



Bakalářská práce

**Mobilní aplikace využívající algoritmy pro zpracování
a rozpoznávání obrazu**

Studijní program:

B0613A140005 – Informační technologie

Studijní obor:

B0613A140005AI – Aplikovaná informatika

Autor práce:

Jakub Káčerek

Vedoucí práce:

doc. Ing. Josef Chaloupka, Ph.D.

Liberec 2025

Tento list nahradte
originálem zadání.

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, 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 nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

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

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

7. 5. 2025

Jakub Káčerek

Mobilní aplikace využívající algoritmy pro zpracování a rozpoznávání obrazu

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací mobilní aplikace, která využívá algoritmy pro zpracování a rozpoznávání obrazu v reálném čase. Hlavním cílem je vytvořit multiplatformní nástroj schopný detektovat a rozpoznávat obličeje i text na základě vstupního obrazového signálu z kamery mobilního zařízení. Práce se zaměřuje na výběr vhodných metod počítačového vidění, jejich optimalizaci pro omezené výpočetní prostředky mobilních zařízení a efektivní integraci v prostředí Flutter. Využity jsou moderní technologie jako Google ML Kit a OpenCV. Výsledkem je funkční aplikace, která demonstruje praktické využití strojového učení a rozpoznávání obrazu v mobilních podmínkách, a zároveň respektuje zásady ochrany soukromí uživatele.

Klíčová slova: mobilní aplikace, zpracování obrazu, rozpoznávání obličeje, OCR, počítačové vidění, Flutter, ML Kit, OpenCV

Mobilní aplikace využívající algoritmy pro zpracování a rozpoznávání obrazu

Abstract

This bachelor's thesis focuses on the design and implementation of a mobile application that utilizes image processing and recognition algorithms in real time. The primary goal is to develop a cross-platform tool capable of detecting and recognizing faces and text based on live input from a mobile device camera. The work emphasizes the selection of appropriate computer vision methods, their optimization for the limited computational resources of mobile devices, and efficient integration within the Flutter framework. Technologies such as Google ML Kit and OpenCV are employed. The result is a fully functional application demonstrating the practical application of machine learning and image recognition in mobile environments, while ensuring user data privacy.

Keywords: mobile application, image processing, face recognition, OCR, computer vision, Flutter, ML Kit, OpenCV

Poděkování

Tento cestou bych chtěl poděkovat mému vedoucímu práce za asistence a navádění v případných problémech.

Dále bych chtěl poděkovat mé rodině a přátelům, kteří mě podpořovali po celou dobu mého studia.

A v hlavní řadě bych chtěl poděkovat mé úžasné přítelkyni, která mě držela před naprostým nervovým zhroucením při vypracování této práce.

Obsah

Seznam zkratek	9
1 Úvod	10
2 Současná problematika a state-of-the-art aplikace	11
2.1 Technické výzvy mobilního zpracování v reálném čase	11
2.2 Kvalita a variabilita vstupních dat	12
2.3 Ochrana soukromí a etické aspekty	12
2.4 State-of-the-art aplikace	13
2.5 Směřování dalšího vývoje	14
3 Teoretické základy zpracování a rozpoznávání obrazu	15
3.1 Předzpracování obrazu	15
3.2 Konverze barev	16
3.3 Normalizace jasu a kontrastu	17
3.4 Odstranění šumu v obrazu	19
3.5 Detekce hran a zvýraznění rysů	20
4 Struktura a uživatelské rozhraní aplikace	23
4.1 Hlavní obrazovka	23
4.2 Kontextové menu	23
4.3 Obrazovka nastavení	24
5 Klíčové algoritmy aplikace	26
5.1 Přehled aplikace	26
5.2 Detekce obličeje	26
5.2.1 Získávání vstupních dat z kamery	27
5.2.2 Zpracování oprávnění a spuštění streamu	27
5.2.3 Příprava snímků pro detekci	27
5.2.4 Princip fungování detekčního modelu ML Kit	27

5.2.5	Výstupní data z detekce	28
5.3	Rozpoznání obličeje	29
5.3.1	Příprava vstupního obrazu pro rozpoznávání	29
5.3.2	Extrahování embeddingu (vektorové reprezentace obličeje) . .	29
5.3.3	Správa databáze známých identit	31
5.3.4	Optimalizace výkonu a real-time nasazení	31
5.3.5	Zabezpečení a soukromí	32
5.3.6	Zpracování a porovnání embeddingů v Kotlinu (OpenCV) . .	32
5.4	Rozpoznávání textu - OCR	32
5.4.1	Získání obrazových dat	32
5.4.2	Zpracování obrazu pomocí ML Kit Text Recognition	33
5.4.3	Předzpracování obrazu pomocí OpenCV v Kotlinu	34
5.5	Flutter framework	34
5.5.1	Architektura a základní principy	34
5.5.2	Programovací jazyk Dart	35
5.5.3	Widgety a deklarativní přístup k UI	35
5.5.4	Multiplatformní vývoj a podpora nativních funkcí	35
5.5.5	Rozšíření a integrace knihoven třetích stran	36
5.5.6	Výkon a efektivita	36
5.5.7	Vývojové nástroje a build	36
5.6	OpenCV – zpracování obrazu	37
5.6.1	Integrace OpenCV v prostředí Flutter	37
5.6.2	Převod YUV na Mat	37
5.6.3	Předzpracování obrazu	37
5.6.4	Optimalizace výkonu	38
5.6.5	Shrnutí	38
6	Závěr	39
6.1	Hodnocení použitého frameworku	39
6.2	Srovnání s existujícími řešeními	39
6.3	Přínos a využití	40
6.4	Možnosti dalšího rozvoje	40
6.5	Shrnutí	41
Použitá literatura		42
Odkaz na GitHub repozitář		45

Seznam zkratek

AI	Artificial Intelligence (Umělá inteligence)
API	Red-Green-Blue (Tříbarevné spektrum počítačového vidění)
CNN	Hue-Saturation-Value
CV	Plugin
FPS	Fakulta textilní
GPU	Fakulta přírodovědně-humanitní a pedagogická
HSV	Ekonomická fakulta
ID	Fakulta zdravotnických studií
JSON	Ústav pro nanomateriály, pokročilé technologie a inovace
ML	Machine Learning (Strojové učení)
ML Kit	Machine Learning Kit (Google knihovna pro ML na mobilních zařízeních)
NV21	Android YUV barevný formát pro kamerová data
OCR	Optical Character Recognition (Optické rozpoznávání znaků)
RAM	Random Access Memory (Operační paměť)
SDK	Software Development Kit (Sada nástrojů pro vývoj)
UI	User Interface (Uživatelské rozhraní)
UX	User Experience (Uživatelská zkušenost)
YUV	Barevný model používaný u videosignálu

1 Úvod

V posledních letech zaznamenal obor zpracování a rozpoznávání obrazu významné pokroky díky rozvoji výpočetní techniky a technologiím jako jsou např. neurono-vé sítě, počítačové vidění strojové učení, nebo umělá inteligence. Tyto technologie umožňují automatickou analýzu enormních množství dat s velice vysokou přesností. Tyto metody se dnes už běžně využívají v různých oblastech, jako je medicína, bezpečnostní systémy, autonomní řízení vozidel, automatizace průmyslu a mnoho dalších. Za posledních několik let je patrně rostoucí trend využívat tyto technologie v počítačích nebo v mobilních aplikacích, které umožňují tyto metody využívat uživateli přímo na jejich telefoních zařízeních nebo tablettech.

Tato bakalářská práce se zaměřuje právě na návrh a implementaci mobilní aplikace pro platformu Android, pro zpracování a rozpoznávání obrazu. Hlavním cílem je vytvořit mobilní aplikaci schopnou analyzovat data v reálném čase pomocí několika různých algoritmů. K dosažení tohoto cíle bude využito několika knihoven a nástrojů. Mezi nejvýznamnější patří: framework Flutter, PLG Google MLKit a další.

Vývoj mobilních aplikací představuje několik výzev. Mezi nejvýznamnější patří právě optimalizace algoritmů pro omezený výpočetní výkon mobilních zařízení. práce s různorodými vývojovými prostředími, zajištění přístupnosti a implementace uživatelského rozhraní. V této práci se proto zaměříme nejen na samotnou implementaci algoritmů, ale i na jejich optimalizaci a zjednodušení práce s nimi pro uživatele.

Práce je strukturovaná následovně. První kapitola slouží k seznámení se základy teorie zpracování obrazu, principy rozpoznávání díky strojovému vidění a souvisejícími algoritmy. Druhá kapitola se bude věnovat samotnému návrhu aplikace, kde budou popsány jednotlivě všechny použité algoritmy, struktura aplikace a implementační detaily. Třetí kapitola se bude zaměřovat na testování a vyhodnocení efektivity navrženého řešení.

Tato práce přispívá k oblasti mobilního zpracování obrazu tím, že kombinuje moderní algoritmy s praktickým využitím v mobilních aplikacích. Výstupem bude plně funkční aplikace demonstrující možnosti nasazení počítačového vidění na mobilních zařízeních s důrazem na efektivitu a přesnost rozpoznávání obrazu.

2 Současná problematika a state-of-the-art aplikace

Mobilní aplikace využívající detekci obličejů a optické rozpoznávání znaků (OCR) v reálném čase představují dynamicky se rozvíjející oblast, která se opírá o pokroky v oblasti strojového učení, zpracování obrazu a mobilních technologií. Tato kapitola popisuje současné výzvy, které vývoj těchto aplikací provázejí, a uvádí přehled nejvýznamnějších moderních řešení dostupných na trhu.

2.1 Technické výzvy mobilního zpracování v reálném čase

Významnou výzvou je omezený výpočetní výkon mobilních zařízení. Na rozdíl od výkonných desktopových počítačů, které běžně disponují dedikovanými GPU akcelerátory, musí být mobilní aplikace optimalizovány pro běh na procesorech s omezeným výkonem a kapacitou paměti. To vede k nutnosti nasazení zjednodušených a efektivních modelů neuronových sítí, jako jsou MobileNet, BlazeFace, EfficientNet-Lite nebo CRNN, které nabízejí rozumný kompromis mezi přesností a rychlostí.

Při zajištění plynulého zpracování obrazu v reálném čase je kritická latence – aplikace by měly reagovat do několika desítek až stovek milisekund. To vyžaduje optimalizace jak na úrovni inference (např. použití kvantizovaných modelů, jako INT8 nebo FP16), tak na úrovni samotného zpracování snímků – včetně konverze barevných prostorů (např. z YUV420 nebo NV21 do RGB), synchronizace mezi vláknenem pro záznam obrazu a vláknenem pro inference, a snížení frekvence snímkování tak, aby byla zachována rovnováha mezi výkonem a výdrží baterie. Výzvou zůstává také správné škálování aplikace pro různá zařízení s různými verzemi Android/iOS, různým hardwarem a verzemi NPU/Tensor jednotek.

2.2 Kvalita a variabilita vstupních dat

Kvalita detekce a rozpoznávání je silně závislá na kvalitě vstupních obrazových dat. Mobilní zařízení jsou často používána v různorodých podmínkách – za špatného osvětlení, při pohybu uživatele nebo cílového objektu, či při silném protisvětle. Tyto faktory způsobují problémy jako je rozmazání, šum, nedostatečný kontrast nebo deformace obrazu způsobené perspektivou.

V oblasti OCR je důležitá schopnost rozpoznávat různé typy písma (tištěné, ručně psané), orientaci textu (rotace, naklonění) i strukturu (víceřádkové odstavce, formuláře). Kromě toho musí být modely robustní vůči jazykovým odchylkám – například správně interpretovat znaky s diakritikou, rozpoznávat slova v různých jazycích nebo přepínat jazykové modely podle detekovaného jazyka vstupního textu. U vícejazykových aplikací je vhodné kombinovat OCR s jazykovou detekcí, případně využívat modely trénované na vícejazyčných korpusech.

2.3 Ochrana soukromí a etické aspekty

Vzhledem k tomu, že aplikace pracují s citlivými daty – ať už se jedná o biometrické rysy obličeje nebo textové informace, které mohou obsahovat osobní údaje – je třeba věnovat zvýšenou pozornost ochraně soukromí a souladu s legislativou. Zvláště důležité je zajistit, že veškeré zpracování probíhá v souladu s pravidly jako je GDPR v Evropské unii nebo CCPA v Kalifornii.

Z technického hlediska to znamená minimalizovat množství dat odesílaných na cloud, šifrovat veškerou komunikaci a ideálně zpracovávat data přímo na zařízení. Moderní přístupy využívají tzv. on-device inference, kdy je model nahrán a spuštěn přímo na mobilním zařízení bez nutnosti odesílání snímků na vzdálený server. Dalším trendem je federativní učení, při kterém se model učí na lokálních datech a sdílí pouze váhové aktualizace, nikoliv samotná data – čímž je výrazně sníženo riziko úniku citlivých informací. V praxi je třeba uživatelům také nabídnout transparentní informace o tom, jaká data jsou sbírána, jak jsou zpracovávána, a možnost jejich smazání.

2.4 State-of-the-art aplikace

- **Google Lens**

Tato aplikace představuje špičku v oblasti vizuální analýzy. Umožňuje v reálném čase rozpoznávat text, objekty, rostliny, zvířata nebo budovy. Kromě OCR nabízí i překlad textu, extrakci kontaktů z vizitek, či vyhledávání produktů podle obrázku. Pro mobilní inference využívá ML Kit a TensorFlow Lite, přičemž pro složitější analýzy spoléhá na cloudové služby Google Cloud Vision.

- **Microsoft Seeing AI**

Jedná se o asistivní aplikaci pro osoby se zrakovým postižením, která integruje OCR, rozpoznávání osob, emocí a popisování prostředí. Například dokáže přečíst text z dokumentu, určit pohlaví a přibližný věk osoby, rozpoznat čárový kód nebo nahlas popsat, co se nachází v záběru kamery. Využívá technologie Microsoft Azure Cognitive Services a strojového učení optimalizovaného pro mobilní prostředí.

- **Snapchat / Instagram Filters**

Tyto aplikace využívají detekci a sledování obličeje pro aplikaci interaktivních filtrů v reálném čase. Díky modelům jako je BlazeFace, MediaPipe Face Mesh nebo i vlastní implementace detekce bodů na obličeji, dokáží přesně mapovat rysy obličeje i při pohybu a různých světelných podmínkách. Tyto technologie nacházejí uplatnění i mimo zábavu – např. v oblasti kosmetiky, zábavy nebo telekonferencí.

- **CamScanner, Adobe Scan**

Tyto nástroje se specializují na digitalizaci papírových dokumentů pomocí mobilního zařízení. Detekují hranice dokumentu, upravují perspektivu, zvyšují kontrast a provádějí OCR pro převod obsahu do vyhledatelného textu. Výsledky lze exportovat jako PDF nebo upravovat přímo v aplikaci. Pro OCR často využívají knihovny jako Tesseract OCR nebo vlastní cloudové služby. “

2.5 Směřování dalšího vývoje

Do budoucna se očekává zrychlování a zefektivňování modelů díky technikám jako je model pruning (odstraňování nadbytečných neuronů), knowledge distillation (přenos znalostí z většího modelu do menšího) a kvantizace. Nové architektury jako MobileOne nebo EfficientFormer jsou navrženy přímo pro mobilní inference a umožňují běh složitých modelů i na zařízeních střední třídy.

Zároveň se zvyšuje důraz na bezpečnost a vysvětlitelnost rozhodnutí modelu – tzv. explainability. To je klíčové především v aplikacích v medicíně, právu nebo veřejné správě, kde je nutné rozumět tomu, proč model dospěl k určitému závěru.

Jedním z nejzajímavějších směrů vývoje je multimodální zpracování – tedy kombinace obrazu, textu a zvuku v reálném čase. Takové systémy mohou například současně detekovat obličej, číst text na obrazovce a interpretovat hlas uživatele, čímž vznikají nové možnosti v oblasti asistivních technologií, bezpečnostních systémů i chytrých průmyslových řešení.

3 Teoretické základy zpracování a rozpoznávání obrazu

Oblast zpracování obrazu a jeho rozpoznávání se v posledních desetiletích výrazně rozvinula díky pokrokům v oblasti počítačového vidění, umělé inteligence a dostupnosti výpočetních prostředků. Algoritmy určené pro analýzu obrazových dat umožňují nejen úpravu a filtraci obrazu, ale především jeho hlubší interpretaci a automatickou klasifikaci objektů. Díky těmto metodám lze efektivně detektovat tvary, rozpoznávat obličeje, analyzovat pohyb či dokonce provádět segmentaci scény v reálném čase.

Tato kapitola poskytuje teoretický přehled metod používaných pro zpracování a rozpoznávání obrazu. Nejprve se zaměříme na obecné principy práce s obrazovými daty, včetně jejich reprezentace a základních operací pro předzpracování. Následně budou představeny klíčové algoritmy, které tvoří základ mobilních aplikací zaměřených na rozpoznávání obrazu. Každý z těchto algoritmů bude podrobně rozebrán v samostatných podkapitolách.

3.1 Předzpracování obrazu

Předzpracování obrazu je klíčovým krokem v počítačovém vidění, který zajišťuje správnou kvalitu vstupních dat pro následné algoritmy zpracování a rozpoznávání. Hlavním cílem této fáze je eliminovat nežádoucí šum, zvýraznit důležité rysy obrazu a normalizovat jeho parametry tak, aby následné modely dosahovaly co nejlepších výsledků.

3.2 Konverze barev

Konverze barev je jedním ze základních kroků předzpracování obrazu v oblasti počítačového vidění a digitálního zpracování obrazu. Tento proces zahrnuje transformaci obrazu z jednoho barevného prostoru do jiného, což umožňuje efektivnější analýzu, detekci a rozpoznávání objektů. Každý barevný model je vhodný pro různé aplikace a umožňuje extrakci specifických vlastností obrazu, jako jsou kontrast, sytost, jas nebo barevné složky.

V kontextu strojového vidění je klíčové zvolit vhodný barevný prostor v závislosti na aplikaci. Například pro rozpoznávání obličejů může být vhodnější pracovat v prostoru **YCrCb**, zatímco pro segmentaci objektů podle barvy se často používá **HSV**.

1. **RGB** - Každá složka má obvykle hodnoty v rozsahu 0–255 (8bitová reprezentace). Kombinací těchto složek vznikají různé barvy. RGB model je vhodný pro vizualizaci, ale není optimální pro analýzu obrazu, protože není dobře oddělen jas a barva
2. Konverze do stupňů šedi (**grayscale**) znamená odstranění barevných informací a ponechání pouze informace o jasu. Obraz je pak reprezentován jediným kanálem s hodnotami od 0 (černá) do 255 (bílá). **Grayscale** obraz se často používá v počítačovém vidění, protože mnoho algoritmů, například detekce hran nebo OCR (optické rozpoznávání znaků), nevyžaduje barevné informace.
3. **YUV** formát využívá jasovou složku (Y) a podvzorkovanými chrominančními složkami (U, V). Pro vstup do Google ML Kit (požadující RGB, tvar [1, 224, 224, 3] je nutné provést konverzi NV21 (nativní formát výstupu z kamery) do RGB.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \cdot \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

4. **YCrCb** je barevný model používaný v televizním a obrazovém zpracování. Odděluje jasovou složku (**Y**) od barevných složek (**Cr** – červeno-modrá a **Cb** – modro-zelená). Tato separace umožňuje efektivní kompresi obrazu a je také využívána pro detekci obličejů.

Při implementaci převodu **YUB** → **RGB** pomocí OpenCV v jazyce **Kotlin** se běžně používá:

Konverze barev je klíčový krok v předzpracování obrazu, který umožňuje lepší analýzu a detekci objektů. Výběr správného barevného modelu závisí na konkrétní aplikaci – zatímco RGB je vhodný pro vizualizaci, YUV a YCrCb jsou efektivní pro segmentaci a detekci. V této práci jsou barevné modely využívány pro detekci obličejů (YCrCb), OCR (grayscale) a rozpoznávání bankovek (RGB).

3.3 Normalizace jasu a kontrastu

Normalizace jasu a kontrastu je klíčový krok předzpracování obrazu, který zajišťuje konzistentní osvětlení a zlepšuje viditelnost detailů. Tento proces se často používá ve strojovém vidění, aby se minimalizovaly rozdíly v osvětlení a umožnilo robustnější zpracování obrazu.

Například při rozpoznávání obličejů může mít různá intenzita světla vliv na detekční algoritmy. Podobně při OCR (optickém rozpoznávání znaků) může být špatný kontrast překážkou při správné segmentaci textu. Cílem normalizace je tedy zajistit, aby byl obraz jednotně osvětlený a obsahoval dobře rozeznatelné prvky.

1. Normalizace jasu

- (a) **Lineární normalizace** jasu Nejjednodušší metodou je lineární transformace hodnot pixelů tak, aby pokryly celý rozsah 0–255. To se provádí podle vzorce:

$$I' = \frac{I - I_{min}}{I_{max} - I_{min}} * 255 \quad (3.1)$$

kde:

- I je původní hodnota pixelu
- I_{min} je nejnižší intenzita v obrazu
- I_{max} je nevyšší intenzita v obrazu

- (b) **Gamma korekce** Gamma korekce umožňuje upravit jas obrazu nelineárním způsobem. Používá se vzorec:

$$I' = 255 \times \left(\frac{I}{255} \right)^{\Gamma} \quad (3.2)$$

kde Γ určuje sílu korekce. Pokud $\Gamma < 1$, obraz se zesvětlí, pokud $\Gamma > 1$, obraz se ztmaví.

Gamma korekce je užitečná při zpracování snímků pořízených za špatných světelných podmínek.

- (c) Klasická ekvalizace histogramu může někdy vést k přehnanému kontrastu. **CLAHE** (Contrast Limited Adaptive Histogram Equalization) je vylepšená metoda, která pracuje s lokálními oblastmi obrazu, čímž se vyhýbá přepáleným světlým nebo tmavým oblastem.

2. Normalizace kontrastu

- (a) **Ekvalizace histogramu** redistribuuje jasové hodnoty tak, aby byly rovnoměrněji rozloženy. Tento postup zvýrazňuje detaily v tmavých i světlých oblastech. Tato metoda je efektivní pro zlepšení kontrastu černobílých snímků.
- (b) **Normalizace kontrastu pomocí NORM_MINMAX** Tato metoda pracuje podobně jako lineární normalizace jasu, ale aplikuje se na kontrast. Tento přístup pomáhá zdůraznit rozdíly mezi objekty a pozadím.
- (c) **Změna kontrastu a jasu ručně** Jas a kontrast lze upravit jednoduchou lineární transformací:

$$I' = \alpha I + \beta \quad (3.3)$$

kde:

- α je koeficient kontrastu (>1 zvýší kontrast, <1 sníží kontrast)
- β je přírůstek jasu

3. Použití v počítačovém vidění

Normalizace jasu a kontrastu je klíčová pro mnoho aplikací v oblasti zpracování obrazu:

- **Rozpoznávání obličejů** – zlepšení viditelnosti rysů obličeje při různých světelných podmínkách.
- **OCR (optické rozpoznávání znaků)** – zajištění lepší čitelnosti textu odstraněním nerovnoměrného osvětlení.
- **Lékařské zobrazování** – zvýraznění detailů v rentgenových snímcích.
- **Automatické řízení vozidel** – zlepšení viditelnosti dopravních značek za různých podmínek.

Normalizace jasu a kontrastu je důležitý krok v předzpracování obrazu, který pomáhá zlepšit kvalitu snímků a umožňuje přesnější analýzu dat. Metody jako ekvalizace histogramu, CLAHE nebo gamma korekce poskytují různé způsoby, jak přizpůsobit obraz konkrétní aplikaci.

3.4 Odstranění šumu v obraze

Odstranění šumu je klíčovým krokem v předzpracování obrazu, protože šum může negativně ovlivnit výkonnost následných algoritmů pro rozpoznávání a analýzu. Šum v obrazech může mít různé příčiny, například elektronické rušení, nízké osvětlení, kompresní artefakty nebo pohybovou neostrost. Existuje několik metod pro redukci šumu, které jsou vybírány na základě povahy šumu a požadované kvality výstupu.

1. Typy šumu v obraze

- (a) **Gaussovský šum** – Náhodné odchylky jasu způsobené například senzory fotoaparátu.
- (b) **Sůl a pepř šum** – Náhodně se vyskytující černé a bílé body v obraze, často způsobené poruchami přenosu.
- (c) **Poissonův šum** – Vyskytuje se při nízkém osvětlení kvůli kvantové povaze světla.
- (d) **Šum způsobený kompresí** – Vzniká při ztrátové kompresi obrazu, například u JPEG.

2. Metody odstranění šumu

- (a) **Gaussovské rozostření (Gaussian Blur)**

Gaussovský filtr je efektivní metoda pro odstranění náhodného šumu, která funguje na principu konvoluce obrazu s Gaussovskou funkcí. Ten-to filtr změkčuje obraz a odstraňuje drobné detaily, které mohou být interpretovány jako šum.

$$G(x, y) = \frac{1}{\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (3.4)$$

kde σ řídí úrověň rozostření.

- (b) **Mediánový filtr (Median Filter)**

Mediánový filtr je vhodný zejména pro odstranění šumu typu "sůl a pepř". Na rozdíl od Gaussovského filtrování zachovává hrany, což je výhodné při zpracování textu nebo ostrých hran v obraze.

- (c) **Bilaterální filtr (Bilateral Filter)**

Bilaterální filtr je pokročilejší metoda, která zachovává hrany, zatímco snižuje šum. Oproti Gaussovskému filtrování lépe rozpoznává rozdíly mezi objekty a zachovává detaily.

(d) Waveletová transformace

Waveletová transformace rozkládá obraz na frekvenční složky a umožňuje odstranění šumu v konkrétních úrovních detailu. Používá se zejména v pokročilých aplikacích, jako je lékařské zobrazování nebo satelitní snímky.

3. Shrnutí

Odstranění šumu je důležitým krokem předzpracování obrazu, který zlepšuje kvalitu dat pro následné zpracování. Volba správné metody závisí na typu šumu a požadavcích na výstup.

- **Gaussovský filtr** je vhodný pro běžné náhodné šumy.
- **Mediánový filtr** je ideální pro odstranění ”sůl a pepř“ šumu.
- **Bilaterální filtr** poskytuje kompromis mezi zachováním hran a odstraněním šumu.
- **Waveletová transformace** je výkonná, ale složitější metoda.

3.5 Detekce hran a zvýraznění rysů

Detekce hran je klíčovou součástí zpracování obrazu, protože umožňuje extrahovat důležité struktury z obrazu, jako jsou obrysy objektů, změny v intenzitě a další rysy důležité pro analýzu. Zvýraznění rysů pomáhá algoritmu pro rozpoznávání obrazu lépe rozlišovat mezi různými objekty a snižuje množství nepotřebných informací.

1. Principy detekce hran

Hranou v obrazu se obvykle rozumí oblast, kde dochází k významné změně jasu nebo barevné intenzity. Tyto změny jsou identifikovány pomocí gradientu, což je matematická míra změny intenzity obrazu v určitém směru. Matematicky lze gradient reprezentovat jako parciální derivace intenzity obrazu $I(x, y)$ podle souřadnic x a y :

$$G_x = \frac{I}{x}, G_y = \frac{I}{y} \quad (3.5)$$

Celková velikost gradientu (hrana) se pak vypočítá jako:

$$G = \sqrt{G_x^2 + G_y^2} \quad (3.6)$$

Směr hrany je dán úhlem:

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right) \quad (3.7)$$

2. Metody detekce hran

(a) Sobelův filtr

Sobelův filtr je metoda založená na výpočtu gradientu obrazu pomocí konvoluce s následujícími jádry (kernel):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

(b) Prewittův filtr

Prewittův filtr je podobný Sobelově metodě, ale používá jiné konvoluční matice:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Použití Prewittova filtru je téměř totožné se Sobelovým filtrem.

(c) Cannyho detektor hran

Cannyho metoda je pokročilejší technika detekce hran, která kombinuje několik kroků:

- **Odstranění šumu** pomocí Gaussova rozostření.
- **Výpočet gradientu** Sobelovým filtrem.
- **Potlačení nesprávných hran** metodou non-maximum suppression.
- **Použití prahování hysterézí** k odstranění slabých hran.

3. Zvýraznění rysů v obraze

Kromě detekce hran existují techniky, které pomáhají zvýraznit určité rysy obrazu:

- **Zvýšení kontrastu** pomocí histogramové ekvalizace.
- **Ostré filtry** jako Laplaceova transformace:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- **Gaborovy filtry** pro zvýraznění textur.

4. Shrnutí

- Detekce hran je založena na analýze gradientu intenzity obrazu.
- Sobelův a Prewittův filtr jsou jednoduché gradientní metody.
- Cannyho detektor hran poskytuje přesnější výsledky díky několika zpracovacím krokům.
- Zvýraznění rysů lze dosáhnout pomocí filtrování a transformací, jako je Laplaceova nebo Gaborova filtrace.

4 Struktura a uživatelské rozhraní aplikace

Uživatelské rozhraní navržené aplikace bylo vytvořeno s důrazem na jednoduchost, přehlednost a funkčnost. Aplikace využívá celý displej zařízení pro zobrazení obrazu z kamery v reálném čase, přičemž všechny ovládací prvky jsou rozmístěny tak, aby nenarušovaly vizuální tok a zároveň byly uživatelsky snadno dostupné.

4.1 Hlavní obrazovka

Po spuštění aplikace je uživateli zobrazena hlavní obrazovka, která je plně překryta živým obrazem z kamery zařízení (tzv. live feed). Kamera je inicializována ve výchozím nastavení podle volby uživatele (přední nebo zadní) a v reálném čase poskytuje vstup pro detekci a rozpoznávání obrazu.

V horní části obrazovky je umístěn jednoduchý textový prvek zobrazující název aplikace. Tento nápis slouží jako vizuální identifikace a nemá žádnou ovládací funkci.

Pod nápisem se nachází informační panel – malý box, který zobrazuje základní stavové informace týkající se běhu aplikace. Tyto údaje se aktualizují v reálném čase a umožňují uživateli sledovat aktivitu systému bez nutnosti přecházet do dalších nabídek.

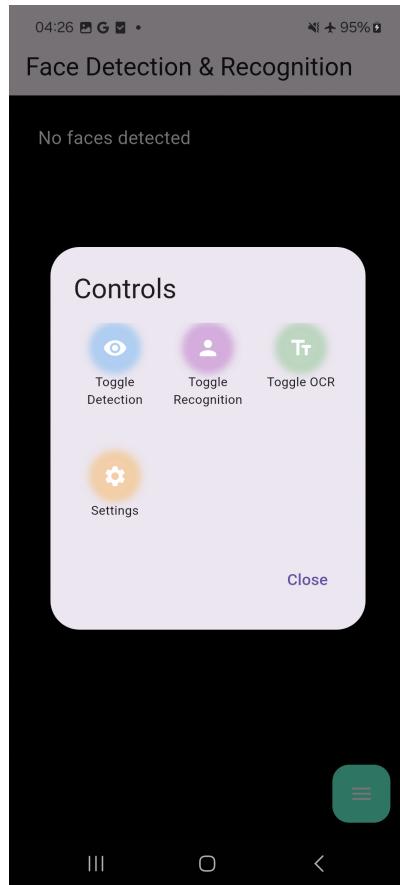
4.2 Kontextové menu

V pravém dolním rohu hlavní obrazovky se nachází ikona pro otevření kontextového menu. Po jejím stisknutí se uživateli zobrazí nabídka obsahující čtyři základní ovládací tlačítka:

- **Toggle Detection** – aktivuje nebo deaktivuje detekci obličejů,
- **Toggle Recognition** – přepíná rozpoznávání identit v detekovaných obličejích,
- **Toggle OCR** – povoluje nebo zakazuje optické rozpoznávání textu,

- **Settings** – otevře obrazovku s rozšířeným nastavením aplikace.

Každé tlačítko má jednoznačně definovanou funkci, která se projeví okamžitě po jeho stisknutí. Tím je zajištěna vysoká míra kontroly nad chodem aplikace a zároveň intuitivní ovládání i pro méně zkušené uživatele.



Obrázek 4.1: Menu aplikace

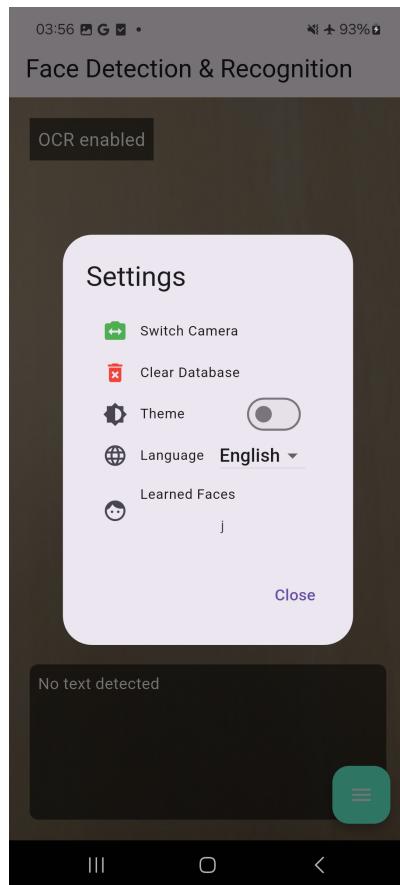
4.3 Obrazovka nastavení

Sekce Settings, přístupná z hlavní nabídky, poskytuje uživateli pokročilé možnosti konfigurace aplikace. Obsahuje následující ovládací prvky:

- **Přepnutí mezi přední a zadní kamerou** – umožňuje změnit zdroj video streamu podle situace nebo preferencí uživatele.
- **Vymazání databáze rozpoznaných obličejů** – tato funkce odstraní veškeré uložené embeddingy a odpovídající identifikace z paměti zařízení.

- **Přepnutí mezi světlým a tmavým režimem (Light/Dark mode)** – mění vzhled uživatelského rozhraní pro zajištění lepší čitelnosti a přizpůsobení světelným podmínekám.
- **Změna jazyka aplikace (CZ/EN)** – umožňuje přepínání mezi češtinou a angličtinou. Texty v uživatelském rozhraní se přepnou okamžitě po výběru jazyka.
- **Zobrazení seznamu naučených obličejů** – poskytuje přehled aktuálně rozpoznaných identit, včetně jejich jmen uložených uživatelem během používání aplikace.

Celkové uspořádání rozhraní je navrženo s ohledem na multiplatformní využití a možnost budoucího rozšíření. Zároveň umožňuje pohodlné ovládání jednou rukou, což je důležité především při práci v terénu.



Obrázek 4.2: Nastavení aplikace

5 Klíčové algoritmy aplikace

V této části se budeme zabývat klíčovými algoritmy, které tvoří základní stavební prvky naší aplikace. Zaměříme se na metody zpracování obrazu a strojového učení, které umožňují rozpoznávání obličejů, extrakci textu z obrazu a identifikaci objektů. Podrobně si vysvětlíme principy fungování těchto algoritmů, jejich teoretické základy a časovou složitost.

5.1 Přehled aplikace

- Detekce a rozpoznání obličeje pomocí knihovny **Google ML Kit Face Detection**.
- Rozpoznávání textu pomocí **Google ML Kit Text Recognition**.
- Zpracování video streamu přímo z kamery mobilního telefonu, na kterém běží aplikace.
- Přepínání režimů rozpoznávání (obličeje, text) prostřednictvím uživatelského prostředí
- Přepínání jazyků a různé režimy rozpoznávání.

Aplikace je implementována ve frameworku **Flutter** a využívá balíky, **Google_mlkit_face_detection**, **Google_mlkit_text_recognition**, **camera**, **image** a **OpenCV** pro efektivní manipulaci s obrazovými daty.

5.2 Detekce obličeje

V rámci mobilní aplikace byla implementována detekce obličejů v reálném čase za použití technologie Google ML Kit Face Detection, která umožňuje rozpoznávání lidských obličejů přímo na mobilním zařízení. Tato knihovna je navržena tak, aby

fungovala efektivně i bez připojení k internetu (on-device inference), čímž zajišťuje nejen nízkou latenci zpracování, ale také ochranu soukromí uživatelů – obrazová data nikdy neopouští zařízení.

5.2.1 Získávání vstupních dat z kamery

Pro přístup ke kameře mobilního zařízení a získání živého video streamu byl využit **Flutter** balíček camera ve verzi 0.11.0. Při spuštění aplikace dojde k inicializaci objektu **CameraController**, který je nakonfigurován s preferovanou kamerou – ideálně přední, pokud je dostupná. Kamera pracuje s rozlišením **ResolutionPreset.medium** a formátem obrazu **ImageFormatGroup.nv21**, který odpovídá běžnému YUV formátu. Jasová složka (Y) je uložena v plném rozlišení, zatímco chrominanční složky (U a V) jsou subsamplovány. Tento formát je efektivní z hlediska paměťové náročnosti a vhodný pro následné zpracování obrazu.

5.2.2 Zpracování oprávnění a spuštění streamu

Před inicializací kamery aplikace zajišťuje ověření oprávnění pomocí balíčku **permission_handler**. Pokud uživatel přístup ke kameře odepře, zobrazí se výzva nebo nabídka otevření nastavení zařízení. Po úspěšném získání oprávnění je kamera inicializována a spuštěn kontinuální obrazový stream pomocí metody **startImageStream()**, která zajišťuje přenos jednotlivých snímků typu **CameraImage** do dalšího zpracování.

5.2.3 Příprava snímků pro detekci

Snímků z kamery ve formátu **NV21** je potřeba převést na objekt typu **InputImage**, se kterým pracuje **ML Kit**. Tento převod je zajištěn funkcí **_convertCameraImageToBytes**, která spojí datové roviny Y, U a V do jednoho bajtového pole. Součástí převodu je i přidání metadat jako je rozlišení, orientace senzoru a formát. Zvláštní pozornost je věnována korekci orientace a zrcadlení, zejména u přední kamery, aby bylo zajištěno, že detekce obličejů bude přesná nezávisle na poloze zařízení.

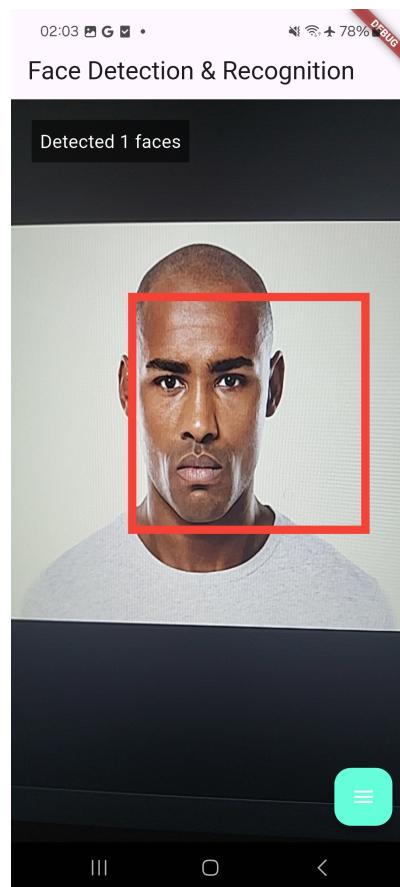
5.2.4 Princip fungování detekčního modelu **ML Kit**

Model detekce obličejů využívá hluboké konvoluční neuronové síťě trénované na rozsáhlých datasetech. Tyto síťě pracují ve více vrstvách – od nízkoúrovňové extrakce hran a tvarů až po vysoce abstraktní reprezentace rysů obličeje. Model obsahuje

i mechanismy jako Region Proposal Networks (RPN), které identifikují potenciální oblasti s výskytem obličeje, a následně se provádí regrese ohraničujících obdélníků (bounding boxes) a detekce klíčových bodů. Model je optimalizován pomocí kvantizace a může být akcelerován pomocí NNAPI nebo GPU, což zajišťuje plynulý chod i na méně výkonných zařízeních.

5.2.5 Výstupní data z detekce

Pro zlepšení kvality obrazu a přípravu na další analytické kroky je do projektu integrováno nativní zpracování obrazu pomocí OpenCV v Kotlinu. Prvním krokem je konverze obrazu z YUV do formátu Mat a následně jeho převedení na RGB (**COLOR_YUV2RGB_NV21**) a šedotónový obraz (**COLOR_RGB2GRAY**). Na výsledný obraz je následně aplikován Gaussův filtr pomocí **Imgproc.GaussianBlur()**, který pomáhá redukovat šum a zvýšit přesnost následného rozpoznávání. Toto zpracování je zvláště důležité v případě složitějších světelných podmínek nebo přítomnosti textur v pozadí.



Obrázek 5.1: Detekce obličeje

5.3 Rozpoznání obličeje

Zatímco detekce obličeje lokalizuje, kde se obličeje na obraze nachází, rozpoznávání obličeje (**face recognition**) se zaměřuje na identifikaci konkrétní osoby na základě unikátních rysů její tváře. Tento proces typicky zahrnuje porovnávání aktuálně detekovaného obličeje s databází známých identit. V rámci aplikace je využíván stejný ekosystém jako pro detekci – Flutter balíček `google_mlkit_face_detection`, `image` a `camera`, doplněný o OpenCV zpracování obrazu v Kotlinu pro předzpracování a vyrovnání snímků obličejů.

5.3.1 Příprava vstupního obrazu pro rozpoznávání

Jakmile je detekován obličeji, je z původního snímku oříznut příslušný výřez podle **boundingBox** hodnoty. Tento výřez je upraven do konzistentního rozměru (např. 112×112 pixelů) a orientace pomocí `face.headEulerAngleY` a `face.headEulerAngleZ`, aby byl obličeji zarovnán čelně. Zarovnání je důležité pro zajištění konzistence mezi různými snímky téže osoby. V Kotlinu je tento proces řešen pomocí OpenCV – detekovaný výřez se konvertuje do šedotónového obrazu a aplikuje se **Gaussian blur** pro odstranění šumu a zvýšení robustnosti rozpoznávání.

5.3.2 Extrahování embeddingu (vektorové reprezentace obličeje)

Google ML Kit sice nativně neposkytuje API pro rozpoznávání osob, ale je možné využít vnější knihovnu nebo si vlastní embedding generovat pomocí integrovaného modelu v zařízení. V praxi to znamená, že každému obličeji se přiřadí embedding – vektor čísel (pro zachování rychlosti aplikace je využito pouze 4 čísel z 128-512 možných), který zachycuje charakteristické rysy obličeje. Tento vektor je invariantní vůči změnám osvětlení, drobným výrazu tváře či úhlu pohledu. Tyto embeddingy se následně porovnávají pomocí metrik jako je kosinová podobnost nebo L2 norma.

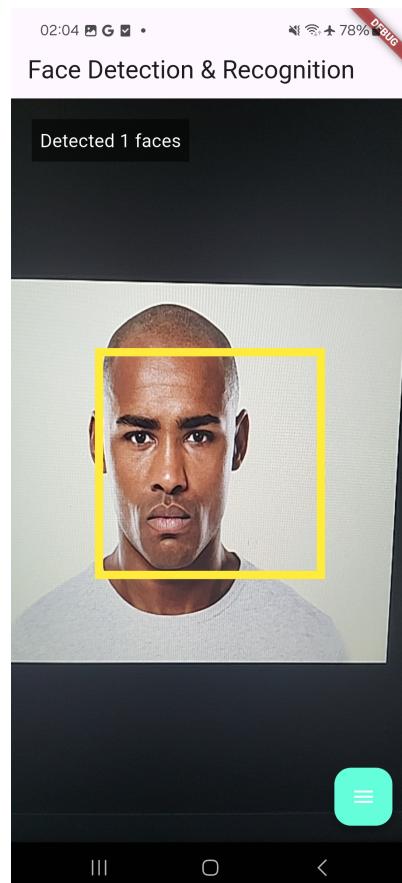
Následně je embedding porovnán s referenčními embeddingy uloženými v databázi. Porovnání probíhá pomocí výpočtu eukleidovské vzdálenosti, čímž se určí míra podobnosti. Pokud podobnost přesáhne stanovený práh, je obličeji považován za známý a přiřadí se mu jméno a pravděpodobnost rozpoznání.

Z hlediska vizualizace je každému detekovanému obličeji přiřazen barevně odlišený bounding box:

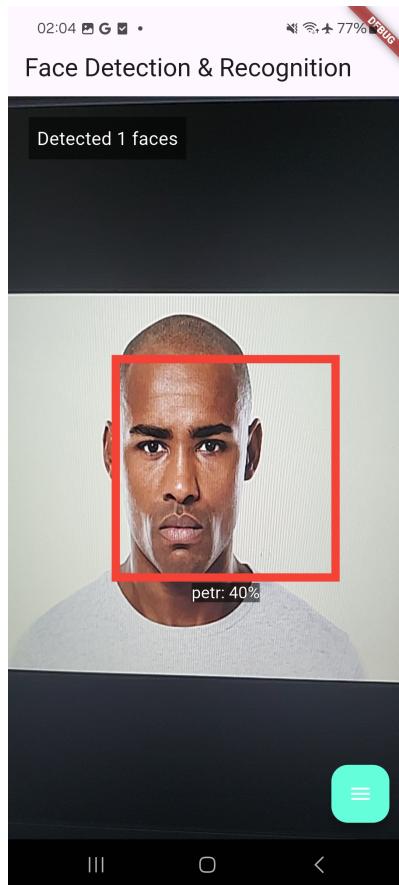
- Pokud obličeji není rozpoznán (embedding je výrazně odlišný od všech zná-

mých), ohraničení má žlutou barvu a aplikace nabízí uživateli možnost kliknout na box, čímž se otevře dialog pro ruční zadání jména. Po zadání se aktuální embedding uloží pod zadaným jménem do databáze.

- Pokud je obličej rozpoznán, bounding box je vykreslen červeně a v jeho dolní části se zobrazí jméno osoby spolu s procentuální pravděpodobností rozpoznání (např. Karel - 92%). Tímto způsobem je uživatel schopen vizuálně odlišit úspěšně rozpoznané osoby od těch, které v databázi ještě chybí.



Obrázek 5.2: Nerozpoznaný obličej



Obrázek 5.3: Rozpoznaný obličej

5.3.3 Správa databáze známých identit

Pro uchování referenčních údajů o uživatelích, jejichž obličej má být rozpoznán, aplikace využívá jednoduchý, ale efektivní způsob – uložení embeddingů do **JSON** souboru přímo na zařízení. Každému uživateli je přiřazeno unikátní jméno nebo identifikátor, pod kterým se ukládá jeho **embedding** – tedy vektor čísel, který reprezentuje rysy obličeje.

5.3.4 Optimalizace výkonu a real-time nasazení

Rozpoznávání obličeje je výpočetně náročnější než samotná detekce. Aby bylo možné tento proces provozovat v reálném čase, je nutné:

- Použít kvantizovaný model (např. tflite float16 nebo int8)
- Využít OpenCV pro vyčištění obrazu (blur, grayscale, ořez)

- Provádět rozpoznávání pouze při splnění podmínek (např. pokud je obličej dostatečně rovně a zepředu)

5.3.5 Zabezpečení a soukromí

Veškeré zpracování probíhá lokálně na zařízení, což zajišťuje, že žádné biometrické údaje nejsou odesílány na servery třetích stran. Data jsou navíc ukládána pouze na zařízení. Tento přístup zajišťuje jak soulad s principy **GDPR**, tak důvěru uživatelů v aplikaci.

5.3.6 Zpracování a porovnání embeddingů v Kotlinu (OpenCV)

Na nativní úrovni v Kotlinu se embeddingy mohou extrahovat pomocí předtrénovaných TensorFlow Lite modelů (např. FaceNet nebo MobileFaceNet), které lze integrovat přímo do Android aplikace. Vstupem je Mat obraz získaný z YUV streamu po převodu do RGB a oříznutí oblasti obličeje. Výstupem je pole float hodnot, které Kotlin předá zpět Flutteru přes platform channel. Porovnávání vektorů se pak může odehrávat buď v Kotlinu (nižší latence), nebo přímo ve Flutteru.

5.4 Rozpoznávání textu - OCR

Pro rozpoznávání textu v reálném čase je využit modul Text Recognition z balíku **google_mlkit_text_recognition**. Tato komponenta využívá hluboké neuronové sítě trénované na rozsáhlých datových sadách různých druhů tištěného a psaného textu, a je optimalizovaná pro běh přímo na zařízení. Text Recognition umožňuje extrahovat textové informace z obrazových dat, včetně rozdělení na bloky, řádky a jednotlivá slova s jejich přesnými souřadnicemi v obrazu.

5.4.1 Získání obrazových dat

Kamera je obsluhována prostřednictvím balíku **camera**, který poskytuje přímý přístup ke snímkům z hardwarové kamery zařízení ve formátu **NV21**. Tento formát, běžný na platformě Android, odděluje jasovou složku (Y) a chrominanční složky (U, V), což je efektivní pro zpracování obrazu a konverzi do podoby, se kterou ML Kit umí pracovat. Data ze snímku jsou pomocí funkce **_convertCameraImageToBytes** převedena do bajtového pole a následně zabalená do objektu **InputImage**, který obsahuje i metadata jako rozlišení obrazu, orientaci senzoru a formát.

5.4.2 Zpracování obrazu pomocí ML Kit Text Recognition

InputImage je následně předán do instance **TextRecognizer**, která provádí samotnou inferenci. Rozpoznávání probíhá plně na zařízení, čímž se zajišťuje nízká latence, možnost offline použití a také maximální ochrana soukromí – žádná data neopouštějí zařízení. Výstupem rozpoznávání je hierarchicky strukturovaný objekt **RecognizedText**, který obsahuje:

- textové bloky (paragraph-like celky),
- řádky a jednotlivé segmenty textu,
- přesný rozpoznaný text (včetně interpunkce a diakritiky).

Rozpoznaný text je výsledně vypsán do **Containeru**, kde se vypíše podle rozpoznaných výsledků.



Obrázek 5.4: Ukázka OCR

5.4.3 Předzpracování obrazu pomocí OpenCV v Kotlinu

Pro zajištění vyšší kvality rozpoznání je obraz před zpracováním optimalizován v Kotlin části aplikace pomocí OpenCV. Tam dochází k:

- převodu YUV dat na strukturu Mat (OpenCV matice),
- konverzi obrazu do odstínů šedi, čímž se sníží komplexita dat a zvýší kontrast textu vůči pozadí,
- aplikaci Gaussovského rozostření, které potlačí šum a nerovnosti, čímž se zlepší konzistence písma a odstraní se rušivé pozadí.

Tato příprava výrazně zvyšuje úspěšnost detekce zejména u slabě osvětlených, rozmazených nebo jinak znehodnocených snímků.

5.5 Flutter framework

Flutter je moderní open-source UI framework vyvinutý společností Google, který umožňuje vývoj nativně komplikovaných aplikací pro více platform – především pro Android, iOS, Web, Windows, macOS a Linux – z jediného společného kódu napsaného v jazyce **Dart**. Jeho hlavní předností je schopnost vytvářet vysoce výkonné, vizuálně bohaté aplikace s nativním vzhledem a chováním při zachování plné kontroly nad renderovacím procesem.

5.5.1 Architektura a základní principy

Flutter se odlišuje od jiných multiplatformních nástrojů tím, že nevyužívá nativní komponenty cílového systému, ale veškeré uživatelské rozhraní vykresluje prostřednictvím vlastního grafického jádra **Skia**. To znamená, že každá aplikace vykresluje svůj vzhled nezávisle na platformě a zajišťuje tak konzistentní vizuální výstup a chování bez ohledu na cílový operační systém. Tento přístup eliminuje potřebu použití mostů (bridge) mezi nativním a sdíleným kódem, jak je tomu např. v React Native.

Architektura Flutteru je vrstvená a skládá se z následujících částí:

- **Dart Virtual Machine** – běhové prostředí pro vývoj a testování aplikací s podporou tzv. hot reloadu.
- **Flutter engine** – jádro napsané v jazyce C++, které obstarává vykreslování přes **Skia**, práci s grafikou, animacemi, vstupy uživatele či přístup k platformně specifickým API.

- **Foundation library** – poskytuje základní sadu tříd a metod pro zpracování asynchronních operací, rozvržení widgetů a správu stavu.
- **Widget layer** – nejvyšší vrstva, ve které vývojář definuje uživatelské rozhraní pomocí stromové struktury widgetů.

5.5.2 Programovací jazyk Dart

Jazyk **Dart** je objektově orientovaný a silně typovaný jazyk, který byl navržen s důrazem na rychlý běh a přehlednost při tvorbě uživatelských rozhraní. Dart podporuje jak just-in-time (JIT) kompliaci pro rychlý vývoj, tak ahead-of-time (AOT) kompliaci pro produkční buildy. Výhodou JIT režimu je schopnost okamžité aktualizace běžící aplikace bez nutnosti jejího restartu, což je známé jako hot reload. AOT režim naopak poskytuje optimalizovaný binární kód, který je efektivně spustitelný přímo na cílovém zařízení.

5.5.3 Widgety a deklarativní přístup k UI

Flutter staví na principu deklarativního návrhu uživatelského rozhraní. Každý vizuální prvek aplikace je reprezentován tzv. widgetem. Widgety mohou být buď stateless (neměnné), nebo stateful (reagující na změny stavu). Uživatelské rozhraní je pak tvořeno jako hierarchický strom těchto widgetů. Tento přístup přispívá k modularitě a snadné údržbě aplikace.

Díky možnosti vnořování widgetů je možné vytvářet složité uživatelské rozhraní s minimální námahou a vysokou mírou přizpůsobení. Flutter poskytuje widgety odpovídající stylu **Material Design** (pro Android) i Cupertino (pro iOS), čímž se zachovává přirozený vzhled aplikace na dané platformě.

5.5.4 Multiplatformní vývoj a podpora nativních funkcí

Hlavní výhodou Flutteru je schopnost vytvořit jednu kódovou základnu, kterou lze přeložit a spustit na různých cílových platformách bez potřeby úprav. V případech, kdy je třeba využít specifických funkcí cílové platformy (např. přístup ke kameře, GPS nebo práci s nativním API systému Android/iOS), Flutter nabízí tzv. platform channels – mechanismus pro komunikaci mezi Dart kódem a nativními knihovnami napsanými například v Kotlinu (pro Android) nebo Swiftu (pro iOS).

Tato architektura dovoluje kombinovat výhody sdíleného kódu s možností hlubší integrace do systému, pokud je to třeba. Vývojáři mohou rovněž využívat pluginy

dostupné prostřednictvím balíčkovacího systému **pub.dev**.

5.5.5 Rozšíření a integrace knihoven třetích stran

Flutter disponuje rozsáhlým ekosystémem balíčků a knihoven, které pokrývají běžné potřeby vývoje aplikací. Mezi využívané knihovny patří například:

- **camera** - přímá práce s kamerou zařízení a získávání obrazového vstupu.
- **image** - zpracování bitmapových obrázků.
- **google_mlkit_face_detection** - detekce obličeje pomocí knihovny Google ML Kit.
- **google_mlkit_text_recognition** - optické rozpoznávání textu (OCR) pomocí ML Kit.
- **permission_handler** - správa systémových oprávnění (kamera, úložiště, atd.).

Díky těmto knihovnám je možné v aplikacích jednoduše realizovat pokročilé funkce, jako je například rozpoznávání obličejů, detekce textu, QR kódů, nebo integrace OCR, a to bez nutnosti psaní nativního kódu.

5.5.6 Výkon a efektivita

Flutter dosahuje vysokého výkonu díky AOT komplikaci do nativního kódu a vlastním mechanismům vykreslování. Oproti jiným frameworkům, které se spoléhají na přemostění mezi jazykem aplikace a nativní vrstvou (např. React Native), poskytuje Flutter nižší latenci a vyšší stabilitu. Animace, přechody a interakce jsou zpracovávány s vysokou snímkovou frekvencí (60–120 FPS), což přispívá k plynulému uživatelskému zážitku.

5.5.7 Vývojové nástroje a build

Vývoj aplikací ve Flutteru je možné provádět pomocí nástrojů jako Visual Studio Code nebo Android Studio, které poskytují podporu pro Dart, integrované debegování, emulaci zařízení a možnost testování na reálném hardware. Build systém pro Android využívá Gradle, pro iOS pak Xcode.

Distribuce výsledných aplikací je možná ve formě **.apk** (Android), **.ipa** (iOS) nebo jako webová aplikace či desktopový program. Flutter také podporuje CI/CD

integraci a automatizované testování, což jej činí vhodným pro profesionální vývoj v rámci větších týmů.

5.6 OpenCV – zpracování obrazu

V rámci této aplikace byla použita knihovna OpenCV (Open Source Computer Vision Library), která poskytuje bohaté nástroje pro práci s obrazem a videem. Jedná se o otevřenou multiplatformní knihovnu vyvíjenou pod licencí BSD, jež nabízí více než 2500 optimalizovaných algoritmů pro zpracování obrazu, počítačové vidění a strojové učení.

5.6.1 Integrace OpenCV v prostředí Flutter

Jelikož Flutter nepodporuje přímé použití OpenCV, bylo nutné využít nativní kód v jazyce Kotlin (pro Android), do kterého se předává obrazová data z Dart vrstvy. Komunikace mezi Flutterem a Kotlinem je realizována pomocí tzv. **platformních kanálů (platform channels)**, což umožňuje efektivní předávání dat a volání funkcí mezi těmito vrstvami.

Obraz získaný z kamery prostřednictvím balíčku **camera** je nejčastěji v barevném prostoru YUV420, což je formát, který je výhodný pro kompresi a práci s videem, avšak nevhodný pro některé algoritmy OpenCV, které očekávají vstup ve formátu Mat.

5.6.2 Převod YUV na Mat

Prvním krokem je převod dat z formátu YUV do struktury Mat, kterou používá OpenCV. Tento převod zahrnuje:

- Extrakci jednotlivých kanálů Y (jas), U a V (barevné informace).
- Jejich následnou transformaci na barevný obraz v prostoru **RGB**.
- Vytvoření instance **Mat** s odpovídajícími rozměry a typem (např. CvType.CV_8UC3 pro **RGB**).

5.6.3 Předzpracování obrazu

Na připraveném obrazu lze provádět různé předzpracovatelské operace pro zvýšení kvality následné detekce nebo OCR. Mezi nejčastěji používané patří:

- **Převod do odstínů šedi** - snižuje datovou náročnost a zvýrazňuje kontrastní hrany.
- **Gaussovské rozmazání (Gaussian blur)** - potlačuje šum a jemné detaily, které by mohly rušit detekční algoritmy.

5.6.4 Optimalizace výkonu

Vzhledem k tomu, že operace nad obrazovými daty jsou výpočetně náročné, je třeba přistupovat k jejich implementaci efektivně. Použití nižších rozlišení pro analýzu a využití nativního běhu na ARM procesoru zařízení zajišťuje, že aplikace zachovává plynulost i při reálném čase zpracování obrazu z kamery.

Převody obrazových dat a operace OpenCV probíhají výhradně na straně Kotlinu, zatímco Dart vrstva nadále obstarává logiku rozhraní a interakci s uživatelem. Toto dělení umožňuje využívat maximální výkon zařízení bez negativního dopadu na uživatelský zážitek.

5.6.5 Shrnutí

OpenCV v této aplikaci slouží jako základní nástroj pro předzpracování obrazu před jeho dalším využitím, ať už se jedná o detekci obličeje, optické rozpoznávání textu (OCR) nebo jinou analýzu. Díky propojení s Flutterem přes nativní Kotlin kód je možné využít robustních funkcí knihovny OpenCV i v moderním multiplatformním prostředí.

6 Závěr

Tato bakalářská práce se zaměřila na návrh, vývoj a testování mobilní aplikace pro zpracování a rozpoznávání obrazu v reálném čase. Výsledná aplikace využívá framework Flutter ve spojení s nativním kódem v jazyce Kotlin a knihovnami Google ML Kit a OpenCV. Hlavním cílem bylo vytvořit multiplatformní řešení schopné efektivní detekce obličejů a optického rozpoznávání textu (OCR), a to s důrazem na rychlosť, stabilitu a ochranu osobních údajů.

Aplikace naplnila všechny stanovené požadavky. Běží stabilně, bez zjevných chyb, a umožňuje detektovat i rozpoznávat obličeje či text v reálném čase přímo na zařízení, bez nutnosti internetového připojení. Veškeré zpracování probíhá lokálně, čímž je zajištěna vysoká úroveň bezpečnosti a soukromí. Díky využití nativního předzpracování obrazu prostřednictvím OpenCV a optimalizovaných modelů ML Kit dosahuje aplikace velmi dobré výkonnosti i na zařízeních střední třídy.

6.1 Hodnocení použitého frameworku

V průběhu vývoje byly vyzkoušeny čtyři různé frameworky: Kivy, Briefcase, Flet a Flutter. Kivy se ukázal jako velmi nestabilní – komplikace často selhávala, vývoj byl pomalý a nespolehlivý. Briefcase byl využit pouze okrajově, bez výrazného praktického nasazení. Flet se osvědčil při tvorbě jednodušších aplikací, avšak nedokázal pokrýt komplexnější algoritmy a interakci s kamerovým vstupem. Flutter se nakonec ukázal jako nejvhodnější volba. Poskytl stabilní multiplatformní základ, moderní nástroje pro tvorbu uživatelského rozhraní, širokou podporu knihoven a možnost integrace s nativním kódem. Díky těmto vlastnostem umožnil rychlý a efektivní vývoj finální aplikace.

6.2 Srovnání s existujícími řešeními

Aplikace byla rovněž srovnána s několika současnými nástroji dostupnými na trhu:

- **Google Lens** nabízí široké možnosti rozpoznávání objektů, textu a překladů, ale spoléhá se na cloudové služby, což může být nevýhodou z hlediska ochrany soukromí. Moje aplikace funguje plně offline.
- **Microsoft Seeing AI** poskytuje asistivní funkce pro nevidomé, včetně čtení textu či identifikace osob, avšak není snadno rozšířitelná pro jiné účely a opět vyžaduje internetové připojení.
- **CamScanner** a **Adobe Scan** se specializují na OCR, ale neumožňují detekci obličejů a nejsou vhodné pro práci s živým obrazem z kamery.
- **Snapchat / Instagram** využívají detekci obličejů pro filtry, nikoliv však pro rozpoznávání nebo analytické účely.

Oproti těmto řešením má vyvinutá aplikace výhodu v kombinaci několika funkcí do jednoho celku, schopnosti pracovat offline a jednoduché možnosti rozšíření o nové algoritmy nebo databáze.

6.3 Přínos a využití

Aplikace má široký aplikační potenciál:

- **Bezpečnost** – přístupové systémy založené na rozpoznávání obličeje.
- **Asistence** – OCR funkce pro osoby se zrakovým omezením.
- **Firemní nasazení** – docházkové systémy nebo evidence osob.
- **Digitalizace** – převod tištěných dokumentů do digitální podoby.
- **Logistika** – čtení údajů z etiket, formulářů nebo štítků.

Díky multiplatformnímu jádru postavenému na Flutteru je aplikace snadno přenositelná mezi Android, iOS i desktopové prostředí.

6.4 Možnosti dalšího rozvoje

Další rozvoj aplikace může směřovat do několika oblastí:

- Integrace rozpoznávání emocí, věku a pohlaví.
- Podpora rozpoznávání objektů, SPZ, QR kódů a dalších vizuálních prvků.

- Přidání překladového modulu pro OCR výstupy.
- Možnost učení na zařízení a adaptivního vylepšování modelu podle zpětné vazby.
- Využití rozšířené reality pro interaktivní vizualizaci detekovaných informací.
- Cloudová synchronizace embeddingů a databází pro více zařízení.

6.5 Shrnutí

Práce prokázala, že moderní technologie jako Flutter, ML Kit a OpenCV lze efektivně kombinovat pro tvorbu výkonných mobilních aplikací v oblasti počítačového vidění. Výsledná aplikace je funkční, efektivní a bezpečná, připravená pro reálné nasazení i další vývoj. Z hlediska přínosu práce byla získána praktická zkušenosť s výběrem vývojových nástrojů, optimalizací algoritmů pro mobilní platformy a návrhem uživatelsky přívětivého softwarového řešení. Aplikace se tak může stát nejen základem pro další výzkum či komerční nasazení, ale také ukázkou možnosti efektivního nasazení umělé inteligence přímo v mobilních zařízeních bez nutnosti externí infrastruktury.

Použitá literatura

- [1] BENNETT, Michael. *Scalable Android Applications in Kotlin: Write and maintain large Android application code bases*. BPB Publications, 2024. ISBN 978-9365899276.
- [2] BEYELER, Michael. *OpenCV 4 with Python Blueprints: Build creative computer vision projects with OpenCV 4 and Python 3*. Birmingham: Packt Publishing, 2019. ISBN 978-1789344912.
- [3] BIESSEK, Alessandro. *Flutter Complete Reference*. Packt Publishing, 2021. ISBN 978-1800565999.
- [4] DARWIN, Ian. *Android Cookbook: Problems and Solutions for Android Developers* [online]. 2. vyd. Sebastopol: O'Reilly Media, 2017 [cit. 2024-05-10]. ISBN 978-1491974056.
- [5] FLUTTER DEV TEAM. *Flutter Documentation* [online]. [cit. 2025-05-07]. Dostupné z: <https://docs.flutter.dev>. Oficiální dokumentace frameworku Flutter.
- [6] FLUTTER TEAM. *Flutter GitHub Repository* [online]. 2025. [cit. 2025-05-07]. Dostupné z: <https://github.com/flutter/flutter>. Zdrojový kód a problémy Flutter SDK.
- [7] GONZALEZ, Rafael C. a Richard E. WOODS. *Digital Image Processing*. Global edition. New York: Pearson, 2017. ISBN 978-1-292-22304-9.
- [8] GOOGLE DEVELOPERS. *ML Kit / Google for Developers* [online]. [cit. 2025-05-07]. Dostupné z: <https://developers.google.com/ml-kit>. Oficiální přehled ML Kit API pro Android/iOS.
- [9] HORTON, John. *Android Programming for Beginners: Build in-depth, full-featured Android apps starting from zero programming experience*. Packt Publishing, 2021. ISBN 978-1800563438.
- [10] MARTIN, D. *Learn Flutter and Dart to Build iOS and Android Apps*. New York: Apress, 2021. ISBN 978-1484262306.

- [11] OPENCV TEAM. *OpenCV Documentation* [online]. [cit. 2025-05-07]. Dostupné z: <https://docs.opencv.org/4.x/>. Kompletní API a tutoriály pro OpenCV 4.x.
- [12] SZELISKI, Richard, Andrew ZISSERMAN a David A. FORSYTH. *Computer Vision: Algorithms and Applications*. 2. vyd. Cham: Springer, 2022. ISBN 978-3030343729.
- [13] ŠONKA, Milan, Václav HLAVÁČ a Roger BOYLE. *Image Processing, Analysis, and Machine Vision*. 4. vyd. Australia: Cengage Learning, 2015. ISBN 978-1-133-59369-0.

Seznam obrázků

4.1	Menu aplikace	24
4.2	Nastavení aplikace	25
5.1	Detekce obličeje	28
5.2	Nerozpoznaný obličej	30
5.3	Rozpoznaný obličej	31
5.4	Ukázka OCR	33

Odkaz na GitHub repozitář

Všechny zdrojové kódy této aplikace můžete naleznete v GitHubovém adresáři:

https://github.com/JakubKacerek/BP_TUL_25

Při jazykové a stylistické úpravě textu této bakalářské práce jsem využil asistenčního nástroje umělé inteligence ChatGPT 4.0 (OpenAI) a Grok 3.0 (xAI). Tyto nástroje byly použity výhradně pro formulaci a zpřesnění již existujícího obsahu vytvořeného autorem, nikoliv pro generování odborného obsahu nebo analýz.