VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE OBJEKTŮ NA MIKROKONTROLERU I.MX RT

OBJECT DETECTION ON THE I.MX RT MICROCONTROLLER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARINA KRAVCHUK**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

153412

Ústav: Ústav inteligentních systémů (UITS)
Studentka: **Kravchuk Marina**
Program: Informační technologie
Název: **Detekce objektů na mikrokontroleru i.MX RT**
Kategorie: Vestavěné systémy
Akademický rok: 2023/24

Zadání:

1. Nastudujte problematiku neuronových sítí. Nastudujte a popište, jak se dají využít neuronové sítě na vestavěných zařízeních a jaká omezení pro tyto technologie přináší mikrokontrolery.
2. Seznamte se s middlewarem a nástroji, umožňujícími využití neuronových sítí na mikrokontroleru i.MX RT. Seznamte se také s nástroji pro tvorbu GUI na PC.
3. Navrhněte a realizujte aplikaci využívající neuronovou síť na i.MX RT, která bude schopná detekovat na jakých souřadnicích se v prostoru snímaném kamerou nachází předmět uživatelem vybrané barvy a tvaru. Dále navrhněte a realizujte jednoduchou aplikaci s GUI na PC, která nabídne uživateli seznam dostupných tvarů a barev, danou volbu sdělí aplikaci na i.MX RT a následně zobrazí výsledek detekce.
4. Otestujte a vyhodnoťte použitelnost finální aplikace, diskutujte možnosti budoucího vývoje.

Literatura:
Dle pokynů vedoucího a konzultanta.

Při obhajobě semestrální části projektu je požadováno:
První 2 body zadání a návrh.

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

Vedoucí práce: **Janoušek Vladimír, doc. Ing., Ph.D.**
Konzultanti: Stružka Petr, Ing. - NXP
Šimek Václav, Ing.
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 6.11.2023

Abstrakt

Tato práce se zaměřuje na využití strojového učení, zejména konvolučních neuronových sítí, v průmyslových aplikacích. Průběh práce zahrnuje zkoumání implementace těchto sítí přímo na vestavěných zařízeních, konkrétně na mikrokontrolérech NXP i.MX RT. Během studia byly prozkoumány materiály týkající se trénování a použití neuronových sítí a jejich optimalizace pro nasazení na zařízeních s nízkým výkonem. Bylo natrénováno a otestováno několik modelů neuronových sítí, z nichž nejlepší byl použit v konečné verzi aplikace. Samotná aplikace je rozdělena do dvou částí: jedna část je napsána v jazyce C/C++ v prostředí MCUXpresso IDE, kde je implementována hlavní funkcionalita programu, zatímco druhá část práce, tj. vytvoření grafického uživatelského rozhraní pro ovládání programu, je provedena v jazyce Python. Výsledkem je funkční aplikace pro mikrokontrolér MIMXRT1170-EVK, která je schopna detekovat a rozpoznávat malé barevné objekty určitých tvarů z předem definované sady dat.

Abstract

This work focuses on the use of machine learning, particularly convolutional neural networks, in industrial applications. The course of work involves investigating the implementation of these networks directly on embedded devices, specifically NXP i.MX RT microcontrollers. During the course of the study, materials related to the training and use of neural networks and their optimization for deployment on low power devices were reviewed. Several neural network models were trained and tested, the best of which was used in the final version of the application. The application itself is divided into two parts: one part is written in C/C++ in the MCUXpresso IDE, where the main functionality of the program is implemented, while the other part of the work, i.e. the creation of a graphical user interface to control the program, is done in Python. The result is a functional application for the MIMXRT1170-EVK microcontroller that is able to detect and recognize small colored objects of certain shapes from a predefined data set.

Klíčová slova

detekce objektů, neuronová síť, konvoluční neuronová síť, zpracování obrazu, vestavěné systémy, NXP i.MX RT.

Keywords

object detection, neural network, convolutional neural network, image processing, embedded systems, NXP i.MX RT.

Citace

KRAVCHUK, Marina. *Detekce objektů na mikrokontroleru i.MX RT*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Vladimír Janoušek, Ph.D.

Detekce objektů na mikrokontroleru i.MX RT

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Vladimíra Janouška Ph.D. Další informace mi poskytli Ing.Petr Stružka a Ing. David Piškula. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Marina Kravchuk
7. května 2024

Poděkování

Ráda bych poděkovala panu Vladimíru Janouškovi za vedení této bakalářské práce a kolegovi Davidu Piskulovi z firmy NXP za pravidelné konzultace, cenné rady a podporu v průběhu studijního roku. Děkuji také svému příteli a kamarádce za jejich významnou podporu a povzbuzení v průběhu mého studia.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V průmyslovém a technologickém světě se strojové učení a umělá inteligence stávají klíčovými prvky přinášejícími inovace a zvýšenou efektivitu. Jedním z důležitých směrů je detekce objektů a jejich následná manipulace v prostředích, jako jsou továrny a montovny. Tyto operace jsou klíčové pro zvyšování produktivity a automatizaci procesů.

Tento výzkum se zabývá aplikací technologie detekce objektů, řešené prostřednictvím neuronových sítí, na vestavěných zařízeních, konkrétně mikrokontroleru i.MX RT. Lokalizace a identifikace objektů jsou kritické pro následné operace v průmyslovém prostředí, zejména pro řízení robotických paží.

Cílem této práce je nastudovat a implementovat aplikaci založenou na neuronové síti, která bude schopná detekovat a lokalizovat objekty zvolených tvarů a barev. Pro dosažení cíle byla provedena důkladná analýza základů neuronových sítí, možnosti jejich využití na vestavěných zařízeních a omezení vyplývající z použití mikrokontrolerů.

Dalším důležitým aspektem této práce je vývoj přívětivého grafického uživatelského rozhraní (GUI) umožňujícího uživatelům výběr barev a tvarů detekovaných objektů. Toto rozhraní je implementováno na standardním počítači, a dále komunikuje s mikrokontrolerem i.MX RT, který provádí samotnou detekci objektů a následně poskytuje výsledky zpět uživateli.

V následujících kapitolách této práce jsou detailně popsány základy konvolučních neuronových sítí, význam a výhody běhu těchto sítí na vestavěných zařízeních, konkrétně na mikrokontroleru MIMXRT1170-EVK. Dále jsou prezentovány použité nástroje a knihovny pro vytvoření aplikace, návrh a zkoumání existujících modelů pro detekci objektů a jejich následné spuštění na mikrokontroleru MIMXRT1170-EVK.

Kapitola 2

Neuronové sítě

Neuronová síť je technika umělé inteligence, která učí počítače zpracovávat data. Jedná se o typ procesu strojového učení nazývaného hluboké učení, který využívá vzájemně propojené uzly nebo neurony ve vrstevnaté struktuře, která se podobá lidskému mozku. Vytváří adaptivní systém, díky němuž se počítače učí ze svých chyb a neustále se zlepšují. Umělé neuronové sítě se tak pokoušejí řešit složité problémy, jako je shrnutí dokumentů nebo rozpoznávání tváří, s vyšší přesností.

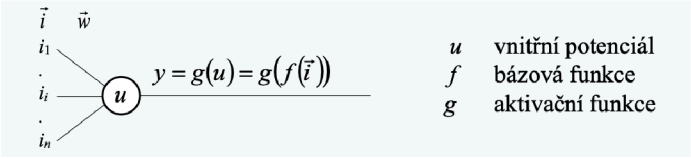
2.1 Neuronové sítě a jejich architektury

Perceptron

Inspirací pro perceptron je základní prvek nervové soustavy živých organizmů, nervová buňka – **neuron** [39]. Každý biologický neuron se skládá z těla (somy), ke kterému je připojeno přibližně tisíc (v rozsahu 102 až 105) 2-3 mm dlouhých dendritů (vstupů) a jeden, 1 cm až několik desítek cm dlouhý axon (výstup) [39].

Hodnoty signálů od jednotlivých dendritů se v těle neuronu sčítají. Přesáhne-li jejich celková hodnota jistý práh, neuron vyšle krátký impuls o velikosti asi 100 mV, který postupuje po jeho axonu rychlostí 0.1 až 10 m/s. Pak se neuron na krátkou dobu stane necitlivým. Pokud je po této době celková hodnota vstupního signálu stále větší než práh, impuls se opakuje. Po axonu se tak šíří pulsy o frekvenci 0.1/s až 100/s [46].

Pro umělý neuron by byla činnost s podobnou výstupní funkcí obtížně realizovatelná. Proto se používají jednodušší modely, jejichž výstupní signály jsou statické.



Obrázek 2.1: Model neuronu. Převzato z [39].

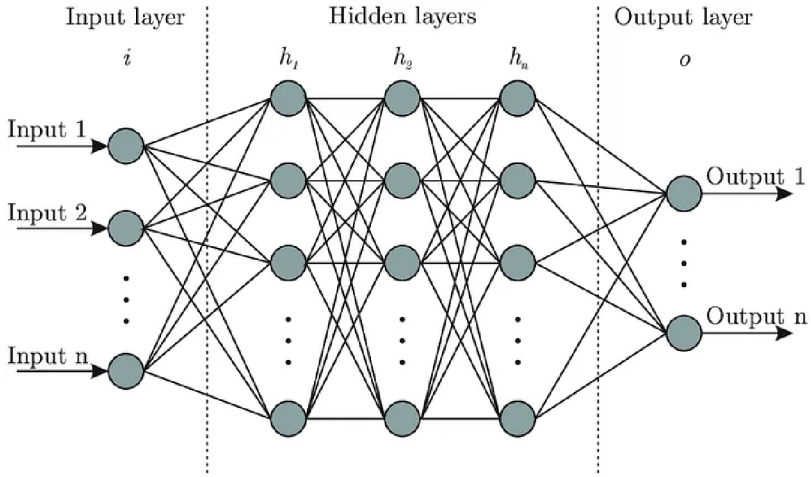
Perceptron je nejznámějším umělým neuronem. Pro zjednodušení vztahů pro výpočet odezvy a pro učení doplňuje v obecném modelu vstupní vektor o nultou složku, která má pevnou hodnotu 1 a jejíž váha se nastavuje na hodnotu záporného prahu [39].

Perceptron je velmi jednoduchý model neuronu. Perceptrony mají omezený výkon a jsou omezeny na řešení lineárních problémů. Jsou schopny řešit jednoduché problémy, jako je

lineární klasifikace, ale selhávají u složitějších problémů, které vyžadují nelineární reprezentaci.

Neuronová síť

Neuronová síť je souhrn mnoha neuronů. Jedna z možných variant architektury neuronové sítě je na obrázku 2.2. Skládá se ze *vstupní*, *výstupní* a *skrytých* vrstev. Vstupní vrstva obsahuje neurony s jedním vstupem, které nic nepočítají a jejich jediným úkolem je předávat vstupní signály ostatním neuronům. Neurony výstupní vrstvy naopak mají pouze jeden výstup. Zbylým vrstvám se říká skryté, jejich neurony jsou navzájem propojeny. Neuronová síť má schopnost učit se a přizpůsobovat se vstupním datům. Samotné neuronové sítě neboli umělé neuronové sítě (ANN) jsou podmnožinou strojového učení, která má napodobit výpočetní výkon lidského mozku. Neuronové sítě fungují tak, že data procházejí vrstvami umělého neuronu [11].



Obrázek 2.2: Architektura neuronové sítě. Převzato z [16].

Každý neuron má uvnitř **bázovou** a **aktivační funkci**. Aktivační funkce určuje výstupní hodnotu neuronu v závislosti na výsledku bázové funkci, například váženého součtu vstupů a prahové hodnoty.

Uvažujme neuron s bázovou funkci f:

$Y = f(n) = f(\sum(weights * inputs) + bias)$

Nyní může být hodnota Y libovolná v rozmezí -nekonečno až +nekonečno. Ve skutečnosti neuron neví, za jakou hranicí následuje aktivace. Odpovězme si na otázku, jak se rozhodujeme, zdá má být neuron aktivován? Uvažujeme aktivační vzorec, protože můžeme použít analogii s biologií. Takto funguje mozek a mozek je dobrým dokladem fungování složitého a inteligentního systému [10].

Za tímto účelem vzniklo rozhodnutí přidat aktivační funkci. Ta kontroluje hodnotu Y produkovanou neuronem, aby zjistila, zdá mají vnější spojení považovat tento neuron za aktivovaný, nebo zdá jej lze ignorovat [10].

Nejjednodušším typem aktivační funkce je kroková funkce, kdy platí

$$f(n) = \begin{cases} 1 & \text{pokud } n > prah \\ 0 & \text{jinak} \end{cases}$$

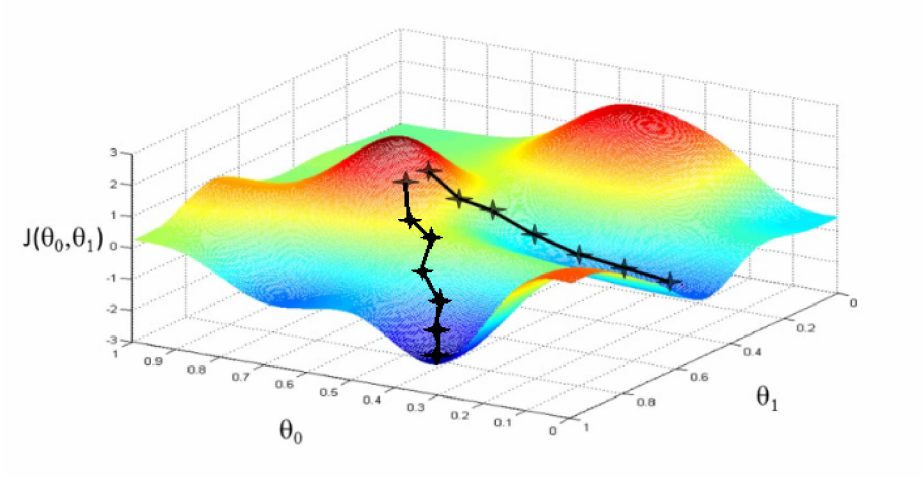
Ztrátová funkce (loss function), také nazývána cílová funkce (objective function), je klíčovým prvkem v procesu učení strojových modelů, včetně neuronových sítí. Slouží k určení, jak dobře model reprezentuje data a jak blízko je jeho predikce k očekávaným výstupům (cílům). Účelem ztrátové funkce je kvantifikovat rozdíl mezi předpovězenými hodnotami a reálnými hodnotami [19]. Pokud je tento rozdíl velký, ztrátová funkce bude vysoká, což signalizuje modelu, že musí zlepšit své predikce během trénování.

Ztrátové funkce mají klíčový vliv na trénování modelu, protože model se snaží minimalizovat hodnotu ztrátové funkce. Jaký typ ztrátové funkce použijete, závisí na povaze problému, který řešíte (například klasifikace, regrese, detekce objektů atd.) [9]. Zde je několik běžně používaných typů ztrátových funkcí:

- **Střední kvadratická chyba (Mean Squared Error, MSE)**: Používá se při problémech regrese. Měří střední hodnoty druhých mocnin mezi predikcemi a reálnými hodnotami [8].
- **Křížová entropie (Cross-Entropy)**: Používá se při klasifikaci. Měří, jak dobře model klasifikuje data tím, jak je blízko jeho predikce k pravděpodobnostem tříd. Běžná varianta zahrnuje binární křížovou entropii pro dvě třídy a kategorickou křížovou entropii pro více tříd [8].
- **IoU Loss (Intersection over Union Loss)**: Používá se při detekci objektů. Měří míru překryvu mezi predikovaným rámečkem a skutečným rámečkem objektu. Cílem je maximalizovat tuto hodnotu [17].
- **Ztrátová funkce váhová (Weighted Loss Function)**: Umožňuje dát různou váhu různým vzorkům v tréninkovém datasetu. Toto je užitečné, pokud máte nevyvážené třídy nebo pokud chcete zdůraznit určité vzorky [43].

Výběr vhodné ztrátové funkce je kritický, protože může významně ovlivnit konvergenci a výkonnost modelu. Různé úlohy a modely mohou vyžadovat různé ztrátové funkce. Je také důležité vzít v úvahu, že ztrátová funkce by měla být diferencovatelná, aby bylo možné provádět **zpětnou propagaci chyby** (backpropagation) a aktualizaci vah modelu během trénování.

Zpětná propagace chyby, známá také jako backpropagation, je klíčovým algoritmem používaným při trénování neuronových sítí. Tento algoritmus umožňuje modelu adaptovat své váhy tak, aby minimalizoval chybu mezi jeho predikcemi a skutečnými hodnotami [5]. Zpětná propagace chyby funguje na základě **gradientního sestupu** a spočívá v iterativním vylepšování váh neuronů sítě tak, aby se chyba co nejvíce snížila. Gradientní sestup je metoda umožňující numerického nalezení jednoho z lokálních minim některých funkcí [5].



Obrázek 2.3: Gradientní sestup. Převzato z [28].

Následující je podrobný popis (inspirováno zdrojem [5]) zpětné propagace chyby:

- **Inicializace vah**: Nejprve jsou váhy neuronů inicializovány náhodnými hodnotami nebo jiným způsobem. Tyto váhy určují, jakým způsobem se vstupy transformují na výstupy sítě.
- **Průchod vpřed (forward pass)**: Během průchodu vpřed se vstupy posílají sítí, a každý neuron provádí váhovou lineární transformaci vstupů a následně použije aktivační funkci na výstup této transformace. Tím se získají výstupy sítě.
- **Výpočet chyby (loss calculation)**: Poté, co jsou získány výstupy sítě, se vypočítá hodnota chybové funkce, která měří rozdíl mezi předpovězenými hodnotami a skutečnými hodnotami cíle. Typ chybové funkce závisí na konkrétním problému a úkolu, například křížová entropie pro klasifikaci nebo kvadratická chyba pro regresi.
- **Zpětná propagace (backpropagation)**: Zpětná propagace začíná výpočtem gradientů chyby vzhledem ke všem váhám sítě. Gradient chyby se vypočítá pro každý váhový parametr pomocí řetězového pravidla (chain rule). Cílem je určit, jakým směrem a o jakou hodnotu se musí váhy upravit, aby se minimalizovala chyba.
- **Aktualizace vah (weight update)**: Váhy neuronů sítě jsou aktualizovány v souladu s vypočítanými gradienty. Nejběžnější metodou aktualizace vah je gradientní sestup (gradient descent), kde váhy jsou upraveny tak, aby chyba klesla. Rychlost učení (learning rate) je hyperparametr, který určuje, jak velký krok je při aktualizaci vah použit.
- **Opakování procesu (iteration)**: Celý proces průchodu vpřed, výpočtu chyby, zpětné propagace a aktualizace vah je opakován pro každý tréninkový vzorek v datasetu. Tím se sítě umožní adaptovat se na různé vzory v datech.
- **Opakování epoch (epochs)**: Trénování sítě obvykle zahrnuje opakované iterace přes celý tréninkový dataset, které se nazývají epochy. Během těchto epoch sítě postupně zlepšují své schopnosti a minimalizují chybu.

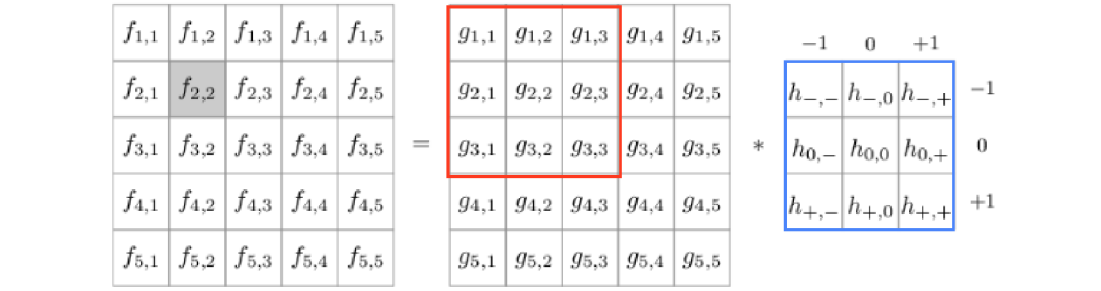
2.2 Základy konvolučních neuronových sítí (CNNs)

Pro úspěšnou detekci objektů se často využívají konvoluční neuronové sítě (Convolutional Neural Networks, CNNs). Tyto sítě jsou optimalizovány pro práci s více-dimenzionálními daty, jako jsou obrazová data. Hlavní charakteristikou CNNs je použití konvolučních vrstev pro automatickou extrakci různých obrazových rysů a hierarchickou reprezentaci dat [27]. CNNs jsou základním stavebním kamenem mnoha úspěšných algoritmů pro detekci a klasifikaci objektů v obrazech a videích.

Konvoluce a konvoluční vrstva

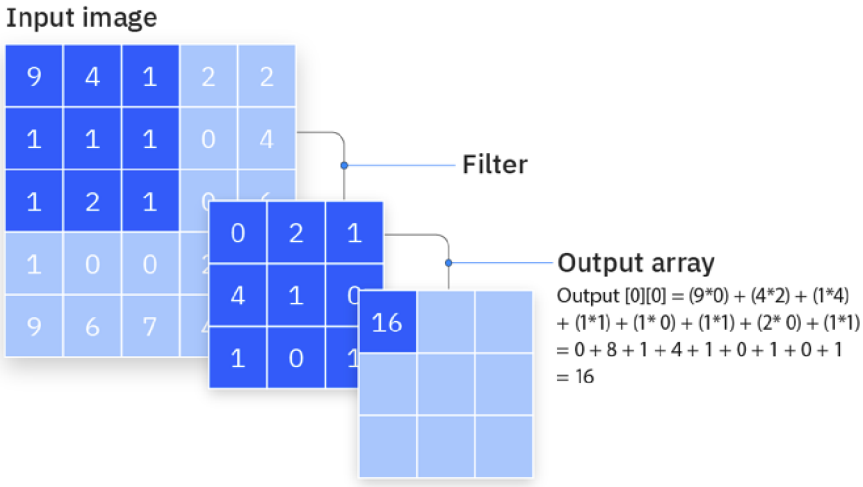
Mnoho výsledků zpracování obrazu vychází z úpravy jednoho pixelu vzhledem k jeho sousedům. Pokud je tato modifikace podobná v celém obrazu g, lze ji matematicky definovat pomocí druhého obrazu h, který definuje sousední vztahy [32]. Výsledkem je třetí obraz . Jedná se o tzv. konvoluci a označuje se pomocí $\star$:

$$f(x,y) = (g \star h)(x,y) = \sum_m \sum_n g(x-m, y-n)h(m,n)[32]$$



Obrázek 2.4: Příklad pro výpočet pixelu f(2,2). Převzato z [32].

Konvoluční vrstva (dale jen CONV) funguje na stejném principu. Parametry vrstvy CONV se skládají ze sady natrénovaných filtrů. Každý filtr je prostorově malý (na šířku a výšku), ale pokrývá celou hloubku vstupního objemu. Například typický filtr v první vrstvě CNN může být 5x5x3 (tj. 5 pixelů na šířku a výšku a 3, protože obrázky mají hloubku 3, tj. barevné kanály). Během dopředného průchodu posunujeme (nebo spíše konvolujeme) každý filtr po šířce a výšce vstupního svazku a počítáme bodový součin mezi výskyty filtru a vstupním signálem v libovolné pozici. Pohybem filtru po šířce a výšce vstupního svazku se vytvoří dvourozměrná aktivační mapa, která odráží odezvy daného filtru v každé prostorové poloze. Intuitivně se síť učí filtry, které se aktivují, když vidí nějaký vizuální prvek, například hranu určité orientace nebo skvrnu určité barvy na první vrstvě nebo identifikuje abstraktnější a složitější vlastnosti, jako jsou tvary, textury nebo dokonce koncepty, a kombinuje základní vlastnosti do reprezentací objektů a obrazů na vyšších vrstvách sítě. Nyní budeme mít v každé vrstvě CONV celou sadu filtrů (např. 12 filtrů), z nichž každý bude tvořit samostatnou 2D aktivační mapu. Pro vytvoření výstupního objemu budeme tyto aktivační mapy skládat na sebe podle hloubky.



Obrázek 2.5: Příklad filtru. Převzato z [27].

Celkově konvoluční vrstvy mají za úkol zpracovat obrazová data a extrahovat různé obrazové rysy, jako jsou hrany, textury, tvary a další vizuální charakteristiky, které jsou důležité pro úkoly jako klasifikace, detekce objektů a segmentace obrazu. Tyto vrstvy umožňují modelu efektivně pracovat s obrazovými daty a dosahovat vysoké úspěšnosti v různých úlohách zpracování obrazu.

2.3 Detekce objektů

Detekce objektů je technika počítačového vidění, která slouží k identifikaci a lokalizaci objektů v obraze nebo videu. Detekce objektů konkrétně vykresluje ohraničující boxy kolem těchto detekovaných objektů, které nám umožňují lokalizovat, kde se tyto objekty v dané scéně nacházejí (nebo jak se v ní pohybují). Pomocí detekce objektů je možné detekovat několik objektů z různých kategorií najednou.

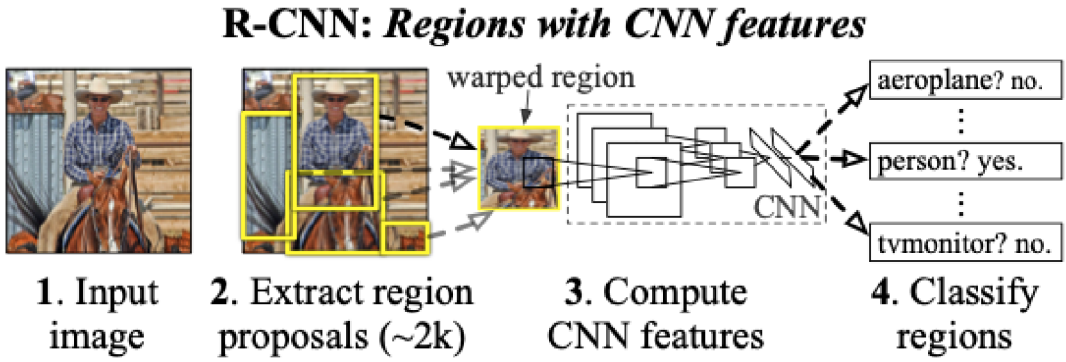
Metriky detekce objektů

Průměrná přesnost (AP), průměrná přesnost detekce (mAP), Intersection over Union (IoU) jsou nejpopulárnější metriky používané k hodnocení modelů detekce objektů, jako jsou mimo jiné Faster R-CNN, Mask R-CNN a YOLO.

R-CNN, Faster R-CNN, Mask R-CNN

Řada populárních modelů detekce objektů patří do rodiny R-CNN. Tyto modely byly klíčovým krokem ve vývoji pokročilých systémů detekce objektů a inspirovaly přístupy, které jsou využívány i v modernějších modelových architekturách, jako je například Faster R-CNN [3]. Běžný popis fungování takového modelu pomocí obrázku 2.6:

Systém R-CNN (1) přijme vstupní obrázek, (2) extrahuje přibližně 2000 návrhů oblastí zdola nahoru, (3) vypočítá rysy pro každý návrh pomocí velké konvoluční neuronové sítě (CNN) a poté (4) klasifikuje každou oblast pomocí lineárních SVM specifických pro danou třídu [24].



Obrázek 2.6: Příklad R-CNN. Převzato z [24].

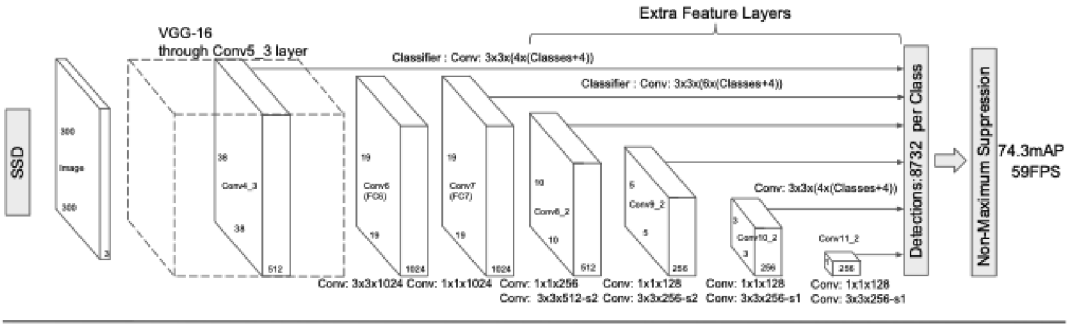
R-CNN modely, i když byly průlomovým krokem, trpěly pomalým procesem kvůli velkému počtu návrhů oblastí a náročnému trénování. Tyto nedostatky vedly k vývoji rychlejších a efektivnějších architektur, jako je Faster R-CNN, která kombinuje generování návrhů oblastí s konvoluční sítí, což vede ke zvýšení rychlosti a přesnosti detekce. Modely R-CNN se poměrně široce používají v současných problémech souvisejících s detekcí objektů. Přesto však nebudou v této práci použity. Tyto modely jsou poměrně velké a časově náročné, což je pro účel práce zcela nevhodné.

Single shot detector family

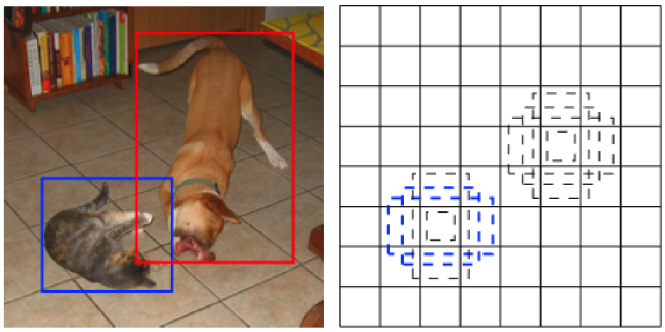
Single Shot Detector je přístup k detekci objektů v obrazech, který byl navržen tak, aby byl rychlý a efektivní. SSD je založen na ideji, že detekce objektů by měla být prováděna jedním průchodem sítě, což znamená, že není třeba generovat návrhy oblastí (region proposals) jako v případě R-CNN modelů.

SSD má dvě části: **backbone** a **head**. Backbone je obvykle předem natrénovaná síť pro klasifikaci obrazu (na obrázku 2.7 je například uvedená VGG-16). Zůstává nám tedy hluboká neuronová síť, která je schopna extrahovat sémantický význam ze vstupního obrázku. Hlava SSD je pouze jedna nebo více konvolučních vrstev přidaných k backbone a výstupy jsou interpretovány jako ohraničující boxy a třídy objektů v prostorovém umístění aktivací koncových vrstev. SSD rozdělí obraz pomocí mřížky a každá buňka mřížky je zodpovědná za detekci objektů v dané oblasti obrazu. Pokud není přítomen žádný objekt, považujeme jej za třídu pozadí a umístění je ignorováno. Každá buňka mřížky je schopna vypsat polohu a tvar objektu, který obsahuje 2.8 [42].

V této práci bude vyzkoušen jeden z populárních SSD modelů – MobileNet-SSD V2. Architektura modelu zůstává stejná jako na obrázku 2.7, s výjimkou backbone. Pro optimalizaci velikosti sítě, nahradíme nakonec síť VGG-16 sítí MobileNetV2 a získáme síť MobileNet-SSD V2 jako model. SSD je mnohem rychlejší a přesnější, zejména rychlejší než R-CNN a přesnější než Yolo. MobileNetV2 je lehká a může splňovat požadavky vestavěných zařízení. Přenesené učení může zlepšit výkon a snížit závislost na velkých souborech dat. V neposlední řadě dokáže systém detekovat masky v reálném čase za složitých scénářů [18].



Obrázek 2.7: Architektura ssd. Převzato z [30].



Obrázek 2.8: SSD process. Převzato z [30].

YOLO

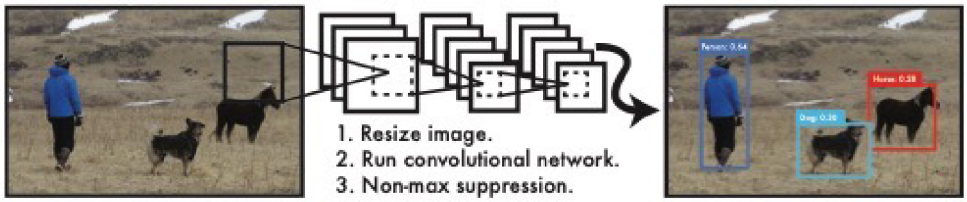
You Only Look Once (YOLO) jedním z populárních přístupů k detekci objektů v obrazech a videích. Jeho klíčovou vlastností je schopnost provést detekci objektů v jediném průchodu sítí.

Jak již bylo uvedeno, přístupy, jako je R-CNN, využívají navrhování oblasti. Tyto metody nejprve vygenerují potenciální ohraničující boxy v obraze a poté na těchto navržených boxech spustí klasifikátor. Po klasifikaci se použije následné zpracování ke zpřesnění hranic, odstranění duplicitních detekcí a opětovnému odhadu hranic na základě dalších objektů ve scéně [24]. Tyto složité pipeline jsou pomalé a obtížně se optimalizují, protože každá komponenta musí být trénována zvlášť.

YOLO považuje detekci objektů za jediný regresní problém, a to přímo z pixelů obrazu na souřadnice ohraničujících boxů a pravděpodobnosti tříd. Při použití tohoto systému „se na obraz podíváte pouze jednou“ (což znamená, že obraz projde modelem pouze jednou), abyste předpověděli, které objekty jsou přítomny a kde se nacházejí [41].

Zpracování obrázků pomocí aplikace YOLO je jednoduché a přímočaré. Příklad fungování modelu na obrázku 2.9. Systém (1) změní velikost vstupního obrázku na 448 × 448, (2) spustí na obrázku jedinou konvoluční síť a (3) stanoví prahové hodnoty výsledných detekcí podle důvěryhodnosti modelu [41].

YOLO modely jsou schopny rychle a efektivně detekovat objekty v reálném čase. Jsou obzvláště vhodné pro scénáře, kde je důležitá rychlost, například v autonomním řízení vozidel nebo sledování objektů v reálném čase. Různé verze YOLO (např. YOLOv1, YOLOv2,

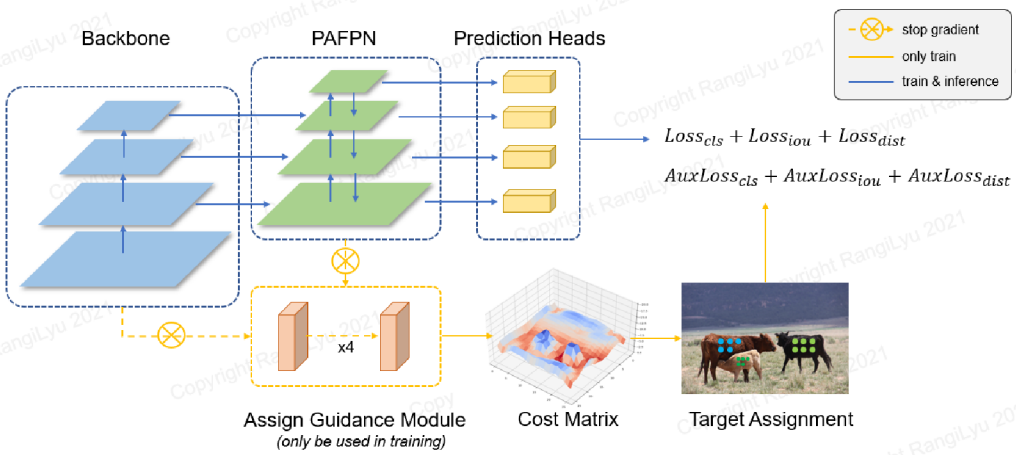


Obrázek 2.9: YOLO process. Převzato z [41].

YOLOv3, atd.) přinesly vylepšení v přesnosti detekce a obecnější použití. V tomto projektu bude testován model YOLOv3.

NanoDet

NanoDet je model detekce objektů, který byl navržen tak, aby byl extrémně lehký a efektivní. Jedná se o méně populární model, který je však vhodný pro účely tohoto projektu. Jeho architektura je optimalizována pro vestavěná zařízení nebo situace, kdy je výpočetní výkon omezený, jako jsou robotické systémy nebo zařízení pro sledování objektů v reálném čase [40].

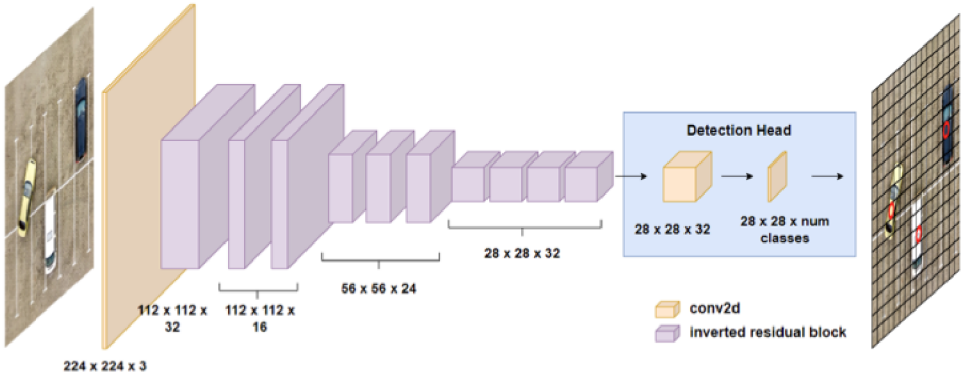


Obrázek 2.10: Nanodet struktura. Převzato z [40].

Architektura NanoDetu je podobná výše popsané struktuře SSD 2.10, ale navíc obsahuje krk nebo-li anglický **neck**. **Backbone**: jako backbone NanoDet používá ShufflenetV2 [31], což je velmi robustní a cenově výhodná struktura. **Neck**: NanoDet používá pro extrakci map příznaků Pyramid Attention Network (PAN), ale všechny konvoluce kromě konvolucí 1×1 byly odstraněny pro vytvoření modelu s nižší komplexitou. **Head**: NanoDet používá v hlavě konvoluci s hloubkou, aby se model zmenšil. Regrese a klasifikace hranic se počítá pomocí stejných konvolucí a později se rozdělí na dvě části, aby se snížil počet výpočtů [34]. Celkově lze říci, že NanoDet ukazuje dobré výsledky na zařízeních s omezenými zdroji. Proto bude i tento model (verze NanoDet-m-0.5x) testován v rámci tohoto projektu.

FOMO

Další zajímavý model se nazývá FOMO. Edge Impulse FOMO (Faster Objects, More Objects) je nový algoritmus strojového učení, který přináší detekci objektů do velmi omezených zařízení. Umožňuje počítat objekty, zjišťovat polohu objektů v obraze a sledovat více objektů v reálném čase s využitím až 30× menšího výpočetního výkonu a paměti než MobileNet SSD nebo YOLOv5 [22].

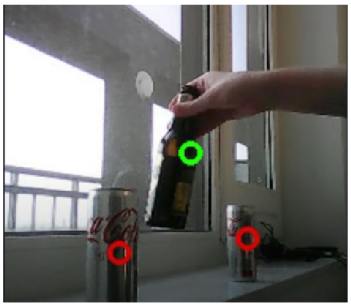


Obrázek 2.11: FOMO struktura. Převzato z [15].

Přehled architektury FOMO. Uvádíme velikosti filtrů pro příklad vstupního obrazu o velikosti 224 × 224 × 3. V tomto příkladu vytváří extraktor příznaků mřížku 28 × 28 příznaků, pro kterou detekční hlava klasifikuje každou buňku mřížky, zda obsahuje objekt, nebo ne. Konečný výstup po zpracování je zobrazen vpravo 2.11 [15].

FOMO v podstatě využívá architekturu MobileNet, ale odstraňuje poslední vrstvy standardního modelu klasifikace obrazu a nahrazuje je mapou pravděpodobnosti tříd pro každou oblast. Vlastní ztrátová funkce pak nutí síť, aby v poslední vrstvě plně zachovávala lokálnost. Tím se v podstatě získá heatmap umístění objektů.

Rozdíl mezi FOMO a ostatními algoritmy detekce objektů spočívá v tom, že neodvozuje ohraničující boxy, ale pouze střed objektu. Ale v praxi velmi často není důležitá velikost objektů, ale spíše jen jejich umístění a počet.



Obrázek 2.12: FOMO output. Převzato z [15].

Kapitola 3

Vestavěné systémy

Vestavěné systémy jsou speciální typy počítačových systémů, které jsou navrženy a integrovány pro specifické úkoly nebo aplikace. Tyto systémy mají omezené prostředky a jsou optimalizovány pro konkrétní účely, což je odlišuje od univerzálních počítačů. Při návrhu mikrokontrolérů je třeba hledat kompromis mezi velikostí a cenou na jedné straně a flexibilitou a výkonem na straně druhé. Pro různé aplikace se může optimální rovnováha mezi těmito a dalšími parametry značně lišit. Proto existuje tolik typů mikrokontrolérů, které se liší architekturou procesorového modulu, velikostí a typem vestavěné paměti, periferními zařízeními, typem pouzdra atd.

3.1 Mikrokontroler i.MX RT1170

V této práci budeme používat desku MIMXRT1170-EVK. Tato deska EVK je platformou navrženou pro prezentaci nejčastěji používaných funkcí procesoru i.MX RT1170 v malém a levném balení. Deska MIMXRT1170-EVK je základní vývojovou deskou, která dává vývojářům možnost seznámit se s procesorem dříve, než investují velké množství financí nebo jiných prostředků do specifičtějších návrhů [35].

3.2 Technické specifikace a schopnosti

MCU i.MX RT1170 Crossover jsou dvoujádrová zařízení s čipy Arm® Cortex®-M7 a Arm® Cortex®-M4 pro výkon mikrokontrolérů (MCU) v reálném čase a vysokou integraci pro automobilové, průmyslové a IoT aplikace [36]. Procesor Cortex-M4 [12] je široce používán v moderních vestavných zařízeních s mikrokontroléry, pro které je speciálně navržen. Procesor využívá třístupňové zpracování vstupních dat, tj. čtení, dekódování a vykonávání. Řada Cortex-M4 je poměrně rychlá, zejména při zpracování digitálního signálu. Díky podpoře operací s plovoucí desetinnou čárkou lze některé instrukce provést za méně hodinových cyklů. Cortex-M7 [12] je nejvýkonnější člen rodiny. Poskytuje všechny možnosti Cortex-M3 [12] a Cortex- M4 s vylepšenou podporou operací s plovoucí řádovou čárkou. Tento procesor cílí na aplikace s rozpoznáním řeči, zpracování obrazu a automobilový průmysl.

Procesor i.MX RT1170 má celkem 2 MB paměti RAM na čipu, včetně 512 KB paměti RAM, kterou lze flexibilně konfigurovat jako TCM nebo univerzální paměť RAM na čipu. Čip i.MX RT1170 integruje pokročilý modul správy napájení s DC-DC a LDO, který snižuje složitost externího napájení a zjednodušuje sekvenci napájení [35]. Poskytuje různá paměťová rozhraní, včetně SDRAM, Raw NAND FLASH, NOR FLASH, SD/eMMC, Quad SPI,

Hyper RAM/Flash. Poskytuje také širokou škálu dalších rozhraní pro připojení periferních zařízení, jako jsou WLAN, BluetoothR, GPS, displeje a senzory kamer. Stejně jako ostatní procesory i.MX má i i.MX RT1170 bohaté audio a video funkce, včetně MIPI CSI/DSI, LCD displeje, grafického akcelerátoru, kamerového rozhraní, S/PDIF a audio rozhraní I2S [35]. Kompletní charakteristiky desky jsou uvedeny v tabulce 3.1. Aplikační procesor i.MX RT1170 lze použít v oblastech, jako jsou průmyslové HMI, IoT, špičkové audio přístroje, low-endové přístrojové clustery, prodejní místa (PoS), řízení motorů a domácí spotřebiče.

3.3 Omezení běhu neuronových sítí na vestavěných zařízeních

Neuronové sítě, zejména hluboké sítě, vyžadují významné množství výpočetního výkonu pro trénování i inferenci. Vestavěná zařízení mohou mít omezené prostředky, což může omezit použití složitějších modelů. Výkon čipu přímo ovlivňuje dobu, po kterou může probíhat výpočet. Čím výkonnější čip máme k dispozici, tím rychleji můžeme provádět výpočty. Paměťové omezení vestavěných systémů může být výzvou, zejména při nasazení hlubokých neuronových sítí. Velikost pamětí určuje, jak složitý a velký model můžeme uložit do FLASH paměti zařízení a jak velká data můžeme využívat pro mezivýpočty v RAM paměti. Pokud máme malou paměť, budeme mít omezení ve velikosti modelu a velikosti dat, se kterými lze pracovat, což může ovlivnit výkon a přesnost modelu. Velikost modelů a datových sad může představovat problém při implementaci na zařízeních s omezenou pamětí. Přesnost modelu je také ovlivněna komplexitou modelu a velikostí vstupu a výstupu. Typicky platí, že čím přesnější je model, tím je složitější a náročnější na výkon a paměť. Neuronové sítě mohou být energeticky náročné, což je klíčový faktor při návrhu pro vestavěná zařízení s omezeným zdrojem energie, například v IoT (Internet of Things) aplikacích.

3.4 Techniky optimalizace pro efektivní inferenci na mikrokontrolerech

Optimalizace běhu neuronových sítí na vestavěných zařízeních je klíčovým aspektem dosažení efektivního a spolehlivého výkonu. V této kapitole se podrobněji podíváme na způsoby, jak optimalizovat běh neuronových sítí.

Co budeme optimalizovat?

Lze optimalizovat různé aspekty neuronové sítě. Například trénování a inferenci. Trénování modelu je výpočetně nákladný proces jak z hlediska času, tak z hlediska prostoru (úložiště). V průběhu let bylo vyvinuto mnoho matematických a programových triků, které mají zlepšit kvalitu a urychlit proces trénování. V tomto projektu však trénování nebude probíhat na mikrokontroléru, takže se nebudeme snažit optimalizovat tento aspekt sítě. Trénování a inference mají velmi odlišné požadavky. Inferencování nemění své parametry, takže jakýkoli algoritmus nebo metadata podporující zpětné šíření jsou zbytečnou režií, kterou je nejlépe odstranit. Při implementaci inferenčního modelu se dbá na výkon, spotřebu a velikost, takže většina úsilí je věnována zmenšení velikosti modelu.

Kvantizace

Kvantizace je proces snižování přesnosti vah, zkreslení a aktivací tak, aby spotřebovávaly méně paměti. Jinými slovy, proces kvantizace je proces, kdy se neuronová síť, která obecně používá 32bitové floaty k reprezentaci parametrů, převede na menší reprezentaci, například na 8bitová celá čísla. Přechod z 32bitové na 8bitovou reprezentaci by například zmenšil velikost modelu čtyřnásobně, takže jednou ze zřejmých výhod kvantizace je výrazné snížení paměti. V praxi existují dva hlavní způsoby kvantizace:

- **Kvantizace po trénování**
- **Trénování s ohledem na kvantizaci**

Jak již název napovídá, kvantizace po trénování je technika, při níž je neuronová síť kompletně natrénována pomocí výpočtu s plovoucí desetinnou čárkou a následně je kvantizována.

Jakmile je natrénována, neuronová síť se zmrazí, což znamená, že její parametry již nelze aktualizovat, a poté se parametry kvantizují. Kvantizovaný model se nakonec nasadí a použije k provádění inference bez jakýchkoliv dalších změn parametrů.

Ačkoli je tato metoda jednoduchá, může vést k vyšší ztrátě přesnosti, protože všechny chyby související s kvantizací se objeví až po dokončení trénování, a nelze je tedy kompenzovat.

Trénování s ohledem na kvantizaci funguje tak, že chyby související s kvantizací kompenzuje trénováním neuronové sítě pomocí kvantizované verze v dopředném průchodu během trénování.

Pruning

Pruning je technika, která zmenšuje model CNN tím, že se zbavuje nepotentních neuronů. Samozřejmě, že když odstraníme část modelu, jeho výkonnost klesne, ale my model přetrénujeme, abychom tento pokles kompenzovali. Tento proces se provádí iterativně, dokud se model nezmenší na přijatelnou velikost a přitom stále vykonává požadovanou úlohu s cílovou přesností [33].

Weight sharing

Sdílení vah je stará technika pro snížení počtu vah v síti, které je třeba natrénovat. Využila ji síť LeCunn-Net kolem roku 1998. Je to přesně to, jak to zní: opakované použití vah v uzlech, které jsou si nějakým způsobem blízké [13].

Typickou aplikací sdílení vah jsou konvoluční neuronové sítě. CNN fungují tak, že vstupní obraz prochází filtrem. Pro triviální příklad obrázku 4x4 a filtru 2x2 s velikostí kroku 2 to znamená, že filtr (který má čtyři váhy, jednu na pixel) je aplikován čtyřikrát, celkem tedy 16 vah. Typickou aplikací sdílení vah je sdílení stejných vah pro všechny čtyři filtry.

V tomto kontextu má sdílení vah následující účinky: snižuje počet vah, které je třeba naučit (v tomto případě ze 16 na 4), což snižuje čas a náklady na trénování modelu. Vyhledávání prvků je díky tomu necitlivé na umístění prvků v obraze. Snižujeme tedy náklady na trénink na úkor flexibility modelu. Sdílení vah je ve všech ohledech formou regularizace. A stejně jako u jiných forem regularizace může v určitých souborech dat s vysokou rozptylností umístění prvků skutečně zvýšit výkonnost modelu tím, že sníží rozptyl více, než zvýší zkreslení.

Processor	NXP Processor	MIMXRT1176DVMAA
DRAM Memory	512 MB SDRAM, 200 MHz	W9825G6KH-5I * 2
DC-DC	MPS	MP2143DJ, MP1613GTL
LDO	UNIONAMS	UM1750S-00, UM1550S-28, AMS1117-1.8
Mass Storage	TF Card Slot	
	128 Mbit Quad SPI Flash	
	512 Mbit Oct Flash	
	4 Mbit LPSPI Flash	
	2 Gbit Parallel NAND Flash(DNP)	
Display Interface	MIPI DSI LCD Connector	
Ethernet	10/100 Mbit/s Ethernet Connector. PHY Chip: KSZ8081RNB	
	10/100/1000 Mbit/s Ethernet Connector. PHY Chip: RTL8211FDI-CG	
USB	USB 2.0 OTG Connector * 2	
Audio Connector	3.5 mm Audio Stereo Headphone Jack	
	Board-Mounted Microphone	
	Left & Right Speaker Out Connectors	
	S/PDIF Interface(DNP)	
Power Connector	5 V DC-Jack	
Debug Connector	JTAG 20-pin Connector (SWD by default)	
	OpenSDA with DAP-Link	
Sensor	FXOS8700CQ: 6-Axis Encompass (3-Axis Mag, 3-Axis Accel)	
Camera	MIPI CSI Interface	
CAN	CAN Bus Connector	
User Interface Button	ON/OFF, POR Reset, Reset, USER Button	
LED Indicator	Power Status, Reset, OpenSDA, USER LED	
Expansion Port	Arduino Interface, M.2 interface	
PCB	5.1968 inch x 6.1024 inch (13.2 cm x 15.5 cm), 6-layer board	

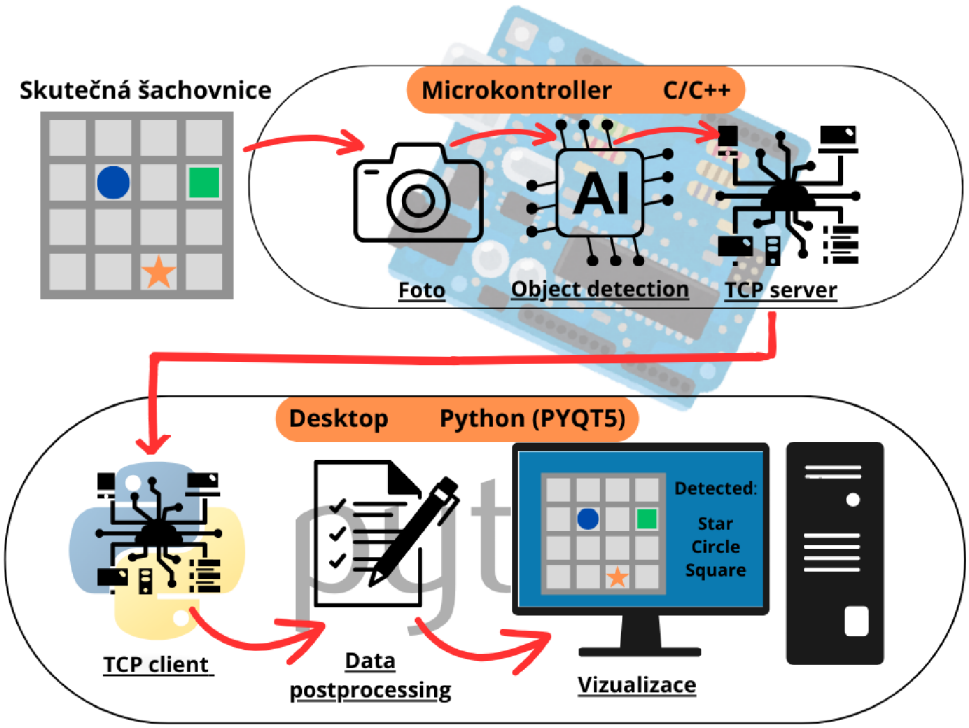
Tabulka 3.1: IMXRT1170 EVK[35].

Kapitola 4

Návrh aplikace a přehled nástrojů a knihoven

V této kapitole popíšu obecný návrh aplikace, svou představu o tom, jak by se měla aplikace chovat a jak by měla komunikovat mezi jednotlivými částmi. A také se podíváme na nástroje, které nám pomohou projekt realizovat.

4.1 Obecný návrh



Obrázek 4.1: Schema

Představa je taková: Máme fyzickou šachovnici se 16 políčky 4x4, na které náhodně umisťujeme objekty (celkem máme objekty 9 různých druhů: oranžová hvězda, oranžový

kruh, oranžový čtverec, modrá hvězda, modrý kruh, modrý čtverec, zelená hvězda, zelený kruh, zelený čtverec). Nad touto šachovnicí máme pevně umístěnou desku MIMXRT1170, ke které je připojena malá kamera. Tato kamera je schopna snímat naši šachovnici shora v rozlišení 720x720. Po zpracování obrazu jsou data odeslána neuronové síti, která vyhledává naše objekty. Po zpracování neuronovou sítí zbývá pouze dekódovat výstupní data a odeslat je do desktopové aplikace v počítači (komunikace server-klient, kde serverem je mikrokontrolér a klientem desktopová aplikace). Aplikace musí také správně vyčíst data ze serveru, zpracovat je a zobrazit uživateli vizualizovanou šachovnici s nalezenými objekty.

4.2 Nástroje a knihovny pro vývoj na mikrokontroleru i.MX RT

V této podkapitole se zaměříme na klíčové nástroje a knihovny, které byly základními komponentami při návrhu, implementaci a testování detekce objektů na mikrokontroleru i.MX RT.

Tensorflow

TensorFlow je bezplatná softwarová knihovna od společnosti Google s otevřeným zdrojovým kódem pro strojové učení a umělou inteligenci. Lze ji použít v celé řadě úloh, ale zaměřuje se zejména na trénování a odvozování hlubokých neuronových sítí [6]. TensorFlow lze používat v celé řadě programovacích jazyků, včetně Pythonu, JavaScriptu, C++ a Javy. Tato flexibilita umožňuje řadu aplikací v mnoha různých odvětvích.

Tensorflow Lite

TensorFlow Lite (TFlite) je sadou nástrojů pro transformaci a optimalizaci klasických modelů TensorFlow pro implementaci modelů na mobilních zařízeních, mikrokontrolérech a dalších vestavěných zařízeních [6]. Vyvinula ji společnost Google pro interní použití a poté ji poskytla veřejnosti.

TensorFlow Object Detection API

TensorFlow Object Detection API [26] je open source framework postavený nad TensorFlow, který usnadňuje konstrukci, trénování a nasazení modelů detekce objektů.

PyTorch

PyTorch [38] je open-source framework pro strojové učení v jazyce Python založený na Torch. Používá se k řešení různých úloh: počítačové vidění, zpracování přirozeného jazyka. Vyvíjí jej především skupina umělé inteligence společnosti Facebook.

Colab

Google Colaboratory [2] neboli **Colab** je as-a-service verze Jupyter Notebook, která umožňuje psát a spouštět kód Pythonu prostřednictvím prohlížeče. Používání tohoto nástroje se ukázalo jako velmi praktické. Colab poskytuje zdroje, jako jsou GPU, TPU a knihovny Pythonu. Není tedy třeba zatěžovat procesor osobního počítače a instalovat velké množství knihoven.

LabelImg

LabelImg [45] je populární open source grafický anotační nástroj, který umožňuje kreslit ohraničující rámečky kolem objektů a přiřazovat jim názvy tříd. Poskytuje intuitivní rozhraní pro efektivní anotování obrázků. Anotace lze ukládat v různých formátech, například ve formátech PASCAL VOC, YOLO a CreateML.

CUDA

Softwarová a hardwarová architektura paralelních výpočtů, která umožňuje výrazně zvýšit výpočetní výkon pomocí grafických procesorů Nvidia.

The Edge AI Platform

Edge Impulse [21] umožňuje vytvářet datové sady, trénovat modely a optimalizovat knihovny tak, aby běžely na jakýchkoli procesorech od MCU s extrémně nízkou spotřebou až po výkonné procesory s operačním systémem Linux a GPU. Pomocí této platformy lze snadno provozovat umělou inteligenci na jakémkoli hardwaru.

eIQ® ML Software

Balíček eIQ [1] od NXP obsahuje softwarové vývojové prostředí NXP® eIQ®, optimalizované knihovny a různé nástroje pro strojové učení. Umožňuje používat algoritmy strojového učení na mikrokontrolérech a mikroprocesorech NXP, včetně crossover MCU i.MX RT a aplikačních procesorů rodiny i.MX. Tento software využívá open-source a proprietární technologie a je plně integrován do vývojových prostředí MCUXpresso SDK a Yocto, což umožňuje snadný vývoj úplných aplikací na úrovni systému [1].

MCUXpresso IDE

MCUXpresso IDE [37] je snadno použitelné vývojové prostředí pro MCU NXP s procesorem Arm Cortex-M. MCUXpresso IDE nabízí pokročilé možnosti editace, kompilace a ladění s přidáním ladění specifického pro MCU, sledování a profilování kódu, vícejádrového ladění a integrovaných konfiguračních nástrojů.

PYQT5

PyQt5 [44] je multiplatformní sada nástrojů pro grafické uživatelské rozhraní, sada vazeb jazyka Python pro Qtv5. Díky nástrojům a jednoduchosti, které tato knihovna poskytuje, lze snadno vyvíjet interaktivní desktopové aplikace.

LwIP

lwIP [7] je malá nezávislá implementace sady protokolů TCP/IP, kterou původně vyvinul Adam Dunkels a nyní v ní pokračuje. Implementace TCP/IP lwIP se zaměřuje na snížení spotřeby prostředků při zachování plnohodnotného protokolu TCP. Díky tomu je lwIP vhodný pro použití ve vestavných systémech s desítkami kilobajtů volné paměti RAM a místem pro přibližně 40 kilobajtů kódu ROM [4].

Kapitola 5

Experimentování

Jak již bylo uvedeno, v této práci byly testovány 4 různé modely neuronových sítí. V této kapitole budou stručně popsány kroky vývoje, trénování a spuštění modelů na mikrokontroléru MIMXRT1170.

MobileNet-SSDv2

Tento model byl vyzkoušen jako první. Při implementaci této verze byly použity následující nástroje:

- Tensorflow
- Tensorflow Lite
- Tensorflow Object Detection API 2
- Tensorflow 2 Detection Model Zoo [25]
- Colab
- LabelImg
- MCUXpresso IDE

S pomocí různých informací v oficiálním repozitáři Tensorflow Object Detection API 2 a návodů pro trénování sítí jsem napsala Colab Notebook[1]. Tam krok za krokem jsou uvedeny pokyny jak model trénovat. Pro trénování byly použity hotové skripty z Object Detection API a vlastní datová sada. Po provedení všech kroků v Notebooku získáme soubor .tflite, který se pokusíme použít na našem vestavěném zařízení. Pro inspiraci byl použit příklad z repozitáře Mcuxpresso SDK. Jednalo se o příklad použití klasifikátoru na RT1170 s využitím knihovny tensorflow micro a souborů .tflite. Bylo třeba upravit několik detailů - nahradit binární soubor modelu vlastním, zvětšit velikost zachycené oblasti, správně načíst výsledek modelu a doplnit načítání souřadnic. Po úpravě projektu a jeho prvním spuštění se ukázalo, že tento model je pro náš MCU příliš náročný. Jeden pracovní cyklus (snímek z kamery - předzpracování snímku - zpracování snímku neuronovou sítí - výsledek) trval přibližně 10 minut, což se zdá jako nepoužitelné v reálném životě. Pokračovala jsem dal hledáním jiného modelu.

[1] https://colab.research.google.com/drive/1a-klg9AK5Zc_LoIizDlUxaTWg6ZSEdiy

YOLO

Yolo na první pohled vypadal jako dobrý nápad. Vyzkoušela jsem natrénovat Yolov3 Darknet by AlexeyAB [14]. Při implementaci této verze byly použity následující nástroje:

- PyTorch
- Colab
- LabelImg
- MCUXpresso IDE

Trénování probíhalo, jako minule, pomocí Google Colab[2]. Problém se v tomto případě objevil ihned po úplném natrénovani modelu. Výchozí formát Yolo je .weights, který bohužel zatím není možné normálně rozchodit na mikrokontrolérech NXP pomocí MCUXpresso IDE. Aplikace podporuje jenom .tflite modely. Několikrát jsem zkoušela prekonvertovat daný model do tensorflow formátu prostřednictvím různých online konvektorů, nástrojů od Tensorflow, ale to se nepodařilo. Nakonec jsem rozhodla hledat jinou možnost.

NanoDet

Další na řadě je NanoDet. Při implementaci této verze byly použity následující nástroje:

- PyTorch
- CUDA
- Colab
- LabelImg
- MCUXpresso IDE

Tento model stejně jako ostatní má svůj repozitář na GitHub [40], odkud jsem brala informace. Nemá moc podrobný Google Colab Notebook na trénování, ale vytvořila jsem vlastní[3], podle pokynů a instrukcí v hlavním README souboru v repozitáři. Jako základní model jsem si zvolila ten nejmenší co tam v Model Zoo měli - NanoDet-m-0.5x. Tady se pak objevil problém jako minule u Yolo - výsledný formát modelu je .onnx, což naše deska nepodporuje. Ale podařilo se mi najít cestu jak to vyřešit. Použila jsem repozitář s konvertorem do .tflite[4]. Postup měl dva kroky: z onnx udělat keras formát a až pak z kerasu udělat tflite. Poté, co jsem získala verzi modelu .tflite, jsem se pokusila ji implementovat do svého projektu. Postup byl podobný tomu, který jsem popsala v MobileNet-SSD. Musela jsem změnit postup načítání vstupních a výstupních dat, získávání souřadnic a mnoho dalších drobností. Projekt byl úspěšně spuštěn a model fungoval poměrně přesně. Problém s rychlostí však zůstal. Doba od pořízení snímku po výsledky detekce byla asi 2 minuty, což je samozřejmě ve srovnání s MobileNet-SSD mnohem lepší výsledek, ale stále to bylo dlouhé. Zbývala ještě jedna možnost modelu, kterou jsem chtěla vyzkoušet, a tak jsem se do ní pustila.

[2] https://colab.research.google.com/drive/10Pg7lFAaGMhf10M05qO58orjU-cRUuxH

[3] https://colab.research.google.com/drive/1y2dB_RNnizGP_vX2cuVoxYZ7_wqwRJ_M#scrollTo=hhhohIF3ytFF

[4] https://github.com/LanpingTech/NANODET-ONNX-TFLite

FOMO

Fomo je jedním z nových modelů vyhledávání objektů vyvinutých společností Edge Impulse [22]. Tento model byl představen jako algoritmus strojového učení navržený speciálně pro běh na vestavěných zařízeních, což byl důvod, proč jsem tento model vyzkoušela ve svém projektu. Při implementaci této verze byly použity následující nástroje:

- Tensorflow
- Tensorflow Lite
- The Edge AI Platform
- LabelImg
- MCUXpresso IDE

Platforma podporuje snadnou integraci modelů až desítek různých MCU s instrukcemi. Bohužel zařízení NXP v tomto seznamu nejsou zahrnuta, ale to neznamená, že bychom tento model nemohli použít na našem MCU. Platforma nám umožňuje trénovat model na naší vlastní sadě dat bez ohledu na konečné zařízení. K dalšímu vývoji si můžeme stáhnout natrénovaný model ve formátu .tflite a knihovnu EgdeImpulse v jazycích C/C++. Když jsem tento model integrovala do projektu, zjistila jsem, že je mnohem rychlejší a má dobrou přesnost. V této variantě modelu nebyly žádné komplikace s trénováním, kompatibilitou a rychlostí zpracování. Proto tento model v této „soutěži“ zvítězil a další podrobnosti vývoje budou popsány v následující kapitole.

Kapitola 6

Návrh a trénování modelu pro detekci objektů

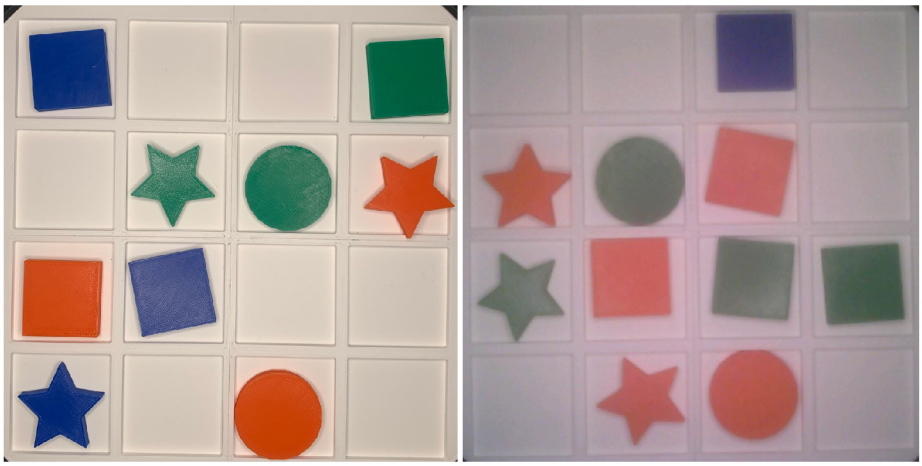
6.1 Výběr architektury modelu

Ze všech testovaných modelů byl FOMO jasným favoritem. Menší velikost modelu, rychlost provozu, vývoj speciálně pro vestavná zařízení, snadná integrace do projektu, žádné problémy s kompatibilitou jako u ostatních kandidátů. Díky tomu všemu jsem si uvědomila, že tento model bude pro můj projekt dobrou volbou.

6.2 Sběr a předzpracování dat

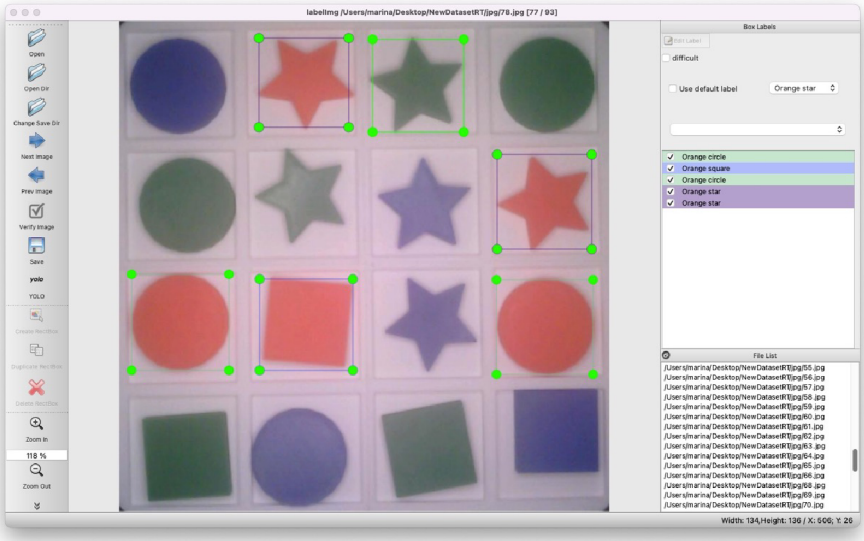
Detekované objekty byly vytištěny na 3D tiskárně.

V úplně první „zkušební“ verzi modelu jsem tyto objekty vyfotila na mobilní telefon. Později však bylo rozhodnuto dataset upravit a sbírat data pomocí kamery připojené k mikrokontroléru. Toto řešení zjednodušuje trénování sítě a zvyšuje mAP. Snímky pořízené mobilem a to, co vidí RT, se z hlediska kvality obrazu zcela liší. Jinými slovy, takto se síť trénuje na „falešných“ datech. Model si zapamatuje jasnější barvy, zřetelnější hranice objektů, světlejší okolí. Zatímco při přímé práci na mikrokontroléru uvidí úplně jiný pohled: nevýrazné barvy, nízkou kvalitu obrazu. Síť bude samozřejmě stále schopna najít naši oranžovou hvězdu, ale přesnost této předpovědi bude nižší, než kdybychom model trénovali na skutečných datech, která uvidí při spouštění na mikrokontroléru.



Obrázek 6.1: Porovnání kvality snímků. IPhone XR vs RT Kamera.

Po nasbírání dat je třeba každý snímek zpracovat. V této práci používám aplikaci LabelImg 4.2 6.2. Dataset se nakonec ukázal jako malý, pouze 90 vzorků. Stačilo to však k natrénování sítě, která rozpoznává relativně malý počet objektů ze statického pohledu.

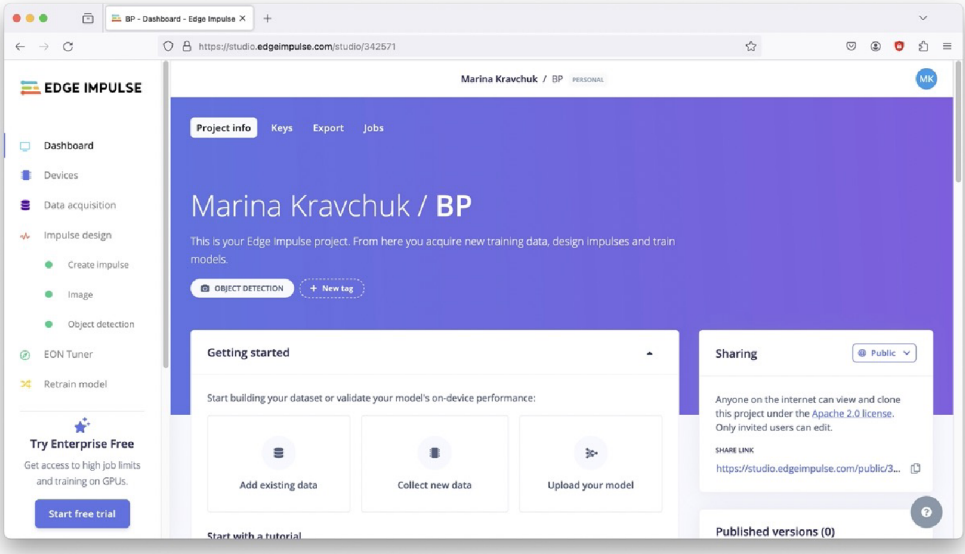


Obrázek 6.2: LabelImg

6.3 Trénování modelu pro detekci specifikovaných barev a tvarů

Trénování modelu probíhá pomocí webového prostředí Egde Impulse. Nejprve se tedy musíme zaregistrovat a vytvořit nový projekt.

Pro přenos naší sady dat do projektu stačí přejít na panel Data acquisition → Upload data a vybrat složku obsahující samotné obrázky a soubory .txt s popisem objektů na snímcích, které jsme získali v aplikaci LabelImg.

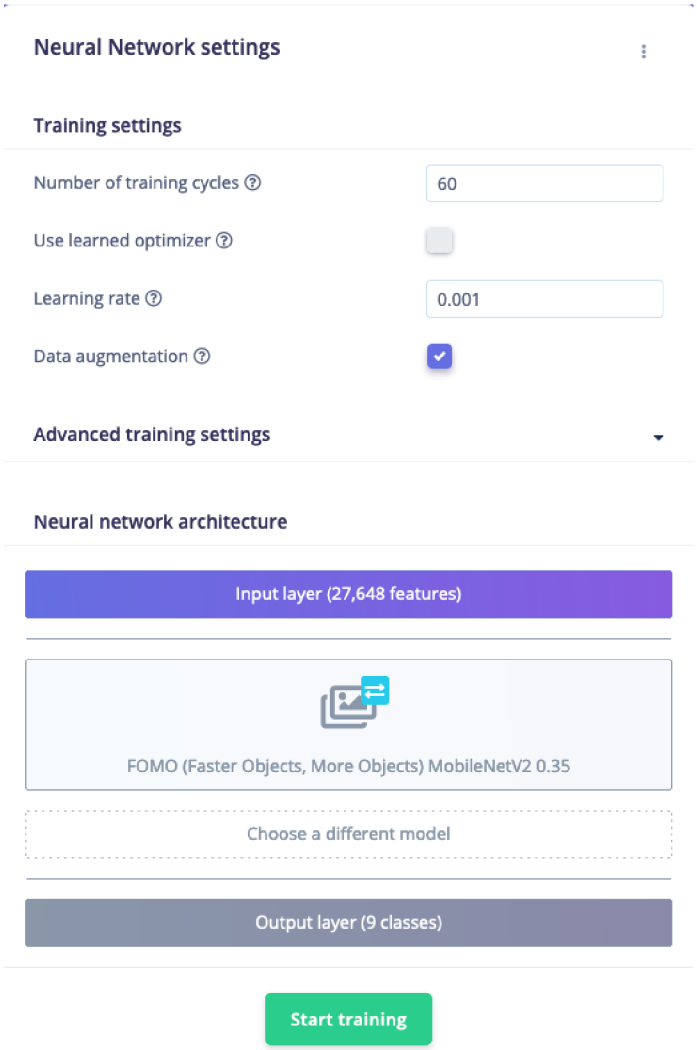


Obrázek 6.3: Úvodní stránka projektu Edge Impulse.

Dále přejdeme do části Impulse design → Create impulse, kde můžeme nakonfigurovat vstupní data a vybrat bloky zpracování a učení. Velikost vstupního obrázku jsem nastavila na 96x96, snažila jsem se velikost obrázku zmenšit, abych zvýšila rychlost modelu. `Resize mode` jsem nastavila na `Squash`, což znamená, že každý obrázek, který není čtvercový, bude zploštěn na čtverec. Ostatní možnosti vedou ke ztrátě dat na fotografii (oříznutí obrázku na čtverec).

Poté jsem vybrala normalizaci a Object Detection Block doporučený platformou jako bloky zpracování a učení. Nyní lze nastavení uložit a přejít k dalšímu kroku.

Na panelu Impulse design → Object detection je možné změnit nastavení trénování neuronové sítě (nastavení v mém projektu 6.4) a začít trénovat model kliknutím tlačítka Start training.



Obrázek 6.4: Nastavení modelu.

Neuronová síť je natrénována poměrně rychle, v mém případě zhruba za 30 minut. Po skončení trénování se můžeme podívat na vygenerovanou tabulku 6.5, kde vidíme, jak přesně model předpovídá určité objekty. V tomto případě si můžeme všimnout, že některé z nich mají hodnocení 100 % a některé 80-90 %. To je způsobeno tím, že jsem při sběru dat objekty náhodně poskládala. To znamená, že některé objekty se vyskytují na více snímcích a některé na méně. A samozřejmě platí, že čím častěji se objekt v trénovací sadě objevuje, tím lépe si ho model zapamatuje.

	BACKGROUND	BLUE CIRCLE	BLUE SQUARE	BLUE STAR	GREEN CIRCLE	GREEN SQUARE	GREEN STAR	ORANGE CIRCLE	ORANGE SQUAR	ORANGE STAR
BACKGROUND	99.6%	0%	0%	0%	0%	0.3%	0%	0%	0.1%	0%
BLUE CIRCLE	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%
BLUE SQUARE	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%
BLUE STAR	9.1%	0%	0%	90.9%	0%	0%	0%	0%	0%	0%
GREEN CIRCLE	18.2%	0%	0%	0%	81.8%	0%	0%	0%	0%	0%
GREEN SQUARE	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%
GREEN STAR	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%
ORANGE CIRCLE	14.3%	0%	0%	0%	0%	0%	0%	85.7%	0%	0%
ORANGE SQUARE	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%
ORANGE STAR	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%
F1 SCORE	1.00	1.00	1.00	0.95	0.90	0.79	1.00	0.92	0.90	1.00

Obrázek 6.5: Výsledky po trénování.

Je však třeba vzít v úvahu, že tyto údaje byly získány při ověřování souboru validačních dat, což není plnohodnotné testování a čistý výsledek přesnosti modelu. Více o testování bude popsáno v kapitole 8.2.

Kapitola 7

Implementace aplikace

7.1 Přehled architektury systému

Celkový návrh systému byl popsán dříve v kapitole 4. Zde přejdeme k podrobnostem procesu vývoje.

MCU

Pro inspiraci jsem použila dva příklady z balíčku MCUXpresso SDK a také jsem použila kód z knihoven Edge Impulse. První příklad[1] ukazoval použití klasifikátoru na RT1170 s využitím knihovny tensorflow micro a modelu tflite. Druhý příklad[2] má za cíl ukázat použití knihovny lwip pro internetovou komunikaci mezi počítačem a ladicí deskou přes ethernet. Převzala jsem části kódu z obou zdrojů a také z knihovny Edge Impulse, vše označené v kódu projektu.

Aplikační kód MCU lze rozdělit do čtyř hlavních částí. Podstatou první části je inicializace/příprava všech potřebných modulů a pinů na naší desce a vytvoření vlákna, které inicializuje vytvoření tcp serveru.

V tomto bodě přejdeme k druhé části - serveru. Pomocí funkce `netconn_new()` se vytvoří nový soket typu TCP. Pomocí funkce `netconn_bind()` navážeme náš soket na požadovanou ip adresu a port. Poté zavoláme funkci `netconn_listen()`, abychom oznámili, že chceme přijímat požadavky na připojení a začít naslouchat. Čekáme na příchozí spojení a po připojení klienta navážeme spojení pomocí funkce `netconn_accept()`.

Po připojení klienta přejdeme ke třetí části kódu - inicializaci a použití neuronového modelu. Pomocí funkce `MODEL_Init()` spustíme řadu funkcí z knihovny tensorflow, které aktivují natrénovaný model. Po úspěšném načtení modelu musíme získat obrázek z kamery mikrokontroléru a předat jej ke zpracování. Pomocí funkce `MODEL_RunInference()` se modelu předají data obrázku a speciální struktura pro uložení výsledku detekce. Pro zjednodušení přenosu dat a následného dešifrování jsem data ze struktury výsledku přenesla do pole 64 prvků typu uint8. V poli jsou uloženy údaje o celé šachovnici - umístění objektů (index buňky i souřadnice) a pravděpodobnost předpovědi pro každý objekt.

Po obdržení výsledku můžeme přistoupit k poslední části kódu, a to k odeslání získaného pole desktopové aplikaci (klientovi) prostřednictvím komunikace TCP (klient-server).

[1] https://github.com/nxp-mcuxpresso/mcux-sdk-examples/tree/main/evkbmimxrt1170/eiq_examples/tflm_label_image

[2] https://github.com/nxp-mcuxpresso/mcux-sdk-examples/tree/main/evkbmimxrt1170/lwip_examples/lwip_ipv4_ipv6_echo/freertos/cm7

Voláme funkci `tcpsrv_sendMessage()`, kde je popsán postup odeslání zprávy. Tam pomocí funkcí `netbuf_ref()` a `netconn_write()` nejprve zapíšeme zprávu, která má být odeslána, do bufferu a poté ji odešleme klientovi.

Akce získání obrázku, zpracování detektorem a odeslání pole s výsledky probíhají v nekonečné smyčce `while(1)` a budou běžet, dokud se aplikace nezavře.

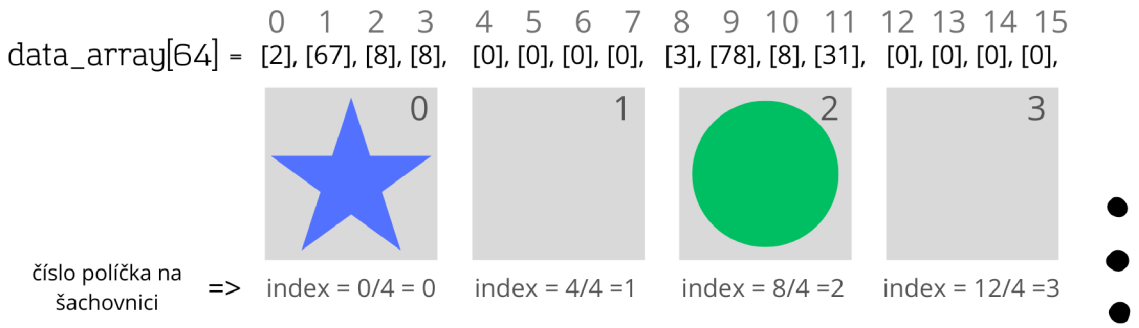
Návrh a vývoj GUI pro PC aplikaci

Grafické uživatelské rozhraní bylo navrženo velmi jednoduše.V aplikaci jsou pouze dvě okna: hlavní okno a okno nastavení. Hlavní okno je rozděleno na dvě části: na pravé straně je tabulka nalezených objektů, která obsahuje název objektu, pravděpodobnost předpovědi a souřadnice umístění. Implementováno pomocí třídy `QtWidgets.QTableWidget()`. Počet řádků se generuje automaticky podle toho, kolik objektů bylo nalezeno. Na levé straně je vizualizace šachovnice se 16 poli. Tato pole jsou vytvořena pomocí třídy `QtWidgets.QLabel()`. Při nalezení objektů program vloží jejich obrázky na místa, kde byly nalezeny. Všechny obrázky jsou uloženy v samostatné složce `resources` ve formátu `png`. V hlavním okně je také tlačítko pro přechod do nastavení a tlačítko `Detect`, ke kterému se vrátíme o něco později. Tlačítko nastavení nás přenese do druhého okna - okna nastavení. Zde se nachází pouze 8 výběrových políček, implementovaných pomocí třídy `QtWidgets.QCheckBox()`. Zde si uživatel může zvolit, které objekty má vyhledat a které ignorovat. Zpočátku jsou při spuštění aplikace vybrány všechny možné objekty. Zde je také tlačítko pro návrat do hlavního okna. Všechna tlačítka jsou realizována pomocí třídy `QtWidgets.QPushButton()` a připojena k akcím pomocí metod `pushButton.clicked.connect()`.

Nyní můžeme probrat hlavní funkci aplikace, která je zabudována v tlačítku Detect. Po stisknutí tlačítka se aktivuje vlákno `Thread()`, ve kterém se vytvoří soket TCP jako klient a připojí se k serveru, který by již měl být spuštěn na mikrokontroléru.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
```

Dále získáme data ze serveru pomocí příkazu `s.recv()`. Data přijdou jako pole typu int se 64 položkami. Data jsou poskládána takto: první prvek nám říká, jaký objekt se v buňce nachází, druhý prvek je pravděpodobnost předpovědi, třetí a čtvrtý prvky jsou X a Y (souřadnice) a to se opakuje 16krát ($16 * 4 = 64$) pro každé pole šachovnice. Index buňky v poli určuje číslo pole na šachovnici, ale protože každé buňce předávám 4 různé informace, počítám index buňky jako index prvku v poli dělený 4. Obrázek pro přehlednost 7.1.



Obrázek 7.1: Schéma.

Po úspěšném načtení pole pomocí cyklu „naskládám“ všechny nalezené objekty a informace o nich do seznamu složeného ze slovníků.

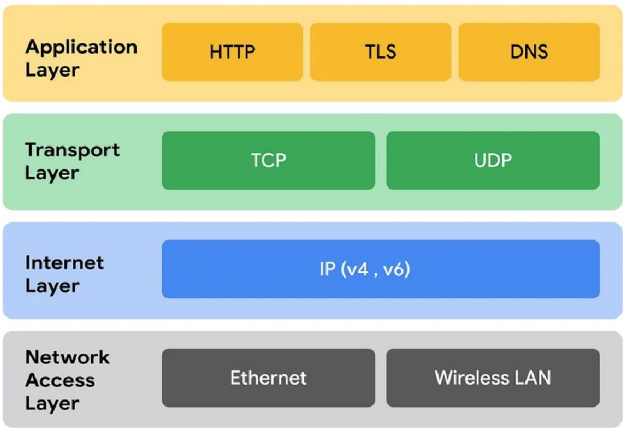
```
data = s.recv(2024)
objectdetecteddata = list(data)
for x in range(0,63,4):
    if objectdetecteddata[x] == 0:
        continue

    udata = dict.fromkeys(['name', 'scores', 'x', 'y', 'index', 'alias'])
    ...
    objectsfromdata.append(udata)
```

Po načtení všech objektů se zavolají metody `regenerate_table()` a `regenerate_grid()`, které obnoví hlavní stránku a vygenerují tabulku a šachovnici s nalezenými objekty.

7.2 Komunikační protokoly mezi PC aplikací a mikrokontrolerem i.MX RT

Pro přenos dat z mikrokontroléru do desktopové aplikace používám architekturu klient-server založenou na protokolu **tcp**. Protokol TCP (Transmission Control Protocol) je jedním z hlavních protokolů internetu, který se používá k zajištění spolehlivého přenosu dat mezi zařízeními v síti.



Obrázek 7.2: TCP/IP. Převzato z [29].

Protokol TCP (Transmission Control Protocol) se vztahuje k transportní vrstvě modelu TCP/IP (Transmission Control Protocol/Internet Protocol). V tomto modelu je transportní vrstva zodpovědná za spolehlivý přenos dat mezi zařízeními v síti a zajišťuje mnoho funkcí, jako je rozdělení dat do paketů, řízení toku, kontrola chyb atd. TCP je jedním ze dvou hlavních protokolů transportní vrstvy TCP/IP, druhým je UDP (User Datagram Protocol).

Na straně mikrokontroléru byla použita knihovna LwIP. Byl zde vytvořen server TCP, který komunikuje s grafickým uživatelským rozhraním. K provozu serveru je nutné mít:

- Mini/micro USB kabel
- Síťový kabel standardu RJ45

- Deska MIMXRT1170-EVKB
- Personální počítač

A také je třeba nakonfigurovat následující věci:

- Připojit kabel USB mezi hostitelský počítač a port USB OpenSDA (nebo USB na sériový port) na cílové desce
- Otevřít sériový terminál na PC pro sériové zařízení OpenSDA(nebo USB to Serial) s těmito nastaveními:
  - 115200 baud rate
  - 8 data bits
  - No parity
  - One stop bit
  - No flow control
- Vložit ethernetový kabel do portu RJ45 (1G ENET) cílové desky a připojit jej k síťovému adaptéru počítače
- Nastavit IP adresu hostitelského počítače na 192.168.0.103
- Nahrát program do cílové desky a spustit ladicí program v IDE

Na straně klienta v aplikaci v jazyce Python byl k vytvoření komunikace TCP použit modul `socket`. Pro připojení klienta k serveru stačí pár řádků kódu, kde se zadá ip adresa serveru a číslo portu.

```
import socket
HOST = "192.168.0.102"  # The server's hostname or IP address
PORT = 7  # The port used by the server
...
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    data = s.recv(2024)
    ...
```

7.3 Integrace modelu neuronové sítě do aplikace mikrokontroleru

Vraťme se k naší platformě Edge Impulse a převezměme odtud můj vytrénovaný model. Webová platforma nám nabízí několik možností pro další vývoj, od souboru ve formátu NPY až po Keras h5. Já ale potřebuji formát Tensorflow Lite kvantizovaný na hodnoty int8 pro lepší optimalizaci.

Interpret TensorFlow Lite pro mikrokontroléry v mém projektu očekává, že model bude reprezentován jako datové pole C, aby jej bylo možné snadno zkompilovat do binárního souboru pro zařízení, která nemají souborový systém. A proto jsem převedla svůj soubor `ei-bp-int8.lite` na datové pole pomocí konzolového příkazu `xxd -i ei-bp-int8.lite > model_data.h` a přidala tento soubor do projektu.

Dále musíme použít několik standardních funkcí z knihovny Tensorflow Lite k rozvinutí modelu. Funkce `tflite::GetModel` nahraje náš model jako proměnnou typu `tflite::Model`.

Po načtení je nutné přidat resolver operací. Deklaruje se instance `MicroMutableOpResolver`. Tu bude interpret používat k registraci a přístupu k operacím používaným modelem. V mém případě model potřebuje zaregistrovat pouze 5 operací: `Conv2D`, `DepthwiseConv2D`, `Pad`, `Add`, `Softmax`. Abych zjistila, které operace model používá, použila jsem webovou stránku netron.app[3], kde je celá síť znázorněna jako diagram. Musíme předem alokovat určité množství paměti pro vstupní a výstupní pole a pole mezivýstupů. To je poskytnuto jako pole `uint8_t` o velikosti `tensor_arena_size` : `uint8_t tensor_arena[tensor_arena_size]`. Dalším krokem je vytvoření instance `tflite::MicroInterpreter` předáním dříve vytvořených proměnných:

```
tflite::MicroInterpreter interpreter(model, resolver, tensor_arena,
tensor_arena_size, error_reporter);
```

V posledním bodě interpreteru řekneme, aby pro modelové tenzory alokoval paměť z `tensor_arena`:

```
interpreter.AllocateTensors();
```

Nyní můžeme model považovat za inicializovaný a připravený k použití.

Abychom mohli model použít, musíme přečíst obraz z kamery připojené k desce a poté jej odeslat k detekci. Pro získání obrázku z kamery se volá funkce `IMAGE_GetImage()`, uvnitř které je standardní sada příkazů pro práci s kamerou z knihovny NXP.

```
static EIQ_VideoWorker_t* s_worker = NULL;
static uint8_t* s_captureBuffer = NULL;
static Dims_t s_captureBufferDims;
...
status_t IMAGE_GetImage(float* output)
{
    if (s_worker == NULL)
    {
        PRINTF(EOL "Camera data processing:" EOL);
        s_worker = EIQ_InitVideoWorker();
        s_captureBuffer = s_worker->base.getData();
        s_captureBufferDims = s_worker->base.getResolution();
        s_worker->base.start();
    }
    ...
}
```

Dále je ve stejné funkci obrázek normalizován pomocí `crop_and_interpolate_rgb888()`, která obsahuje dlouhou řadu dalších funkcí a příkazů, ale výsledkem všech těchto akcí je zmenšení obrázku na velikost 96x96 a jeho převod do formátu rgb888.

Knihovna Egde Impulse používá k předávání dat neuronovému modelu strukturu `signal_t`, kterou jsem převzala pro svůj projekt. `signal` poskytuje jednoduchou implementaci zpracování signálů pro vestavěné cíle.

```
typedef struct ei_signal_t {
    /**
     * A function to retrieve part of the sensor signal
     * No bytes will be requested outside of the ‘total_length‘.
     * @param offset The offset in the signal
     * @param length The total length of the signal
     * @param out_ptr An out buffer to set the signal data
     */
```

[3] https://netron.app

```
    std::function<int(size_t offset, size_t length, float *out_ptr)> get_data;
    size_t total_length;
} signal_t;
```

Po zpracování obrázku naplníme strukturu `signal` vstupními daty pomocí `image_get_data()`. Nakonec opustíme funkci `IMAGE_GetImage()` a pokračujeme v hlavním cyklu. K ukládání výsledků detekce používáme strukturu `ei_impulse_result_t` (taktéž převzatou z knihovny Egde Impulse). Struktura obsahuje všechny údaje o nalezených objektech.

```
typedef struct {
    ei_impulse_result_bounding_box_t *bounding_boxes;
    uint32_t bounding_boxes_count;
    ei_impulse_result_classification_t classification[EI_CLASSIFIER_MAX_LABELS_COUNT];
    float anomaly;
    ei_impulse_result_timing_t timing;
} ei_impulse_result_t;
```

A jsme připraveni spustit detektor. K tomu se zavolá funkce `MODEL_RunInference()`, která následně zpracuje vstupní data (obrázek) a zavolá metodu `interpreter->Invoke()`, což je v podstatě spuštění modelu. Zpracováním obrazu myslím kvantizaci. Každý pixel (každá hodnota v poli inputu) by se měl vypočítat podle vzorce:

$$q =< int8 > (round(p/scale) + zero\_point),$$

kde q je pixel po kvantizaci a p před ní, *scale* a *zero_point* jsou proměnné v neuronovém modelu.

Až po kvantizaci vstupních dat můžeme spustit model. Poté, co Invoke proběhl, získáme pole 1440 prvků typu int8, které musíme transformovat do čitelného a srozumitelného výsledku.

```
// out_width = out_height = 12
for (size_t y = 0; y < out_width; y++) {
    for (size_t x = 0; x < out_height; x++) {
        size_t loc = ((y * out_height) + x) * (label_count + 1);

        for (size_t ix = 1; ix < label_count + 1; ix++) {
            int8_t v = data[loc+ix];
            float vf = static_cast<float>(v - zero_point) * scale;

            ei_handle_cube(&cubes, x, y, vf, labels[ix - 1], 0.5);
        }
    }
}
fill_result_struct_from_cubes(result, &cubes, out_width_factor,
                              object_detection_count);
```

V cyklu for procházíme celé pole a vybíráme segmenty po 10 prvcích (viz změna proměnné loc). V samé vnitřní smyčce procházíme každý prvek a provádíme inverzní kvantizaci pomocí vzorce:

$$p =< float > (q - zero\_point) * scale$$

A v každé iteraci posíláme data do funkce `ei_handle_cube()`, kde jsou uloženy tzv. bounding boxy, kde `x` a `y` jsou budoucí souřadnice, `vf` je procento pravděpodobnosti predikce,

`labels[ix-1]` je označení objektu a poslední parametr 0,5 je nejmenší možná hodnota `vf`, predikce s pravděpodobností pod 50 % se neberou v úvahu.

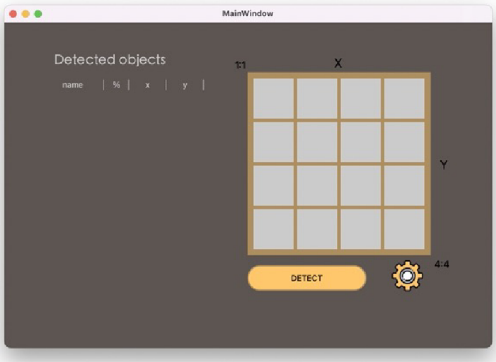
Nakonec zavoláme funkci `fill_result_struct_from_cubes()`, která zpracuje data z boxů a naplní naši strukturu výsledků(`ei_impulse_result_t *result`).

Kapitola 8

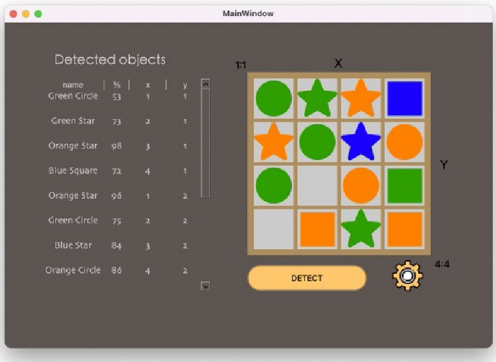
Hodnocení a výsledky

8.1 Výsledky

Nakonec se mi podařilo vybraný model spustit na mikrokontroléru MIMXRT1170. Aplikace se skládá ze dvou částí: část v jazyce C/C++ napsaná v MCUXpressoIDE a druhou částí je grafické rozhraní napsané v jazyce Python pomocí frameworku PYQT5. Zde je vidět, jak aplikace vypadá při spuštění 8.1 a při detekci objektů 8.2.



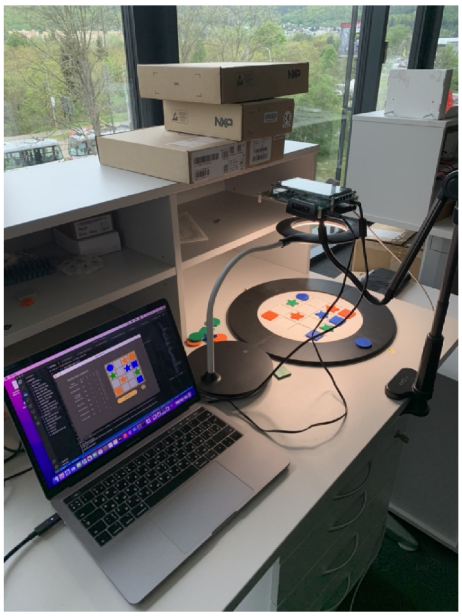
Obrázek 8.1: Aplikace po spuštění



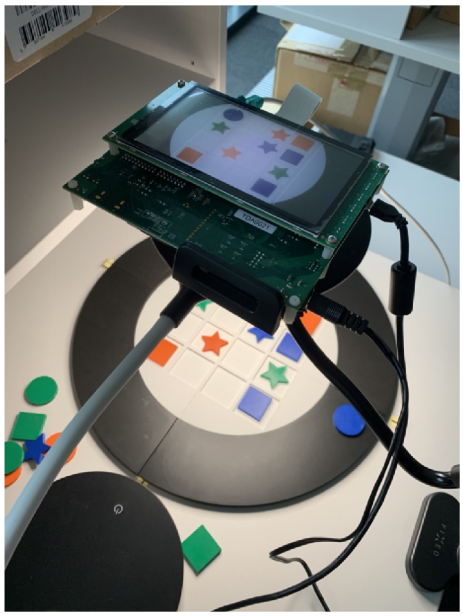
Obrázek 8.2: Aplikace po detekci

Pro vývoj a testování byla sestavena konstrukce z lampy a držáku, na kterém byla deska umístěna. Připravila jsem také video[1], které názorně ukazuje, jak aplikace funguje.

[1] https://www.youtube.com/watch?v=VJ-1PWt2Anc



Obrázek 8.3: Konstrukce



Obrázek 8.4: Konstrukce

Doba běhu (počítám od okamžiku stisknutí tlačítka **Detect** do vykreslení tabulky a šachovnice) je v průměru 5,7 sekundy. Velikost získaného modelu v kvantizované podobě je 56KB. Testování bylo provedeno na systému MacOS Monterey, více informací o testování naleznete v následující podsekci.

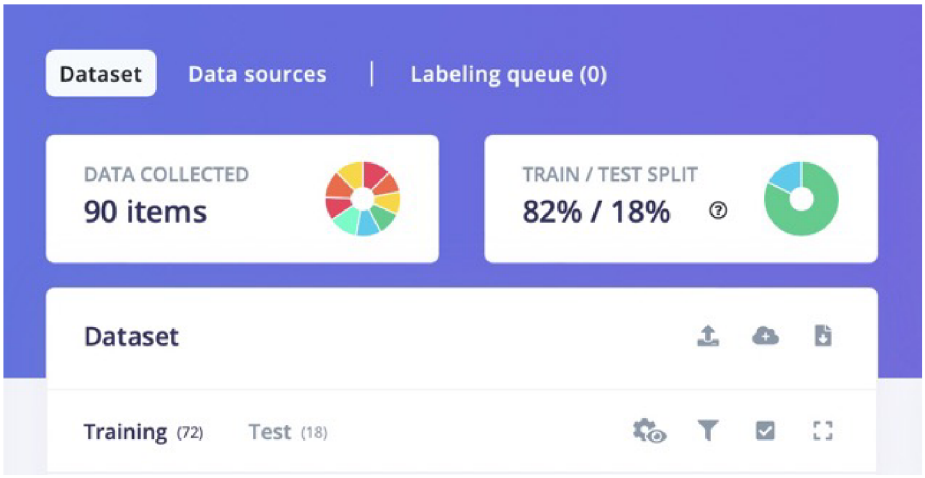
8.2 Testovani

Testování prostřednictvím platformy Edge Impulse

Model jsem testovala dvěma způsoby. Prvním způsobem je testování hotového modelu na platformě Edge Impulse. Dříve v kapitole 6.3 byly uvedeny výsledky testování neuronové sítě na validační sadě dat. K trénování a testování modelu jsou zapotřebí celkem tři datové sady:

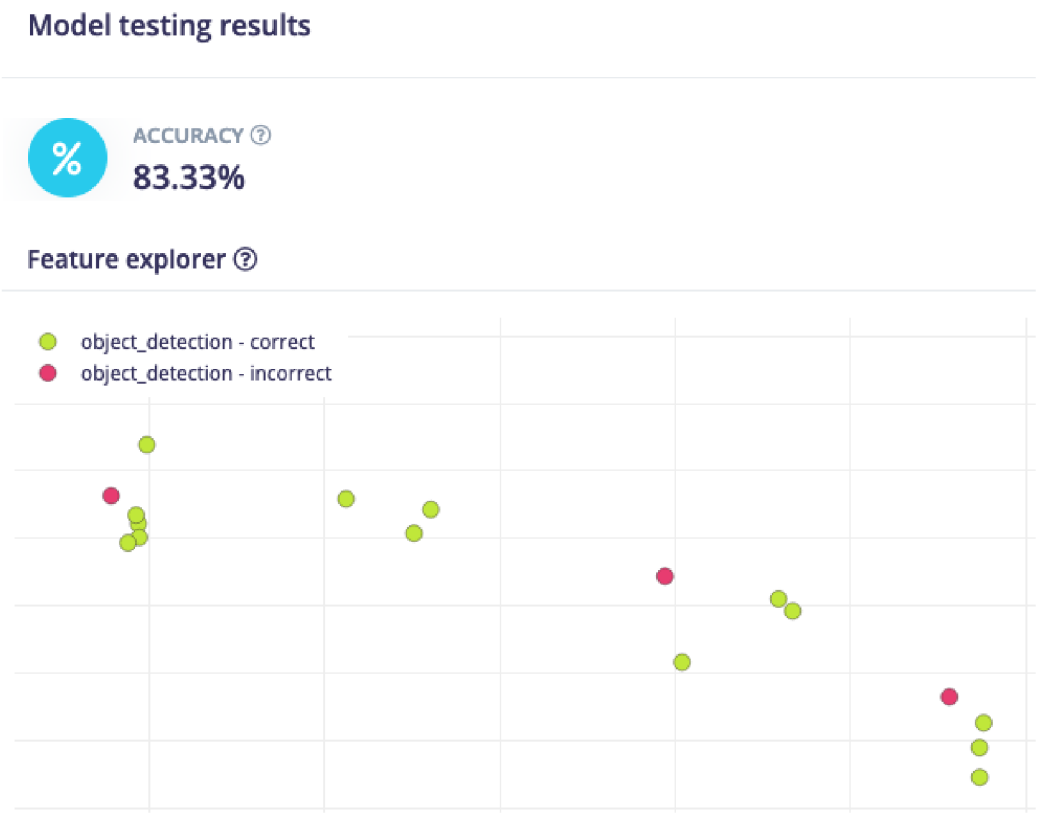
- **Training Dataset** - Vzorek dat použitý k trénování modelu.
- **Validation Dataset** - Validační datové sady používají vzorek dat, který se nepoužívá v procesu trénování. Tato data se pak používají k odhadu případných zjevných chyb. Na základě těchto dat lze pak upravit hyperparametry modelu - laditelné parametry, které slouží k řízení chování modelu. Proto model tato data občas vidí, ale nikdy se z nich „neučí“. Tento proces slouží jako nezávislý soubor dat pro porovnání výkonnosti modelu [23].
- **Test Dataset** - Vzorek dat, který slouží k objektivnímu vyhodnocení finální způsobilosti modelu na trénovacím souboru dat. Tato data model nikdy předtím neviděl.

V sekci Model testing můžeme výsledný model otestovat na testovacích datech. Jak již bylo uvedeno, kompletní soubor dat se skládal pouze z 90 obrázků. Po nahrání datové sady do platformy byly fotografie automaticky rozděleny na tréninkovou a testovací část v poměru 80 % / 20 % 8.5. Po tomto roztřídění zůstalo v testovacím souboru dat pouze 18 snímků.



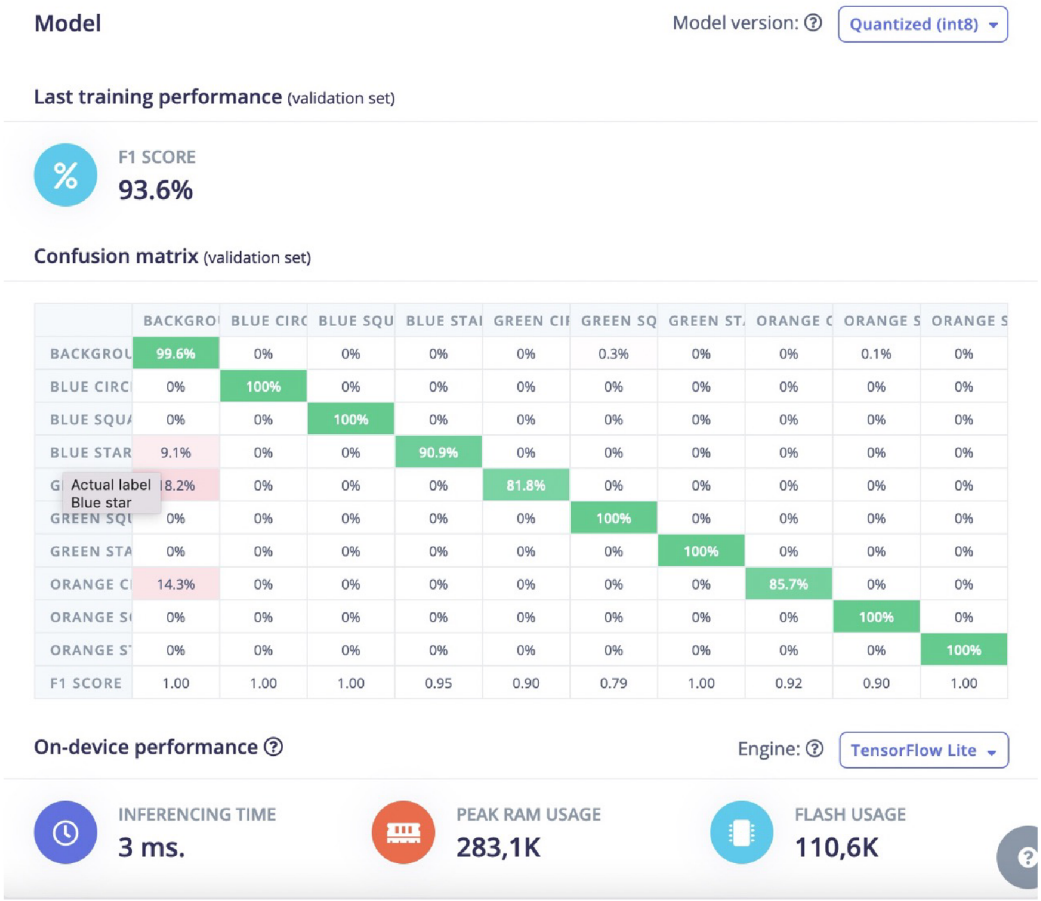
Obrázek 8.5: Dataset

Po kliknutí na tlačítko „otestovat vše“ model klasifikuje všechny vzorky testovací sady a zobrazí celkovou přesnost modelu 8.6.



Obrázek 8.6: Výsledky testování

Jak vidíte, FOMO používá trochu jiné základní metriky hodnocení než SSD MobileNetV2 / Yolo / RetNet atd., které používají průměrnou přesnost (mAP). Použití průměrné přesnosti (mAP) jako jediné metriky hodnocení může někdy poskytnout omezený pohled



	BACKGRO	BLUE CIRC	BLUE SQU	BLUE STAI	GREEN CII	GREEN SQ	GREEN ST.	ORANGE C	ORANGE S	ORANGE S
BACKGROU	99.6%	0%	0%	0%	0%	0.3%	0%	0%	0.1%	0%
BLUE CIRC	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%
BLUE SQUA	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%
BLUE STAR	9.1%	0%	0%	90.9%	0%	0%	0%	0%	0%	0%
G	8.2%	0%	0%	0%	81.8%	0%	0%	0%	0%	0%
GREEN SQU	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%
GREEN STA	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%
ORANGE C	14.3%	0%	0%	0%	0%	0%	0%	85.7%	0%	0%
ORANGE S	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%
ORANGE S	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%
F1 SCORE	1.00	1.00	1.00	0.95	0.90	0.79	1.00	0.92	0.90	1.00

Obrázek 8.7: Výsledky testování validačního datasetu

na výkonnost modelu. To platí zejména v případě datových sad s nevyváženými třídami, protože měří pouze přesnost predikce, aniž by zohledňovala, jak dobrý nebo špatný je model pro jednotlivé třídy. Kombinace F1 Score a Confusion matrix nám poskytuje rovnováhu mezi **precision** a **recall** modelu a ukazuje, jak si model vede pro jednotlivé třídy [20].

Zde je několik pojmů a vzorců:

- **True Positive** - (TP) je počet pravdivě pozitivních výsledků
- **True Negative** - (TN) je počet pravdivě negativních výsledků
- **False Positive** - (FP) je počet falešně pozitivních výsledků
- **False Negative** - (FN) je počet falešně negativních výsledků
- **Precision** - Přesnost udává přesnost pozitivních předpovědí [20]. Je definována jako

$$precision = \frac{TP}{TP + FP}$$

- **Recall** - měří schopnost modelu najít všechny relevantní případy v souboru dat [20]

$$recall = \frac{TP}{TP + FN}$$

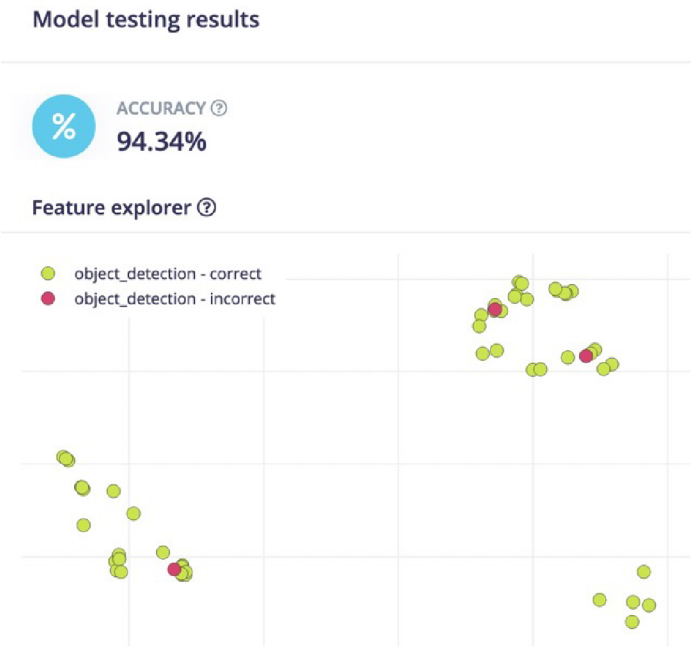
- **Accuracy** - Přesnost je podíl správných předpovědí našeho modelu [20]. Je definována jako:

$$accuracy = \frac{TP + TN}{TP + TN + FT + FN}$$

- **F1 Score** - Skóre F1 je harmonickým průměrem přesnosti a odvolávky, který zajišťuje jejich rovnováhu [20]. Vypočítá se jako:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

Pro jistotu ověření jsem se rozhodla rozšířit testovací soubor dat o dalších 36 snímků s větším počtem objektů na fotografii (od 10 do 16). Nyní je v testovaném souboru dat 54 obrázků.



Obrázek 8.8: Opakované testování

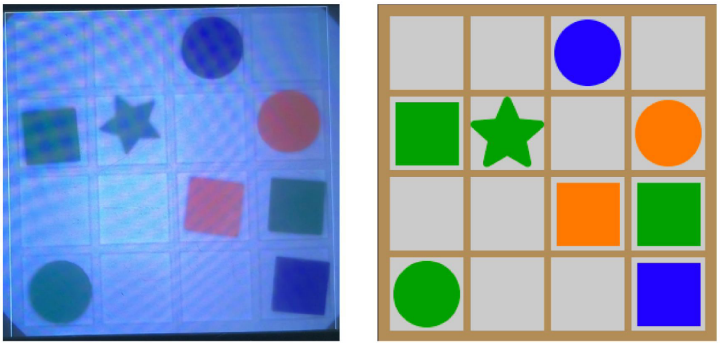
Procento správných odhadů se zvýšilo 8.8, což vypadá jako dobrý výsledek. To znamená, že model identifikuje většinu objektů správně.

Ruční testování

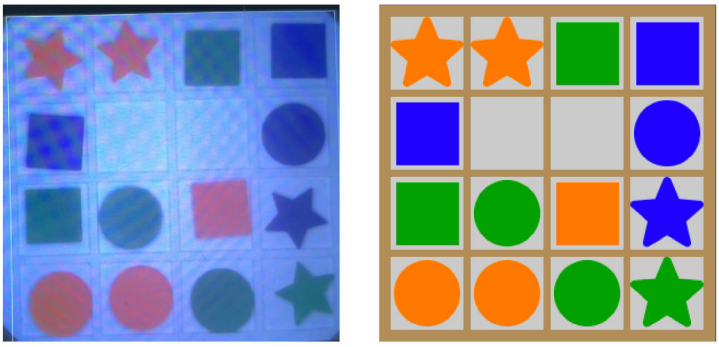
Samozřejmě bylo nutné otestovat nejen samotnou neuronovou síť, ale i celou aplikaci jako celek a vyhodnotit výsledky přímo na mikrokontroléru. Za tímto účelem jsem 35krát ručně spustila detektor prostřednictvím své aplikace a porovnala výsledky se skutečnou polohou objektů. Výsledkem bylo, že 35 snímků obsahovalo 405 objektů, z nichž 9 objektů bylo klasifikováno chybně a 3 objekty nebyly nalezeny vůbec.

Pár příkladů naleznete zde 8.11 8.9 8.10 8.12, zbytek jsem nahrála do repozitáře na githubu[2].

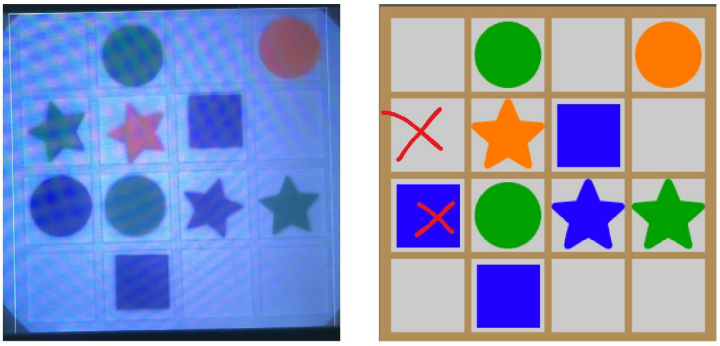
[2] https://github.com/kravcc/BP_dataset_and_results/tree/main/manual_testing_result



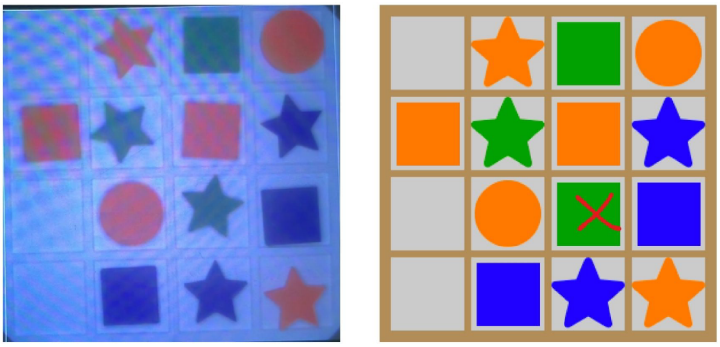
Obrázek 8.9: Ruční testování. Příklad č.1



Obrázek 8.10: Ruční testování. Příklad č.2



Obrázek 8.11: Ruční testování. Příklad č.3



Obrázek 8.12: Ruční testování. Příklad č.4

Kapitola 9

Závěr

V této práci jsem vytvořila aplikaci pro vyhledávání objektů na principu neuronové sítě. Při psaní práce jsem studovala principy konvolučních neuronových sítí a problematiku strojového vidění. Důležitým aspektem byla také oblast použití neuronových sítí ve vestavěných zařízeních s nízkým výkonem. Proto bylo nutné se seznamit s způsoby optimalizace a hledat vhodné modely konkrétně pro tuto úlohu. Za účelem výběru vhodného modelu neuronové sítě byli prozkoumáné a otestováné: MobileNet SSDv2, Yolov3, NanoDet, FOMO. Nakonec volba padla na model strojového učení od Edge Impulse: FOMO (Faster Objects, More Objects).

Pro vytvoření datové sady jsem na 3D tiskárně vytiskla různé objekty. Jsem je ručně vyfotila, každou fotografii zpracovala a jednotlivé objekty označila. Po natrénování modelu přesnost předpovědí na validační sadě dat byla 93,6 %, na testovací sadě dat 83,3 %. Jako základ pro vytvoření aplikace na mikrokontroléru jsem použila příklad ze sady MCUXpresso IDE SDK.

Jednalo se o příklad použití klasifikátoru, který má s detekcí objektů mnoho společného. Rozdíl je v tom, že klasifikátor je schopen rozpoznat pouze jeden objekt a jako výsledek produkuje ID objektu, zatímco detekce se zabývá rozpoznáváním množiny objektů, jejich klasifikací a zjišťováním souřadnic. Jako výsledek generuje na první pohled nesrozumitelné a chaotické pole čísel. Abych pochopila, v jaké podobě mají být vstupní data a jak mám zpracovávat výstupní data, podrobně jsem prostudovala dokumentaci a pomocné kódy platformy Edge Impulse.

Po mnoha změnách v kódu se podařilo projekt spustit a potvrdit, že model funguje a dělá vše, jak má. Zbývalo dokončit některé vedlejší záležitosti, jako je vytvoření grafického rozhraní a vytvoření komunikace na úrovni TCP mezi deskou a desktopovou aplikací. V závěru bylo dosaženo úspěšné implementace detekce objektů na mikrokontroleru i.MX RT s možností ovládání pomocí jednoduchého uživatelského rozhraní na PC, což považuji za kladný výsledek.

Tento projekt se plánuje použít jako ukázková aplikace na veletrzích/konferencích společnosti NXP. Projekt bude zabudován do skleněné krabice s robotickým ramenem, které bude schopno na žádost uživatele zvedat nalezené předměty. Projekt lze v budoucích verzích různě upravovat. Například změnit datovou sadu tak, aby neuronová síť mohla bez problémů vyhledávat objekty při různých úhlech pohledu. Zvětšit datovou sadu pro dosažení lepšího procenta přesnosti. Model lze vlastně natrénovat na cokoli, třeba na detekci skutečných dílů/výrobků a využít chytrého robota jako třídičku ve výrobě.

Literatura

[1] *EIQ® ML Software Development Environment* [online]. NXP. Dostupné z: https://www.nxp.com/design/design-center/software/eiq-ml-development-environment:EIQ.

[2] *Google Colaboratory* [online]. google. Dostupné z: https://colab.google.

[3] *Object Detection Guide* [online]. fritz.ai. Dostupné z: https://fritz.ai/object-detection/.

[4] *LwIP - A Lightweight TCP/IP stack* [online]. 2002. Dostupné z: https://savannah.nongnu.org/projects/lwip/.

[5] 3Blue1Brown. *What is backpropagation really doing?* [online]. 2018. Dostupné z: https://www.youtube.com/watch?v=Ilg3gGewQ5U&t=33s.

[6] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Dostupné z: https://www.tensorflow.org/lite.

[7] Adam, D. a Leon, W. *LwIP - Overview* [online]. 2002. Dostupné z: https://www.nongnu.org/lwip/2_1_x/index.html.

[8] Alake, R. *Loss Functions in Machine Learning Explained* [online]. 2023. Dostupné z: https://www.datacamp.com/tutorial/loss-function-in-machine-learning.

[9] Algorithimia's website. *Introduction to Loss Functions* [online]. 2023. Dostupné z: https://www.datarobot.com/blog/introduction-to-loss-functions/#:~:text=Further%20reading-,What's%20a%20loss%20function%3F,ll%20output%20a%20lower%20number.

[10] Ali, M. *Introduction to Activation Functions in Neural Networks* [online]. 2023. Dostupné z: https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks.

[11] Amazon. *What is a neural network?* [online]. 2023. Dostupné z: https://aws.amazon.com/ru/what-is/neural-network/#:~:text=A%20neural%20network%20is%20a,that%20resembles%20the%20human%20brain.

[12] ARM. *Cortex-M* [online]. 2023. Dostupné z: https://www.arm.com/products/silicon-ip-cpu?families=cortex-m&showall=true.

[13] BILOGUR, A. *Notes on weight sharing* [online]. 2019. Dostupné z: https://www.kaggle.com/code/residentmario/notes-on-weight-sharing.

[14] Bochkovskiy, A., Wang, C.-Y. a Liao, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection.* 2020.

[15] Boyle, L., Baumann, N., Heo, S. a Magno, M. *Enhancing Lightweight Neural Networks for Small Object Detection in IoT Applications.* 2023.

[16] Bre, F. *Artificial neural network architecture* [online]. Dostupné z: https://www.researchgate.net/figure/Artificial-neural-network-architecture-ANN-i-h-1-h-2-h-n-o_fig1_321259051.

[17] Chandra, P. *IoU Loss Functions for Faster More Accurate Object Detection* [online]. 2023. Dostupné z: https://learnopencv.com/iou-loss-functions-object-detection/#disqus_thread.

[18] Cheng, C. *Real-Time Mask Detection Based on SSD-MobileNetV2.* 2022.

[19] Data Robot. *Introduction to Loss Functions* [online]. 2018. Dostupné z: https://www.datarobot.com/blog/introduction-to-loss-functions/#:~:text=At%20its%20core%2C%20a%20loss,ll%20output%20a%20lower%20number.

[20] Edge Impulse. *Detect objects with FOMO* [online]. 2023. Dostupné z: https://docs.edgeimpulse.com/docs/tutorials/end-to-end-tutorials/object-detection/detect-objects-using-fomo?

[21] Edge Impulse. *Edge Impulse* [online]. 2023. Dostupné z: https://docs.edgeimpulse.com.

[22] Edge Impulse. *FOMO: Object detection for constrained devices* [online]. 2023. Dostupné z: https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/fomo-object-detection-for-constrained-devices.

[23] Gillis, A. S. *What is a validation set?* [online]. 2023. Dostupné z: https://www.techtarget.com/whatis/definition/validation-set.

[24] Girshick, R., Donahue, J., Darrell, T. a Malik, J. *Rich feature hierarchies for accurate object detection and semantic segmentation.* 2014.

[25] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A. et al. *TensorFlow 2 Detection Model Zoo* [https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md]. 2017.

[26] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A. et al. *TensorFlow Object Detection API* [https://github.com/tensorflow/models/tree/master/research/object_detection]. 2017.

[27] IBM. *What are convolutional neural networks?* [online]. Dostupné z: https://www.ibm.com/topics/convolutional-neural-networks.

[28] Isakov, S. *Gradient descent: everything you need to know* [online]. 2018. Dostupné z: https://neurohive.io/ru/osnovy-data-science/gradient-descent/.

[29] Law, K. *[Networking Theory] Understanding TCP/IP: The Backbone of the Internet* [online]. 2023. Dostupné z: https://medium.com/@kylelzk/networking-theory-understanding-tcp-ip-the-backbone-of-the-internet-c435f50d7a9a.

[30] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. et al. *SSD: Single Shot MultiBox Detector.* 2015.

[31] Ma, N., Zhang, X., Zheng, H.-T. a Sun, J. *ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design.* 2018.

[32] Mazet, V. *Convolution* [online]. Dostupné z: https://vincmazet.github.io/bip/filtering/convolution.html.

[33] Molchanov, P., Tyree, S., Karras, T., Aila, T. a Kautz, J. *Prining convolutional neural networks for resource efficient inference* [https://openreview.net/pdf?id=SJGCiw5gl]. 2017.

[34] Nasscom Company. *ANPR using NanoDet, an Anchor-free Object Detection model* [online]. 2023. Dostupné z: https://wire19.com/anpr-using-nanodet-an-anchor-free-object-detection-model-2/.

[35] NXP Semiconductors. *MIMXRT1170 EVK Board Hardware User's Guide* [https://www.nxp.com/webapp/Download?colCode=MIMXRT1170EVKHUG]. 2020.

[36] NXP Semiconductors. *I.MX RT1170: 1 GHz Crossover MCU with Arm® Cortex® Cores* [online]. 2023. Dostupné z: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus/i-mx-rt1170-1-ghz-crossover-mcu-with-arm-cortex-cores:i.MX-RT1170.

[37] NXP Semiconductors. *MCUXpresso Integrated Development Environment (IDE)* [online]. 2023. Dostupné z: https://www.nxp.com/design/design-center/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE.

[38] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E. et al. Automatic differentiation in PyTorch. In: *NIPS-W.* 2017. Dostupné z: https://pytorch.org.

[39] Radek, K. a Zbořil, F. *Zdroje z přednašky IZU. Předaška č.9 - Rozpoznávání.* [online]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cwk.php.cs?title=Main_Page&src=2122izu_09.pdf&ns=IZU&action=download&csid=769752&id=14726.

[40] RangiLyu. *NanoDet-Plus: Super fast and high accuracy lightweight anchor-free object detection model.* [https://github.com/RangiLyu/nanodet]. 2021.

[41] Redmon, J., Divvala, S., Girshick, R. a Farhadi, A. *You Only Look Once: Unified, Real-Time Object Detection.* 2016.

[42] Shukueian, S. *How Single Shot MultiBox Detector (SSD) Real-Time Object Detection Technique works?* [online]. 2021 [cit. 2019-10-02]. Dostupné z: https://shukueian.medium.com/how-single-shot-multibox-detector-ssd-real-time-object-detection-technique-works-47e87bee2562.

[43] TANTAI hengtao. *Use weighted loss function to solve imbalanced data classification problems* [online]. 2023. Dostupné z: https://medium.com/@zergtant/use-weighted-loss-function-to-solve-imbalanced-data-classification-problems-749237f38b75.

[44] THE QT COMPANY. *Qt for Python* [online]. 2023. Dostupné z: https://www.qt.io/qt-for-python.

[45] TZUTALIN. *LabelImg* [https://github.com/tzutalin/labelImg]. 2015.

[46] WALINGA, J. a STANGOR, C. *The Neuron Is the Building Block of the Nervous System* [online]. 2014. Dostupné z: https://opentextbc.ca/introductiontopsychology/chapter/3-1-the-neuron-is-the-building-block-of-the-nervous-system/.

Příloha A

Obsah přiloženého paměťového média

Paměťové médium obsahuje:

- xkravc02-BP.pdf - elektronická verze písemné zprávy
- latex.zip - zdrojové soubory písemné zprávy
- bp-project.zip - zdrojové soubory programu
- README.txt - informace o souborech