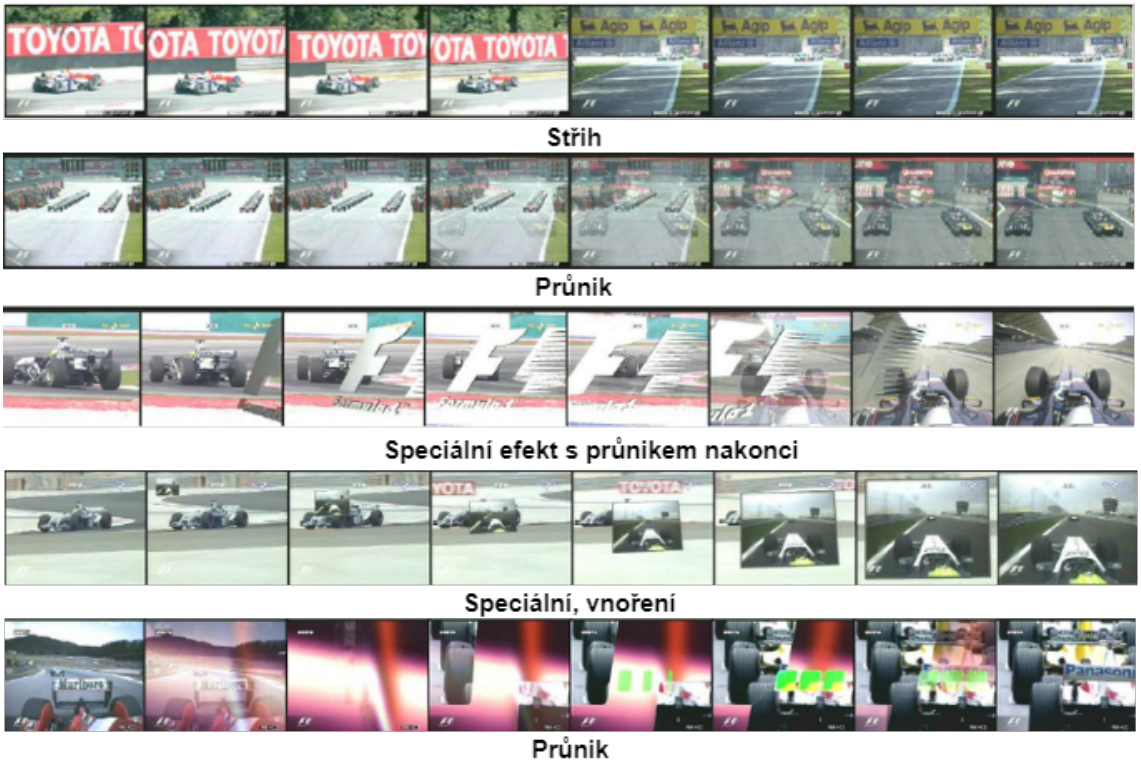
3.2 Snímkové přechody

Jedná se o to, jak spojit dva nezávislé snímky dohromady. Lze tedy říci, že se jedná o jakýsi časový úsek, jež vede diváka od jednoho záběru k druhému. Přechody se dělají v postprodukci. Propojení mezi záběry je stejně důležité jako záběry samy o sobě se všemi přechody reagujícími na změnu v čase nebo prostoru. Existují dva typy přechodů, které mohou nastat mezi záběry. Nespojité přechody označované jako střih nebo postupné a spojité přechody jako rozsvětlení/ztmavení, průnik, clona nebo změna scény pohybem. Podrobněji mohou být tyto transformace definovány jako:

- střih: instantní přechod z jednoho snímku na další;
- rozsvětlení: snímek se postupně objeví z konstantního obrazu;
- ztmavení: snímek postupně mizí do konstantního obrazu;

- průnik: momentální snímek postupně mizí, zatímco další se postupně objevuje;
- clona: další snímek je odhalen pohybující se hranicí ve formě čáry nebo vzoru;
- změna scény pohybem: následující snímek posune předešlý snímek do směru;
- speciální: kombinace předešlých, případně nedefinovaných;

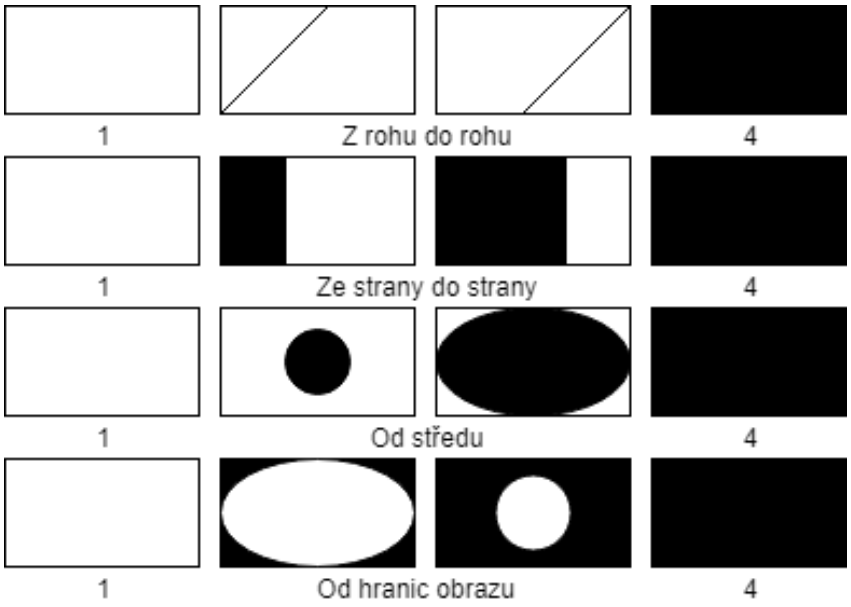
Existují další stovky různých změn scény pohybem a clon, a právě tyto jsou nazývány speciálními přechody. Příklady takových přechodů jsou znázorněny v obrázku 3.2. Detekcí všech zařaditelných přechodů rozdělí video sekvenci na její jednotlivé záběry, kde každý reprezentuje jiný čas nebo prostor. Po této operaci je nahrávka připravena pro následnou vyšší úroveň zpracování k její charakterizaci.



Obrázek 3.2: Přechody mezi snímkami [2]

Většina lidí viděla nespočet hodin televizních nebo filmových pořadů a sdílí svůj implicitní názor na filmy, obzvláště jde-li o přechody. Například průnik z jedné scény na další obvykle znamená relativně krátký časový úsek. Producenti tuto znalost využívají, aby divák pochopil nahrávku. Porušením této skryté domněnky může vést u diváka k frustraci. Střih je nejlehčím a nejčastějším způsobem, jak se dostat do dalšího záběru. Existuje i jemný střih, kdy se mezi dvěma obrazy vyskytuje kontinuita. Ta se například vyskytuje u rozhovorů, kdy se obraz pohybuje od reportéra ku dotazované osobě. Průnik zase ovlivňuje vnímání času v obraze a rytmu událostí. Naznačuje tématickou vazbu mezi dvěma záběry. Průnik může být zase použit ke

zkrácení dlouhých akcí, jako jsou lety letadlem a přeskočit z odletové destinace do té cílové. Změna scény pohybem naznačuje začátek nebo konec scény, epizody. Tento přechod často naznačuje výraznější změnu v místě nebo čase než průnik.



Obrázek 3.3: Ilustrace různých průnikových přechodů

Díky této skryté filmové gramotnosti jsou nejpoužívanějšími přechody nalezené ve video sekvencích střihy, rozsvětlení/ztmavení a průniky.

3.3 Detekce střihu

V případě střihu je jeden obraz ihned vyměněn dalším. Proto cílem jakékoliv metody na detekci střihu je vybrání určitých vlastností, jež souvisí s vizuálním obsahem videa jako:

- všechny snímky ve stejném záběru vykazující podobné vlastnosti
- snímky, které patří k různým záběrům, by měly vykazovat odlišné vlastnosti.

Většina existujících metod používá rozdílovou metriku mezi snímky. Pár snímků, který má rozdíl větší než předdefinovaný práh je považován, že obsahuje střih.

3.3.1 Porovnání na úrovni obrazových bodů

Jednoznačně nejlehčí cestou jak vyčíslit rozdíl mezi dvěma snímky je porovnáním jejich odpovídajících hodnot intenzit. Pokud je absolutní změna v intenzitách obrazových bodů větší než práh T_{cut} předpokládá se, že se mezi dvěma snímky vyskytuje střih. Mějme dva snímky f_{n-1} a f_n pak se dá definovat vzorec:

$$\frac{\sum_p |f_{n-1}(p) - f_n(p)|}{w \cdot h} \begin{cases} > T_{cut} & střih \\ \leq T_{cut} & neobsahuje \end{cases} \tag{3.1}$$

kde $f_n(p)$ je intenzita obrazového bodu p v f_n. Potenciálním problémem tohoto přístupu je citlivost na pohyb kamery a objektu. Použitím absolutního rozdílu nelze rozlišit, zda se jedná o velkou změnu v malé oblasti nebo o menší změnu ve velké oblasti. Tím pádem může přítomnost velkých pohybů v obraze vést ke špatné detekci střihu.

Tento postup se dá vylepšit tím, že budeme pozorovat změnu obrazových bodů v procentech, které se výrazně změnily mezi dvěma snímky. Tento přístup je znám také jako párové srovnání obrazových bodů. Obrazový bod je považován za změněný, pokud je jeho rozdíl vyšší než daná prahová hodnota. Přítomnost střihu je poté vyhodnocena na základě druhého prahu, jež sleduje procento změněných obrazových bodů. Ačkoliv se jedná o zlepšení, je tento přístup stále citlivý na pohyb kamery a objektu. Například fotoaparát může způsobit vadu, že většina obrazových bodů se bude jevit jako změněná. Aby se dále omezil vliv pohybu je doporučeno použít vyhlazující filtr na každý obrázek před porovnáním.

3.3.2 Porovnání na globální úrovni

Ve snaze dále překonat problém pohybu kamery a objektu vzniklo toto porovnání. Namísto porovnávání individuálních obrazových bodů byly navrhnuty alternativní přístupy, jež porovnávají globální vlastnosti každého snímku. Měření průměrné intenzity bere průměrnou hodnotu každého barevného kanálu v momentálním obraze a porovnává ji s hodnotami získanými v předešlých a následujících snímcích. Ačkoliv je tato metoda méně citlivá vůči pohybu než srovnání na úrovni obrazových bodů tak může nastat, že dva záběry s rozdílnou distribucí barev můžou mít podobnou průměrnou intenzitu, která povede k chybné detekci.

Rozdílným přístupem je porovnání globálních histogramů. Tato metoda je založena na předpokladu, že dva snímky se stálým pozadím a jejich objekty budou prokazovat malé změny v jejich příslušných histogramech. Tento přístup by měl být méně citlivý vůči pohybu než porovnávání na úrovni obrazových bodů, protože ignoruje změny v prostorovém rozložení uvnitř obrazu. Nicméně zde také spočívá jeho slabost. Mohou existovat dva sousedící záběry se stejnými histogramy ale naprosto odlišnými obsahy, což má za následek neshodu podobnou tomu, kterou způsobuje kamera a pohyb objektu. To znamená, že je problematické detekovat všechny pozitivně detekovatelné střihy, aniž by došlo k zahrnutí negativní detekce. Histogram však přináší rozumný kompromis mezi přesností a složitostí na výpočet a jedná se tedy dodnes o nejpoužívanější metodu.

Nagasaka a Tanaka navrhli porovnání černobílých histogramů mezi dvěma snímky [8]. Histogram $H_n(k)$ je získán sečtením počtu obrazových bodů ve snímku f_n, kde k představuje černobílý kanál. Rozdíl mezi dvěma histogramy je poté určen:

$$DH_n = \sum_{k=1}^{K} |H_{n-1}(k) - H_n(k)| \tag{3.2}$$

Kde K je počet černobílých kanálů. Pokud je DH_n větší než daný práh, jedná se o střih. V experimentech Nagasaka a Tanaka bylo používáno 64 rozdílných kvantizačních úrovní. Nicméně uvedli, že metrika není dostatečně robustní v přítomnosti momentálního šumu, jako například při pohybu velkých objektů [8]. Byl vytvořen robustnější algoritmus pro porovnávání dvou barevných histogramů. Rozdíl byl stále počítán vzorcem 3.2 ale s $H_n(k)$ získaným sečtením počtů obrazových bodů s barevným kódem k. Autoři navrhli použít 6 bitový barevný kód získaný odebráním dvou nejvýznamnějších bitů každé komponenty RGB, což má za následek 64 barevných kódů. Aby byl rozdíl mezi dvěma snímky obsahující střih co největší, navrhli také použití chí-kvadrát testu, který může být použit k změření rozdílu [9].

Existuje mnoho kombinací, jak vypočítat rozdíl histogramů mezi dvěma snímky pro účely detekce střihu. Histogramy mohou být získány v jiných barevných prostorech jako RGB, HSV, YIQ, Munsell, jež byly v minulosti použity k získání obrazové databáze na základě barvy. Níže je zobrazena jedna z dalších metrik vycházejících z 3.2 a χ^2 testu:

$$INT_n = \frac{\sum_{k=1}^{K} min(H_{n-1}(k), H_n(k))}{w \cdot h} \tag{3.3}$$

kde $w \cdot h$ je počet obrazových bodů v každém snímku [10]. Rozdíl mezi dvěma snímky je tedy potom definován následovně:

$$INTD_n = 1 - INT_n \tag{3.4}$$

Ukázalo se však, že jednoduchá konverze mezi RGB nebo YUV barevným prostorem s každou barevnou složkou kvantizovanou na 2^b různých hodnot, kde b je obvykle nastaveno na 2 nebo 3 je jednoduchá ale účinná metoda na detekci ostrých střihů [8]. Ve skutečnosti se ukázalo, že vylaďování barevných prostorů a metrik rozdílů přináší malé zlepšení.

3.3.3 Bloková porovnání

Slabinou srovnávání na globální úrovni je, že mohou vynechat změny v prostorovém rozložení mezi dvěma různými záběry. A zase porovnávání na úrovni obrazových bodů postrádá odolnost proti pohybu kamery a objektů. Jako kompromis mezi těmito dvěma metodami navrhl Zhang a ostatní porovnávání odpovídajících regionů (bloků) ve dvou po sobě následujících snímcích [11]. Bloky byly porovnány na základě statistických charakteristik druhého řádu[3] jejich hodnot intenzity pomocí poměru pravděpodobností. Střih byl pak detekován, pokud počet bloků s pravděpodobností byl větší než T_{diff} a překračoval práh T_{cut}. Počet bloků požadovaných na indikaci výrazné změny a střihu ve snímku samozřejmě závisí na tom, jak byl snímek rozdělen.

Nagasaka a Tanaka navrhli rozdělit každý obraz do 4×4 regionů a porovnávat histogramy těchto regionů [8]. Také navrhli, že v případě šumu, jako je blesk kamery a pohyb obvykle ovlivňuje méně než polovinu snímku. Na základě této myšlenky

[3]Statistická charakteristika druhého řádu - druhé nejmenší číslo v řadě

byly bloky seřazeny a prvních 8 bloků s největším rozdílem bylo vyřazeno. Průměr zbývajících bloků byl určen k detekci střihu. Ueda a kol. doporučili použít 48 bloků a určovat rozdíl mezi dvěma obrazy na základě celkového počtu bloků, jež měly rozdíl v histogramech větší než daný práh T_{cut} [12]. Tato metoda se osvědčila jako citlivější na detekci střihů než předešlé způsoby. Ačkoliv odstraněním osmi největších rozdílových bloků v předešlém řešení efektivně odstranilo vliv šumu, tak to také vedlo k potlačení rozdílnosti dvou snímků z jiných záběrů. Naproti tomu přístup Ueda klade důraz na bloky, které se nejvíce liší od jednoho snímku k dalšímu. Kombinace tohoto a skutečnosti, že bloky byly menší vedlo k větší senzitivitě na pohyb kamery a objektu [13]. To poukazuje na problém výběru vhodné škály pro porovnávání vlastností vizuálního obsahu mezi dvěma snímky. Použitím lokálnějšího měřítka se zvyšuje citlivost algoritmu na pohyb kamery a objektu, kdežto použití globálnějšího měřítka snižuje citlivost algoritmu na změny v prostředí.

3.3.4 Změny pohybu

K překonání citlivosti na pohyb kamery a objektu bylo navrženo hned několik metod, které se snaží o odstranění rozdílů mezi dvěma snímky způsobenými takovými pohyby před samotným porovnáním. Byly navrženy metody, které zahrnují proces shody bloků k získání podobnosti inter-framu[4] na základě pohybu [14, 15, 16]. Pro každý blok ve snímku f_{n-1} je hledán nejvhodnější blok v okolí kolem odpovídajícího bloku ve snímku f_n. Porovnávání bloků se provádí na základě intenzity v obrazových datech. Blok s nejvyšší shodou je vybrán tak, aby maximalizoval normalizovaný korelační koeficient. Tento koeficient je poté použit na porovnávání podobnosti dvou bloků.

Hlavní rozdíl mezi těmito přístupy tkví v tom, jak se vyhodnotí všechny bloky a získá se globální parametr shody. Akatsu a spol. použili průměr koeficientu maximální korelace pro každý blok [14]. To mělo za následek smíchání negativních shod s těmi pozitivními pro dosažení shody mezi dvěma snímky patřícími do stejného záběru. Shahraray použil nelineární statistický filtr [15]. To umožnilo přiřazovat blokům váhu, jež mohla dále ovlivnit prioritu dobře vyhovujících bloků. Došlo tak ke zlepšení v případech, kdy některé z bloků, které se porovnávaly, měly velkou míru neshody. Nevýhodou ovšem bylo, že mezi dvěma snímky z různých záběrů může existovat dobrá shoda, což má za následek méně významnou změnu, která naznačuje, že došlo ke střihu. K překonání tohoto problému autoři navrhli, aby se bloky vážily tak, že několik nejlepších odpovídajících bloků bude vyloučeno. To znamená, že koeficienty nelineárního průměrovacího filtru musí být zvoleny pozorně, když se rozdělení hodnot podobnosti mezi dvěma snímky může značně lišit.

Lupatini a kol. sečetli hodnoty rozdílů pohybem kompenzovaných obrazových bodů pro každý blok [16]. Pakliže tato suma překročila daný práh mezi dvěma snímky, byl deklarován střih. Naopak nový přístup byl navržen Vlachosem, který používal fázovou korelaci pro získání míry podobnosti obsahu mezi dvěma snímky [17]. Tato metoda je imunní proti změnám v globálním osvětlení a nabízí se pro implementaci na frekvenční doméně.

[4]Inter-frame - snímek, jež je popsán/vyjádřen na základě jednoho nebo více sousedících snímků

Nakonec Fernando a spol. využili skutečnosti, že vektory pohybu jsou náhodné během náhlého střihu [18]. Byl určen průměr vektorů pohybu mezi dvěma snímky a Euklidovská vzdálenost vzhledem k průměru vektoru vypočítanému pro všechny vektory pohybu. Pokud existuje střih, tak většina pohybových vektorů bude mít velkou odchylku v důsledku špatné korelace mezi dvěma snímky. Velká změna v Euklidovské vzdálenosti může být poté použita k detekci střihu. Myšlenka že dva snímky na obou stranách střihu jsou zcela nekorelované takové, že jsou získány nesouvislé odhady pohybu, je také využívána ostatními [14, 19].

3.3.5 Přístup založen na příznacích v obraze

Dalším prvkem na jehož základě lze detekovat přítomnost střihu v záběrech jsou hrany. Zabih a spol. navrhli metodu na detekci střihů kontrolou prostorového rozložení výstupních a vstupních hranových obrazových bodů, známé jako poměr změn hran (ECR) [20]. Tato metoda využívá skutečnosti, že hrany objektů ve snímku před střihem se nemohou nacházet ve stejné lokaci v prvním snímku po střihu, tj. nové hrany se objevují vzdáleně od lokace zmizení starších hran. Byla použita registrační technika, jež kompenzuje globální pohyb mezi dvěma snímky. Ke kompenzování pohybů malých objektů se hraniční obrazové body v jednom snímku v malé vzdálenosti od hraničních obrazových bodů v druhém nezapočítávaly jako vstupní nebo výstupní hrany. Tudíž jakýkoliv rozdíl mezi hranovými obrazovými body by měl být pouze výsledkem střihu. Nechť E_n je celkový počet hraničních obrazových bodů v obraze f_n a I_n, O_{n-1} je počet vstupujících a výstupních hranových obrazových bodů ve snímku n a $n-1$. Poměr změn hran mezi f_{n-1} a f_n je definován jako:

$$ECR_n = max(I_n/E_n, O_{n-1}/E_{n-1}) \tag{3.5}$$

Podle vzorce $0 \leq ECR_n \leq 1$ kde 0 indikuje rovnost. Ačkoliv tato metoda ukázala proveditelnost detekce na základě hran, byl její výkon neuspokojivý ve srovnání s více jednoduchými metrikami, které jsou méně výpočetně náročné [21, 16, 22].

Novodobý způsob detekce střihů, a nejen jich by mohl vypadat následovně:

1. změnit velikost obrázku na 192×256
2. histogram orientovaných gradientů s bloky o velikosti 64×64
3. tři hodiny manuálně segmentovat videonahrávku
4. natrénovaná CNN (konvoluční neuronová síť) skládající se z 5 vrstev (3 konvoluční a 2 plně propojené)
5. výstup:
    - není segment
    - ostrý střih
    - průnik
    - posun pohybem

3.4 Výběr klíčového snímku

Hlavním postupem k automatizaci procesu indexování videa je výběr reprezentativních klíčových snímků ze záběrů nebo scén pro vytvoření video abstraktu. Kolekce klíčových snímků může být poté využita k další charakterizaci a následnému třídění video dat. Stojí za to poznamenat, že výběr klíčových snímků nemá žádná pravidla a je subjektivní a často závislý na aplikaci. Pro efektivní prohlížení a získávání videa by vybrané klíčové snímky měly být schopny reprezentovat celý záznam [23]. Naproti tomu Dufaux navrhl techniku automatického stahování jediného klíčového snímku z videa určeného pro systém, jež hledá videa na webu [24].

Byly navrženy dva hlavní postupy k získání klíčového snímku:

1. s explicitní detekcí přechodu záběru
2. bez této detekce

Původní přístup byl, že první snímek v každém záběru je klíčový [25, 26]. Seřazená množina klíčových snímků je někdy označována jako filmový pás. Tento přístup není vždy vhodný, protože mohou existovat výrazné změny ve snímku díky pohybu kamery nebo objektu. Pro zvýšení počtu snímků v záběru Ardizzone a Cascia navrhli, aby počet snímku byl odvozen od délky záběru [27]. Pokud je záběr kratší než jedna sekunda, byl zvolen prostřední snímek. A pokud je záběr delší, byl vybrán klíčový snímek v každé sekundě. Jakmile jsou klíčové snímky vybrány, tak jsou charakterizovány svým optickým tokem pro účely video indexace. Tento přístup může převzorkovat sekvenci, neboť záběr může být dostatečně dlouhý, ale obsahovat zanedbatelné změny v obsahu.

Zhang a kol. navrhli extrahovat klíčový snímek pomocí podobných způsobů nalezených u detekce střihu [28]. V záběru byl zvolen vždy první snímek. Následující snímky v záběru byly porovnány s posledním vybraným klíčovým snímkem na základě nějakého porovnání podobnosti, jako jsou histogramy. Pokud byla zaznamenána velká změna v obsahu, byl momentální snímek zvolen jako další klíčový. Bylo navrženo, aby každá výrazná akce byla reprezentována klíčovým snímkem, zatímco statické záběry měly pouze jeden klíčový snímek. Kim a Park navrhli použít kumulativní četnost mezi klíčovými snímky [29]. Jakmile byly klíčové snímky vybrány, jejich podobnost mezi různými video záběry byla vyhodnocena modifikovanou Hausdorffovou vzdáleností na množině klíčových snímků. Chang a kolektiv zase navrhli zajímavou metodu jak určit minimální množinu klíčových snímků pro záběr tak, aby vzdálenost mezi každým snímkem v záběru a alespoň jedním klíčovým v množině byla menší než jakýsi práh [23]. Použili příklady barevných histogramů a korelaci jako ukazatel odlišnosti. Nejkratší cestou v grafu byla vyhodnocena ideální množina klíčových snímků.

Alternativní cestou, jak najít optimální množinu klíčových snímků, jejíž snímky jsou maximálně odlišné a nesou většinu informací, navrhl Vermaak a kol. [30]. Vstupní video bylo transformováno do sekvence příznakových vektorů. Pomocí této reprezentace byla definována obslužná funkce a posloupnost klíčových snímků, která tuto funkci maximalizuje.

Namísto použití kritéria vzdálenosti použil Wolf optický tok k identifikaci lokálního minima pohybů ve snímku k identifikaci klíčových snímků [31]. Bylo navrhnuto, aby se klíčové snímky identifikovaly v klidu. Součet veličin optického toku na každém obrazovém bodu byl vypočítán a body lokálních minim v sekvenci byly použity pro výběr klíčových snímků.

Jak bylo zmíněno výše, existují přístupy, které se specificky nezaměřují na jednotlivé záběry pro výběr klíčových snímků. Obvykle jsou snímky reprezentovány na zmenšeném prostoru rozměrů používající reprezentace podobné těm, které se používají pro detekci změny záběru jako jsou SVD (Singular-value decomposition) [32] a PCA (Principal component analysis) [33]. Seskupování snímků se typicky provádí prahováním [32], chamtivým seskupováním [34] nebo zjednodušením přímky [34] ve zmenšeném prostoru rozměrů.

4 Detekce obličejů

Základním problémem k vyřešení je navrhnutí algoritmu pro detekci obličejů v obraze. Existuje celá řada různých metod a algoritmů. V současné době je nejpoužívanější Viola-Jones a Dlib.

Viola-Jones se používá na výpočetně slabších zařízení a jeho nevýhodou je větší množství falešných detekcí. Aby se tento problém zjednodušil, je Viola-Jones limitován pouze na čelní, mírně vychýlené pohledy. To znamená, že k úspěšné detekci musí směřovat celá tvář naproti kameře a neměla by být nakloněna ani na jednu stranu.

Viola-Jones je metoda, kdežto Dlib je knihovna funkcí a zastřešuje momentálně 2 způsoby, jak detekovat tvář v obraze. Tyto detektory jsou založeny na:

- Histogramech orientovaných gradientů
- Konvolučních neuronových sítích

Jedná se o široce používaný model detekce obličeje, založený na HoG (Histogram of oriented gradients) příznacích a SVM (Support Vector Machine). Tento model je postaven z 5 HoG pohledových filtrů, přední, pravý, levý, přední rotovaný doprava, přední rotovaný doleva. Dataset použitý na trénování se skládá z 2825 obrázků, které jsou získány z LFW datasetu a manuálně anotované Davisem Kingem, tvůrcem Dlib. V této metodě záleží na zvětšení obrazu. Čím větší je obraz, tím větší je šance na odhalení malých tváří. Nicméně zvětšování obrazu se negativně podepisuje na výkonu. Jedná se o nejrychlejší metodu na CPU (Central processing unit) a funguje velice dobře s čelními a mírně vychýlenými pohledy. Nevýhodou je, že dokáže detekovat pouze tváře o velikost 80×80 a více. Důvod je ten, že natrénovaný model nebyl trénován na menších tvářích. Musí se proto zajistit, aby velikost obličeje byla větší než zmíněná hodnota, nebo použít vlastní detektor tváří pro menší rozměry obličeje. Dále jeho ohraničující rámečky často vynechávají čelo a část brady. V neposlední řade nefunguje na tváře z boku a extrémně nečelní pohledy jako při pohledu nahoru nebo dolu.

Metoda konvolučních neuronových používá *Maximum-Margin Object Detector* (MMOD) s CNN založenými příznaky. Trénování je velice jednoduché a není potřeba velké množství dat k natrénování vlastního detektoru. K trénování je použit dataset, který je ručně označován jejím autorem Davisem Kingem. Tento dataset se skládá z různých datasetů jako ImageNet, PASCAL, VOC, VGC, WIDER, Face Scrub. Obsahuje 7220 obrázků. Toto řešení funguje na různé natočení tváří, je velice rychlé na GPU a obsahuje jednoduchý trénovací proces. Na druhou stranu je pomalé na

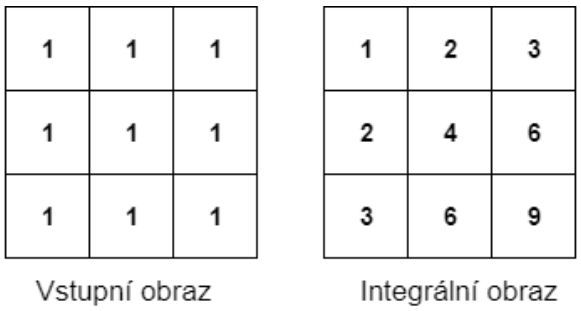
CPU a opět nedokáže detekovat tváře menší než 80×80. Navíc jsou jeho ohraničující rámečky ještě menší než u předchozí metody.

4.1 Viola-Jones

Základním principem Viola-Jonesova algoritmu je skenování okénka schopného detekovat tváře napříč celým vstupním obrazem. Standardní přístup ke zpracování obrazu spočívá ve změně měřítka vstupního obrazu na různé velikosti a následné spuštění vyhledávacích oken s pevnou velikostí přes tyto obrazy. Tento přístup se ukazuje jako velice výpočetně náročný díky výpočtu různých velikostí obrazu. Naproti tomu Viola-Jones namísto vstupního obrazu mění velikost vyhledávacího okna a nechává běžet vyhledávací okno několikrát obrazem pokaždé s jinou velikostí. Na první pohled by se mohlo zdát, že oba přístupy jsou stejně výpočetně náročné. Nicméně Viola-Jones vymyslel invariantní velikostní vyhledávací okno, jež vyžaduje stejný počet kroků k výpočtu nehledě na velikost. Toto vyhledávací okno je sestrojeno za pomoci takzvaného integrálního obrazu a některých jednoduchých obdélníkových příznaků připomínajících Haarovy vlnky.

4.1.1 Invariantní velikostní vyhledávací okno

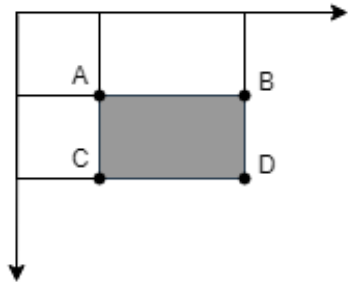
Prvním krokem Viola-Jonesova detektoru tváří je transformace vstupního obrazu do integrálního obrazu. Toho je docíleno tak, že se každý obrazový bod rovná celkové sumě všech obrazových bodů nad a vlevo od dotyčného obrazového bodu. Tento výpočet je demonstrován na obrázku 4.1.



Obrázek 4.1: Integrálový obraz

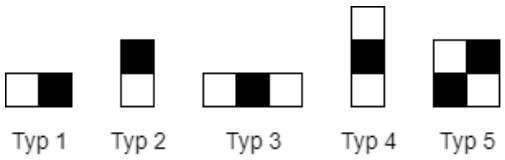
To dále umožňuje vypočítat sumu všech obrazových bodů uvnitř daného obdélníku za pomocí pouhých čtyř hodnot. Tyto hodnoty jsou obrazové body integrálního obrazu, které se shodují s rohy obdélníku vstupního obrazu. Ukázku vidíme na obrázku 4.2. Sumu tohoto šedého obdélníku vypočteme jako:

$$D - (B + C) + A \tag{4.1}$$



Obrázek 4.2: Kalkulace sumy

Vzhledem k tomu, že obdélník B a C obsahuje obdélník A, musí být do výpočtu přidán součet A. Na základě těchto znalostí bylo demonstrováno, že součet obrazových bodů v libovolné obdélníkové velikosti lze vypočítat v konstantním čase. Detektor obličejů analyzuje daná okénka pomocí příznaků, skládajících se ze dvou nebo více obdélníků. Výběr rozdílných příznaků je ukázán na obrázku 4.3.



Obrázek 4.3: Různé typy příznaků

Každý příznak vyústí v jednu hodnotu, která se počítá odečtením součtu bílých obdélníků od sumy těch černých. Viola-Jones pokusem a omylem zjistil, že vyhledávací okno s rozlišením 24×24 obrazových bodů poskytuje uspokojivé výsledky. Když povolíme všechny možné kombinace velikostí a poloh příznaků z obrázku 4.3, pak může být vytvořeno přibližně 160 000 různých příznaků. Čili počet všech možných příznaků zdaleka převažuje 576 obrazových bodů obsažených ve vyhledávacím okně při základním rozlišení. Tyto příznaky se mohou zdát až velice lehké na provedení pro tak komplexní úkol jako je detekce obličeje. Co příznakům chybí v komplexnosti, jim zdaleka nechybí ve výpočetní efektivitě.

Příznaky se dají pochopit jako způsob, jak počítač vnímá vstupní obraz. Předpokládá se, že některé příznaky budou vykazovat velké hodnoty, bude-li se jednat o tvář. Na řadě je sestavení vhodné množiny příznaků schopné detekovat tváře.

4.1.2 Modifikovaný AdaBoost

Jak již bylo řečeno, tak v základním vyhledávacím okně může být vypočítáno přibližně 160 000 příznaků. Mezi všemi těmito vlastnostmi se očekává, že pár z nich bude konzistentně vracet vysoké hodnoty, když budou na tváři. K nalezení těchto příznaků používá Viola-Jones modifikovanou verzi AdaBoost algoritmu vyvinutou Freundem a Schapirem v roce 1996.

AdaBoost je algoritmus strojového učení, který je schopný vytvořit silný klasifikátor prostřednictvím vážené kombinace slabých klasifikátorů. Slabý klasifikátor je definován tak, že korektně klasifikuje lehce nad polovinu všech případů. K přirovnání této terminologie k prezentované teorii, je každý prvek považován za potenciálně slabý klasifikátor. Slabý klasifikátor je matematicky popsán jako:

$$h(x, f, p, \theta) = \begin{cases} 1 & pf(x) > p\theta \\ 0 & jinak \end{cases} \tag{4.2}$$

Kde x je 24×24 okénko obrazových bodů, f je aplikovaný příznak, p pro polaritu a θ práh, který rozhoduje jestli má být x klasifikován jako pozitivní výsledek (tvář) nebo negativní. Očekává se, že pouze malé množství příznaků z možných 160 000 kombinací budou potenciální slabé klasifikátory a tak se AdaBoost upraví, aby vybíral pouze ty nejlepší příznaky.

Důležitým krokem upraveného AdaBoost algoritmu je určení nejlepší vlastnosti polarity a prahu. Na tento problém neexistuje chytré řešení a Viola-Jones navrhuje prostý útok hrubou silou. To znamená, že k určení každého nového slabého klasifikátoru se musí vyhodnotit každý příznak na všech trénovacích datech, aby se našel nejlepší příznak. Dost pravděpodobně se tak jedná o nejvíce časově náročnou operaci v oblasti trénování.

Nejlepší příznak je zvolen na základě vážených chyb, které produkuje. Tato vážená chyba je funkcí vah, které patří k trénovacím datům. Váha správně klasifikovaného příkladu je snížena a váha nesprávně klasifikovaného příkladu je udržována konstantní. V důsledku toho je pro druhý příznak v konečném klasifikátoru těžší nesprávně klasifikovat příklad, který byl také nesprávně klasifikován podle prvního příznaku, než příklad správně klasifikovaný. Toto tvrzení se dá přeformulovat tak, že druhý příznak je nucen se více zaměřit na příklady nesprávně klasifikované prvním. Jde o to, že váhy jsou nedílnou součástí algoritmu AdaBoost.

S integrálním obrazem, výpočetně nenáročnými příznaky a modifikovaným AdaBoostem chybí už pouze poslední eso v rukávu pro implementaci tohoto algoritmu a to kaskádový klasifikátor.

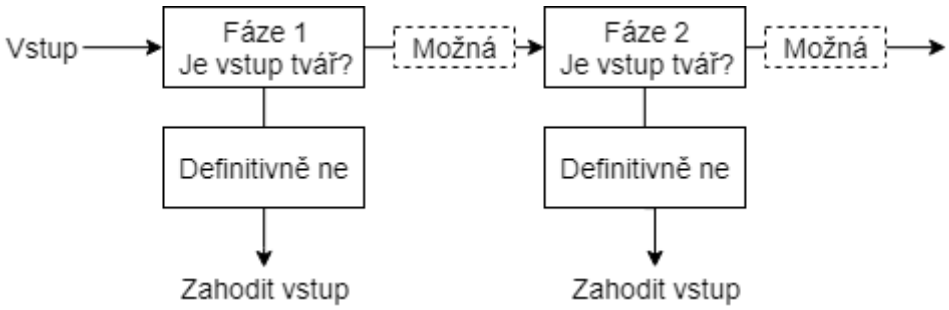
4.1.3 Kaskádový klasifikátor

Základním principem algoritmu pro detekci obličeje je několikrát skenovat vyhledávací okno přes stejný obraz pokaždé s jinou velikostí. I když by měl obraz obsahovat jednu nebo více tváří je zřejmé, že nadměrně velké množství vyhodnocených okének bude stále negativními bez tváře. Tato realizace vede k jiné formulaci problému. Namísto hledání tváří by měl algoritmus vyřadit ne-tváře.

Myšlenkou tohoto tvrzení je, že je rychlejší se zbavit ne-tváře než ji najít. S tímto vědomím se zdá, že vyhledávací okno skládající se pouze z jednoho silného klasifikátoru se náhle jeví neefektivní, protože doba vyhodnocení je konstantní bez ohledu na vstup. Z tohoto důvodu je potřeba kaskádového klasifikátoru.

Kaskádový klasifikátor se skládá z fází, kde každá obsahuje silný klasifikátor. Úkolem každé fáze je určení, zda se v okénku definitivně nenachází tvář nebo tam možná existuje. Pakliže je okénko klasifikováno že neobsahuje vůbec tvář, je v dané

fázi zahozeno. A naopak okénko označené jako možná tvář je předáno další fázi v kaskádě. Z toho vyplývá, že čím více fází okénko projde, tím je větší šance, že tam je tvář. Funkcionalita je znázorněna na obrázku 4.4, kde má klasifikátor dvě fáze.



Obrázek 4.4: Kaskádový klasifikátor

V klasifikátoru s jednou fází by za normálních podmínek bylo možné přijmout falešné negativy, aby se snížila falešně pozitivní hodnota. Nicméně pro první fáze v kaskádovém klasifikátoru se nejedná o problém, protože se předpokládá, že je setřídí následující fáze. Proto Viola-Jones toleruje přijetí mnoha falešných pozitiv v počátcích. Očekává se tedy, že množství falešných negativů bude velmi malé u finálních fází klasifikátoru.

4.2 Dlib

Dlib je moderní C++ knihovna, obsahující algoritmy a nástroje pro strojové učení a tvorbu komplexního softwaru, jež řeší reálné problémy. Používá se jak v průmyslu tak v akademické sféře v široké škále oblastí. Licence Dlib je na bázi otevřeného zdrojového kódu a lze ji bezplatně využívat v jakékoli aplikaci. Dlib obsahuje celkem dva různé detektory, jak vyřešit problematiku detekcí tváří.

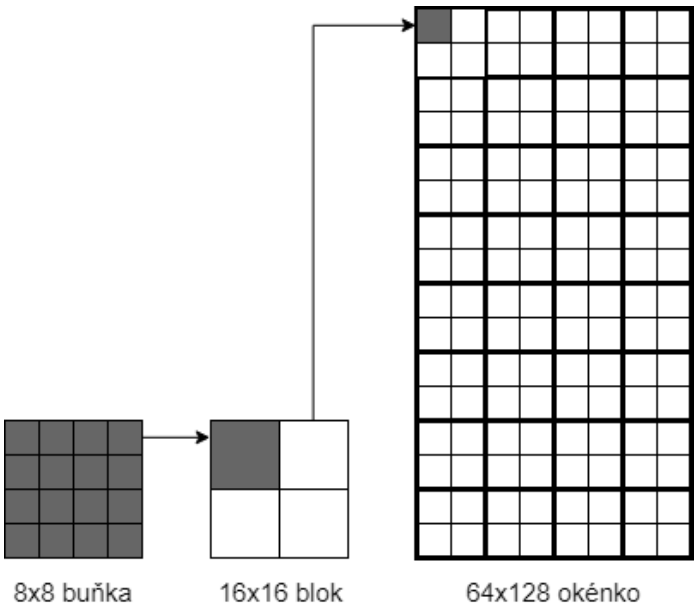
V této práci bude pojednáno o HoG a SVM metodách, jež byly použity v praktické části. Tato konkrétní implementace je nejrychlejší na CPU. Funguje velice dobře na přímých a mírně nepřímých tvářích a její model je oproti ostatním Dlib detektorům úspornější na úložný prostor. Nevýhodou je, že předem natrénovaný detektor nedokáže detekovat tváře menší než 80×80 a ohraničující sektory občas vynechávají čelo nebo bradu.

4.2.1 Extrakce příznaků HoG

Histogram orientovaných gradientů byl navrhnut kolem roku 2005 Dalalem a spol. k detekci chodců. HoG příznaky jsou robustní a nezávisí na osvětlení ani na geometrických změnách v obraze. Výpočetní složitost HoG příznaků je mnohem menší než u původních dat. Hlavními kroky extrakce HoG příznaků jsou následující:

1. Okénko obrazových bodů 64×128 je rozděleno na 8×8 buňky, jež tvoří $8 \times 16 = 128$ buněk, jak je zobrazeno na obrázku 4.5. Gradientní komponenta každého

obrazového bodu (x, y) v horizontálním a vertikálním směru je vypočítána pomocí vzorce 4.3 a 4.4. Velikost a směr gradientu pro každý obrazový bod vychází ze vzorečků 4.5 a 4.6.



Obrázek 4.5: Rozdělení okénka na dílčí bloky

$$G_x(x, y) = I(x + 1, y) - I(x - 1, y) \tag{4.3}$$

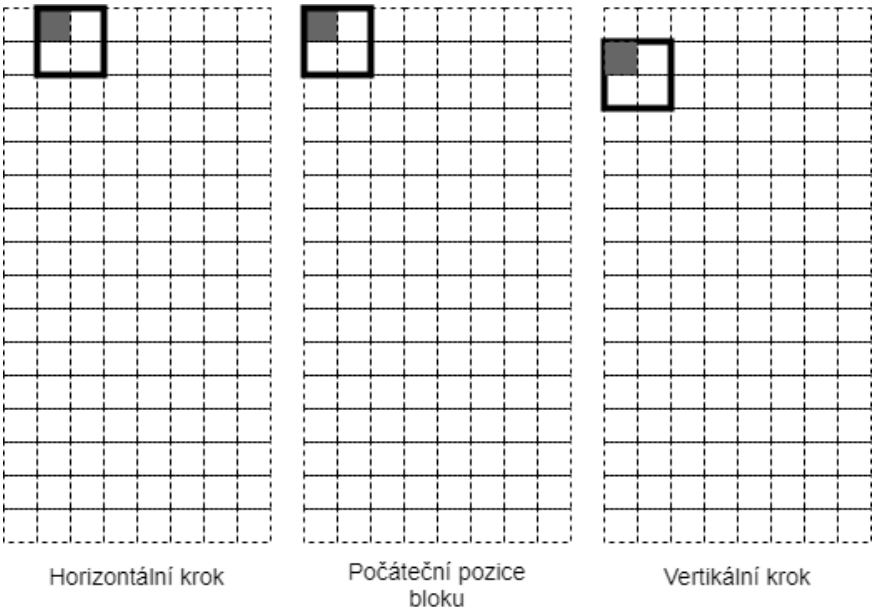
$$G_y(x, y) = I(x, y + 1) - I(x, y - 1) \tag{4.4}$$

$$m(x, y) = \sqrt{(G_x(x, y))^2 + (G_y(x, y))^2} \tag{4.5}$$

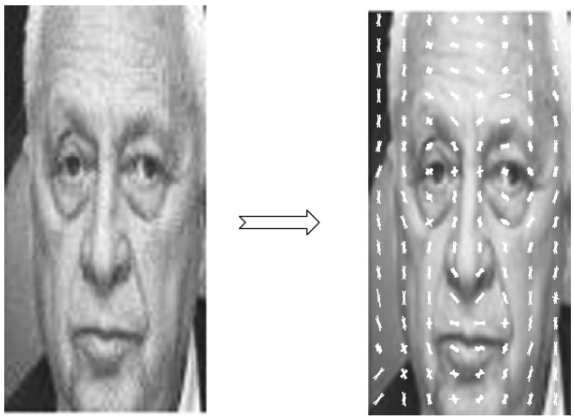
$$\theta(x, y) = \arctan \frac{G_y(x, y)}{G_x(x, y)} \tag{4.6}$$

2. Blok obrazových bodů 16×16 se skládá z $2 \times 2 = 4$ buněk a $7 \times 15 = 105$ bloků je vytvořeno. Velikost kroku bloku je 8 obrazových bodů. Počet bloků v horizontálním směru je $(64 - 16)/8 + 1 = 7$, a počet bloků ve vertikálním směru je $(128 - 16)/8 + 1 = 15$ podle obrázku 4.6.

3. Vezmeme histogram 9 gradientů pro každou buňku, podle obrázku 4.7. Takový blok bude mít $4 \times 9 = 36$ příznakových vektorů. Následuje spojení 105 bloků s příznakovými vektory do série k vytvoření obrazu o $36 \times 105 = 3780$ HoG příznaků.



Obrázek 4.6: Krokovací proces bloku



Obrázek 4.7: Vizualizace extrakce HoG příznaků [6]

4.2.2 PCA

Principem analýzy hlavních komponent je transformace originálních dat do množiny lineárně nezávislých dat pomocí lineárních transformací. Z této množiny se pak dají extrahovat hlavní komponenty dat. Proto se toto často používá k redukci rozměrů dat s vysokými rozměry. Pro problematiku rozpoznávání tváří se často stává, že rozměry příznaků jsou mnohdy větší než samotný počet dat.

4.2.3 SVM

Support vector machines, neboli metoda podpůrných vektorů, je široce používaný klasifikátor, díky jeho silné klasifikační schopnosti pro malá i rozměrově velká data. Algoritmus SVM odděluje různé kategorie tím, že se snaží nalézt optimální nadrovinu mezi různými daty. Optimální rovina může být vyjádřena jako:

$$y = \omega^T \phi(x) + b \tag{4.7}$$

kde ω je normálový vektor nadroviny, a b je offset nadrovinového vektoru.

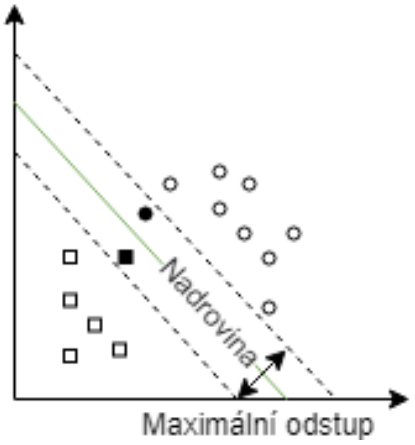
Pro problém lineární nedělitelnosti je třeba převést nelineární klasifikační problém na kvadratický optimalizační problém. Poté je použita metoda Lagrangeových multiplikátorů k transformaci klasifikačního problému na jeho dvojí problém. Konečná funkce nadroviny je:

$$f(x) = sign(\sum_{i=1}^{n} \alpha_i y_i (\phi(x) \cdot \phi(x_i)) + b) \tag{4.8}$$

kde $sign$ je funkce signum a α_i je Lagrangeův multiplikátor. Použitím funkce jádra $k(x_i \cdot x)$ namísto $\phi(x) \cdot \phi(x_i)$ vnikne:

$$f(x) = sign(\sum_{i=1}^{n} \alpha_i y_i k(x_i \cdot x) + n) \tag{4.9}$$

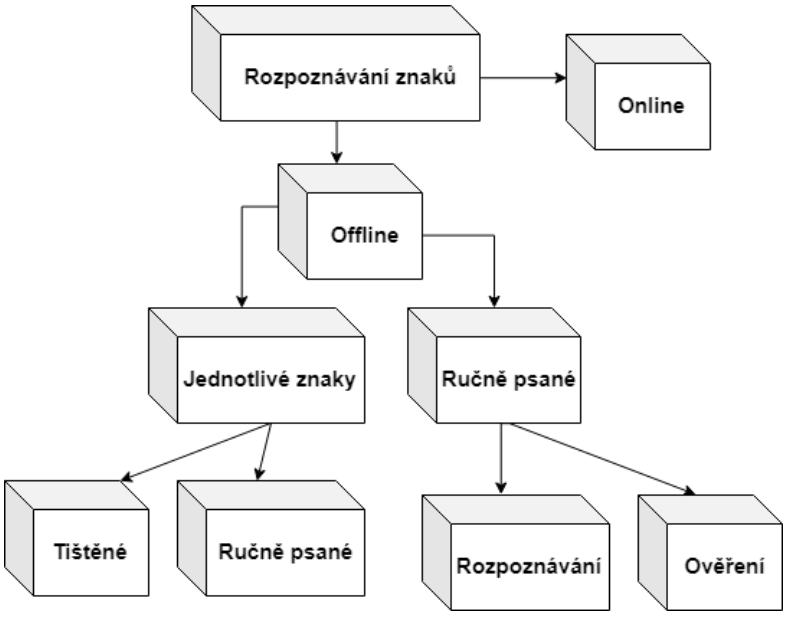
Mezi běžné funkce jádra patří funkce lineárního jádra, Gaussova radiální základní funkce jádra (RBF), funkce polynomiálního jádra a funkce jádra Sigmoid.



Obrázek 4.8: Optimální nadrovina a její odstup

5 Optické rozpoznávání znaků

Optické rozpoznávání znaků patří do skupiny technik automatické identifikace. Do této skupiny patří dále rozpoznávání hlasu, radiové frekvence, barových kódů atd. OCR (Optical character recognition) se zabývá problémem offline rozpoznáváním opticky zpracovaných znaků poté, co byly napsány nebo vytisknuty. Oproti online rozpoznávání kde stroj rozpoznává znaky jakmile jsou nakresleny. Mohou být rozpoznávány jak tištěné tak psané znaky, ale výkon je přímo závislý na kvalitě vstupního dokumentu.



Obrázek 5.1: Různé skupiny optického rozpoznávání znaků

5.1 Historie OCR

Původ optického rozpoznávání se rýsuje zpět až do roku 1870. Jednalo se o rok, kdy C.R.Carey vynalezl skener sítnice, jež přenášel obraz na základě mozaiky fotobuněk. O dvě dekády později vynalezl P.Nipkow sekvenční skener, který byl průlomem pro moderní televize. Během prvních desetiletí 19. století bylo učiněno několik pokusů k vytvoření přístrojů, jež měly pomoci nevidomým prostřednictvím experimentů s OCR. Nicméně objevení prvních moderních OCR nastalo až po roce 1940 s vývojem digitálního počítače.

Rokem 1950 se technologická revoluce pohybovala vpřed a zpracování elektronických dat se stalo důležitým oborem. Zadávání dat bylo realizováno přes děrované karty a bylo potřeba přijít s novými postupy, jak zpracovávat přibývající data. Ve stejné době se stala technologie čtení znaků dostatečná pro tuto aplikaci. Rokem 1950 se objevily první komerční OCR stroje. První opravdový OCR stroj byl instalován v *Reader's Digest* roku 1954. Toto vybavení bylo určeno ke konverzi psaných dat prodejů na děrované karty pro vstup do počítače.

První generace OCR systémů se začala objevovat v rozmezí roků 1960-1965. Tato generace OCR byla charakterizována hlavně omezenými tvary písmen. Symboly byly speciálně navrženy pro strojové čtení a první z nich nevypadaly ani přirozeně. Časem se začaly objevovat multifontové stroje, jež dokázaly číst až deset různých stylů písma. Počet fontů byl omezen díky aplikované metodě na rozpoznávání vzorů jmenující se porovnávání šablon. Ta porovnává znak ve formě obrazu s knihovnou obrazů pro každý znak každého fontu.

Čtecí stroje druhé generace se objevily okolo roku 1960 a 1970. Tyto systémy byly schopny rozeznat jak tištěné znaky tak ručně psané znaky. Psané znaky byly omezeny na množinu čísel, pár písmen a symbolů. První známý systém tohoto typu byl IBM 1287, jež byl představen roku 1965 v New Yorku. Toshiba v tomto roku také vyvinula první automatický stroj na třídění poštovních směrovacích čísel. A Hitachi představila první OCR stroj s vysokým výkonem a nízkými náklady. V tomto časovém období došlo k výraznému pokroku v oblasti normalizace. V roce 1966 byla definována množina znaků pro Ameriku jako OCR-A. Toto písmo bylo vysoce stylizované a navrženo tak, aby usnadnilo optické rozpoznávání a bylo stále čitelné člověkem. Evropský standard OCR-B byl také navržen a obsahoval více přirozených fontů než americký. Byly provedeny pokusy o sjednocení těchto standardů, ale namísto toho stroje podporovaly obě množiny znaků.

Obrázek 5.2: OCR-A (nahoře), OCR-B (dole) [3]

Výzvou třetí generace OCR systémů objevujících se roku 1970 měly být dokumenty špatné kvality a velké psané i tištěné znaky. Nízké náklady a vysoký výkon byly také důležitými cíli, kterým pomáhal dramatický rozvoj hardwaru. Ačkoliv se začaly objevovat komplexnější OCR stroje, jednoduché stroje byly stále velice užitečné. Předtím než osobní počítače a laserové tiskárny začaly dominovat textové produkci bylo psaní speciální pro OCR. Jednotné mezery mezi písmeny a nízký počet fontů udělaly jednoduše navržené OCR stroje velice užitečnými. Návrhy mohly být vytvořeny na obyčejných psacích strojích a dodávány do počítače skrz OCR přístroje pro finální editaci. Tímto způsobem byly zpracovány textové procesory, které byly tenkrát drahé, ale mohly podporovat několik uživatelů zároveň.

Ačkoliv OCR stroje byly komerčně přístupné, okolo roku 1950 pouze několik tísíc systémů bylo prodáno celosvětově do roku 1986. Hlavním důvodem byla cena. Nicméně jak se hardware stával levnějším a OCR systémy začaly být dostupné v podobě softwaru, prodeje rostly. Dnes se prodává několik tisíc systémů každý týden a náklady na OCR klesly desetkrát každý druhý rok za posledních 6 let.

1870	První pokusy
1940	Moderní verze OCR
1950	První objevení OCR strojů
1960-1965	První generace OCR
1965-1975	Druhá generace OCR
1975-1985	Třetí generace OCR
1986 a dál	OCR lidem

Tabulka 5.1: Přehled vývoje OCR v čase

5.2 Tesseract OCR

Tesseract je volně dostupný OCR engine, jež byl vyvíjen Hewlett Packardem v roce 1984 až 1994. Začal jako disertační práce v laboratořích HP (Hewlett-Packard) v Bristolu a získal momentum jako jeden z možných doplňkových softwarů případě hardwarů pro jejich řadu skenerů.

Po společném projektu mezi laboratořemi v Bristolu a Coloradu měl Tesseract významný náskok v přesnosti oproti komerčním enginům, ale nestal se produktem. Další etapa vývoje pokračovala zpět v Bristolu, kde se odborníci zabývali použitím OCR na kompresi. Etapa se zabývala více na zlepšení účinnosti zamítnutí než zlepšení přesnosti. Na konci roku 1944 vývoj zcela zastavil. Engine byl poslán na univerzitu v Nevadě pro výroční zkoušku v roce 1995 v přesnosti OCR, kde obstál proti konkurenci z té doby. V roce 2005 vydal HP Tesseract jako open-source.

5.2.1 Architektura

Jelikož HP navrhla svoji vlastní technologii analýzy rozložení stránky, jež byla použita v jejich produktech (nejedná se o open-source), tak Tesseract nikdy ne-

potřeboval svoji vlastní analýzu stránky. Tesseract proto předpokládá, že jeho vstupem bude binární obraz s volitelnými polygonálními oblastmi textu.

Zpracování následuje tradičním zřetězením kroků za krokem. Nicméně některé fáze byly nezvyklé a možná i tak zůstávají v současnosti. Prvním krokem je propojená analýza komponent, ve které jsou uloženy obrysy komponent. V té době se jednalo o výpočetně náročný design, který měl nesporné výhody. Inspekcí vnořovaných obrysů a počtu obrysů potomků a vnuků je lehké detekovat inverzní text a rozpoznat ho stejně snadno jako černobílý text. Tesseract byl pravděpodobně jedním z prvních OCR nástrojů, který dokázal rozeznat černobílý text tak snadno. V této fázi jsou obrysy shromážděny dohromady čistě vnořováním do tzv. *Blobs*[1].

Bloky jsou uspořádány do textových řádků a řádky a oblasti jsou analyzovány na neproporcionální nebo proporcionální text. Textové řádky jsou rozděleny do slov různě podle typu znakových mezer. Neproporcionální text je okamžitě detekován znakovými buňkami. Proporciální text je rozdělen do slov díky určitým mezerám a fuzzy mezerám.

Rozpoznání pak pokračuje do dvouúrovňového procesu. V první úrovni je pokus o rozpoznání každého slova. Každé takové slovo, které projde je předáno adaptivnímu klasifikátoru jako trénovací data. Adaptivní klasifikátor dostane později větší šanci ke správnému rozpoznání textu na pozdějších stránkách.

Jelikož se adaptivní klasifikátor mohl naučit něco užitečného pozdě, je blízko vrchu stránky spuštěn druhý průchod přes stránku, ve kterém jsou špatně rozpoznaná slova identifikována znovu.

Finální fáze řeší fuzzy mezery a kontroluje alternativní hypotézy pro výšku x k nalezení malých textů.

5.2.2 Hledání řádku

Hledání řádku je jedním z mála částí Tesseractu, jež byl dříve zveřejněn. Je navržen tak, aby zkosenou stránku nemusel transformovat a tím se zachovává kvalita obrazu. Klíčovými prvky tohoto procesu jsou filtrace bloků a konstrukce řádku.

Za předpokladu, že analýza rozložení stránky již proběhla a poskytla textové oblasti s přibližně jednotnou velikostí textu, je použit jednoduchý filtr výšky, který odstraňuje iniciály a vertikálně se dotýkající znaky. Průměrná výška se přibližně rovná velikosti textu v oblasti, takže je bezpečné odfiltrovat bloky, které jsou menší než určitý zlomek střední výšky, což je s největší pravděpodobností interpunkční znaménko, diakritické znaménko nebo šum.

Filtrované bloky mají větší šanci pasovat do modelu nepřekrývajících se paralelních ale šikmých řádků. Tříděním a zpracováním bloků podle souřadnice x nám umožňuje přiřadit blokům jedinečný textový řádek. Zatímco sledováním zešikmení celé stránky značně snižujeme riziko špatného přiřazení řádku v momentě zkosení. Jakmile jsou filtrované bloky přiřazeny k řádkům, je použita metoda nejmenších čtverců k určení základní hladiny textu a odfiltrované bloky se vloží zpět k příslušným řádkům.

[1] Blob - v češtině neexistuje ekvivalent v této terminologii proto bude dále hovořeno o "bloku"

V posledním kroku hledání řádků se sloučí bloky, které se překrývají horizontálně alespoň z poloviny, přičemž se správně přidávají diakritická znaménka a některé poničené znaky.

5.2.3 Přizpůsobení základních hladin

Jakmile jsou nalezeny řádky s textem, jsou základní hladiny textu rekalibrovány pomocí kvadratického splajnu. To byla opět další novinka pro OCR systém a umožnila Tesseractu zvládat stránky s křivými základními hladinami, jež se běžně vyskytují při skenování a u knižních vazeb.

Hladiny jsou upraveny na základě rozdělení bloků na skupiny s přiměřeně kontinuálním posunem vůči původní přímé hladině. Kvadratická splajna je použita na největší oblast. Předpokládá se, že se jedná o základní hladinu. Kvadratická splajna má tu výhodu, že její výpočet je relativně stabilní, nicméně můžou se objevit nespojitosti při požadavku na více splajn segmentů. Více tradiční kubický splajn může fungovat lépe.

Obrázek 5.3: Příklad přizpůsobených základních hladin v textu [4]

Obrázek 5.3 ukazuje řádek s textem s přizpůsobenými základními hladinami dolní, střední a horní linie. Všechny tyto linie jsou "paralelní", y je konstantní ale mírně křivá.

5.2.4 Detekce neproporcionálního písma

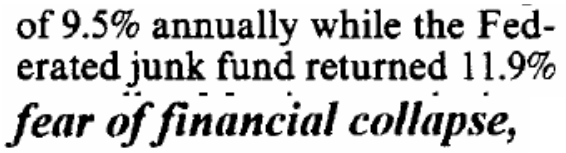
Tesseract testuje řádky s textem a vyhodnocuje zda se jedná o neproporcionální text. Tesseract pak jednotlivá slova rozdělí na znaky pomocí mezer a ty poté předá na rozeznání slov. Obrázek 5.4 ukazuje typický příklad neproporcionálního písma.

Obrázek 5.4: Neproporcionální slovo a detekované znaky [4]

5.2.5 Detekce proporcionálního písma

Jedná se o velice netriviální případ. Obrázek 5.5 zobrazuje typické problémy. Mezera mezi desítkami a jednotkami *11.9%* má podobnou velikost jako obecná mezera a

je jistě větší než prostor mezi *erated* a *junk*. Mezi ohraničujícími rámečky *of* a *financial* není žádná vodorovná mezera. Tesseract většinu těchto problémů řeší tak, že porovnává mezery v omezeném vertikálním rozsahu mezi základní linií a střední linií. Mezery, které se blíží prahu v tomto stádiu jsou konvertovány na fuzzy mezery, takže konečné rozhodnutí může být provedeno po rozpoznání slov.



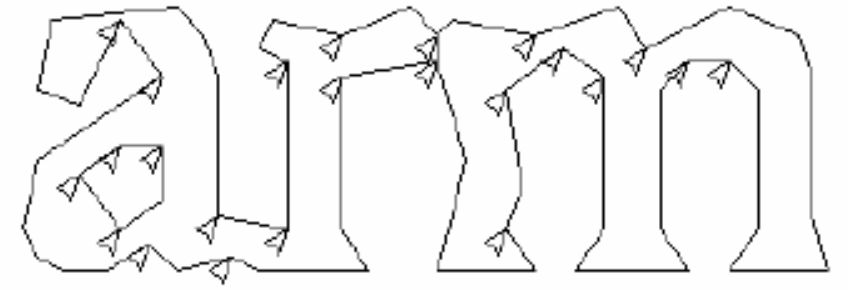
Obrázek 5.5: Složité mezery mezi slovy [4]

5.2.6 Detekce slov

Částí rozeznávacího procesu pro jakýkoliv rozpoznávací nástroj je identifikace, jak by mělo být slovo rozděleno na jednotlivé znaky. Prvotní výstup segmentace z hledání řádků je klasifikován jako první. Zbytek kroku rozpoznávání slov platí pouze pro texty, které nejsou neproporcionální.

5.2.7 Rozdělení spojených znaků

Tesseract se snaží rozdělit blok s nejhorší konfidencí ze znakového klasifikátoru k zlepšení výsledku. Kandidátní body jsou nalezeny z konkávních vrcholů polygonální aproximace obrysů a můžou mít buď jiný konkávní vrchol naproti nebo úsečku. K rozdělení spojených znaků z ASCII množiny můžou být použity až tři páry rozdělovacích bodů. Obrázek 5.6 znázorňuje šipkami množinu možných bodů. Rozdělování je děláno na základě priority. Každé oříznutí, jež nezlepší konfidenci výsledku, je vráceno zpět, ale tato informace není úplně odstraněna, takže se tento bod dá využít později asociátorem.



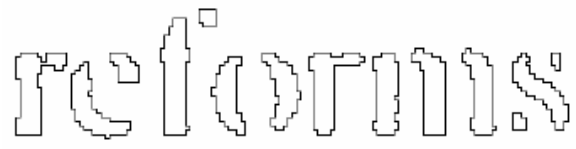
Obrázek 5.6: Kandidátní body pro rozdělení složeného textu [4]

5.2.8 Asociace poničených znaků

Když jsou potenciální rozdělovací body vyčerpány a slovo stále není dosti dobré, je předáno *asociátorovi*. Ten používá algoritmus uspořádaného prohledávání segmentovaného grafu možných kombinací maximálně rozdělených slov do kandidátních znaků. To se provádí za běhu, bez vytváření takového segmentovaného grafu díky uchovávání hashovací tabulky s navštívenými stavy. A* prohledávání[2] pokračuje vytahováním nových kandidátních stavů z prioritní fronty a vyhodnocuje je klasifikováním neklasifikovaných kombinací fragmentů.

Někdo by mohl navrhnout, že tento přístup s prvotním kompletním rozdělením a následnou asociací je přinejlepším neefektivní. V nejhorším případě by mohl vynechat důležité rozdělovací body. Výhodou tedy tohoto přístupu je, že zjednodušuje datové struktury, které by byly nutné pro udržení úplného segmentovaného grafu.

Roku 1989, když byl algoritmus uspořádaného prohledávání segmentace implementován, byla přesnost Tesseractu na poničených znacích daleko před konkurencí. Obrázek 5.7 je typickým příkladem, důležitou částí tohoto úspěchu je to, že znakový klasifikátor dokázal snadno detekovat tyto rozbité znaky.



Obrázek 5.7: Snadno rozeznatelné slovo [4]

5.2.9 Klasifikace

Klasifikace probíhá jako dvouúrovňový proces. V prvním kroku se vytvoří užší seznam znaků, které by se mohly shodovat s neznámým znakem. Každý příznak dostane z vyhledávací tabulky bitový vektor tříd, který by mohl odpovídat. Bitové vektory se sečtou přes všechny příznaky. Třídy s největším počtem se dostanou na užší seznam pro další zpracování.

Každý příznak neznámého znaku vyhledá bitový vektor prototypu dané třídy, jež by mohl pasovat. Následně je mezi nimi vypočítána reálná podobnost. Každá znaková prototypová třída je reprezentována logickým součtem produktu, kde každý term je nazýván konfigurací, takže výpočetní proces vzdálenosti si uchovává záznam o celkové podobnosti všech příznaků v každé konfiguraci stejně tak i prototypu. Nejlepší kombinovaná vzdálenost, která je vypočtena na základě součtu příznaků a prototypů, je celkově nejlepší uložená konfigurace třídy.

5.2.10 Trénovací data

Jelikož klasifikátor je schopen lehce rozeznat poničené znaky, tak klasifikátor nebyl trénován na takových znacích. Ve skutečnosti byl klasifikátor učen na pouhých 20

[2] A* prohledávání - Uspořádané prohledávání

vzorcích z 94 znaků z 8 fontů v jedné velikosti a ve stylu kurzívy, tučné kurzívy, normálním a tučném textu, což činí 60 160 trénovacích dat. To je výrazně méně než u jiných publikovaných klasifikátorů, jako je třeba Calerův s více než milionem vzorků a Bairdův 100 fontový klasifikátor s 1 175 000 vzorky.

5.2.11 Lingvistická analýza

Tesseract obsahuje poměrně málo lingvistické analýzy. Kdykoliv modul na rozeznání slov uvažuje nad novou segmentací, vybere lingvistický modul nejvhodněji odpovídající slovo v těchto kategoriích: Nejvíce frekventované, nejlepší slovo ve slovníku, nejlepší numerické slovo, nejlepší slovo s velkými písmeny, nejlepší slovo s malými písmeny, nejlepší slovo podle klasifikátoru. Konečné rozhodnutí pro danou segmentaci je jednoduše slovo s nejmenší vzdáleností, kde každá z výše uvedených kategorií má jinou váhu.

Slova z různých segmentací mohou mít rozdílnou délku. Je složité porovnávat taková slova přímo. Tento problém řeší Tesseract generováním dvou čísel pro každý znak klasifikace. První číslo z nich nazývaným konfidence je mínus normalizovaná vzdálenost od prototypu. To má za následek vytvoření opravdové *"konfidence"* ve smyslu, že čím větší jsou čísla tím lepší, ale stále se jedná o vzdálenost, protože čím dál je číslo větší od nuly, tím je větší vzdálenost. Druhé číslo se nazývá hodnocení a násobí se jím vzdálenost normalizované vzdálenosti od prototypu celkovou délkou obrysu z neznámého znaku. Hodnoty pro znak v rámci slova lze smysluplně sčítat, protože celková délka obrysu všech znaků v rámci slova je vždycky stejná.

5.2.12 Adaptivní klasifikátor

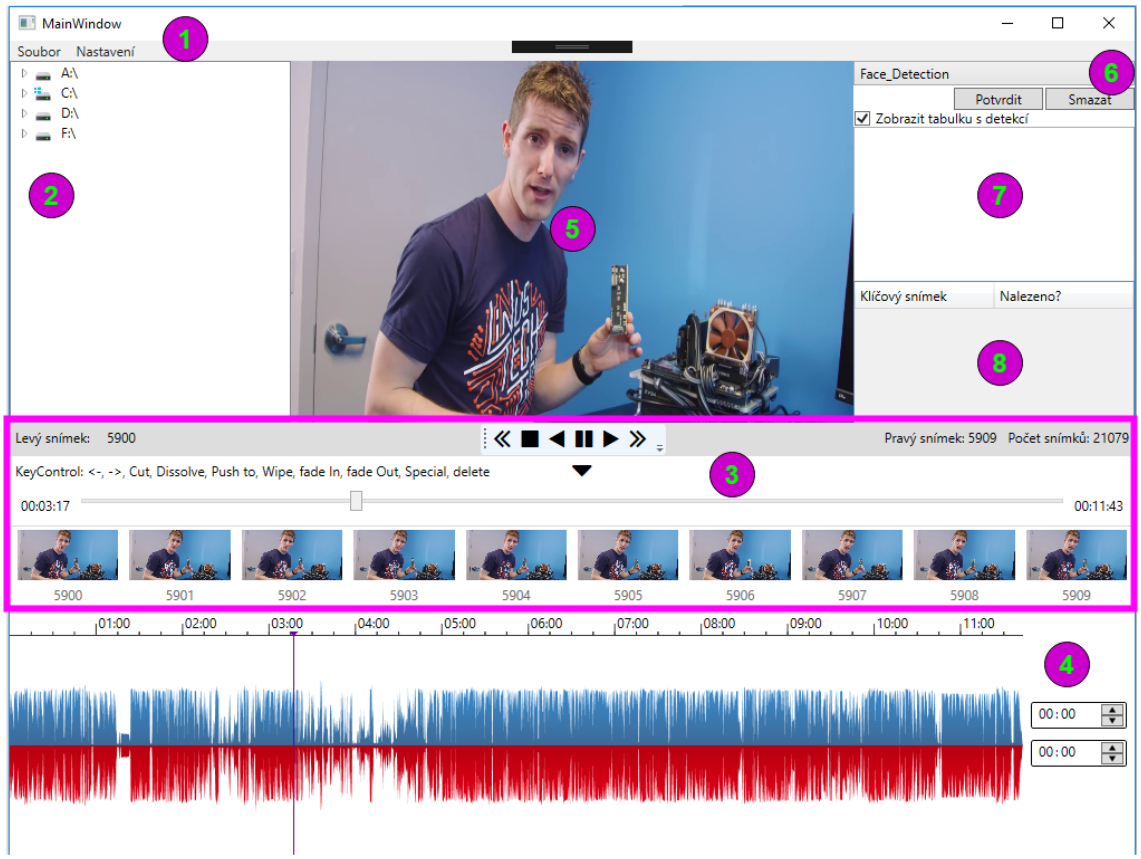
Bylo navrhnuto a demonstrováno, že OCR stroje dokáží těžit z použití adaptivního klasifikátoru. Protože statický klasifikátor musí být správně abstraktní na jakýkoliv typ písma, jsou jeho schopnosti rozlišovat mezi různými znaky nebo mezi znaky a symboly oslabeny. Adaptivní klasifikátor, jež je více přizpůsobivý fontům, je trénován na základě výstupu ze statického klasifikátoru. Je běžně používán k získání větší různorodosti v rámci každého dokumentu, kde je počet fontů omezen.

Tesseract nepoužívá šablonový klasifikátor, ale používá stejné příznaky a klasifikátor jako statický klasifikátor. Jediný významný rozdíl mezi statickým klasifikátorem a adaptivním, kromě trénovacích dat je to, že adaptivní klasifikátor používá izotropní x-výškovou/základní hladinovou normalizaci, zatímco statický klasifikátor normalizuje znaky centroidem, osa y (první moment) pro polohu a osa x (druhý moment) pro normalizaci anizotropní velikosti.

Obrázek 5.8: Základní hladina u textu a normalizované znaky [4]

Výšková normalizace na ose x ulehčuje rozlišení mezi velkými a malými znaky a stejně tak zlepšuje odolnost vůči šumu. Hlavní výhodou normalizací centroidem je odstranění poměru stran písma a do určité míry šířky tahu písma. Také to ulehčuje rozeznání horních a dolních indexů u slov. Současně to vyžaduje dodatečnou funkci klasifikátoru pro rozlišení některých horních a dolních znaků. Obrázek 5.8 ukazuje příklad třech znaků s x-výškovou normalizací a centroidovou.

6 Navržená a realizovaná aplikace



Obrázek 6.1: Realizace aplikace s načtenou videonahrávkou

Výsledný postavený systém na základě informací z kapitoly 7 je zobrazen na obrázku 6.1. Tento program se dá popsat pomocí 8 nezávislých celků, nakreslených v obraze, jež jsou:

1. Soubor a nastavení:
    - Soubor - obsahuje operace pro načtení videonahrávky, obličeje nebo již vytvořené segmentace.
    - Nastavení - jazykového prostředí, počtu zobrazovaných obrázků.

2. Průzkumník souborů - automaticky otevírá po kliknutí soubory CSV, XML a MP4.

3. Extrahované snímky a segmentace - udává přehled o nejkrajnějším pravém a levém indexu snímku, celkovém počtu snímků ve videonahrávce a momentálním čase v nahrávce. Případný výskyt přechodu mezi záběry je zvýrazněn.

4. Vizualizace audio stopy - lze se pohybovat v audio stopě výběrem myší nebo příslušných ovládacích prvků. Vybraná oblast se poté může nepřetržitě přehrávat dokola.

5. Vybraný extrahovaný snímek - představuje snímek na který ukazuje šipka v 3. nezávislém celku. Zjednodušeně se jedná o prostřední snímek mezi extrahovanými (zaokrouhleno dolů).

6. Detekce objektů - textu, tváří a identifikace osob na klíčových snímcích.

7. Výsledek detekcí - slouží pro výpis informací pro operace detekce textu a tváří.

8. Výsledek identifikace - slouží pro výpis informací pro operaci identifikace osob.

7 Implementace aplikace

Implementace aplikace probíhala v nativním vývojářském studiu pro C#, Visual Studio 2017, za doprovodu *NuGet*[1] pro správu balíčků a případného rychlejšího a bezproblémového nasazení aplikace. Složitější algoritmy typu detekce obličeje jsou prováděny pomocí Python skriptů s knihovnou OpenCV, Dlib a spouštěny jako nový proces uvnitř C# za pomoci PyInstaller[2]. Detekce textu je zase prováděna volně dostupnou knihovnou Tesseract OCR a tentokrát se jedná o Wrapper[3] a funkcionality jsou volány přímo zevnitř prostředí, čímž se celý ekosystém stále drží v prostředí C#, což má nespočetné výhody. Nejvíce lukrativní kapitoly budou pravděpodobně 7.6 a 7.8, jelikož se jedná o nejsložitější část tohoto projektu a bylo jim věnováno nejvíce času.

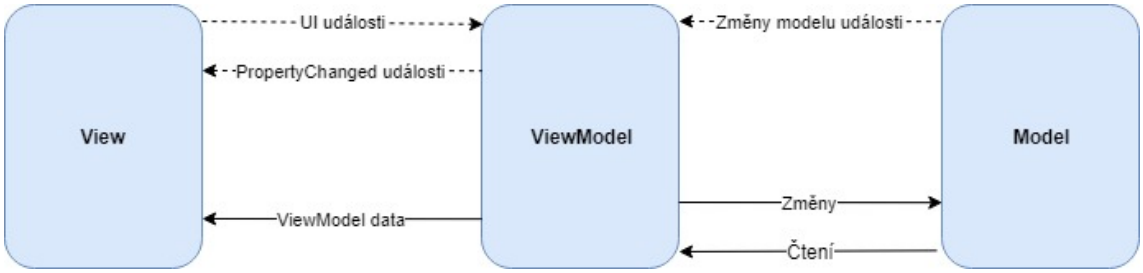
7.1 Návrhový vzor MVVM

Aplikace byla navrhována architekturou MVVM (Model–View–ViewModel) se snahou co nejvíce se držet tohoto návrhového vzoru a nejlepším praktikám pro daný jazyk. To znamená, View obsahuje pouze vzhled a žádný kód na pozadí. ViewModel pouze zaštiťuje komunikaci mezi Modelem a View. Model jsou třídy s daty. Tento návrhový vzor ovšem v praxi není tak jednoduchý, jak se na první pohled může zdát. Otázkou je, kam vložit logiku do celé aplikace. Některé diskuze vedou spíše směrem k Modelu a jiné zase naopak k ViewModelu. Existuje i separátní skupina jež stojí za názorem další speciální třídy mezi Modelem a ViewModelem s názvem Services.

Tento systém původně obsahoval logiku ve ViewModelu, ale v jedné fázi byl přesunut do Modelové části kvůli přehlednosti. Na obrázku 7.1 lze tedy vidět jednotlivé komponenty jak mezi sebou komunikují. Některé části jako klávesové zkratky jsou řešeny ve View na pozadí, jelikož implementace na straně ViewModelu by byla složitější a méně přehledná. Komunikace mezi View a ViewModelem je zastoupena commandy[4] a bindingem[5]. Každý Model ve kterém se za běhu mění jeho data dědí od třídy BaseModel, která má implementováno rozhraní INotifyPropertyChanged, jež se stará o notifikaci změn dat. Konkrétní ViewModel, jež obsahuje dotyčné Modely,

[1] Nuget – open source balíčkovací systém pro Microsoft ekosystém
[2] PyInstaller – rozšíření pro Python, umožnuje vytvářet spustitelné soubory pro Windows
[3] wrapper – zabaluje funkce originální knihovny a zpřístupňuje je v nativním prostředí
[4] Command – zjednodušeně se jedná o MVVM event
[5] Binding – propojení View s Viewmodelem a jeho daty

poté reaguje na tyto změny. Naprosto každý ViewModel dědí od BaseViewModel s implementovaným INotifyPropertyChanged, jelikož každý View se svým propojeným ViewModelem musí vědět, kdy se mění data.



Obrázek 7.1: Návrhový vzor MVVM

7.2 Uživatelské rozhraní

7.2.1 Průzkumník souborů

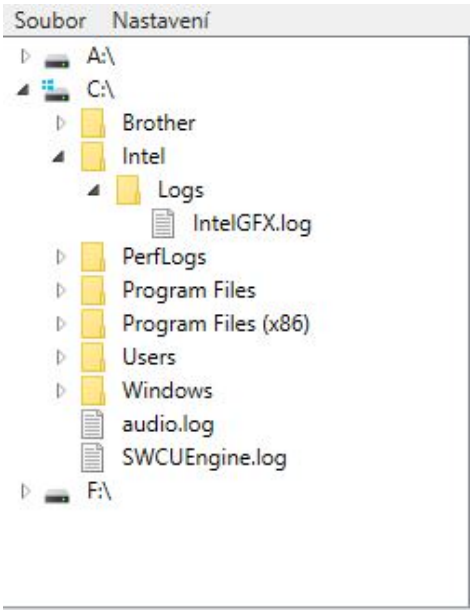
K ulehčení navigace je implementován základní průzkumník souborů, který po kliknutí na validní typ podporovaného souboru jako CSV, XML dokáže reagovat. Tato komponenta není přímo k nalezení mezi ovládacími prvky pro WPF (Windows Presentation Foundation) a vyskytují se na výběr v podstatě tři možnosti:

1. Windows API Code Pack
2. Placené knihovny
3. WebBrowser ovládací prvek
4. Vlastní řešení

Zaprvé se dá použít rozhraní od firmy Microsoft, nicméně toto je verze pouze pro WinForms a na WPF aplikace neexistuje. V případě, že bychom chtěli použít toto řešení, museli bychom element v prostředí WPF takzvaně hostovat. To je z hlediska budoucnosti velice riskantní, jelikož do projektu přinášíme věci, které jsou zastaralé. Navíc Microsoft toto API (Application programming interface) stáhl bez vyjádření a přestal nadále vyvíjet. Jsou ovšem dohledatelná na GitHubu v rámci třetích stran.

Další řešení je často poměrně drahé a nikdy se nevyskytuje osamocené. Je tím míněno to, že je potřeba si koupit celý balíček dalších komponent, které nebudou využity. V době vytváření aplikace nebyla zjištěna volně dostupná knihovna.

Z názvu ovládacího prvku WebBrowser je patrné, že se nejedná o správné řešení. Nicméně je možné tuto komponentu přesvědčit, aby zobrazovala místo obsahu stránky stromovou hierarchii systému. Je možné se pohybovat ve stromové struktuře systému, avšak nemožné reagovat jakkoliv na uživatelovo kliknutí



Obrázek 7.2: Vlastní průzkumník souborů

událostí případně Commandem. WebBrowser má vyloženě prvky pro práci s HTML dokumentem.

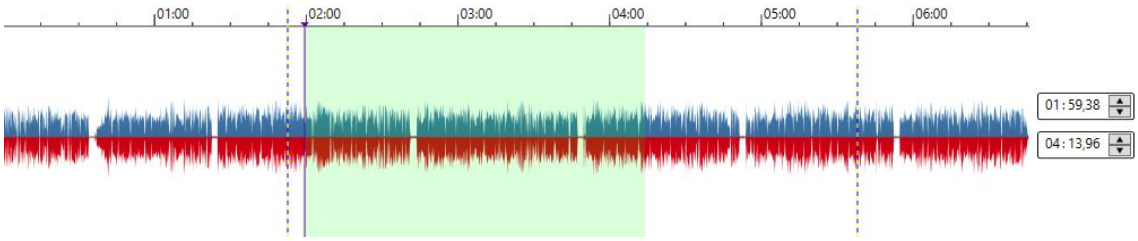
Poslední varianta je relativně časově náročná na implementaci a pochopení. Toto je zvolená varianta v tomto konkrétním systému. Jedná se tedy o vlastní ovládací prvek, jehož základem je TreeView komponenta. TreeView jako komponenta sama o sobě nic moc neumí, protože je velice abstraktní, to je daň za všestrannost. Má změněný zdroj dat a bere data ze svého příslušného Modelu BaseObject. Ovládací prvek reaguje na expandování jednotlivých uložišť neboli TreeViewItems díky třídě FileSystemObjectInfo a generuje jejich případné potomky k zobrazení. Ikonky bere z dynamické Windows knihovny shell32.dll. Jak už bylo řečeno TreeView toho z hlediska implementace obsahuje pramálo, proto bylo potřeba ještě vytvořit DependencyProperty SelectedPathProperty, která bude zpřístupňovat momentálně vybranou cestu uživatelem v podobě vlastnosti ovládacího prvku navenek. Díky tomu lze poté sledovat ve ViewModelu na co uživatel kliknul a patřičně reagovat. Na obrázku 7.2 je vidět vlastní implementace komponenty na základě těchto informací.

7.2.2 WPFSVL

Jedná se o kolekci WPF ovládacích prvků pro analýzu a zpracování zvukových signálů. Pomocí této volně dostupné knihovny například lze sestavit svůj vlastní zvukový přehrávač. Knihovna obsahuje analyzátor spektra, časovou osu zvuku, ekvalizér a selektor času pro časovou osu. Ke knihovně jsou dodávány zdrojové kódy k testování s populárními zvukovými knihovnami založenými na BASS.NET, či NAudio, nebo jakékoliv jiné zvukové knihovny, ale ke zmiňovaným dvěma je sestaven návod jak je zprovoznit.

Tato aplikace využívá pouze časovou osu zvuku a k ní selektor času. Lze tedy

vybírat zvukovou pasáž buď definováním času v komponentách, nebo používat myš k výběru případně kombinací předchozích možností.



Obrázek 7.3: Zvuková stopa nahrávky

7.2.3 GridExtra

Další z použitých knihoven se soustředí na rozšíření samotných panelů jako je Grid, Wrap panel a tak dále. V projektu byla tato knihovna použita v brzkých začátcích vývoje uživatelského rozhraní, neboť umožňuje programátorovi ulehčit práci s Gridem samotným tím, že zobrazuje definice sloupců a řádků, a to i při běhu. Při vývoji sice toto Visual Studio také podporuje, nicméně při odlaďování aplikace tyto linie zmizí. Obsahuje také šablonovací systém, pro představu co jaký sektor bude obsahovat.

Hlavní přednosti ovšem leží naprosto jinde. Tkví v Responsive Gridu, který je naprosto identický k Bootstrapu pro vývoj webu akorát ve verzi pro XAML. Opět se definují sloupce pro jednotlivé velikosti zařízení typu extrémně malé, malé, střední a velké. Elementy v tomto prvku se tedy poté smršťují, případně roztahují, v závislosti co se děje s velikostí okna.

7.2.4 Extended WPF Toolkit

Tato knihovna byla použita pouze z jednoho důvodu, a tak je její potenciál nevyužit jakožto knihovny zaměřené na velké množství ovládacích prvků. Z nějakého důvodu WPF neadoptovalo jednu z velice užitečných komponent na předávání a validaci celočíselných dat NumericUpDown a to je důvod importu této knihovny. K vyřešení vedou samozřejmě další tři alternativní cesty, které byly v rámci vývoje tohoto systému zváženy:

1. Vlastní UserControl a implementace
2. Knihovna třetích stran
3. Hostování WinForm komponenty uvnitř prostředí WPF

Ovládací prvek by nebyl problém vytvořit od začátku, jelikož existují velké množství návodů, jak docílit základní funkcionality.

Knihovna třetích stran je zdánlivě nejspolehlivějším a nejrychlejším řešením, neboť vývojář k funkcionalitě přidává většinou ještě vzhled a extra rozšíření, které původní komponenta neobsahovala.

Posledním řešením je naimportování reference WindowsFormsIntegration a System.Windows.Forms. Po tomto kroku lze hostovat ovládací prvky WinForms uvnitř WPF. Realizace je jednoduchá a funkční. Otázkou však zůstává, zda-li se jedná o optimální řešení. Použitím této metody začínají vznikat jisté problémy, které mohou ale nemusí celou aplikaci afektovat. Velké množství těchto problémů je sepsána na stránce MSDN (Microsoft Developer Network). Závažnějším problémem je ale to, že by tato integrace nemusela být podporována v budoucích verzích .Net frameworku. Z hlediska míry rizika jsou tedy první dvě cesty implementace nejlepší.

7.2.5 FontAwesome

Jedná se o hojně používanou knihovnu v oblasti webových aplikací zaměřenou na různé ikonky, jejichž licence je otevřená pro vývojáře. Například na obrázku 7.4 je celý prostřední panel s řídícími prvky videonahrávky řešen právě touto knihovnou. Jedná se o Wrapper pro C# a tudíž je práce s touto knihovnou v XAML velice příjemná.

7.2.6 Zvýraznění přechodů v nahrávce

Dalším požadavkem na systém bylo, aby dokázal zobrazovat ve dvou barvách přechody v modré a červené. Bylo potřeba si uchovávat informace o všech barvách v celé video sekvenci. Řešení, které je zde znázorněno a implementováno, pravděpodobně nebude ideální. Další idea nebyla zjištěna, nicméně funguje.

K vyřešení této problematiky byly vytvořeny dvě statické proměnné SEMAFOR a COLOR_ARRAY. První z nich je typu boolean a slouží k přepínání barev mezi modrou a červenou. Následující proměnná je pole integerů, jež se vytváří vždy s načtením nového videa a na základě EmguCV a jeho metod je zjištěna jeho velikost. Tyto proměnné se mění vždy při aktivaci klávesové zkratky z kapitoly 7.2.7. Toto pole si v zásadě drží tři hodnoty a to 0,1,2. První hodnota z nich naznačuje, že se jedná o defaultní barvu neboli nic, další červená a modrá respektive.

K poli jsou přístupné ještě dvě metody, jež se starají o mazání a nastavování barev v celém jednom spektru. Jejich jediným parametrem je celočíselný vstup "od". Tyto metody pomocí cyklu procházejí celé pole barev zpět k 0 od daného indexu a jejich stopovací podmínkou je buďto nalezení existující barvy nebo konce pole.

S definovanou proměnnou pole barev bylo potřeba propojit komunikaci s již existující segmentací videonahrávky v ovládacím prvku ListView. Jelikož má tento ovládací prvek již přetypovanou ItemsControl.ItemTemplate bylo potřeba nastavovat styl u ItemsControl.ItemContainerStyle, konkrétněji Setter nastavoval styl vlastnosti pozadí. Hodnota, která se bindovala, byla index obrázku ze segmentace a k ní konvertor NumberToColorConverter, jež na základě hodnoty v poli vracel proměnnou typu Brushes, neboli barvu.

Jak už bylo řečeno v momentě, kdy něco popisuje náhodné číslo, tak se jedná z hlediska čitelnosti o nepřehledný kód. Mírným zlepšením by mohlo být použití enum hodnot k reprezentaci barev.

7.2.7 Klávesové zkratky

Aby byl program uživatelsky přívětivý, bylo rozhodnuto, že se hlavní část programu tj. segmentace bude dát ovládat bez použití myši. Definice tohoto chování se v zásadě nachází na dvou místech a to v kódu hlavního okna a jeho kódu na pozadí. Důvod umístění logiky zde je ten, že podle definice by mělo být vše ve ViewModelu, nicméně taková implementace je složitá, i když by měla být podle .Net standardu správná. Níže je uveden přehled všech klávesových zkratek v programu:

- V hlavním okně
  - CTRL+O - otevřít videonahrávku
  - CTRL+S - uložit momentální anotaci videonahrávky
  - CTRL+L - načíst abstraktní anotaci
- V logice pozadí hlavního okna
  - C,D,P,W,I,O,S - začáteční písmena pro anglické přechody označující přechody snímků.
  - D - smaž přechod
  - Levá, pravá šipka - posun videa ve směru

Samozřejmě bylo nutné nezapomenout, že klávesové šipky mají defaultně nastavené funkce dopředu a dozadu. Proto například když zůstal focus na prohlížeči souborů z kapitoly 7.2.1 a operátor použil šipku, tak místo pohybu ve videu se pohyboval v kořenové sekci TreeView. Jediné co bylo potřeba, tak nastavit událost Handled na true. Systém si tak myslel, že událost byla obsloužena.

7.2.8 WriteableBitmapEx

Aby bylo možné zpětně rekonstruovat vykreslení nalezených tváří a textů, byla použita tato knihovna, která rozšiřuje datový typ WriteableBitmap. Pakliže uděláme detekci nad obrazem a posuneme se dále v nahrávce a poté zase zpět, informace o detekci budou zaznamenány, ovšem vizuálně nikoliv a tuto informaci je třeba obnovit z již známých dat a to nejlépe co nejrychleji. To samé nastává pokud se načtou nové labely s video abstraktem.

WriteableBitmap umožňuje přímou manipulaci s bitmapou a lze ji použít pro manipulaci s obrazem. Problémem je, že tento datový typ v základu neobsahuje metody pro vykreslení např. čtverce, obdélníku. Rozhraní API WriteableBitmap je tedy velmi minimalistické a pro tyto operace existují pouze manipulace s poli obrazových bodů. Knihovna WriteableBitmapEx se snaží zaplnit tuto mezeru s doplňkovými

metodami, které jsou snadno použitelné jako vestavěné metody a nabízejí funkce podobné GDI+ (Graphics Device Interface). Knihovna rozšiřuje třídu WriteableBitmap o elementární a rychlou 2D kresbu, konverzní metody a funkce pro kombinování WriteableBitmaps dohromady. Metody rozšíření jsou seskupeny do různých tříd C# pomocí klíčového slova partial. Je možné zahrnout jen několik metod pomocí specifických souborů se zdrojovým kódem, nebo použít plnou funkčnost prostřednictvím zabudovaných binárních souborů.

7.3 ImageUtilities třída

Jelikož WPF obsahuje velké množství struktur pro uložení obrazových dat jako jsou Bitmap, BitmapImage, BitmapSource, Image, BitmapFrame atd. a každá knihovna nebo metoda pracuje s jinými strukturami. Bylo výhodné si tyto konvertory jednou napsat a poté je používat stále opakovaně. Například knihovna EmguCV vrací jeden konkrétní zachycený snímek jako typ Bitmap. Nicméně abychom mohli tento obrázek zobrazit ve View, bylo ho potřeba překonvertovat na typ BitmapSource.

Dále bylo potřeba sjednotit formát obrazových dat při konverzi. To znamená použít ideálně stejný kodér a dekodér obrazových dat jako png, jpeg nebo bmp. V tomto projektu byl použit bmp kodér a dekodér.

V neposlední řadě dalším parametrem na zvolení a udržení byl formát obrazových bodů. Jelikož tyto kódy jsou volně dostupné na internetu v hojném množství a mnohdy se pouze kopírují, může právě zde nastat zásadní problém. Tento problém se projevuje velice rychle a srozumitelně. A to sice tak, že buďto program spadne s chybovou hláškou "bad color range"a nebo budou zaměněné barevné kanály. Zde byl použit formát *Format32bppArgb*, který používá 8 bitů na každý kanál alfa, červená, modrá, zelená.

Je také velice důležité si kontrolovat parametr "Freeze"a v případě, že se data obrázku nebudou dále měnit, mu dát hodnotu true a tím ho tzv. zamrazit.

V případě že se v aplikaci vyskytují vlákna, je také nutné při konverzi definovat parametry FileAccess a FileShare podle potřeby. Velice se tak zvyšuje pravděpodobnost, že nenastane chyba.

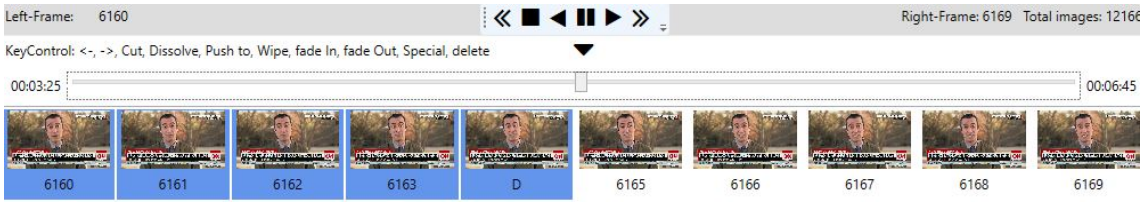
7.4 Segmentace videa

Způsoby jak extrahovat snímky z videa byly buďto moc staré, anebo nefungovaly. Knihoven ke spouštění videa jako takového je dostatek, ale extrakce jednoho konkrétního snímku je problém. Možnosti které byly brány v potaz:

1. EmguCV
2. FFmpeg(CLI)
3. FFmpeg(C# Wrapper)

EmguCV je jen jiný název pro OpenCV. Jedná se o .Net wrapper, který umožňuje volat funkce OpenCV v prostředí C#. Z knihovny je nejvíce používána třída VideoCapture, která obsahuje všechny nástroje pro čtení hlaviček souborů a posouvání se ve videu na určité pozice, a to už buďto pomocí milisekund nebo definovaného čísla snímku. Jedná-li se o videa s vysokou kvalitou, dá se pozorovat náročnost na najití konkrétního snímku při dlouhém posunu ve videu. Hledat snímek pomocí času je efektivnější, než ho hledat pomocí jeho indexu. Nevýhodou jsou občasné chyby. Například metoda pro získávání informace o snímcích za sekundu vrací jiné číslo, než které by reálně mělo být.

FFmpeg nemělo v době psaní pro .Net funkční Wrapper, takže jeho volání je o něco složitější a navíc případná distribuce koncovým uživatelům by byla problematická, protože by každý musel mít na stroji tuto knihovnu. Uvnitř C# by se musely volat procesy s danými parametry a příkazy pro práci s multimediálními soubory. Nutno dodat, že kdyby měl FFmpeg v .Net dostatečné zastoupení, tak by velice pravděpodobně dostal přednost před použitou technologií EmguCV.



Obrázek 7.4: Příklad segmentace

7.4.1 Plynulé přehrávání

Program umí přehrávat video jak klasicky tak i pozpátku. Každopádně díky knihovně EmguCV zvolené v kapitole 7.4 je přehrávání nazpátek extrémně neefektivní a náročné na výkon procesoru. Zvláště jedná-li se o kvalitní nahrávku. V programu je spíše z důvodu, že se vůbec taková věc dá udělat, ale s vybranou technologií ztrácí reálně smysl. Zkrátka tato knihovna není určena pro profesionální přehrávání videí. Nicméně pro poloautomatické zpracování videonahrávek a tuto problematiku segmentace a analýzu videonahrávek bohatě stačí. Způsoby jak celé přehrávání s EmguCV zprovoznit jsou tři a to následující:

1. DispatcherTimer
2. Timer
3. While a wait

DispatcherTimer oproti Timeru běží rovnou na UI (User interface) vlákně a není třeba se starat o komunikaci vláken na rozdíl od Timeru, který běží na novém. Jelikož je potřeba aktualizovat řadu obrázků na straně uživatelského rozhraní, je jednodušší použít první způsob. Nicméně byly vyzkoušeny všechny tři cesty a všechny s

úspěchem. První způsob byl čistě vybrán kvůli dané problematice zásahu do zobrazení, kde while cyklus byl nepřehledný. Časovače jsou spouštěné s nejvyšší možnou prioritou kvůli problematice zmíněné v kapitole 7.6.

7.4.2 Optimalizace přehrávání

V prvotních fázích, kde ještě neexistovalo plynulé přehrávání, byla data uložena do pole, kde se vždy všechny prvky kopírovaly na další nebo předchozí pozici podle směru pohybu ve videu. Ve chvíli, kdy se v nahrávce pohybovalo ručně pomocí šipek a příslušných událostí, byl tento způsob naprosto dostačující bez známek na výkonu. Později se však ukázala nedokonalost řešení při úpravě na plynulé přehrávání, protože například při 30 snímcích za sekundu se musí posouvat celé pole za zhruba 33 ms. Toho zkrátka nebylo možné docílit a byla pozorována samotná náročnost na uživatelském rozhraní, kde se vše sekalo. Celý problém byl tedy vyřešen kolekcí, která se chová jako hybridní fronta, kdy se mění vždy poslední nebo první položka podle potřeby. Po tomto zásahu bylo na celé aplikaci vidět, že změna datové struktury výrazně ulehčila časovou náročnost.

7.4.3 Únik v nespravované paměti

Při dlouhém procházení videonahrávek a testování stability programu, byla zjištěna bobtnající paměť, která po pár minutách vyvolala výjimku nedostatek paměti. Při testování kvalitnějších nahrávek tato chyba nastala rychleji a bylo jasné, že se správně neuvolňuje paměť a to konkrétně u bitmap. K tomu garbage collector běžel každých 0,5 sekund, a to mělo taktéž výrazný dopad na celý výkon. Ačkoliv GC (Garbage collector) běžel, tak nedokázal uvolňovat prostředky. Jak samotné Visual Studio tak dodatečně diagnostické nástroje ReSharperu[6] hlásily únik v nespravované paměti. Bylo vyzkoušeno proměnné po dokončení jejich potřeby nullovat a volat metody dispose, nicméně ani to nevedlo k řešení.

Nakonec v dokumentaci je v posledním odstavci zmíněno, že veškerá správa paměti bitmap je na programátorovi a byl uveden správný příklad uvolnění. Takže jediné, co bylo potřeba udělat, bylo zabalit kód, kde se pracuje s bitmapou do using s try a finally. Při dokončení práce s bitmapou nastane blok finally a smaže se ukazatel na bitmapu. Na obrázku 7.5 je vidět správný postup práce s bitmapou.

7.4.4 Export a import označení snímků

Program plně podporuje export a import dat o snímcích z formátů CSV a XML. Jedná se o model, který je plně flexibilní a při zachování interních datových struktur lze snadno podporovat další formáty souborů. Interní datové struktury jsou dvě, a to jednoduché pole znaků pro segmentaci (přechody) a kolekce obsahující model, který zaštiťuje informace o detekovaném textu, obličejích a jejich identifikace, které se vytváří vždy při načtení nové videonahrávky a pomocí EmguCV a třídy VideoCapture je zjištěna potřebná velikost na základě počtu snímků.

[6] ReSharper – placené rozšíření do Visual Studia

```
1  [DllImport("gdi32.dll")]
2  private static extern bool DeleteObject(IntPtr hObject);
3  using(Bitmap bmp = new Bitmap()) {
4    IntPtr hBitmap = bmp.GetHbitmap();
5    try {
6      //...
7    } finally {
8      DeleteObject(hBitmap);
9    }
10 }
```

Obrázek 7.5: Uvolnění bitmapy

Například pokud bude označen snímek s číslem pět jako střih, pak v interní datové struktuře bude na čtvrté pozici znak "C", podle anglického označení cut. Tato struktura se potom dále dá uložit do čitelného XML, případně do textového souboru, kde jedné hodnotě náleží jeden řádek. V tomto řešení parser CSV pracuje pouze s polem znaků a ty ukládá. XML parser je pokročilejší a pracuje s oběma interními strukturami a jedná se o doporučený postup k tvorbě video abstraktů.

Při exportu se počítá s nepsaným pravidlem, že první snímek je vždy střih a poslední průnik.

7.5 Přehrávání zvuku

Příliš knihoven zabývající se touto problematikou nebylo nalezeno. Některé mají poměrně restriktivní licence, i když jsou zadarmo. A zase na druhou stranu, některé z nich měly poslední aktualizace v rámci roku, což určitě není také žádoucí. Každopádně v projektu byly vyzkoušeny dvě velice vychvalované a oblíbené knihovny online komunitou BASS.NET a NAudio. Jejich výběr byl ovlivněn také zvolenou audiovizuální knihovnou WPFSVL zmíněnou v kapitole 7.2.2.

Zkoušené NAudio k přehrávání zvuku fungovalo výborně, nicméně problém nastal při vizualizaci dat, kdy v přiloženém příkladu nastával problém u určitých zvukových formátů, které by měly fungovat podle dokumentace k chybě o neplatném formátu, a to se jednalo o mp4. Problém nebyl nadále zkoumán a místo toho byla odzkoušena konkurenční knihovna, zda se vyskytne stejná chyba či nikoliv.

Konkurent zvládl přehrát zvukový soubor a následně i zobrazit bez jakékoliv chyby. BASS.NET má tu výhodu, že podporuje základní formáty a v případě, že je potřeba podporovat jiný formát typu FLAC, lze si ho snadno stáhnout jako samostatnou knihovnu .dll a tu importovat uvnitř programu. Nevýhodou je ovšem licence kde NAudio je open-source tak zde záleží na tom, zda je program výdělečný a pro osobní použití či nikoliv. Zobrazující se plovoucí logo lze odstranit registrací na stránce vydavatele, ale to přináší opět problémy, jak importovat registrovací token bez zneužití třetích stran. Je třeba si zapnout manuální konstruktor v třídě App a ještě před spuštěním programu přidat jedné z registračních metod BASS.NET váš klíč a email.

7.6 Synchronizace zvuku a videa

Přidáním přehrávání zvuku do systému se projevilo velice negativně s mnoha problémy. Na jedné straně stála nezávislá knihovna EmguCV, jejíž jedinou doménou byla extrakce snímků z nahrávky a na druhé zase BASS.NET, který se zajímá pouze o zvukovou část. Doplněno o nepřesné časovače prostředí .Net se jednalo o velkou překážku. U časovačů obecně je problematika, kdy nastane přerušení a vyvolání události. Podle dokumentace MSDN je dáno, že časovače se nespustí před daným intervalem, ale po intervalu může nastat jejich událost teoreticky kdykoliv s určitým zpožděním. Problémem tedy bylo, jak donutit tyto dva samostatné moduly komunikovat. Pakliže se spustilo přehrávání zvuku a obrazu zároveň v určitý okamžik se začaly obě stopy předbíhat a to zvuková tu obrazovou. Při odlaďování celého systému, bylo prakticky ověřeno, že časovače se opravdu před definovaným intervalem nespustí. Nicméně se spouštěly s náhodným zpožděním, které nebylo možné predikovat.

První myšlenkou bylo zrychlit konstantně přehrávání videa, ale jediné čeho bylo docíleno, bylo zpoždění, kdy zvuková stopa předběhne obrazovou. Druhá myšlenka vycházela z masivní diagnostiky dat, kterou lze vidět v samotném kódu. Bylo sledováno, jak dlouho trvala operace časovače a následně upraven interval tohoto časovače. Aby se spustil dříve či později na základě zjištěné hodnoty doby trvání. Tento přístup byl nefunkční, ačkoliv na papíře vypadal slibně. Řešením nakonec bylo sledování hodnot v milisekundách jak přehrávání obrazu tak zvuku a při události skoro předběhnutí artificiálně zrychlit časovač pro dodání dalšího obrazu z videa o jakousi definovanou konstantu. Frekvence spouštění této výjimky závisí výhradně na kvalitě videa, čím kvalitnější nahrávka tím častěji nastává a naopak. Jelikož se jedná o velice malé zrychlení, tak není rozpoznatelné.

7.7 Nastavení aplikace

Aplikace v sobě uchovává možnost individuálních nastavení pro jednotlivé uživatele. Lze v ní měnit jazyk, počet segmentů, ve kterých se budou zobrazovat jednotlivé snímky a v neposlední řadě od volby jazykového prostředí se mění i datová sada pro detekci textu knihovnou Tesseract. Data o nastavení si uchovává aplikace ve svém interním uložišti a na každém klientském stroji bude rozdílná.

7.7.1 Vícejazyčnost

Vícejazyčnost je prováděna dynamicky za běhu programu. Toho je docíleno bindováním a uchováváním singletonu o aktuálním jazykovém rozložení a objektu resources s aktuálními daty pro daný jazyk, při změně jazyka se vyvolává událost, která mění resource pro danou kulturu. Se změnou jazykové kultury se každý dotčený prvek změní. Při zachování interní abstraktní struktury defaultního resource lze podporovat další jazyky. Momentálně aplikace rozeznává češtinu a angličtinu.

7.8 Integrace pythonu

Jelikož detekce obličejů probíhá za pomoci knihovny Dlib a OpenCV, které jsou napsány pro Python, bylo potřeba integrovat do celého systému komunikaci s jazykem Python. Pro řešení této problematiky byly prozkoumány čtyři následující cesty:

1. IronPython
2. Pythonnet
3. Volat skript přes Python na klientském stroji
4. Volat .exe s Python kódem vytvořeným přes PyInstaller

IronPython je řešením přímo od Microsoftu. Výhodou je, že .Net je úzce spjat s Pythonem a lze psát Python kód uvnitř C# například. Nevýhodou je však, že díky hluboké integraci je afektován celý vývoj. IronPython je v momentální době k dispozici pouze s Pythonem ve verzi 2.7, což se jeví jako velice nedostačující, neboť tato verze brzy ztratí podporu a je zastaralá. Navíc IronPython má problém, že nepodporuje CPython a má tedy problémy s C++ knihovnami typu Numpy, Dlib. Python 2.7 vyšel naposledy v roce 2010. Existuje experimentální verze na GitHubu s novějším Pythonem 3.X, avšak ten se stále nachází v experimentálním vývoji a nelze předpovídat jeho chování a vývoj.

Pythonnet by byl býval použitou technologií nebýt pracného zprovoznění. Koncový uživatel musí na svém stroji mít nainstalovaný Python a je jedno v jaké verzi. Problémem však je více verzí na cílovém stroji. V projektu C# je tedy potřeba naimportovat správné cesty k Pythonu a jeho příslušným knihovnám. V tomto řešení se toto nepovedlo a nejednalo se o ojedinělý neúspěch. Na GitHubu se běžně vyskytují tyto problémy se zprovozněním. Někomu rady pomohou někomu ne. Byly dokonce vykonány následné reinstalace celých systémů, aby tato knihovna fungovala. Nutno podotknout, že s čistou instalací a jedním interpretem Pythonu na cílovém stroji se dotyčnému podařilo tuto knihovnu zprovoznit. Funkcionálně je to velice podobné IronPythonu s tím rozdílem, že každá proměnná a výsledek funkce z Pythonu volaný uvnitř C# je typu dynamic. Výrazně by se tedy ulehčila práce s následným zpracováním dat i případným odlaďováním. Dále Pythonnet podporuje většinu vědeckých a matematických knihoven díky CPythonu jako Numpy, Scipy, Pandas, Dlib.

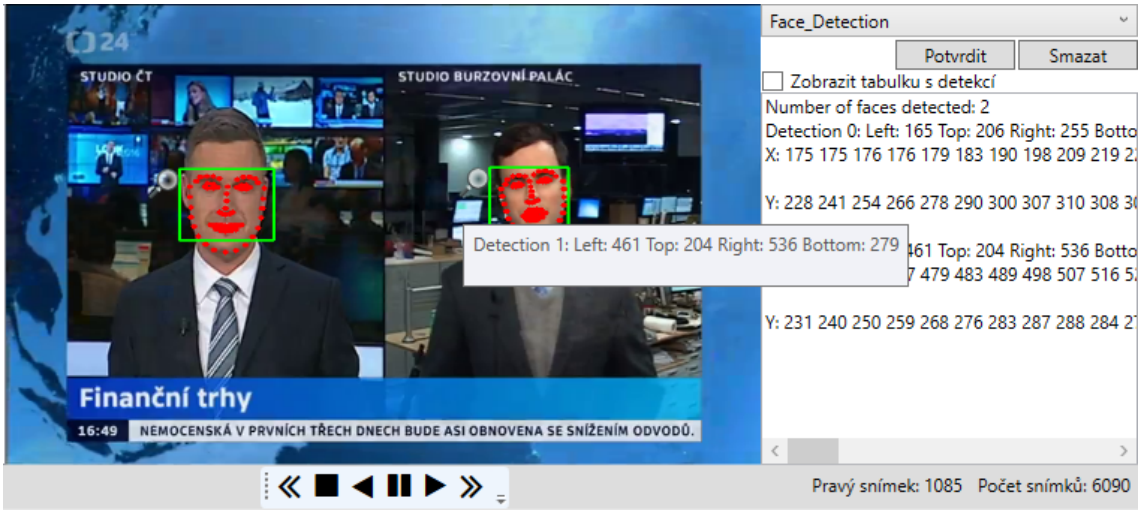
S dvěma nefunkčními přístupy přišly poslední dva nápady. Spouštět nový proces přes příkazovou řádku s parametry. Šlo tedy vlastně o to, napsat skript v Pythonu, zavolat nový proces s parametry Python, název skriptu, argumenty skriptu. Tento přístup je naprosto validní a funkční, nicméně sdílel stejnou problematiku jako Pythonnet. Koncový uživatel musí mít na stroji distribuci Pythonu plus ještě všechny dané knihovny, který daný skript používal. Distribuovat dále takové řešení by bylo složité nebo jedině v Dockeru.

Poslední vyzkoušená cesta použitá v tomto řešení vychází z předešlé problematiky a odstraňuje otázku přenositelnosti. Napsané Python skripty jsou převáděny

do jednoduchého spustitelného souboru .exe se všemi potřebnými knihovnami pomocí PyInstalleru. Samozřejmě má i své zápory a ty se vyskytují v podobě špatného ladění, protože při vydání nebo drobné úpravě skriptu je potřeba ho znovu sestavit do celistvého celku. Navíc takový soubor při zavolání nelze odlaďovat běžně skoky. A poslední nevýhodou je samostatná velikost takového skriptu. Skripty v projektu na detekci obličeje mají kolem 30 MB. A u identifikace obličeje to činí okolo 350 MB. Na druhou stranu je to ovlivněno tím, že aby se dal script spustit nezávisle na distribuci Pythonu, je třeba zabalit jak knihovny tak natrénované rozsáhlé modely.

7.9 Detekce obličejů

Detekce probíhá právě pomocí Pythonu, který byl zmíněn v kapitole 7.8. Skript používá čtyři knihovny sys, OpenCV, Dlib a Numpy. Knihovna sys je zde použita pro zpracování parametrů, které ji přišly na vstup z .Net prostředí. Jelikož se nepodařilo zprovoznit Pythonnet, tak jediné čemu rozumí spouštění nového procesu jsou znaky a žádné struktury. Původně skript přebíral pole bitů jako text, které tvořily výsledný obrázek, avšak ukázalo se, že při tomto způsobu přetéká vstupní pole argumentů. Výsledné vstupní parametry skriptu jsou tedy: jaký obrázek, kam ho uložit a prediktor pro detekci tvarů v obličeji. Na straně C# byla vytvořena pomocná třída PythonBridge a PythonVariables pro ulehčení práce s předáváním parametrů mezi těmito dvěma jazyky. OpenCV společně s Numpy je zde pro zpracování obrazu, jeho načtení, uložení a vykreslování výsledných hraničních boxů na základě detekce z knihovny Dlib. Poslední knihovna je právě Dlib, která se stará o samotnou detekci obličejů a 68 příznaků v ní. Obličeje jsou zvýrazněny zelenou barvou a 68 příznaků v obličeji zase červenou. Skript sám o sobě ještě vypisuje kolik tváří nalezl a na jakých obrazových bodech začíná jejich ohraničení. Tento výstup je potom přemostěn z konzolového výpisu do textové podoby a nadále zpracováván a zobrazován uživateli. Samozřejmostí je spuštění nového vlákna, jelikož se jedná o poměrně časově náročnou práci. Hlavním problémem zde byl souběh, který byl vyřešen zámky.



Obrázek 7.6: Příklad detekce obličeje, jedná se o výřez okna z aplikace

7.10 Rozpoznání obličejů

S funkční detekcí obličejů přišla na řadu identifikace již detekovaných tváří. Algoritmus je sestaven v jazyce Python s doprovodnou knihovnou face_recognition vycházející z Dlib. Nejedná se o nic jiného než předem naučené modely pro identifikace tváře z již existujících knihoven jako Dlib, Numpy, Pillow, Scipy. Jedná se o problematickou knihovnu k instalaci. A to i s balíčkovacím systémem Anaconda v novém prostředí, jelikož zapouzdřuje mnoho závislostí, které mají své další závislosti a tak dále. Od toho se odvíjel další problém a to se samostatným zapouzdřením do spustitelného souboru .exe, kde si PyInstaller nedokázal poradit sám a bylo potřeba sestavit specifikaci, co vše se má naimportovat a odkud. Aby se ovšem tento soubor dal spustit na platformě Windows, nezávisle na instalaci Pythonu, potřebných knihoven a naučených modelů, nese si sebou tento soubor přibližně 350 MB.

Otestovaný, funkční a zabalený script se spouští přes nový proces. Je navržen, aby zpracovával hlavní snímky mezi záběry a výsledky identifikace poté ukládal do tabulky, která reaguje na kliknutí a přesunuje do oblasti zájmu. Zároveň jsou tato data uložena do interní struktury pro následné generování XML souboru. Nicméně bylo zjištěno, že taková operace oproti detekci trvá příliš dlouho a kompletně blokuje hlavní UI vlákno. Navíc uživatel nemá přehled, zda se něco děje. Pro tuto operaci bylo tedy vytvořeno vlákno BackgroundWorker, které podle názvu běží na pozadí a zanechává informace uživateli o stavu operace ve formě ukazatele průběhu.

Výkon stále nebyl dostatečný a po hlubším zjištění bylo objeveno, že jedna konkrétní tvář se parametrizuje stále dokola podle počtu detekovaných obličejů v každé iteraci, což je zbytečná operace. Chyba byla do programu zanesena díky ukázkovému příkladu, kdy se tato operace v jejich případě odehrávala pouze jednou, a ne v cyklu. Byl tedy vytvořen nový script, který se spouštěl před samostatnou identifikací a vracel 138 parametrických příznaků tváře, které se potom

předávaly v parametrech do původního algoritmu a porovnávaly vůči neznámým tvářím. Ulehčení na náročnosti byla zásadně znatelná, nicméně přinesla další negativa, jelikož bylo potřeba vytvořit nový samostatný .exe soubor, který měl opět okolo 350 MB a script pro identifikaci se nedal nadále spouštět z příkazové řádky, neboť obsahoval 140 parametrů.

Existovalo však řešení, které bylo úspornější na paměť, ale bylo pomalejší. Jednalo se o parametrizování tváře při vstupu do scriptu identifikace. Tedy při identifikaci osob ve třech snímcích se jednalo o dvakrát promarněnou operaci a čas. Tento způsob byl implementován a jak bylo očekáváno, tak se jednalo o zlatý střed mezi vytížeností paměti a časovou náročností. V tomto systému byl ovšem zvolen výkon na úkor paměti.

Obrázek 7.7: Příklad identifikace konkrétní osoby na videonahrávce

Na obrázku 7.7 je vidět příklad konkrétní segmentace jedné osoby. Předtím, aby bylo možné vůbec tuto funkcionalitu použít, je potřeba mít definované na videonahrávce nějaké segmentace a načíst si vyřezaný obličej osoby. Jedině poté se zpřístupní možnost provést tento algoritmus. Inicializuje se nový proces na pozadí, který si nejdříve vybere hlavní snímky z videonahrávky, tj. snímek mezi dvěma hranicemi segmentace. Následuje jednorázová parametrizace vstupního známého obličeje, který je potom předáván dalšímu procesu, jež spouští samotnou identifikaci s 138 parametry příznaků známé tváře, kam a odkud načítat obrazová data. Proces vrací zpět booleovské hodnoty, které se musí příslušně zpracovat a následně přidat do tabulky. Tento postup je opakován dokud se neprojdou všechny hlavní snímky.

7.11 Optické rozpoznávání znaků

Posledním zajímavým implementovaným algoritmem bylo optické rozpoznávání znaků. K tomuto účelu byla použita knihovna Tesseract OCR, která se vyskytuje volně na Gitu a na vyřešení této problematiky existovaly dvě cesty:

1. Nainstalovat Tesseract OCR na cílový stroj, CLI aplikace
2. C# Wrapper pro Tesseract OCR

První řešení by se dalo použít za předpokladu, že bude ucelené prostředí, ve kterém bude na každém cílovém stroji nainstalován lokální Tesseract OCR a správně nastavené systémové proměnné. Takovéto řešení je naprosto validní, nicméně pro aplikaci by to znamenalo další přítěž v rámci omezení na další požadavky plus nepříjemné volání nového procesu s parametry.

Jelikož pro Tesseract OCR existuje aktivně udržovaný a funkční Wrapper, tak se řešení přímo vybízelo touto cestou. Algoritmus běží stále v prostředí C# bez nutnosti spouštět nový proces a plně podporuje proměnné daného prostředí, což značně ulehčuje revizi kódu. Algoritmus má pro sebe navržen nový model Tesseract, který přijímá obrázek datového typu bitmap a vrací tuple dvou hodnot list textů a list objektů RectangleF, který obsahuje souřadnice x, y, šířku a výšku. Ty poté předává do ViewModelu, který se stará o vykreslení příslušných hraničních boxů a vypsání hodnoty detekce textu. Logicky při prvním spuštění takového algoritmu nastala blokace hlavního UI vlákna a bylo potřeba opět vytvořit nové vlákno a zajistit, aby nedošlo k souběhu.

Obrázek 7.8: Detekce textu u aplikace

8 Závěr

V rámci této bakalářské práce byl vytvořen systém v jazyce C# a Pythonu pro poloautomatické zpracování videonahrávek pořadů, jež obsahuje zajímavé algoritmy jako detekce obličeje, detekce textu a v neposlední řadě identifikace konkrétní osoby z výřezu. Byl také kladen důraz, aby se hlavní komponenty jako posun ve videu a detekce přechodů daly provádět bez pomoci myši a tím se značně ulehčilo ovládání. Výsledný systém pak poskytuje prostředí pro analýzu rozsáhlých videonahrávek pořadů a tvorbu video abstraktů.

Vlastně poprvé jsem se seznámil s tvorbou rozsáhlejšího systému, jež vyžadoval návrh určité struktury. Bez ní by nebylo možné tuto práci udržet pohromadě, protože obsahuje nespočet knihoven a funkcí, jež na sobě závisí. V tomto byla velice nápomocná stránka `www.stackoverflow.com`, která obsahuje mnoho odpovědí na otázky ohledně oblasti IT. Ačkoliv ne vždy se na tuto komunitu dalo spoléhat, neboť v této době nikdo neřešil problém s EmguCV a BASS.NET. Vyvstala také otázka, zda není špatný přístup, případně je to už natolik specifické a nadrámec této komunity, když neexistuje žádné založené vlákno na této stránce.

Tři nejsložitější a z mého pohledu nejzajímavější kapitoly, se kterými jsem se setkal v rámci projektu, byly pravděpodobně úniky v nespravované paměti u bitmap, synchronizace mezi BASS.NET a EmguCV a komunikace C# a Pythonu. Hned u prvního problému se ukázalo, že se nelze vždy spoléhat pouze na automatiku, ale že v tomto případě je třeba číst dokumentaci a nezapomínat na poučky z jazyků typu C/C++ o uvolňování všeho co se inicializuje. Druhý problém bylo nesmírně těžké odhalit. Hledaly se dlouze konkrétní hodnoty, které bylo třeba diagnostikovat. A i s odhaleným problémem bylo poté i nadále složité vymyslet řešení. Poslední problematika komunikace Pythonu a C# byla ovšem nejsložitější, jelikož se jednalo o klíčovou část systému a vyplatilo by se jí zpracovat co nejdokonaleji. Nicméně jelikož knihovny pro komunikaci s Pythonem v rámci C# nepřinesly nic jiného než velké množství problémů, tak přišlo dosavadní řešení, jež má nakonec také své výhody. Při odstraňování tohoto problému se naskytla i otázka, zda nenapsat celý systém v Pythonu s nadstavbou PyQt5.

Největší přínos této práce vidím ve svém zdokonalení programování, aplikování některých vědomostí získaných během studia a rozšíření si vědomostí o této problematice. Neustále se vyskytující problémy mě nutily hledat nové a lepší postupy řešení pro tuto aplikaci. Poprvé jsem se setkal v praxi s časovou náročností metod a souběhem u BackgroundWorker vláken.

V poslední řadě by se daly definovat návrhy na zlepšení v podobě vzhledu aplikace, Task Parallel Library (TPL) místo zastaralého BackgroundWorkeru a im-

plementací lepší komunikace mezi C# a Pythonem. Samozřejmě by mohly býti rozšířeny i algoritmy o další typy detekcí objektů v případě nutnosti.

Literatura

[1] PORTER, Sarah Victoria. *Video Segmentation and Indexing using Motion Estimation.* 2004.

[2] GRANA, C., G. TARDINI a R. CUCCHIARA. MPEG-7 Compliant Shot Detection in Sport Videos. In: *Seventh IEEE International Symposium on Multimedia (ISM'05)* [online]. IEEE, 2005, s. 395-402 [cit. 2019-04-18]. DOI: 10.1109/ISM.2005.82. ISBN 0-7695-2489-3. Dostupné z: http://ieeexplore.ieee.org/document/1565860/

[3] EIKVIL, Line. *OCR - Optical Character Recognition.* 1993, 7 - 10. DOI: 10.1.1.25.3684. Dostupné také z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.3684

[4] SMITH, R. An Overview of the Tesseract OCR Engine. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2* [online]. IEEE, 2007, 2007, s. 629-633 [cit. 2019-04-18]. DOI: 10.1109/ICDAR.2007.4376991. ISBN 0-7695-2822-8. ISSN 1520-5363. Dostupné z: http://ieeexplore.ieee.org/document/4376991/

[5] JENSEN, OH. *Implementing the Viola-Jones Face Detection Algorithm.* Denmark, 2008, 7 - 14. ISBN 87-643-0008-0.

[6] LI, Xiang-Yu a Zhen-Xian LIN. Face Recognition Based on HOG and Fast PCA Algorithm. KRÖMER, Pavel, Enrique ALBA, Jeng-Shyang PAN a Václav SNÁŠEL, ed. *Proceedings of the Fourth Euro-China Conference on Intelligent Data Analysis and Applications* [online]. Cham: Springer International Publishing, 2018, 2018-09-24, s. 10-21 [cit. 2019-04-18]. Advances in Intelligent Systems and Computing. DOI: 10.1007/978-3-319-68527-4_2. ISBN 978-3-319-68526-7. Dostupné z: http://link.springer.com/10.1007/978-3-319-68527-4_2

[7] Face Detection - OpenCV, Dlib and Deep Learning — Learn OpenCV. Learn OpenCV (C++ / Python) [online]. Dostupné z: https://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/

[8] NAGASAKA, Akio a Yuzuru TANAKA. *Automatic video indexing and full-video search for object appearances (abstract).* Tokyo, 1992. ISBN 0-444-89609-0.

[9] PRESS, William H. *Numerical recipes: the art of scientific computing. 3rd ed.* New York: Cambridge University Press, 2007. ISBN 9780521880688.

[10] SWAIN, Michael J. a Dana H. BALLARD. Color indexing. *International Journal of Computer Vision* [online]. 1991, 7(1), 11-32 [cit. 2019-04-18]. DOI: 10.1007/BF00130487. ISSN 0920-5691. Dostupné z: http://link.springer.com/10.1007/BF00130487

[11] ZHANG, HongJiang, Atreyi KANKANHALLI a Stephen W. SMOLIAR. *Automatic partitioning of full-motion video. Multimedia Systems* [online]. 1993, 1(1), 10-28 [cit. 2019-04-18]. DOI: 10.1007/BF01210504. ISSN 0942-4962. Dostupné z: http://link.springer.com/10.1007/BF01210504

[12] UEDA, Hirotada, Takafumi MIYATAKE a Satoshi YOSHIZAWA. IMPACT. In: *Proceedings of the SIGCHI conference on Human factors in computing systems Reaching through technology - CHI '91* [online]. New York, New York, USA: ACM Press, 1991, 1991, s. 343-350 [cit. 2019-04-18]. DOI: 10.1145/108844.108939. ISBN 0897913833. Dostupné z: http://portal.acm.org/citation.cfm?doid=108844.108939

[13] HANJALIC, A. Shot-boundary detection: unraveled and resolved?. *IEEE Transactions on Circuits and Systems for Video Technology* [online]. 12(2), 90-105 [cit. 2019-04-18]. DOI: 10.1109/76.988656. ISSN 10518215. Dostupné z: http://ieeexplore.ieee.org/document/988656/

[14] AKUTSU, Akihito, Yoshinobu TONOMURA, Hideo HASHIMOTO, Yuji OHBA a Petros MARAGOS. [online]. In:*SPIE Visual Communication and Image Processing*. 1992-11-1, s. 1522-1530 [cit. 2019-04-18]. DOI: 10.1117/12.131425. Dostupné z: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1003418

[15] NIBLACK, Wayne, Ishwar K. SETHI, Nilesh V. PATEL a Ramesh C. JAIN. Statistical approach to scene change detection [online]. In:*SPIE Proceedings on Storage and Retrieval for Image and Video Databases III*. 1995-3-23, s. 329- [cit. 2019-04-18]. DOI: 10.1117/12.205299. Dostupné z: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.205299

[16] LUPATINI, G., C. SARACENO a R. LEONARDI. Scene break detection: a comparison. In: *Proceedings Eighth International Workshop on Research Issues in Data Engineering. Continuous-Media Databases and Applications* [online]. IEEE Comput. Soc, 1998, s. 34-41 [cit. 2019-04-18]. DOI: 10.1109/RIDE.1998.658276. ISBN 0-8186-8389-9. Dostupné z: http://ieeexplore.ieee.org/document/658276/

[17] VLACHOS, T. Cut detection in video sequences using phase correlation. *IEEE Signal Processing Letters* [online]. 2000, 7(7), 173-175 [cit. 2019-04-18]. DOI: 10.1109/97.847360. ISSN 1070-9908. Dostupné z: http://ieeexplore.ieee.org/document/847360/

[18] W. A. C. Fernando, C. N. Canagarajah, and D. R. Bull. Video segmentation and classification for content based storage and retrieval using motion vectors. *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases VII.* 1998. DOI: 10.1117/12.333889

[19] BOUTHEMY, P., M. GELGON a F. GANANSIA. A unified approach to shot change detection and camera motion characterization. *IEEE Transactions on Circuits and Systems for Video Technology* [online]. 9(7), 1030-1044 [cit. 2019-04-18]. DOI: 10.1109/76.795057. ISSN 10518215. Dostupné z: http://ieeexplore.ieee.org/document/795057/

[20] ZABIH, Ramin, Justin MILLER a Kevin MAI. A feature-based algorithm for detecting and classifying scene breaks. In: *Proceedings of the third ACM international conference on Multimedia - MULTIMEDIA '95* [online]. New York, New York, USA: ACM Press, 1995, 1995, s. 189-200 [cit. 2019-04-18]. DOI: 10.1145/217279.215266. ISBN 0897917510. Dostupné z: http://portal.acm.org/citation.cfm?doid=217279.215266

[21] DAILIANAS, Apostolos, Robert B. ALLEN, Paul ENGLAND, Andrew G. TESCHER a V. Michael BOVE, JR. [online]. In:*Proceedings of the SPIE Conference on Integration Issues in Large Commercial Media Delivery Systems*. 1996-1-3, s. 2-16 [cit. 2019-04-18]. DOI: 10.1117/12.229193. Dostupné z: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1010456

[22] LIENHART, Rainer W., Minerva M. YEUNG, Chung-Sheng LI a Rainer W. LIENHART. [online]. In:*Proceedings of the SPIE Conference on Storage and Retrieval for Media Databases* . 2001-1-1, s. 219-230 [cit. 2019-04-18]. DOI: 10.1117/12.410931. Dostupné z: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=905124

[23] HYUN SUNG CHANG, SANGHOON SULL a SANG UK LEE. Efficient video indexing scheme for content-based retrieval. *IEEE Transactions on Circuits and Systems for Video Technology* [online]. 9(8), 1269-1279 [cit. 2019-04-18]. DOI: 10.1109/76.809161. ISSN 10518215. Dostupné z: http://ieeexplore.ieee.org/document/809161/

[24] DIRFAUX, F. Key frame selection to represent a video. In: *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)* [online]. IEEE, 2000, 2000, 275-278 vol.2 [cit. 2019-04-18]. DOI: 10.1109/ICIP.2000.899354. Dostupné z: http://ieeexplore.ieee.org/document/899354/

[25] GUNSEL, B. a A.M. TEKALP. Content-based video abstraction. In: *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269)* [online]. IEEE Comput. Soc, 1998, s. 128-132 [cit. 2019-04-18]. DOI: 10.1109/ICIP.1998.727150. ISBN 0-8186-8821-1. Dostupné z: http://ieeexplore.ieee.org/document/727150/

[26] HERNG-YOW CHEN a JA-LING WU. A multi-layer video browsing system. *IEEE Transactions on Consumer Electronics* [online]. 41(3), 842-850 [cit. 2019-04-18]. DOI: 10.1109/30.468075. ISSN 00983063. Dostupné z: http://ieeexplore.ieee.org/document/468075/

[27] ARDIZZONE, E. a M. LA CASCIA. Video indexing using optical flow field. In: *Proceedings of 3rd IEEE International Conference on Image Processing* [online]. IEEE, 1996, s. 831-834 [cit. 2019-04-18]. DOI: 10.1109/ICIP.1996.560876. ISBN 0-7803-3259-8. Dostupné z: http://ieeexplore.ieee.org/document/560876/

[28] ZHANG, Hong Jiang, Jianhua WU, Di ZHONG a Stephen W. SMOLIAR. An integrated system for content-based video retrieval and browsing. *Pattern Recognition* [online]. 1997, 30(4), 643-658 [cit. 2019-04-18]. DOI: 10.1016/S0031-3203(96)00109-4. ISSN 00313203. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0031320396001094

[29] S. H. Kim and R. H. Park. A novel approach to video indexing using luminance projection. *Proceedings of the IASTED International Conference on Signal and Image Processing*. 2002. s. 359–362 [cit. 2019-04-18].

[30] ALMEIDA, Jurandy, Ricardo da S. TORRES a Neucimar J. LEITE. Rapid Video Summarization on Compressed Video. In: *2010 IEEE International Symposium on Multimedia* [online]. IEEE, 2010, 2010, s. 113-120 [cit. 2019-04-18]. DOI: 10.1109/ISM.2010.25. ISBN 978-1-4244-8672-4. Dostupné z: http://ieeexplore.ieee.org/document/5693830/

[31] WOLF, W. Key frame selection by motion analysis. In: *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*[online]. IEEE, 1996, s. 1228-1231 [cit. 2019-04-18]. DOI: 10.1109/ICASSP.1996.543588. ISBN 0-7803-3192-3. Dostupné z: http://ieeexplore.ieee.org/document/543588/

[32] YIHONG GONG a XIN LIU. Video summarization using singular value decomposition. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)* [online]. IEEE Comput. Soc, 2000, s. 174-180 [cit. 2019-04-18]. DOI: 10.1109/CVPR.2000.854772. ISBN 0-7695-0662-3. Dostupné z: http://ieeexplore.ieee.org/document/854772/

[33] HAN, K.J. a A.H. TEWFIK. Eigen-image based video segmentation and indexing. In: *Proceedings of International Conference on Image Processing* [online]. IEEE Comput. Soc, 1997, s. 538-541 [cit. 2019-04-18]. DOI: 10.1109/ICIP.1997.638827. ISBN 0-8186-8183-7. Dostupné z: http://ieeexplore.ieee.org/document/638827/

[34] YOON, K., D. DEMENTHON a D. DOERMANN. Event detection from MPEG video in the compressed domain. In: *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000* [online]. IEEE Comput. Soc, 2000, s.

819-822 [cit. 2019-04-18]. DOI: 10.1109/ICPR.2000.905531. ISBN 0-7695-0750-6. Dostupné z: http://ieeexplore.ieee.org/document/905531/

[35] Vision AI — Derive Image Insights via ML — *Google Cloud. Cloud Computing Services — Google Cloud* [online]. Dostupné z: https://cloud.google.com/vision/

[36] Amazon Rekognition – Video and Image - *AWS. Amazon Web Services (AWS) - Cloud Computing Services* [online]. Copyright © 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. [cit. 19.04.2019]. Dostupné z: https://aws.amazon.com/rekognition/