

Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta



Rozpoznávání směru uchopení těles

Bakalářská práce

Viktor Andrusyak

Vedoucí práce: Ing. Václav Novák CSc.

České Budějovice 2023



Přírodovědecká fakulta
Faculty of Science
Jihočeská univerzita
v Českých Budějovicích
University of South Bohemia
in České Budějovice

ZADÁVACÍ PROTOKOL BAKALÁŘSKÉ/DIPLOMOVÉ¹ PRÁCE

Student: Viktor ANDRUSYAK

Program studia/specializace: Aplikovaná informatika - specializace Embedded systémy

Pracoviště PŘF JU, kde bude práce vypracována a obhájena: Ústav aplikované informatiky

Školitel: Ing. Václav Novák CSc., vacnovak@prf.jcu.cz

Garant z PŘF:

Školitel – specialista, konzultant:

Téma práce: Rozpoznávání směru uchopení těles

Práce má vyřešit směr uchopování těles - kvádrů, válců a dalších těles z dopravníku. Pro manipulaci s nimi je potřeba nastavit správný úhel manipulačních kleští. Tím se značně omezí vypadávání výrobků při přenášení a manipulaci.

Cíle práce:

Student navrhne rozpoznávací systém tvarů těles pomocí Artificial Intelligence (AI) pro možnost technologie uchopení těchto těles robotickým podavačem na pohybujícím se povrchu.

1. Navrhne kamerový řetězec, který pomocí umělé inteligence rozpozná tvar na běžícím pásu.
2. Vyhodnotí možnost uchopení pohybujícího se tělesa a předá informaci manipulátoru.
3. Provede testy jednak rychlosti rozpoznání, ale i věrohodnosti informací pro uchopení.

Pro systém je doporučeno použití mikropočítačů STM32 a příslušných knihoven (X-CUBE-AI, FP-AI-SENSING1, a dalších).

Základní doporučená literatura:

- [1] - STM32 solutions for Artificial Neural Networks [online]. USA: STMicroelectronics, 2021 [cit. 2021-8-23]. Dostupné z: https://www.st.com/content/st_com/en/ecosystems/stm32-ann.html
- [2] - PATTERSON, Josh a Adam GIBSON. Deep learning: a practitioner's approach. Sebastopol, CA: O'Reilly, 2017. ISBN 9781491914250.

¹ Nehodící se škrtněte/smažte



Přírodovědecká fakulta
Faculty of Science
Jihočeská univerzita
v Českých Budějovicích
University of South Bohemia
in České Budějovice

Zvláštní poznámky pracoviště:

Financování práce:

Školitel práce

podpis :
V. L. S.

U externích vedoucích fakultní garant práce

podpis :

Garant programu/specializace²³

podpis :
[Signature]

Vedoucí pracoviště PŘF JU, kde proběhne obhajoba

podpis :

Souhlas vedoucího ústavu AV nebo jiné instituce⁴

podpis :

V Českých Budějovicích dne 13.10.2021

podpis studenta:
[Signature]

² v případě prací v bakalářském programu Biologie není podpis garanta programu vyžadován

³ v případě magisterských prací v programech učitelství pro SŠ podpis proděkana pro učitelské obory

⁴ v případě, že práce bude vypracována jinde než v prostorách PŘF, například na ústavu AV

Bibliografické údaje

Andrusyak V., 2023: Rozpoznávání směru uchopení těles. [Recognition of the direction of grasping bodies. Bc. Thesis, in Czech.] – 42 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Annotation

The aim of bachelor thesis is to analyse and design a body shape recognition system using artificial intelligence. This system could be used for grasping these bodies by a robotic feeder on moving surface. The system should operate completely autonomously on the basis of input data obtained from the camera. Testing of the proposed system is also part of the bachelor thesis.

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne 13.4.2023

.....

Viktor Andrusyak

Poděkování

Rád bych poděkoval Ing. Václavu Novákovi, CSc. za odbornou pomoc, vstřícnost, důvěru a trpělivost při vedení této práce. Děkuji také mé přítelkyni za podporu a obětavost, bez které by psaní práce nebylo tak snadné.

Obsah

Úvod	1
1 Mikrokontroléry a umělá inteligence	2
1.1 Strojové učení	2
1.1.1 Typy strojového učení	2
1.1.2 Konvoluční neuronové sítě	3
1.1.3 Doporučený hardware	4
1.2 STM32H747I-DISCO	5
1.2.1 Architektura a procesory	6
1.2.2 Paměti	7
1.2.3 Napájení	7
1.2.4 B-CAMS-OMV	8
1.2.5 Vývojové prostředí a programování	8
2 Metodika rozpoznávání obrazu	11
2.1 Výběr architektury	11
2.1.1 MobileNet	12
2.2 Předzpracování dat	13
2.2.1 Augmentace	13
2.2.2 Normalizace	15
2.3 Trénování modelu	15
2.3.1 Transfer learning	15
2.3.2 TensorFlow	16
2.3.3 Keras	16
2.4 Optimalizace modelu	17
2.4.1 Kvantizace	17
2.4.2 X-CUBE-AI	18
3 Návrh a implementace systému	19
3.1 Fyzické propojení	19
3.1.1 Aktualizace firmwaru	20
3.2 Příprava trénovacích dat	21
3.2.1 Nahrání potřebné aplikace	21
3.2.2 Získávání dat	21
3.2.3 Augmentace	23

3.2.4	Normalizace	24
3.3	Trénování modelu	24
3.3.1	Základní testování přesnosti	25
3.4	Optimalizace	26
3.4.1	Kvantizace	26
3.4.2	STM32CubeMX	26
3.5	Vývoj aplikace	28
3.5.1	Programování	28
3.5.2	Výstup aplikace	29
3.6	Testování funkčnosti	29
4	Testování a analýza	31
4.1	Přesnost	31
4.2	Rychlost	32
4.3	Použití jiných nástrojů	32
4.4	Plánovaný rozvoj	33
	Závěr	34
	Seznam použité literatury	35
	Seznam obrázků	39
	Seznam použitých zkratk	40
	Seznam tabulek	41
	Přílohy	42

Úvod

Za poslední desetiletí zájem o strojové učení prudce stoupá, a to ve všech odvětvích. Příklady tohoto učení se objevují jak ve vědě, tak i v průmyslu. V zásadě, strojové učení používá určitý algoritmus k získání informace z počátečních dat a následně jejich reprezentaci v určitém modelu. Tento model pak můžeme použít pro odvozování dalších výsledků z dat, které jsme dosud nezpracovali. [1]

Takový proces učení a následná implementace naučeného modelu vždy vyžadovala relativně vysoké hardwarové prostředky. Avšak dnes, po prudkém nárůstu výpočetního výkonu, tato záležitost nachází místo i v kompaktnějších řešeních, jako například mikrokontroléry.

Tématem této práce je návrh a implementace rozpoznávacího systému tvarů těles pomocí umělé inteligence, který zcela autonomně funguje na jednom z vybraných mikrokontrolérů. Tento systém by mohl být využit pro uchopení těles robotickým podavačem na pohyblivém povrchu. Systém funguje na základě vstupních dat, získaných z kamerového řetězce, který pomocí umělé inteligence rozpozná tvar na běžícím pásu a přepošle tuto informaci pro nastavení správného úhlu manipulačních kleští.

První část práce obsahuje rešerši na základní témata, potřebná pro efektivní tvorbu zadaného systému. Jedná se o stručnou analýzu umělé inteligence v práci s mikrokontrolérem, popis možných algoritmů a analýza doporučeného hardwaru. Další neméně důležitou částí je průzkum řešení zpracování obrazu, potřebného pro implementaci systému.

Druhá část práce obsahuje praktický návrh systému, spojení všech jeho součástí a popis jeho funkčnosti. Nejdůležitější kroky této části jsou modelování systému, příprava testovacích dat, trénování umělé inteligence, závěrečné testování, včetně měření rychlosti a přesnosti.

Práce řeší hned několik problémů různých typů. Jedná se především o zprovoznění funkční umělé inteligence, která dokáže ve spojení s mikrokontrolérem dostatečně rychle a přesně vyhodnotit možnost uchopení pohyblivého se tělesa a následně tuto informaci přeposlat. Dále řeší správné nastavení připojení a komunikaci mezi mikrokontrolérem a jinými zařízeními, např. kamerou, která má za úkol generovat vstupní data pro umělou inteligenci. Na závěr je v práci provedeno testování výsledného modelu pro určování efektivity celého systému.

1. Mikrokontroléry a umělá inteligence

S rychlým vývojem hardwaru a softwaru různého typu se Internet věcí (IoT) začíná široce používat v našem každodenním životě. Vzhledem k tomuto vývoji se naše požadavky na tuto technologii mění a dále rostou. [2]

Dle výzkumu [3] z roku 2022 se předpokládá, že počet zařízení IoT na celém světě se téměř ztrojnásobí z 9,7 miliardy v roce 2020 na více než 29 miliard v roce 2030. Používají se ve všech typech průmyslů a také na spotřebitelských trzích, přičemž spotřebitelský segment v roce 2020 představoval přibližně 60 % všech připojených zařízení. Předpokládá se, že tento podíl zůstane na této úrovni i v příštích letech.

Tyto zařízení s pomocí IoT senzorů pořád generují data, které je třeba určitým způsobem zpracovat. Pro tento účel se stává běžnějším používat strojové učení, které se úspěšně projevuje v počítačovém vidění, zpracování přirozeného jazyka a analýze velkých dat. [4]

1.1 Strojové učení

Strojové učení je vyvíjející se odvětví výpočetních algoritmů, které jsou navrženy tak, aby napodobovaly lidskou inteligenci učním se od přesně definovaných vstupních dat. [5] Prostřednictvím strojového učení, které je odvíjeno od umělé inteligence, se počítačové systémy učí provádět úkoly, jako například klasifikace, předpovědi, shlukování, rozpoznávání vzorů a jiné. Vzorky dat jsou obvykle charakterizované měřitelnými charakteristikami a algoritmus strojového učení se pokouší najít korelaci mezi těmito charakteristikami a některými výstupními hodnotami. Informace získané ve fázi trénování se používají k identifikaci vzorů nebo k rozhodování na základě nových dat. [6] Jedná se o učení, s jehož pomocí můžeme navrhnout logiku rozpoznávání těles na téměř libovolném koncovém zařízení. Pro přesnější pochopení dané problematiky je třeba rozebrat na jaké základní typy se strojové učení rozděluje.

1.1.1 Typy strojového učení

Strojové učení lze rozdělit [6] dle stylu učení na čtyři hlavní typy:

- Učení pod dohledem - proces učení je rozdělen do dvou kroků: trénování a testování. V procesu trénování jsou vzorky v trénovacích datech brány jako vstupní množina pro trénovací algoritmus, který následně generuje učící se model. V procesu testování se tento model používá k predikci pro testovací nebo produkční data. [7] Učení pod dohledem se obvykle zabývá problémy zahrnujícími regresi, jako jsou například

předpověď počasí nebo předpověď růstu populace, a to pomocí algoritmů, jako je lineární regrese nebo rozhodovací strom. [6]

- Učení bez dohledu - v této kategorii se systému poskytují data, která neobsahují informace o cílové proměnné nebo výstupu, který by měl být naučen. Cílem učení bez dohledu je najít přirozené shluky, vzory, závislosti nebo latentní reprezentace dat, aniž by bylo třeba ručně definovat vstup nebo výstup. Je užitečné v případech, kdy máme velké množství dat a nevíme, jakým způsobem je strukturovat. Tento přístup umožňuje odhalit skryté vztahy a vzorce v datech, které by jinak mohly zůstat skryté. [8]
- Učení s polodohledem - kategorie, ve které původní data jsou částečně označeny a částečně neoznačeny. Cílem je naučit stroj rozpoznávat vzory a struktury v datech a předpovídat označení pro neoznačené příklady. Obvykle se tato metoda používá v případech, kdy je označení dat nákladné nebo je nemožné je získat v dostatečném množství. [9] Je to kombinace učení pod dohledem a bez dohledu.
- Posilovací učení - se snaží naučit model jak maximalizovat odměnu, která je spojena s určitými akcemi v určitém prostředí. Model nezahrnuje odpověď, ale seznam pravidel a konečných stavů. V průběhu učení aktualizuje svou strategii na základě zpětné vazby od prostředí v podobě odměn a nových stavů. Posilovací učení se používá například v robotice, řízení dopravy nebo předvídaní poptávky. [10]

Ze základních kategorií strojového učení se pro rozpoznávání těles na mikrokontrolérech nejčastěji používá učení pod dohledem. Tato kategorie umožňuje trénovat klasifikační modely na datasetech, kde jsou jednotlivým třídám přiřazeny konkrétní klasifikátory, což je u rozpoznávání obrázků běžně využívaný způsob. Konkrétně pro tento úkol se často používají konvoluční neuronové sítě. Tyto sítě jsou speciální typy neuronových sítí a jsou efektivní v extrahování rysů z obrázků a jsou schopny rozpoznat vzorce v datech, což je klíčové pro rozpoznávání obrázků. [11]

1.1.2 Konvoluční neuronové sítě

Konvoluční neuronová síť (CNN) je speciálním případem hluboké neuronové sítě. Algoritmus pro predikci pomocí učení se skládá z několika vrstev. Při odvozování pomocí hlubokého učení vstupní data procházejí postupně přes vrstvy a v každé vrstvě se provádí násobení matic. Výstup vrstvy je obvykle vstupem do následující vrstvy. Poté, co jsou data zpracována poslední vrstvou, je výstupem buď příznak nebo výstup klasifikace. V konvoluční neuronové síti násobení matic zahrnuje konvoluční filtrové operace, což je běžné u neuronové sítě, která je navržena pro analýzu obrazu a videa (Obrázek 1.1). [4]



Obrázek 1.1: Rozpoznávání obrazu pomocí CNN [1]

CNN je významnou součástí v oblasti strojového vidění, což má zřejmé využití pro samonávodící automobily, robotiku nebo bezpilotní letouny. [1] Z tohoto důvodu, použití CNN je primární způsob zpracování obrazu u mikrokontrolérů. Avšak, i pro optimalizované modely hlubokých neuronových sítí je třeba vybrat správný hardware.

1.1.3 Doporučený hardware

Optimální zpracování problematiky rozpoznávání těles je úzce spojeno s výběrem příslušného hardwaru. Parametry, o kterých je třeba uvažovat, jsou založeny na přesnosti, spotřebě energii, výkonu a nákladech určitých mikrokontrolérů.

Přesnost algoritmu strojového učení musí být měřená na dostatečně velkém souboru dat, aby bylo možné potvrdit, že získaný výsledek je platný.

Energetická účinnost zase úzce souvisí s velikostí neuronové sítě, popřípadě s její optimalizací. Velikostí neuronové sítě rozumíme počet vrstev, který procesor musí zpracovávat. V případě zvýšení velikosti sítě se zvětšuje výpočetní náročnost a počet dat, ke kterým je třeba přistupovat. Toto obecně znamená vyšší energetickou náročnost. [12] Je tedy třeba zvažovat na mikrokontroléru, který by měl dostatečně velkou paměť pro ukládání modelu a mohl být přizpůsoben k výkonu, optimalizovanému pro zpracování obrazu pomocí algoritmů umělé inteligence.

Implementace takových algoritmů pro mikrokontroléry může být náročná. Mikrokontrolér STM32, který byl doporučený vedoucím práce, má připravený ekosystém pro zpra-

cování úkolů daného typu. Například, dobrým nástrojem, usnadňujícím implementaci neuronové sítě na mikrokontroléru je knihovna X-CUBE-AI, vhodná pouze pro zařízení společnosti STMicroelectronics. Jedná se o rozšíření STM32CubeMX prostředí, které umožňuje automatickou konverzi předem natrénovaných dat na hardware s nižší výpočetním výkonem. [12]

Mikrokontroléry od společnosti STMicroelectronics jsou k dispozici v různých variantách, od jednoduchých 8bitových mikrokontrolérů až po výkonné 32bitové. Tyto výkonné varianty fungují na bázi procesoru Arm Cortex-M a dle výrobce [13] to nabízí velmi vysoký výkon, schopnost práce v reálném čase, nízkou spotřebu energie a snadný programový vývoj.

Řada	Procesor	Frekvence [MHz]	Test celkového výkonu
STM32F2	Cortex-M3	120	398
STM32F4	Cortex-M4	180	608
STM32F7	Cortex-M7	216	1082
STM32H5	Cortex-M33	250	1023
STM32H7	Cortex-M7	až 550	3224

Tabulka 1.1: Porovnání řad STM32 dle výkonu

Vzhledem k parametrům modelových řad STM32, pro zpracování obrazu pomocí konvolučních neuronových sítí, je optimální použití STM kontroléru z řad STM32H7 nebo STM32F7. Z tabulky 1.1 je patrné, že největší frekvence i celkový výkon je u řady STM32H7, která je založena na procesoru Cortex-M7. V této řadě výrobce disponuje vícero typy mikrokontrolérů, avšak pro úlohu rozpoznávání těles budeme používat doporučený typ s označením STM32H747. Tento mikrokontrolér je založen na dvojitěm běhu procesorů Cortex-M7 a Cortex-M4, které dokážou běžet na frekvenci 480 MHz a 240 MHz. Také obsahuje 1 MB RAM paměti a 2 MB FLASH paměti. Disponuje dotykovou LCD obrazovkou a konektorem pro vstup kamery. Zároveň výrobce nabízí příslušný kamerový modul s označením B-CAMS-OMV, který funguje na základě snímače s rozlišením 5 Mpx.

Na těchto zařízeních se pokusíme implementovat rozpoznávání uchopení těles. Přesto, před samotným výzkumem problematiky zpracování obrazu je třeba se seznámit s mikrokontrolérem, jeho parametry a metodami vývoje.

1.2 STM32H747I-DISCO

Výrobce uvádí [14], že STM32H747I-DISCO (Obrázek 1.2) je vysoce výkonný a všestranný mikrokontrolér, který poskytuje vysoký výpočetní výkon, mnoho periférií a funkcí. Je

určený pro širokou škálu aplikací, a to například průmyslových, zdravotnických nebo automobilových. Tento mikrokontrolér je navržený tak, aby poskytoval nejlepší kombinaci výkonu, energetické účinnosti a flexibilitu, což umožňuje snadno navrhovat a implementovat aplikace.



Obrázek 1.2: STM32H747I-DISCO

1.2.1 Architektura a procesory

Mikrokontrolér STM32H747I-DISCO je založen na architektuře ARM Cortex-M7 a Cortex-M4, která umožňuje výkonné výpočty a vysokou energetickou efektivitu. Jádru Cortex-M7 zajišťuje vysoký výkon s 480 MHz taktovací frekvencí a má přímý přístup k perifériím a paměti. Jádru Cortex-M4 je navrženo pro úlohy s nízkou latencí a nízkou spotřebou energie, jako jsou řízení motorů a digitální signálové zpracování.

Mikrokontrolér je vybavený vysokorychlostním paměťovým subsystémem, který zahrnuje víceúrovňovou cache pro procesor, rozhraní pro externí paměti a rychlým řadičem pro externí paměť typu FLASH s podporou QSPI rozhraní. Tato architektura umožňuje rychlé načítání a zpracování kódu či dat a zlepšuje výkon celého systému.

Procesory Cortex-M7 a Cortex-M4 jsou vybaveny hardwarovou podporou pro výpočty s plovoucí desetinnou čárkou, což zlepšuje výkon a snižuje nároky na software pro zpracování složitých matematických operací, což se dá využít například pro počítání vah u

neuronových sítí. Cortex-M7 procesor je navíc vybaven hardwarovou podporou pro kryptografické operace, jako jsou AES, DES, MD5 a SHA-1/SHA-2, což umožňuje rychlé a bezpečné šifrování a dešifrování dat. Tyto procesory jsou také vybaveny rozsáhlými sadami periférií, včetně sériových a paralelních komunikačních rozhraní (USART, UART, SPI, I2C), časovačů nebo analogových a digitálních vstupů/výstupů. Periferie mohou být konfigurovány pomocí různých registrů a řídicích mechanismů, což umožňuje přizpůsobit je potřebám konkrétní aplikace. [15]

1.2.2 Paměti

Vnitřní Flash paměť je použita k ukládání programového kódu a může být programována pomocí různých programovacích metod, včetně sériového bootloaderu nebo rozhraní JTAG a SWD. Velikost Flash paměti v mikrokontroléru STM32H747I-DISCO je 2 MB.

Mikrokontrolér obsahuje 1 MB RAM paměti (Tabulka 1.2), z toho 192 KB TCM paměti k ukládání kritických dat a kódu, které je třeba rychle a efektivně zpracovávat procesorem a 864 KB volatilní SRAM paměti pro ukládání dat používaných během provádění programu.

Typ paměti	Kapacita
TCM RAM	192 KB
User SRAM	864 KB
Backup SRAM	4 KB

Tabulka 1.2: Rozdělení RAM pamětí

Mikrokontrolér také obsahuje nevolatilní paměť pro ukládání dat, které mohou být uchovány i po vypnutí napájení. Tato paměť se nazývá EEPROM a má kapacitu 4 KB. EEPROM může být použita pro ukládání konfiguračních dat a dalších informací, které musí být uchovány i po vypnutí napájení. [15]

1.2.3 Napájení

Pro napájení a programování mikrokontroléru je možné použít k počítači připojený USB kabel. V případě STM32H747I-DISCO se vývojáři spoléhají na to, že napájení přes USB kabel bude dodáváno s maximálním napětím 5 V a maximálním proudem 500 mA, které je standardem pro USB 2.0. [15]

1.2.4 B-CAMS-OMV

Kamerový modul B-CAMS-OMV (Obrázek 1.3) je založený na snímači s označením OV5640, nabízejícím rozlišení 5 Mpx s bitovou hloubkou 8 bit.



Obrázek 1.3: Kamerový modul B-CAMS-OMV

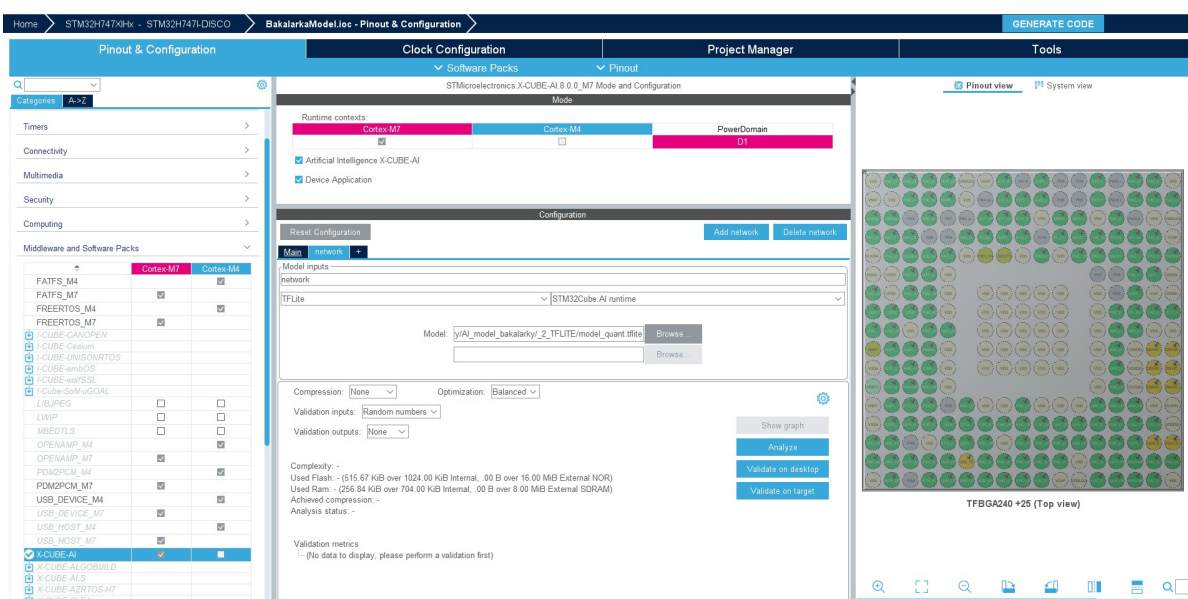
Modul disponuje speciálním 30pinovým ZIF konektorem, který umožňuje přenos dat a napájení. Konektor spojuje mikrokontrolér s adaptérem pro kamerový modul a umožňuje jejich vzájemné propojení. Pro zpracování naší úlohy použijeme nezastíněný flexibilní plochý kabel od společnosti Würth Elektronik, jehož použití minimalizuje vyzářené emise a je preferované pro funkční nastavení. [16]

1.2.5 Vývojové prostředí a programování

Jak bylo uvedeno dříve, pro implementaci neuronové sítě na mikrokontrolér lze použít knihovnu X-CUBE-AI. Tato knihovna poskytuje optimalizaci neuronových sítí z hlediska výpočetní náročnosti a paměti (RAM a Flash). V zásadě, převádí předtrénované neuronové sítě z nejpoužívanějších frameworků pro hluboké učení (například Keras nebo TensorFlow) do knihovny, která je integrována do konečného uživatelského projektu. X-CUBE-AI také rozšiřuje vývojové prostředí STM32CubeMX o výběr správných mikrokontrolérů, které splňují specifická kritéria požadavků pro neuronové sítě. [17]

STM32CubeMX je grafický nástroj pro konfiguraci a inicializaci projektů pro mikrokontroléry STM32. [18] Tento nástroj umožňuje:

- přidávat potřebné knihovny do projektu
- nastavovat piny a periferie
- konfigurovat základní parametry (hodinový signál, časovače, napájení a jiné)
- nastavovat HAL pro konkrétní typ mikrokontroléru
- vybírat vývojové prostředí, které použijeme k sestavení firmwaru



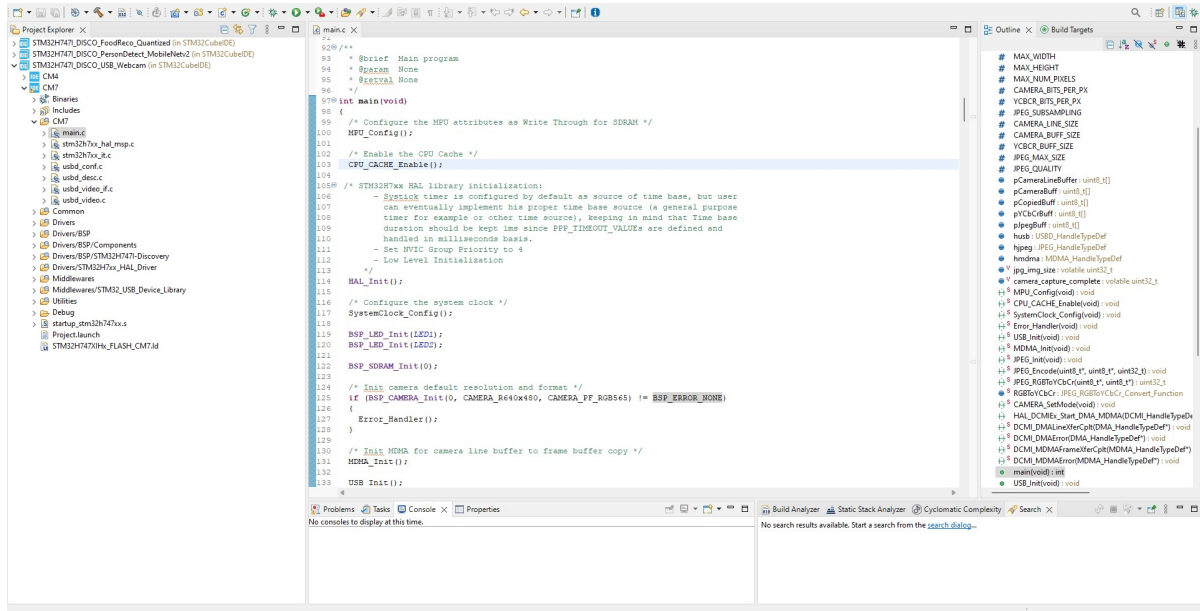
Obrázek 1.4: STM32CubeMX 6.8.0

Primární záložkou pro zpracování úlohy rozpoznávání těles je konfigurace a optimalizace modelu umělé inteligence. Jedná se o prostřední záložku na obrázku 1.4 s názvem a verzi X-CUBE-AI. S její pomocí můžeme nastavit, optimalizovat a vygenerovat naučený model přímo pro použití na mikrokontroléru. Umožňuje také provádět kompresi modelu, zobrazit jeho vlastnosti nebo i validovat model na cílovém zařízení.

Závěrečné nastavení projektu umožňuje vybrat IDE, které budeme používat pro další vývoj aplikace pomocí programování. Na výběr máme EWARM, MDK-ARM nebo STM32CubeIDE. Pro zpracování úlohy z důvodu jednoduché kompatibility a podpory STM ekosystému budeme používat STM32CubeIDE.

STM32CubeIDE je integrované vývojové prostředí pro vývoj firmwaru na STM32 mikrokontroléry. Poskytuje všechny potřebné nástroje pro vývoj a ladění aplikací, včetně

editoru kódu, kompilátoru, debuggeru a nástrojů pro konfiguraci periférií a integraci vygenerovaného kódu pomocí STM32CubeMX. IDE je postaveno na bázi Eclipse a GNU Compiler Collection. [19] Z toho vyplývá, že programování firmwaru probíhá v jazyce C nebo C++. Na obrázku 1.5 je zobrazen příklad main metody v jazyce C v prostředí STM32CubeIDE.



Obrázek 1.5: STM32CubeIDE 1.12.0

Po základní rešerši problematiky strojového učení a mikrokontrolérů můžeme začít s výzkumem konkrétního řešení pro úlohu práce.

2. Metodika rozpoznávání obrazu

V knize "Intelligence at the Edge"[20] od společnosti STMicroelectronics jsou popsány hlavní kroky pro vývoj modelů strojového učení: výběr vhodného datasetu, předzpracování dat, výběr a trénování modelu, testování a evaluace, vyladění a optimalizace, implementace na cílovém zařízení. Tyto kroky jsou opakovatelné a lze je přizpůsobit podle potřeby. Mají za cíl dosáhnout požadované přesnosti a výkonu modelu strojového učení v reálném prostředí.

Pro tuto bakalářskou práci je třeba provést výzkum nejdůležitějších kroků a jejich možnosti pro následné praktické zpracování. Přesto, jako první bod vybereme architekturu modelu, která je optimální pro řešení problému práce.

2.1 Výběr architektury

Jak bylo uvedeno dříve, pro strojové učení budeme používat konvoluční neuronové sítě. Tyto sítě jsou tvořeny několika vrstvami, z nichž každá má svou specifickou funkci.

Základními vrstvami v CNN jsou konvoluční vrstvy, pooling vrstvy a plně propojené vrstvy. Konvoluční vrstvy obsahují mnoho konvolučních filtrů, které jsou zpracovány na vstupním obrázku. Tyto filtry se snaží najít určité vlastnosti, jako jsou hranice, úhly nebo textury. Filtry jsou aplikovány na celý obrázek pomocí konvoluce, což je matematická operace, která vypočítává součin vstupního obrázku a konvolučního filtru v každé pozici daného obrázku. Pooling vrstvy se používají k redukci rozměrů výstupu z konvolučních vrstev. V plně propojených vrstvách jsou všechny vstupy propojeny s každým výstupem a používají se ke klasické kategorizaci a klasifikaci. [1]

Konvoluční neuronové sítě na mikrokontrolérech vyžadují zpravidla architekturu s nízkým počtem parametrů a nízkou výpočetní náročností. Podle výzkumu Andrew Howarda a jeho kolegů ze společnosti Google [21], nejefektivnější řešení pro použití CNN na zařízeních s omezenými výpočetními zdroji je architektura pod názvem MobileNet.

Architektura	Počet parametrů [mil.]	Přesnost klasifikace [%]
AlexNet	60	57,2
VGGNet	138	71,5
GoogleNet	6,8	69,8
MobileNet	4,2	70,6

Tabulka 2.1: Porovnání architektur CNN

Tato architektura se vyznačuje tím, že používá speciální konvoluční vrstvy, které mají

méně parametrů než standardní konvoluční vrstvy. To umožňuje dosáhnout vysoké přesnosti klasifikace obrazů při současném snížení výpočetní náročnosti. Výsledky porovnání v tabulce 2.1 ukazují, že architektura MobileNet v porovnání s alternativními architekturami dosahuje výrazně menšího počtu parametrů a menší výpočetní náročnosti než ostatní architektury, a to při zachování podobné přesnosti klasifikace. Příkladem je, že MobileNet dosáhne přesnosti 70,6 % při použití pouze 4,2 milionu parametrů, zatímco architektura VGGNet dosáhne přesnosti 71,5 % až s 138 miliony parametrů.

2.1.1 MobileNet

MobileNet je architektura sítě pro klasifikaci obrazů, která je navržena s cílem dosáhnout vysoké přesnosti klasifikace při zachování nízkého výpočetního výkonu. Hlavní myšlenkou MobileNetu je použití konvolučních filtrů o velikosti 1×1 a 3×3 namísto tradičních filtrů o velikosti 5×5 nebo 7×7 . To vede ke snížení počtu parametrů, a tedy i výpočetní náročnosti sítě.

Pro zpracování práce budeme používat novější MobileNetV2. Jedná se o vylepšenou verzi, kde hlavní inovací je použití bloků nazvaných Inverted Residuals, jež umožňují zachování nízké výpočetní náročnosti a zároveň zlepšují přesnost klasifikace. [22] Tyto bloky se používají v prvních vrstvách, kde se vstupní obraz převádí na nízkoúrovňové funkce, které jsou dále zpracovávány v dalších vrstvách.

Vstup	Operátor	Výstup
$h \times w \times k$	1×1 conv2d, ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3×3 dwise $s=s$, ReLU6	$h/s \times w/s \times (tk)$
$h/s \times w/s \times (tk)$	linear 1×1 conv2d	$h/s \times w/s \times k'$

Tabulka 2.2: Struktura Inverted Residual bloku [23]

Jak je uvedeno v tabulce 2.2, blok je složen ze tří kroků, které probíhají sekvenčně:

- Prvním krokem je využití konvoluční vrstvy s filtrem o velikosti 1×1 , která umožňuje snížit počet parametrů sítě a tím i výpočetní náročnost. Vstupní data o rozměru $h \times w \times k$ (kde h je výška, w je šířka a k je počet kanálů) projdou 1×1 konvoluční vrstvou a výstupem je feature mapa o rozměru $h \times w \times (tk)$, kde t je hyperparametr, který určuje, o kolikrát se zvětší počet kanálů. Feature mapy procházejí aktivační funkcí ReLU6, která zajistí, že výstupní hodnoty jsou omezeny v rozmezí 0 až 6.
- Druhý krok využívá separabilní konvoluci. Jedná se o speciální typ konvoluce, která se skládá ze dvou částí: nejprve se aplikuje konvoluční filtr pouze na jednom

kanálu (tedy pouze na červeném, zeleném nebo modrém kanálu obrázku), a poté se výsledky z jednotlivých kanálů zkombinují do jedné výstupní feature mapy. Tento typ konvoluce je výpočetně efektivní a zároveň umožňuje dosahovat dobrých výsledků při klasifikaci obrázků. 3x3 separabilní konvoluční filtry jsou aplikovány na feature mapy z předchozího kroku pomocí $s=s$ kroku, kde s je hyperparametr, který určuje, o kolikrát se sníží výška a šířka výstupu. Výstup druhého kroku znovu prochází aktivační funkcí ReLU6.

- V posledním kroku je použita 1x1 lineární konvoluční vrstva, která slouží k redukci počtu kanálů. Tato vrstva umožňuje zachovat informace z předchozích vrstev a zlepšit kvalitu klasifikace s pomocí lineární kombinaci příznaků z každého kanálu.

Výhodou MobileNetV2 oproti ostatním architekturám je použití techniky lineárního bottlenecku, která umožňuje efektivní použití vrstev s větším počtem kanálů. Spočívá v použití jednoduchého lineárního konvolučního filtru o rozměru 1x1 pro snížení počtu kanálů, následované konvolučním filtrem 3x3 pro získání nezbytných vysokoúrovňových funkcí. Díky této technice je možné dosáhnout vyšší přesnosti klasifikace při zachování nízké výpočetní náročnosti. [23]

MobileNetV2 dosahuje vysoké přesnosti klasifikace a zároveň zachovává nízkou výpočetní náročnost, což ho činí vhodným pro použití v mobilních zařízeních a dalších zařízeních s omezenými výpočetními zdroji. [22]

Z toho vyplývá, že pro tuto práci MobileNetV2 je optimálním řešením při výběru architektury konvoluční neuronové sítě.

2.2 Předzpracování dat

Pro konvoluční neuronové sítě není možné, aby dosáhly dobré klasifikační přesnosti bez správné přípravy vstupních dat na samotné učení. Proto existují techniky předzpracování, které hrají zásadní roli pro přípravu jakéhokoli souboru dat. [24]

2.2.1 Augmentace

Jedna ze základních technik, kterou je třeba provést na původním datasetu, je augmentace. Je to technika, která zahrnuje generování nových trénovacích dat z existujících trénovacích dat pomocí transformací jako rotace, posunutí, zvětšení, zmenšení, převrácení a dalších podobných úprav. Cílem augmentace je zvýšit množství trénovacích dat a zlepšit tak výkonnost modelu i v případě menší trénovací sady dat. [25]

Základními typy augmentace jsou:

- Přidávání šumu - přidání náhodného šumu do obrázku může pomoci modelu lépe se vyrovnat s nečistotami nebo šumem v reálných snímcích. Výsledkem může být zlepšení přesnosti modelu.
- Změna jasu a kontrastu - může pomoci při rozpoznávání objektů na snímcích s různým osvětlením. Změna jasu a kontrastu může vést k větší rozmanitosti dat a k lepší generalizaci modelu.
- Horizontální a vertikální otočení - obrázek je otočen o určitý úhel, což může pomoci modelu při rozpoznávání objektů v různých orientacích. Výsledkem může být znovu zlepšení přesnosti modelu.
- Škálování - zvětšení nebo zmenšení obrázku může pomoci modelu při rozpoznávání objektů různých velikostí. Opět může mít vliv na přesnost modelu.

Tyto základní typy lze rozdělit do třech různých kategorií jako jsou geometrická augmentace, která se zabývá transformacemi v obraze, intenzivní augmentace pro změnu kontrastu, jasu, stínu nebo barvy a morfologická augmentace pro přidání detailů nebo vlastností do původních obrazů. [26] Vzhledem k relativně jednoduchým tělesům, potřebným pro rozpoznávání, augmentace pro jejich dataset nebude složitá. Pro tyto účely existují různé připravené knihovny, které lze setřídit dle umožnění pracovat s každou z kategorií augmentace.

Název knihovny	Geometrická	Intenzivní	Morfologická
imgaug	Ano	Ano	Ano
Augmentor	Ano	Ne	Ano
Keras ImageDataGenerator	Ano	Ano	Ne
TorchIO	Ano	Ano	Ano
MONAI	Ano	Ano	Ano

Tabulka 2.3: Porovnání knihoven pro augmentaci obrazových dat

Pro augmentaci našeho datasetu můžeme použít jednu z knihoven, uvedených v tabulce 2.3. Konkrétněji použijeme knihovnu imgaug [27]. Jedná se o Python knihovnu, která nabízí širokou škálu funkcí pro generování náhodných augmentací a umožňuje vytvářet složitější augmentační řetězce.

2.2.2 Normalizace

Další metodou, kterou je třeba provést na datovou sadu, je normalizace. Jedná se o transformaci nezpracovaných dat do podoby užitečného formátu. Cílem normalizace je dosažení stabilnějšího a efektivnějšího učení modelu. [20]

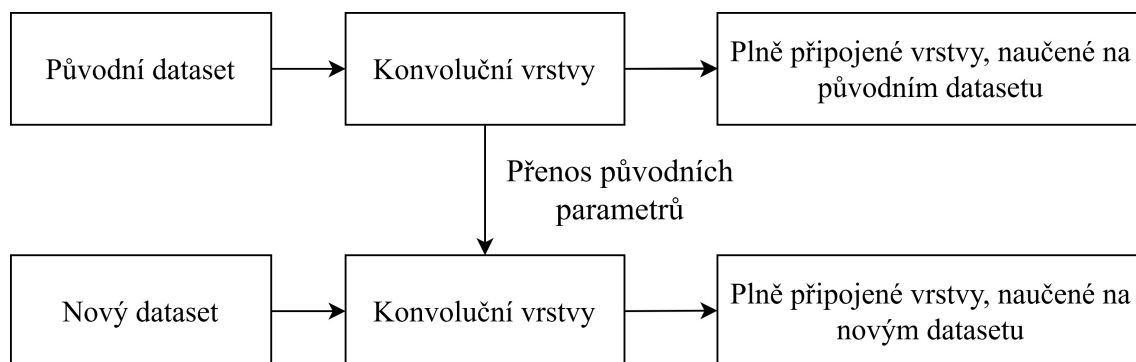
Každá architektura sítě disponuje vlastním nastavením pro vstupní data. Normalizaci pro určité nastavení by měla umožňovat knihovna, použitá pro samotné učení. Proto je třeba provést výzkum nástrojů, které lze použít pro následné zpracování práce.

2.3 Trénování modelu

Poté, co vybereme vhodnou architekturu modelu, lze přistoupit k samotnému trénování, pro které je třeba předem připravit velký soubor trénovacích dat. Tento soubor je možné poměrně zkrátit díky technice pod názvem transfer learning. Tato technika je v posledních letech neoddělitelná od konvolučních neuronových sítí a je třeba rozebrat její princip a výhody. [28]

2.3.1 Transfer learning

Transfer learning je technika, která umožňuje využít znalosti naučené sítě na jedné úloze pro řešení jiné úlohy. V kontextu rozpoznávání obrazů, předtrénovaná síť, která byla naučena rozpoznávat určité charakteristiky, může být použita jako výchozí bod pro trénování nové sítě na podobném úkolu. [29]



Obrázek 2.1: Transfer learning

Princip této techniky (Obrázek 2.1) spočívá v tom, že dřívější vrstvy neuronové sítě se převezmou od předem naučeného modelu, který řeší podobnou problematiku. Například, pro učení modelu rozpoznávání těles lze použít připravený model, který již umí v prvních vrstvách rozpoznávat hrany.

Transfer learning umožňuje výrazně zlepšit výkon modelu, především pokud je k dispozici omezené množství dat pro novou úlohu. To znamená, že tuto techniku lze efektivně použít i pro zpracování této bakalářské práce.

Po úspěšné přípravě dat, můžeme provést výzkum postupu samotného trénování. Je třeba vybrat knihovnu, s jejíž pomocí se bude provádět učení. Měla by podporovat vybranou architekturu modelu a být optimální pro úlohu rozpoznávání těles.

2.3.2 TensorFlow

Jednou z nejznámějších knihoven pro trénování neuronových sítí je TensorFlow. Jedná se o knihovnu od společnosti Google, která je velmi výkonná, flexibilní a umožňuje tvorbu a trénování různých typů neuronových sítí. [30] Poskytuje mnoho nástrojů a funkcí pro trénování konvolučních neuronových sítí. Oproti dalším populárním knihovnám pro hluboké učení, jako například PyTorch, Caffé nebo Theano, poskytuje víc nástrojů pro zpracování úlohy rozpoznávání těles. Z toho vyplývá, že použití této knihovny je optimální pro typ sítě bakalářské práce.

Trénování konvolučních neuronových sítí v TensorFlow se skládá ze tří kroků:

- Definice modelu a vrstev sítě. Je to proces vytvoření struktury sítě, při kterém se určuje počet vrstev, jak budou vrstvy propojeny a jaké aktivační funkce budou použity.
- Definice loss funkce a optimalizačního algoritmu. Loss funkce porovnává výstupy sítě s očekávanými výstupy a určuje jejich rozdíl. Cílem trénování neuronové sítě je minimalizovat hodnotu této funkce. Optimalizační algoritmus se používá k nalezení optimálních hodnot vah neuronové sítě, které minimalizují loss funkci.
- Samotné trénování sítě na trénovacích datech.

Dle autorů publikace "Deep Learning With TensorFlow: A Review"[30], je pro tyto kroky doporučeno použití knihovny pod názvem Keras, která je součástí TensorFlow a je jedním z nejpoužívanějších nástrojů pro tvorbu a trénování neuronových sítí.

2.3.3 Keras

Keras je knihovna pro tvorbu neuronových sítí, napsaná v jazyce Python. Vzhledem k TensorFlow, se jedná o vysokoúrovňové API, které je navrženo tak, aby bylo jednoduché a intuitivní. Zároveň ale nabízí pokročilé funkce pro úpravy a optimalizace modelů. [31] Tato knihovna nabízí širokou škálu vrstev pro tvorbu různých typů sítí, včetně konvolučních neuronových sítí. Součástí Kerasu je implementovaná architektura MobileNetV2,

která obsahuje předtrénovaný model, použitelný i pro Transfer learning. Keras je dobrou volbou pro trénování modelů tohoto typu, tudíž je vhodný pro splnění zadání této práce. Funkce pro trénování lze rozebrat pomocí dokumentace dané knihovny během samotné implementace řešení.

2.4 Optimalizace modelu

Po úspěšném získání natrénovaného modelu, přichází potřeba provést optimalizaci a připravit daný model pro použití přímo na mikrokontroléru. Po trénování je neuronová síť uložena jako model s mnoha parametry, které jsou v podobě desetinných čísel s vysokou přesností (32 bitů na hodnotu). To může způsobit, že velikost modelu bude velká a výpočetní náročnost bude vysoká. Tuto velikost lze zredukovat s pomocí převodu parametrů z vysoce přesných hodnot na nižší. [32]

2.4.1 Kvantizace

Kvantizace je proces převodu vah a aktivačních hodnot z reprezentace s vysokou přesností na nižší přesnost. Například, pokud všechny hodnoty, reprezentující váhy, jsou uloženy jako 32bitová čísla s plovoucí desetinnou čárkou, kvantizace převede tyto hodnoty na 8bitová celá čísla. To umožňuje redukovat paměťové a výpočetní nároky sítě, což je výhodné pro nasazení na zařízení s omezenými výpočetními prostředky, jako jsou například mobilní telefony nebo mikrokontroléry. [32]

Parametr	Sčítání [pJ]	Násobení [pJ]
8bitový Integer	0,03	0,2
32bitový Integer	0,1	3,1
16bitový Float	0,4	1,1
32bitový Float	0,9	3,7

Tabulka 2.4: Tabulka spotřeby energie dle operace

Velikost parametrů má vliv i na spotřebu energie. V publikaci z roku 2014 [33] Mark Horowitz spočítal rozdíl spotřeby u různých velikostí hodnot dle operace sčítání a násobení (Tabulka 2.4). Například, rozdíl mezi násobením 8bitového celého čísla a 32bitového čísla s desetinnou čárkou je 3,5 pJ. Je to tedy 18,5krát vyšší spotřeba, což ukazuje důležitost dané optimalizace.

Samotný převod se provádí pomocí kvantizační funkce, která přiřazuje každé hodnotě novou hodnotu z určité kvantizované reprezentace. Například, lze použít funkci, která rozděljuje rozsah hodnot na intervaly a každé hodnotě přiřazuje nejbližší hodnotu z daného intervalu. [32] V kontextu zadání bakalářské práce, tato optimalizace je důležitá

vzhledem k použití modelu na mikrokontroléru, což je zařízení s omezenými výpočetními prostředky.

2.4.2 X-CUBE-AI

Jak bylo zmíněno dříve v kapitole 1.2.5, X-CUBE-AI optimalizuje již hotovou neuronovou síť z hlediska výpočetní náročnosti a paměti a připravuje ji pro použití na STM32 mikrokontroléru. To znamená, že knihovna provádí konverzi trénovaných modelů do kódu vhodného pro mikrokontroléry a generuje zdrojový kód v jazyce C pro použití v STM32CubeIDE. Jedním ze způsobů samotné optimalizace je odstranění zbytečných funkcí a proměnných, které nejsou použity v konkrétním modelu. [17]

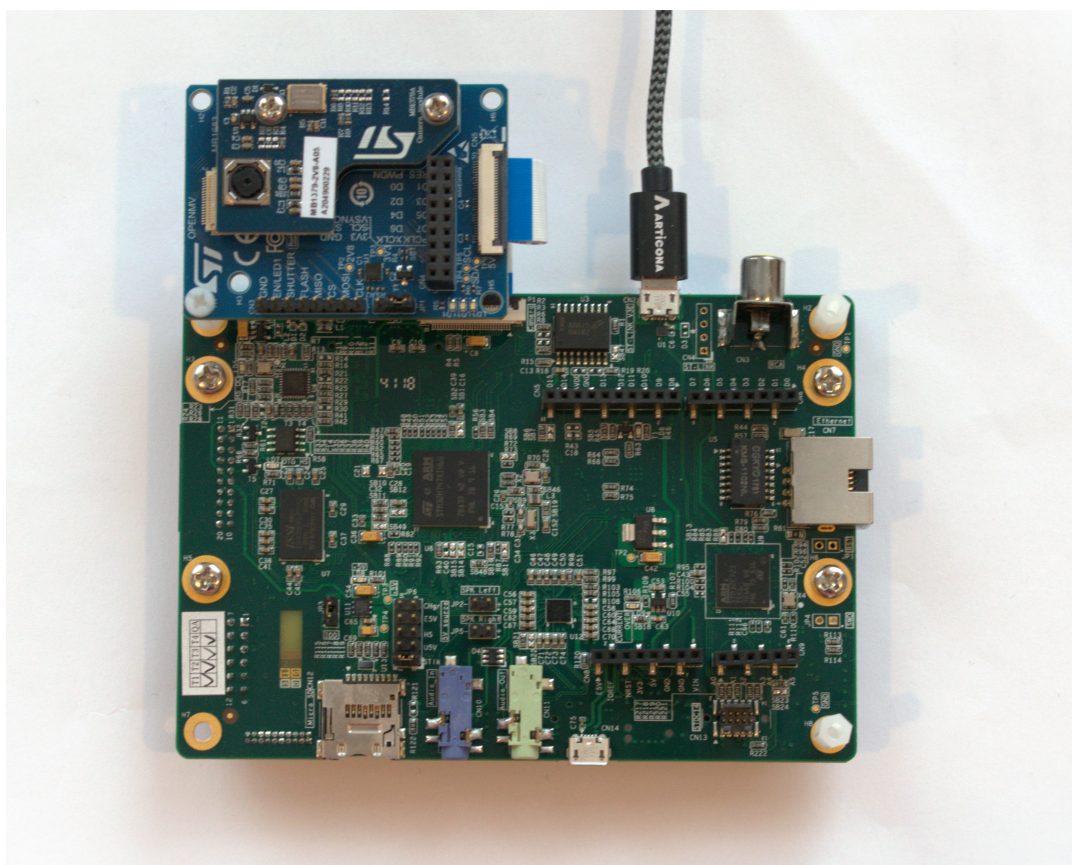
Daný nástroj poskytuje širokou podporu různých neuronových sítí a STM mikrokontrolérů. Vzhledem k této kompatibilitě, jeho použití je důležité i pro implementaci modelu bakalářské práce.

3. Návrh a implementace systému

Po teoretické analýze možných řešení, je na čase přistoupit k praktické části práce. Tato část zahrnuje popis provedených aktivit pro implementaci celého systému.

3.1 Fyzické propojení

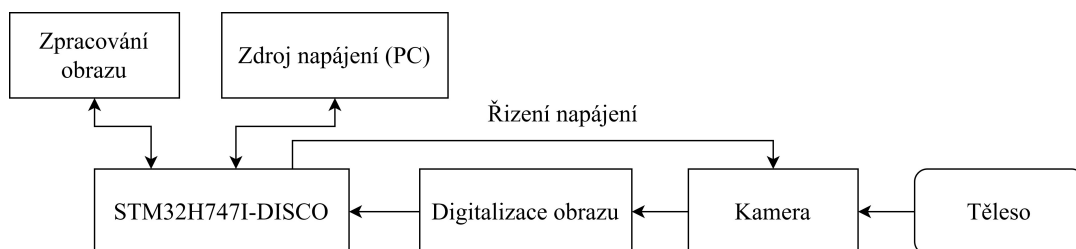
Pro správné fungování celého systému je třeba připojit všechny jeho součástky. Jedná se hlavně o propojení kamery a samotného mikrokontroléru. Jak bylo uvedeno v kapitole 1.2.4, pro propojení lze použít plochý kabel od společnosti Würth Elektronik (Obrázek 3.1).



Obrázek 3.1: Zadní strana STM32H747I-DISCO

Port na mikrokontroléru, který slouží pro připojení konektoru z kamery, má označení "Camera". Samotný kamerový modul lze přišroubovat na mikrokontrolér pomocí speciální plastové hmoždinky. Obraz tedy nebude odpovídat stejnému úhlu jako u displeje, což v případě rozpoznávání těles není problém.

Pro připojení mikrokontroléru s počítačem jsem použil USB kabel, připojený k ST-LINK portu na mikrokontroléru. Tento kabel má konektor typu A pro připojení k počítači a konektor typu Micro-B pro připojení k samotnému mikrokontroléru. Schéma zapojení celého systému je znázorněna na obrázku 3.2.

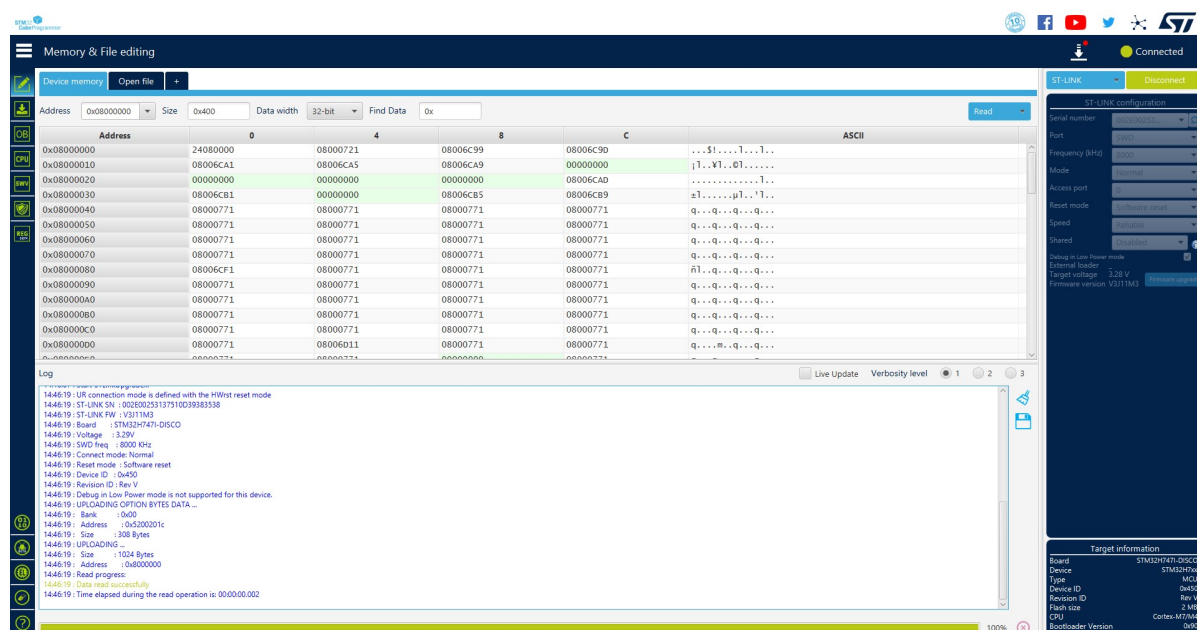


Obrázek 3.2: Schéma zapojení systému

Výsledkem zapojení by měl být funkční systém, umožňující fungování všech prvků aplikace.

3.1.1 Aktualizace firmwaru

Základní testování funkčnosti mikrokontroléru lze provést pomocí aplikace STM32CubeProgrammer [34]. Je to užitečný nástroj pro správu mikrokontrolérů řady STM32. Kromě čtení a zápisu do paměti mikrokontroléru, umožňuje zaktualizovat firmware, na kterém daný mikrokontrolér běží.



Obrázek 3.3: STM32CubeProgrammer 2.13.0

V této aplikaci jsem vyzkoušel připojení přes ST-LINK port dle specifických nastavení a zároveň provedl již zmíněnou aktualizaci (Obrázek 3.3).

3.2 Příprava trénovacích dat

Po připojení mikrokontroléru a kamery k počítači, jsem se rozhodl připravit trénovací data pro učení. Nejlepším způsobem této přípravy je získat data přímo z kamerového modulu. To nám určí, že po konečné implementaci modelu přesnost rozpoznávání zůstane stejná, a nikoliv menší kvůli jinému nastavení kamery.

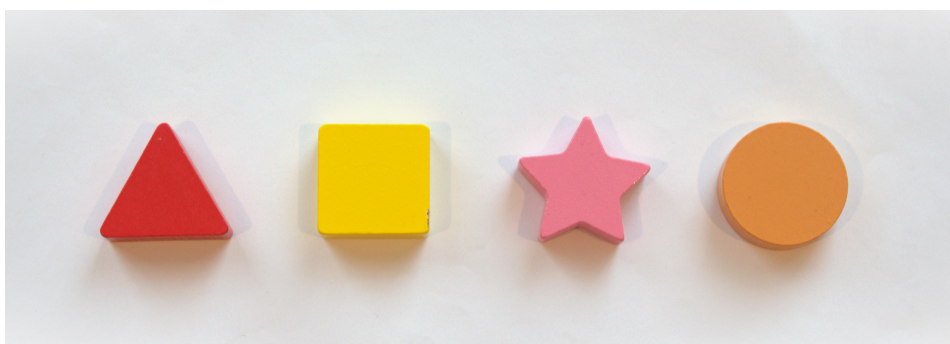
3.2.1 Nahrání potřebné aplikace

Pro získávání dat na mikrokontroléru jsem použil aplikaci z balíčku FP-AI-VISION1 [35]. Jedná se o balíček od společnosti STMicroelectronics, obsahující příklady aplikací, založených na konvolučních neuronových sítích. Jednou z ukázek je aplikace pod názvem "USB_Webcam", kterou jsem použil pro přípravu trénovacích dat. Její funkčnost spoleská v simulaci kamery, kterou lze připojit k počítači pomocí USB kabelu.

Tuto aplikaci lze jednoduše nahrát na mikrokontrolér pomocí prostředí STM32CubeIDE. Stačí ve složce balíčku spustit projektový soubor a v otevřeném prostředí spustit projekt. Po provedení těchto kroků, mikrokontrolér fungoval v operačním systému počítače jako obyčejná kamera, se kterou bylo možné nafotit potřebnou sadu dat v rozlišení 640 na 480 pixelů.

3.2.2 Získávání dat

Jako hlavní tělesa, které bude muset model rozpoznávat, jsem si vybral trojúhelník, čtverec, hvězdu a kruh. Jedná se o trojrozměrné figurky v různobarevném provedení (Obrázek 3.4).



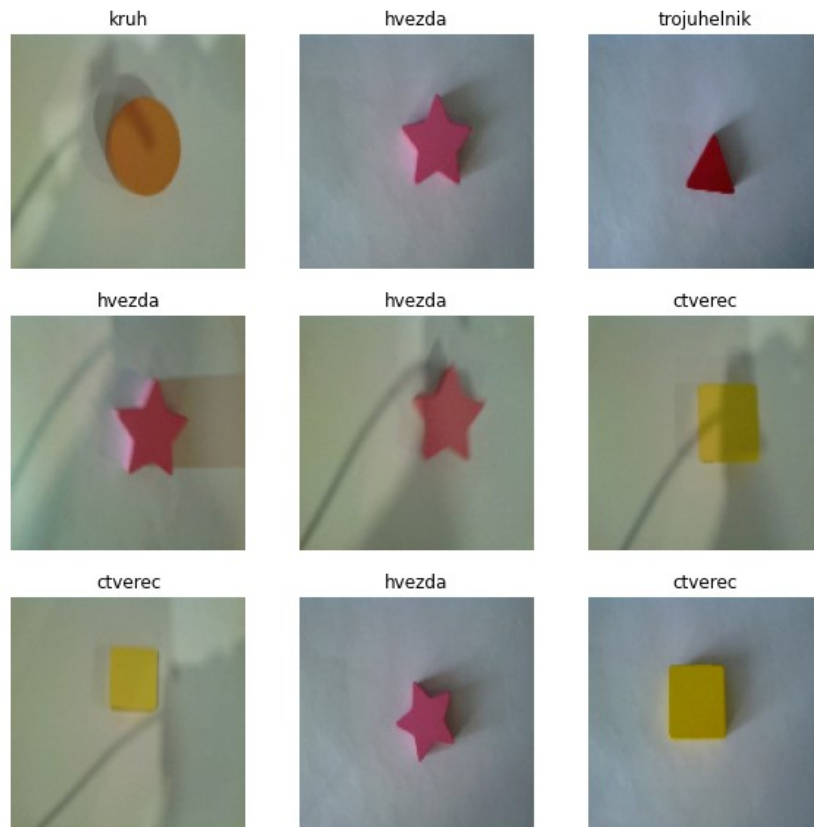
Obrázek 3.4: Tělesa pro rozpoznávání

Pro potřeby trénování jsem nafotil každé těleso v počtu, uvedeném v tabulce 3.1. Vzhledem k použití Transfer learningu, by měl tento počet dat vyhovovat pro úspěšné a efektivní trénování modelu. Počet snímků u kruhu je vzhledem k jeho stejnému tvaru při jakémukoliv směru otáčení a k jeho jednoduchosti menší než u ostatních těles.

Těleso	Počet snímků
Trojúhelník	50
Čtverec	50
Hvězda	50
Kruh	25

Tabulka 3.1: Počet snímků dle tělesa

Navíc jsem nafotil 25 obrázků, kde na popředí není žádné těleso, ale jenom pozadí. Důvodem je kvalitnější rozpoznávání a lepší uživatelské rozhraní u budoucí aplikace.



Obrázek 3.5: Vizualizace sady dat

Každý tvar byl nafocen při různém osvětlení a z různých úhlů. Vizualizace ukázkových obrázků je zobrazena na obrázku 3.5. Rozměr těchto obrázků byl zredukován na 128 na 128 pixelů. Každý prvek byl pojmenován bez diakritiky pro jednodušší budoucí kompatibilitu při programování.

Některé obrázky byly použity i pro testovací množinu, potřebnou pro validaci modelu.

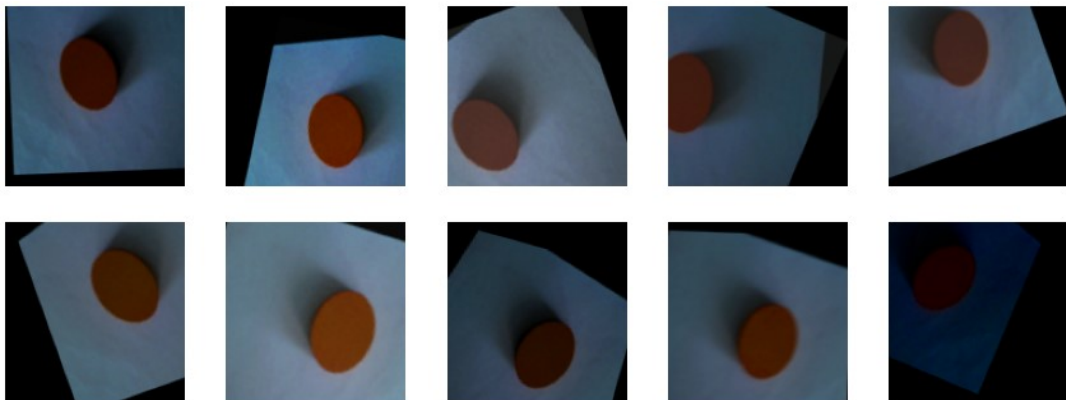
3.2.3 Augmentace

Po připraveném datasetu, je na čase provést augmentaci. Jak bylo zmíněno v kapitole 2.2.1, pro danou úlohu použijeme knihovnu `imgaug`. Její použití jsem přidal do společného souboru, který má na starosti učení modelu. Funkcionalita této knihovny umožňuje používat filtry s různým nastavením. [36]

Konkrétněji funkce pro různé typy augmentace jsou znázorněny v kódu:

```
augmenter = iaug.Sequential([
    iaug.Sometimes(0.3,
        iaug.MotionBlur((3, 4)),
        name='MotionBlur'),
    iaug.Sometimes(0.3,
        iaug.GaussianBlur((0.0, 0.75)),
        name='GaussianBlur'),
    iaug.GammaContrast((0.7, 1.5)),
    iaug.MultiplySaturation((0.9, 1.5)),
    iaug.MultiplyAndAddToBrightness(),
    iaug.Fliplr(p=0.5),
    iaug.Affine(scale=(1, 1.3),
        translate_percent={"x": (-0.3, 0.3), "y": (-0.3, 0.3)},
        rotate=(-25, 25) )
])
```

Zpracování těchto funkcí na obrázku způsobí odlišné parametry daného obrázku, například změnu kontrastu v rozmezí od 0,7 do 1,5. Příklad augmentace je znázorněn na obrázku 3.6.



Obrázek 3.6: Příklad augmentace

3.2.4 Normalizace

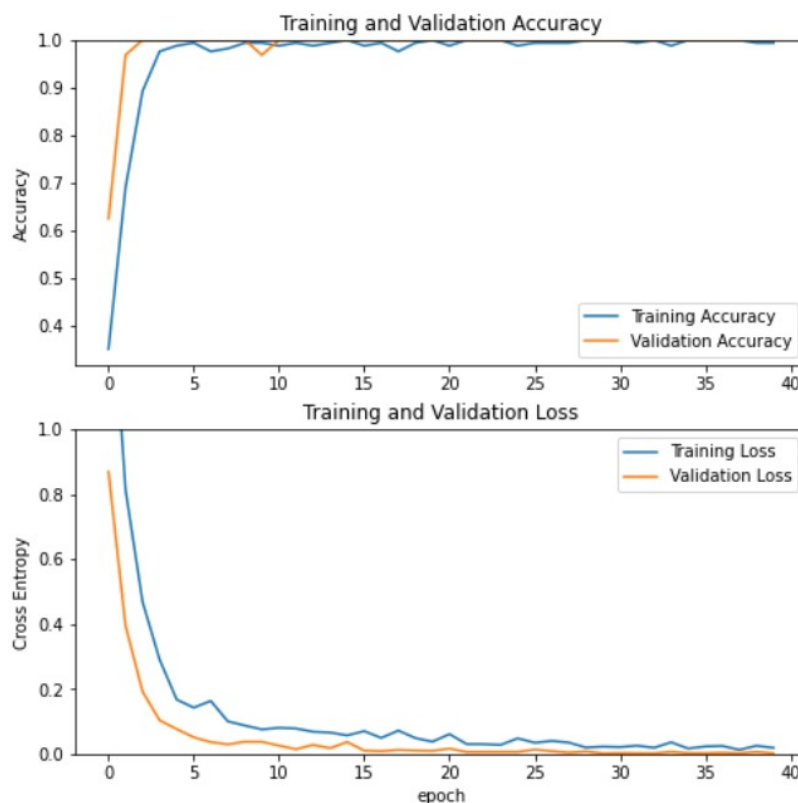
Pro normalizaci dat jsem použil funkci předzpracování, která připravuje model pro architekturu MobileNetV2. Tato funkce normalizuje hodnoty pixelů z rozsahu $[0, 255]$ na hodnoty v rozsahu $[-1, 1]$ a zajišťuje správnou reprezentaci dat.

Toto předzpracování pomáhá zajistit, že data jsou v přesném formátu, který očekává vybraná architektura. Tím pomáhá zajistit dosažení nejlepších výsledků při učení.

3.3 Trénování modelu

Samotné trénování modelu probíhalo pomocí knihovny TensorFlow, konkrétněji její vysokoúrovňového API Keras (viz kapitola 2.3.3). Součástí kompilace modelu v Kerasu je definování optimalizačního algoritmu, který se snaží najít takové hodnoty parametrů modelu, aby byla minimalizována ztrátová funkce. Pro dané zadání jsem použil optimalizační algoritmus pod názvem Adam, který patří k nejčastěji používaným optimalizačním algoritmům v oblasti strojového učení. [37]

Proces trénování je zobrazen na obrázku 3.7.



Obrázek 3.7: Graf trénování

Na prvním grafu jsou přesnosti trénování a souběžné validace. Na druhém grafu jsou ztrátové funkce během trénování. Z obou grafů je patrné, že proces samotného trénování probíhal dobře a počet epoch, potřebných pro kompletně naučený model byl menší než 40. Výsledkem trénování je model, jehož počet vrstev je 69 a počet parametrů je znázorněn v tabulce 3.2.

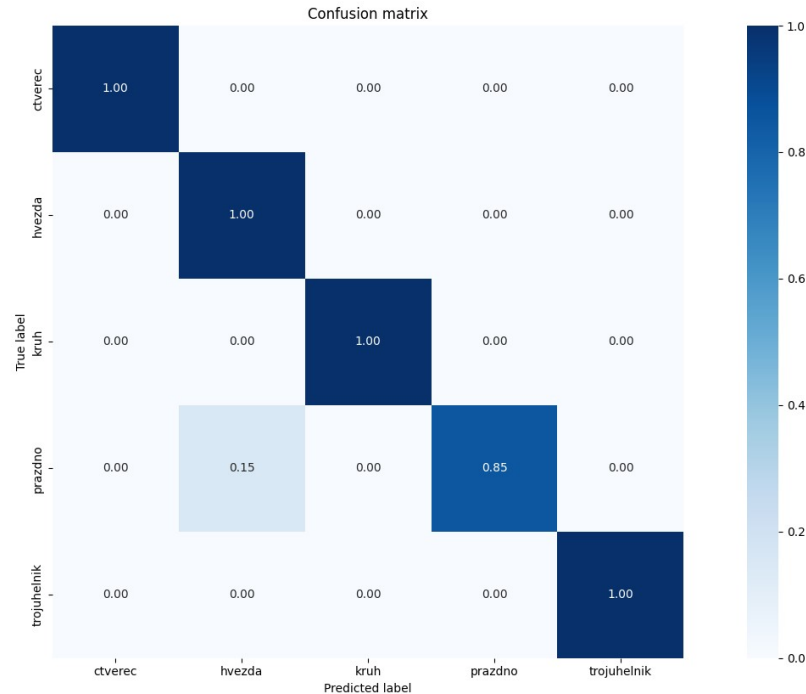
Celkový počet parametrů	416613
Trénovatelné parametry	6405
Netrénující parametry	410208

Tabulka 3.2: Počet parametrů modelu

Z tabulky je patrné, že trénovatelných parametrů je jenom 6405, což oproti celkovému počtu parametrů je přibližně 0,0154 %. Takový poměr trénovatelných parametrů je způsoben transfer learningem, který obecné rysy rozpoznávání obrazů měl připravený v původním Keras modelu.

3.3.1 Základní testování přesnosti

Testování přesnosti modelu je znázorněno v matici záměn na obrázku 3.8.



Obrázek 3.8: Matice záměn

Její výsledek ukazuje, že přesnost modelu dosahuje skoro ideálních výsledků. Jediný problém, který se ukázal, může být spojen s nepřesností různých variant prázdného

pozadí, jako například různé souběžné odstíny. Avšak, tato odchylka by neměla mít význam u konečné aplikace.

3.4 Optimalizace

Po úspěšném testování přesnosti je třeba provést optimalizaci pro nasazení na mikrokontrolér. Jako součást optimalizace, potřebujeme převést model na formát, známý pro prostředí STM32CubeMX.

3.4.1 Kvantizace

Výsledkem kapitoly 2.4.1 je názorná ukázka důležitosti kvantizace modelu. Tento způsob optimalizace je již implementovaný v TensorFlow knihovně:

```
def representative_data_gen():
    for x, y in validation_dataset.take(100):
        yield [x]

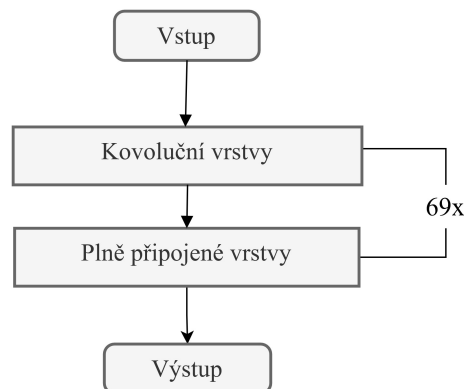
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_data_gen
converter.target_spec.supported_ops = [tf.lite.OpsSet
                                       .TFLITE_BUILTINS_INT8]

converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.uint8
tflite_model_quant = converter.convert()
```

Ze zdrojového kódu je patrné, že s pomocí funkcionality knihovny lze provést kvantizaci modelu a zároveň převést model do formátu pod názvem "tflite". Tento formát lze použít pro nasazení na mikrokontrolér, konkrétněji pro tvorbu modelu v prostředí STM32CubeMX.

3.4.2 STM32CubeMX

Posledním krokem, který je třeba provést pro přípravu modelu, je optimalizace v prostředí STM32CubeMX. V této aplikaci jsem zanalyzoval model pro použití na mikrokontroléru a vygeneroval samotné soubory, které jsem následně použil v projektu. V tomto prostředí lze prozkoumat strukturu neuronové sítě modelu. Jedná se hlavně o počet vrstev, který v daném případě je 69 (viz obr. 3.9).



Obrázek 3.9: Počet vrstev neuronové sítě

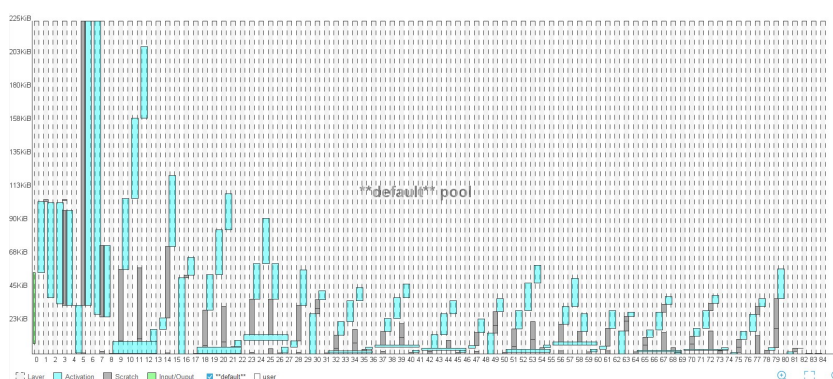
Na obrázku 3.10 je výstup samotné analýzy naučeného modelu. Tato analýza ukázala, že celkové použití Flash paměti pro model je 516 KB, což je plně dostačující pro funkčnost aplikace.

```

Total Flash: 528044 B (515.67 KiB)
Weights: 416628 B (406.86 KiB)
Library: 111416 B (108.80 KiB)
Total Ram: 263001 B (256.84 KiB)
Activations: 229888 B (224.50 KiB)
Library: 33108 B (32.33 KiB)
Input: 49152 B (48.00 KiB included in Activations)
Output: 5 B
Done
Analyze complete on AI model
  
```

Obrázek 3.10: Výstup analýzy modelu

Podrobnější znázornění použití RAM paměti je vidět na obrázku 3.11.



Obrázek 3.11: Použití RAM paměti

Z obrázku je patrné, že maximální použití RAM paměti je 224,5 KB. To tedy znamená, že model vyhovuje pro použití v konečné aplikaci.

3.5 Vývoj aplikace

Jednou z nejdůležitějších částí práce je implementace modelu do konečné aplikace. Výsledkem generování potřebného kódu v prostředí STM32CubeMX je skupina sedmi souborů, obsahujících nastavení a parametry modelu. Konkrétněji se jedná o soubory:

- `network.c` a `network.h` - obsahují implementaci neuronové sítě v programovacím jazyce C. Soubor `network.c` obsahuje funkce pro inicializaci, trénování a ověřování neuronové sítě. Soubor `network.h` obsahuje definice funkcí pro neuronovou síť, které jsou použity v ostatních souborech projektu.
- `network_data.c` a `network_data.h` - obsahují datové struktury pro neuronovou síť, včetně vah neuronů, množství vstupních a výstupních neuronů a počtu skrytých vrstev.
- `network_config.h` - obsahuje konfigurační nastavení pro neuronovou síť.
- `network_data_params.c` a `network_data_params.h` - obsahují další nastavení pro neuronovou síť, například rozsah hodnot vstupních dat.

Tyto soubory jsou nezbytnou součástí následné implementace.

3.5.1 Programování

Před začátkem programování aplikace, jsem v nově vytvořeném projektu nastavil potřebné knihovny a ovladače. Jedná se o nastavení cest ke knihovně X-CUBE-AI 8.0.0 a ovladačům pro zpracování výstupu kamerového modulu, nastavení periférií a programování mikrokontroléru. Tyto nástroje jsou k dispozici v balíčku FP-AI-VISION1, o kterém jsem se zmínil v kapitole 3.2.1.

Pro programování aplikace jsem použil jednu z ukázkových aplikací, která funguje na principu rozpoznávání obrazu pomocí umělé inteligence. Hlavním směrem programování tedy byl výzkum existujícího kódu pro potřeby aplikace a jeho následná implementace.

Hlavním souborem, znázorňujícím funkčnost aplikace je `fp_vision_app.c`. Konkrétně v tomto souboru jsou definovány funkce pro inicializaci a spuštění rozpoznávání obrazu, včetně konfigurace vstupního a výstupního rozhraní. Také obsahuje zpracování vstupních dat z kamery dle příslušných knihoven.

Soubor `main.c` inicializuje všechny periferie, provádí základní konfiguraci mikrokontroléru a spouští hlavní smyčku programu. Seznam všech nastavení a inicializací v tomto souboru je znázorněn v kódu s komentáři:

```

/* Konfiguracni procesy */
MPU_Config(); // Konfigurace MPU atributu
CPU_CACHE_Enable(); // Povoleni CPU Cache
SystemClock_Config(); // Konfigurace hodin na 400 MHz
__HAL_RCC_CRC_CLK_ENABLE(); // Povoleni CRC bloku

/* Inicializacni procesy */
HAL_Init(); // HAL
APP_Init(&App_Context); // Aplikace (fp_vision_app.c)
TEST_Init(App_Context.Test_ContextPtr); // Testovací kontext
UTILS_Init(App_Context.Utils_ContextPtr); // Pomocna struktura
DISPLAY_Init(App_Context.Display_ContextPtr); // Displej
AI_Init(App_Context.Ai_ContextPtr); // AI

```

Tyto nastavení používají různé soubory, které v celku umožňují běh aplikace.

3.5.2 Výstup aplikace

Vzhledem k možnému dalšímu vývoji na bázi aplikace, jsem implementoval základní výstup rozpoznávání ve tvaru fronty. Jedná se o dva soubory queue.c a queue.h, které umožňují uložit do fronty vyhodnocení obrazu. Příklad vkládání do fronty je znázorněn v kódu:

```

float accuracy = (((float*)(App_Context_Ptr->
    Ai_ContextPtr->nn_output_buffer)+0) * 100);
if (accuracy >= NN_BAD_RES)
{
    Result result =
        {NN_OUTPUT_CLASS_LIST[App_Context_Ptr->ranking[0]], accuracy};
    enqueue(&queue, result);
}

```

Daný kód získává aktuální přesnost modelu a v případě její vyšší hodnoty přidává pomocí funkce enqueue informace o vyhodnoceném tělese do fronty.

3.6 Testování funkčnosti

Po dokončení programové části aplikace, je možné ji nasadit na konečné zařízení. Pro tento účel jsem vytvořil konfiguraci běhu, kterou jsem nastavil na příslušný soubor ve

formátu "elf". Tento soubor se generuje při sestavení projektu a obsahuje kód a data pro nasazení na mikrokontroléry. Příklad použití konečné aplikace lze vidět na obrázku 3.12.



Obrázek 3.12: Použití aplikace

Aplikace byla úspěšně spuštěna na mikrokontroléru a je funkční. Výstup kamery je zobrazen na displeji a všechna tělesa jsou rozpoznávány.

4. Testování a analýza

Kontrola funkčnosti ukázala, že implementace modelu byla úspěšná. Avšak, je důležité provést detailnější testování modelu, konkrétně jeho přesnosti a rychlosti.

4.1 Přesnost

Výsledky měření přesnosti ukázaly, že detekce těles dosáhla úspěšnosti v rozmezí od 98 do 100 %. Jedná se o měření v nejlepších podmínkách, kdy bylo dostupné dobré osvětlení a barva pozadí byla čistě bílá. Pokles o 1-2 % může být brán jako odchylka nepřesnosti vnějších podmínek, jako například vedlejší stín z okolních předmětů. Zajímavostí také je, že model umí rozpoznávat i podobná tělesa v jiných barvách. Například, nakreslený čtverec modrou propiskou na papír model vyhodnocuje na 70 % jako čtverec. I s vyšší entropií, výsledek měření ukazuje, že model funguje dobře a s vysokou přesností. Je však nutné zdůraznit, že se během testování vyskytlo několik zajímavých poklesů přesnosti.

Jeden z poklesů přesnosti byl způsoben jiným umístěním tělesa, než uprostřed záběru. V tabulce 4.1 je znázorněn přibližný pokles v procentech. Je třeba brát v úvahu, že kamerový modul při měření byl přibližně ve výšce 10 cm. To znamená, že posun tělesa o víc než 4 cm od středu je překročení hranice celého záběru.

Délka od středu [cm]	Pokles přesnosti [%]
2	5
3	8
4	12

Tabulka 4.1: Pokles přesnosti dle vzdálenosti od středu záběru

Pravděpodobnou příčinou tohoto poklesu je na tento faktor nepřípravená sada dat. Všechny tělesa byly vyfoceny uprostřed záběru. I když augmentace datasetu měla za úkol provádět transformaci obrázků, na mírný pokles přesnosti modelu tento úkon neměl vliv.

Druhým poklesem přesnosti byla rotace některých těles. Jedná se o otáčení čtverce a trojúhelníku přibližně o 45 stupňů, kdy těleso vypadá nejméně podobně k původnímu datasetu. Tento pokles je odůvodněn stejně jako předchozí. Původní sady dat, kde se objevují i rotace těchto těles, bylo málo pro nejvyšší možnou přesnost. Avšak, daný pokles nebyl nikdy větší, než 5 %, což ukazuje, že entropie modelu je pořád velmi nízká. To znamená, že model umí bez problémů rozpoznávat všechny očekávané typy těles.

4.2 Rychlost

Rychlost aplikace a samotného rozpoznávání bylo jednoduché změřit vzhledem k předem připravené funkcionalitě zobrazování odezvy. Výsledná latence aplikace tedy je 110-111 ms. Na toto zpoždění nemá vliv ani fáze rozpoznávání, ani konkrétní těleso, což je logické vzhledem k optimalizovanému počtu parametrů. Dle mého názoru, je tato odezva dostatečně dobrá, jelikož se jedná o zařízení s omezenými výpočetními prostředky.

Jednou z částí rychlosti je také počet snímků za vteřinu. V tomto směru mikrokontrolér zvládá 8,9-9 snímků za sekundu. Je to tedy dostačující pro plynulý výstup z kamery a dobrou uživatelskou zkušenost.

4.3 Použití jiných nástrojů

V dnešní době existuje více nástrojů, umožňujících automatické generování modelu dle samotného datasetu. Jedním z takových nástrojů je Teachable Machine od společnosti Google [38]. Jeho funkcionalita spočívá v učení modelu rozpoznávání obrazů dle předem přidané sady dat bez jiných složitých úprav. Vzhledem k této jednoduchosti, je daný nástroj dobrým výběrem pro porovnání s modelem bakalářské práce.

Po provedení potřebných úkonů k natrénování nového modelu, jsem provedl jeho analýzu pomocí prostředí STM32CubeMX. Oproti výsledku z kapitoly 3.4.2, nový model měl o 23,5 % větší použití Flash paměti a 2,8krát větší použití RAM paměti. Celkový výsledek je znázorněn na obrázku 4.1.

```
Total Flash:      651614 B (636.34 KiB)
  Weights:        539128 B (526.49 KiB)
  Library:        112486 B (109.85 KiB)
Total Ram:        736809 B (719.54 KiB)
  Activations:    702976 B (686.50 KiB)
  Library:        33828 B (33.04 KiB)
  Input:          150528 B (147.00 KiB included in Activations)
  Output:         5 B
Done
Analyze complete on AI model
```

Obrázek 4.1: Výstup analýzy modelu Teachable Machine

Podle mého názoru, lepší výsledek původního modelu je podmíněný použitím augmentace, která rozšířila dataset o nutné prvky pro lepší fungování optimalizačního algoritmu. To způsobilo menší počet parametrů a následně menší použití prostředků.

I přes horší výsledek analýzy, pro diagnostiku přesnosti a rychlosti na mikrokontroléru, jsem implementoval daný model do konečné aplikace. Porovnání výsledného testování

těchto modelů je ukázáno v tabulce 4.2

Model	Přesnost [%]	Latence [ms]	FPS
Původní	98-100	110-111	8,9-9
Nový	98-100	543-544	1,8

Tabulka 4.2: Porovnání modelů

Z tabulky je patrné, že nový model má násobně nižší odezvu a je méně přizpůsobený pro danou úlohu při původní sadě dat. Proto, původní postup trénování byl optimální pro zpracování této práce.

4.4 Plánovaný rozvoj

Konečná aplikace má perspektivu pro její následné použití v dalších aplikacích. Jedná se hlavně o systém uchopování těles, který by s pomocí dané aplikace zjistil polohu tělesa a nastavil manipulační kleště pod potřebným úhlem. Krom této možnosti, aplikaci nebo její úpravy lze dále rozvíjet pro veškeré potřeby, spojené s umělou inteligencí a rozpoznáváním obrazu v oblasti Internetu věcí.

Tato práce slouží také jako příručka pro analýzu a implementaci umělé inteligence na mikrokontroléru, což může být přínosné pro jiné práce tohoto typu.

Závěr

Táto práce se zabývá návrhem a implementací systému rozpoznávání tvarů těles pomocí umělé inteligence. Pojednává o výhodném modelu neuronové sítě a příslušných algoritmech. To vše v kontextu využití na mikrokontroléru.

Jednou z hlavních výzev návrhu byla výpočetní náročnost modelu, jelikož se jedná o zařízení s omezenými výpočetními prostředky. Proto je v první části práce provedena rešerše strojového učení, zejména pro řešení konkrétní úlohy rozpoznávání obrazu na mikrokontroléru. Součástí této rešerše je také analýza doporučeného mikrokontroléru STM32H747I-DISCO, jeho parametry a metody vývoje. Výsledkem daného výzkumu je porozumění problematice konvoluční neuronové sítě a možnostem její implementace na doporučený mikrokontrolér.

Po rešerši obecné problematiky se práce zaměřuje na metodiku zpracování obrazu pomocí modelu neuronové sítě. Jedná se o výběr optimální architektury, výzkum jejího následného učení a optimalizaci. Takovou architekturou se stala MobileNetV2, která ukazuje dobré výsledky nejenom dle odborné literatury, ale i konkrétně v této práci.

Další částí práce je praktický návrh a implementace celého systému. Jedná se o propojení součástí, získávání potřebných dat, trénování modelu a jeho optimalizace. Pro provedení těchto kroků byly použity poznatky z předchozího výzkumu.

Závěrem této práce je testování systému a jeho analýza. Tato část zahrnuje měření přesnosti a rychlosti, které ukázalo, že pro zpracování této práce byly vybrány optimální nástroje. Tyto měření potvrdily možnost použití daného systému pro další vývoj, zejména pro technologii uchopení těles pomocí robotického podavače na pohybujícím se povrchu.

Tato bakalářská práce je určitým náhledem do využití umělé inteligence na zařízeních, které nemají vysoké výpočetní možnosti. Zkoumá možnosti a nejlepší varianty pro implementaci fungujícího systému rozpoznávání obrazu, což je v dnešní době velice populární odvětví.

Seznam použité literatury

1. PATTERSON, Josh; GIBSON, Adam. *Deep learning: A practitioner's approach*. O'Reilly, 2017. ISBN 9781491914250.
2. GUO, Hung Jr.; MAGID, Evgeni; HSIA, Kuo-Hsien; SU, Kuo-Lan. Development of IoT Module with AI Function Using STM32 Chip. *Journal of Robotics, Networking and Artificial Life*. 2021, roč. 7, č. 4. ISSN 2405-9021. Dostupné z DOI: 10.2991/jrnal.k.201215.009.
3. STATISTA. *Number of connected IoT devices worldwide from 2018 to 2030 (in billions)*. 2021. Dostupné také z: <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>.
4. JIASI CHEN, Xukan Ran. Deep Learning With Edge Computing: A Review. *Proceedings of the IEEE*. 2019, roč. 107, č. 1, s. 259–274. Dostupné z DOI: 10.1109/JPROC.2019.2921977.
5. EL NAQA, I.; LI, R.; MURPHY, M. J. (ed.). *Machine Learning in Radiation Oncology*. Springer, 2015. ISBN 978-3-319-18304-6. Dostupné z DOI: 10.1007/978-3-319-18305-3.
6. ZANTALIS, Fotios; KOULOOURAS, Grigorios; KARABETSOS, Sotiris; KANDRIS, Dionisis. A Review of Machine Learning and IoT in Smart Transportation. *Future Internet*. 2019, roč. 11, č. 4, s. 94. Dostupné z DOI: 10.3390/fi11040094.
7. NASTESKI, Vladimir. An overview of the supervised machine learning methods. *HORIZONS.B*. 2017, roč. 4, s. 51–62. Dostupné z DOI: 10.20544/HORIZONS.B.04.1.17.P05.
8. JORDAN, Michael I; MITCHELL, Tom M. Machine learning: Trends, perspectives, and prospects. *Science*. 2015, roč. 349, č. 6245, s. 255–260. Dostupné z DOI: 10.1126/science.aaa8415.
9. CHAPELLE, Olivier; SCHÖLKOPF, Bernhard; ZIEN, Alexander. *Semi-Supervised Learning*. MIT Press, 2006. ISBN 978-0262033589.
10. SUTTON, Richard S.; BARTO, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press, 2018. ISBN 978-0262039246.
11. BROWNLEE, Jason. *Deep Learning for Computer Vision*. Machine Learning Mastery, 2018.

12. MERENDA, Massimo; PORCARO, Carlo; IERO, Demetrio. Edge Machine Learning for AI-Enabled IoT Devices: A Review. *Sensors*. 2020, roč. 20, č. 9. ISSN 1424-8220. Dostupné z DOI: 10.3390/s20092533.
13. STMICROELECTRONICS. *Microcontrollers-Microprocessors* [online]. 2023. [cit. 2023-03-27]. Dostupné z: <https://www.st.com/en/microcontrollers-microprocessors.html>.
14. STMICROELECTRONICS. *STM32H7 Microcontroller Datasheet*. STMicroelectronics, 2023. Dostupné také z: <https://www.st.com/resource/en/datasheet/stm32h747xi.pdf>.
15. STMICROELECTRONICS. *Discovery kit with STM32H747XI MCU* [online]. 2023. [cit. 2023-03-28]. Dostupné z: <https://www.st.com/en/evaluation-tools/stm32h747i-disco.html>.
16. STMICROELECTRONICS. *UM2779: Camera module bundle for STM32 boards*. STMicroelectronics, 2020. Dostupné také z: <https://www.st.com/en/evaluation-tools/b-cams-omv.html#documentation>.
17. STMICROELECTRONICS. *UM2526: Getting started with X-CUBE-AI Expansion Package for Artificial Intelligence (AI)*. STMicroelectronics, 2022. Dostupné také z: <https://www.st.com/en/embedded-software/x-cube-ai.html#documentation>.
18. NOVIELLO, Carmine. *Mastering STM32*. Leanpub, 2018. Dostupné také z: <https://leanpub.com/mastering-stm32>.
19. STMICROELECTRONICS. *Data brief STM32CubeIDE Integrated development environment for STM32 products* [STMicroelectronics Data brief]. 2021. Dostupné také z: <https://www.st.com/en/development-tools/stm32cubeide.html#documentation>.
20. STMICROELECTRONICS. *Intelligence at the Edge*. Mouser Electronics, 2022. Dostupné také z: <https://www.mouser.com/news/st-intelligenceattheedge-infographic/st-intelligenceattheedge-ebook.html>.
21. HOWARD, Andrew G; ZHU, Menglong; CHEN, Bo; KALENICHENKO, Dmitry; WANG, Weijun; WEYAND, Tobias; ANDREETTO, Marco; ADAM, Hartwig. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*. 2017.

22. DONG, Ke; ZHOU, Chengjie; RUAN, Yihan; LI, Yuzhi. MobileNetV2 Model for Image Classification. *2020 International Conference on Information Technology and Computer Application (ITCA)*. 2020, s. 476–480. Dostupné z DOI: 10.1109/ITCA52113.2020.00106.
23. SANDLER, Mark; HOWARD, Andrew; ZHU, Menglong; ZHMOGINOV, Andrey; CHEN, Liang-Chieh. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, s. 4510–4520. Dostupné z DOI: 10.1109/CVPR.2018.00474.
24. PAL, K. K.; SUDEEP, K. S. Preprocessing for Image Classification by Convolutional Neural Networks. In: *2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE, 2016.
25. SHORTEN, Connor; KHOSHGOFTAAR, Taghi M. A survey on image data augmentation for deep learning. *Journal of Big Data*. 2019, roč. 6, č. 1, s. 60.
26. CHLAP, Peter; MIN, Hyeyoung; VANDENBERG, Nicholas; DOWLING, Jason; HOLLOWAY, Lois; HAWORTH, Annette. A review of medical image data augmentation techniques for deep learning applications. *Journal of Medical Imaging and Radiation Oncology*. 2021, roč. 65, č. 5. Dostupné z DOI: 10.1111/1754-9485.13261.
27. JUNG, Alexander. *imgaug: Image augmentation for machine learning experiments*. GitHub, 2020. Dostupné také z: <https://github.com/aleju/imgaug>.
28. TIAN, Youhui. Artificial Intelligence Image Recognition Method Based on Convolutional Neural Network Algorithm. *IEEE*. 2020, roč. 8, s. 125731–125744. Dostupné z DOI: 10.1109/ACCESS.2020.3006097.
29. SHAHA, Manali; PAWAR, Meenakshi. Transfer Learning for Image Classification. In: *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. 2018, s. 656–660. Dostupné z DOI: 10.1109/ICECA.2018.8474802.
30. PANG, Bo; NIJKAMP, Erik; WU, Yi-Nung. Deep Learning With TensorFlow: A Review. *Journal of Educational and Behavioral Statistics*. 2019, roč. 45, č. 2, s. 227–248. Dostupné z DOI: 10.3102/1076998619872761.
31. LEE, Hagyeong; SONG, Jongwoo. Introduction to convolutional neural network using Keras; an understanding from a statistician. *Communications for Statistical Applications and Methods*. 2019, roč. 26, č. 6, s. 591–610.

32. HUBARA, Itay; COURBARIAUX, Matthieu; SOUDRY, Daniel; EL-YANIV, Ran; BENGIO, Yoshua. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*. 2017, roč. 18, č. 1, s. 6869–6898.
33. HOROWITZ, Mark. 1.1 computing’s energy problem (and what we can do about it). In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, s. 10–14.
34. STMICROELECTRONICS. *STM32CubeProg* [online]. [B.r.]. [cit. 2023-04-06]. Dostupné z: <https://www.st.com/en/development-tools/stm32cubeprog.html>.
35. STMICROELECTRONICS. *FP-AI-VISION1* [online]. [B.r.]. [cit. 2023-04-07]. Dostupné z: <https://www.st.com/en/embedded-software/fp-ai-vision1.html>.
36. STMICROELECTRONICS. *STM32AI Repository: Transfer Learning* [online]. [B.r.]. [cit. 2023-04-07]. Dostupné z: https://github.com/STMicroelectronics/stm32ai/tree/master/AI_resources/VISION/transfer_learning.
37. JAIS, Imran Khan Mohd; ISMAIL, Amelia Ritahani; NISA, Syed Qamrun. Adam optimization algorithm for wide and deep neural network. *Knowledge Engineering and Data Science*. 2019, roč. 2, č. 1, s. 41–46.
38. *Teachable Machine* [online]. Google, 2021 [cit. 2023-04-09]. Dostupné z: <https://teachablemachine.withgoogle.com>.

Seznam obrázků

1.1	Rozpoznávání obrazu pomocí CNN [1]	4
1.2	STM32H747I-DISCO	6
1.3	Kamerový modul B-CAMS-OMV	8
1.4	STM32CubeMX 6.8.0	9
1.5	STM32CubeIDE 1.12.0	10
2.1	Transfer learning	15
3.1	Zadní strana STM32H747I-DISCO	19
3.2	Schéma zapojení systému	20
3.3	STM32CubeProgrammer 2.13.0	20
3.4	Tělesa pro rozpoznávání	21
3.5	Vizualizace sady dat	22
3.6	Příkald augmentace	23
3.7	Graf trénování	24
3.8	Matice záměn	25
3.9	Počet vrstev neuronové sítě	27
3.10	Výstup analýzy modelu	27
3.11	Použití RAM paměti	27
3.12	Použití aplikace	30
4.1	Výstup analýzy modelu Teachable Machine	32

Seznam použitých zkratek

IoT - Internet of Things

CNN - Convolutional Neural Network

STM - STMicroelectronics

MB - Megabyte

RAM - Random Access Memory

SRAM - Static Random Access Memory

USB - Universal Serial Bus

LCD - Liquid Crystal Display

HAL - Hardware Abstraction Layer

QSPI - Queued Serial Peripheral Interface

JTAG - Joint Test Action Group

SWD - Serial Wire Debug

EEPROM - Electrically Erasable Programmable Read-Only Memory

IDE - Integrated Development Environment

API - Application Programming Interface

ELF - Executable and Linkable Format

FPS - Frames Per Second

PC - Personal Computer

Mpx - Megapixel

MHz - Megahertz

MPU - Memory Protection Unit

CPU - Central Processing Unit

CRC - Cyclic Redundancy Check

AI - Artificial intelligence

mil. - milion

ms - milisekunda

pJ - pikojoule

cm - centimetr

obr. - obrázek

Seznam tabulek

1.1	Porovnání řad STM32 dle výkonu	5
1.2	Rozdělení RAM pamětí	7
2.1	Porovnání architektur CNN	11
2.2	Struktura Inverted Residual bloku [23]	12
2.3	Porovnání knihoven pro augmentaci obrazových dat	14
2.4	Tabulka spotřeby energie dle operace	17
3.1	Počet snímků dle tělesa	22
3.2	Počet parametrů modelu	25
4.1	Pokles přesnosti dle vzdálenosti od středu záběru	31
4.2	Porovnání modelů	33

Přílohy

Přílohu tvoří soubor `Andrusyak_Viktor_2023_BP_prilohy.zip`, který se skládá ze tří složek. Tyto složky obsahují:

- Soubory hotového AI modelu dle kapitoly 3.4.2.
- Důležité soubory, které byly zmíněny během implementace.
- Ostatní soubory projektu.