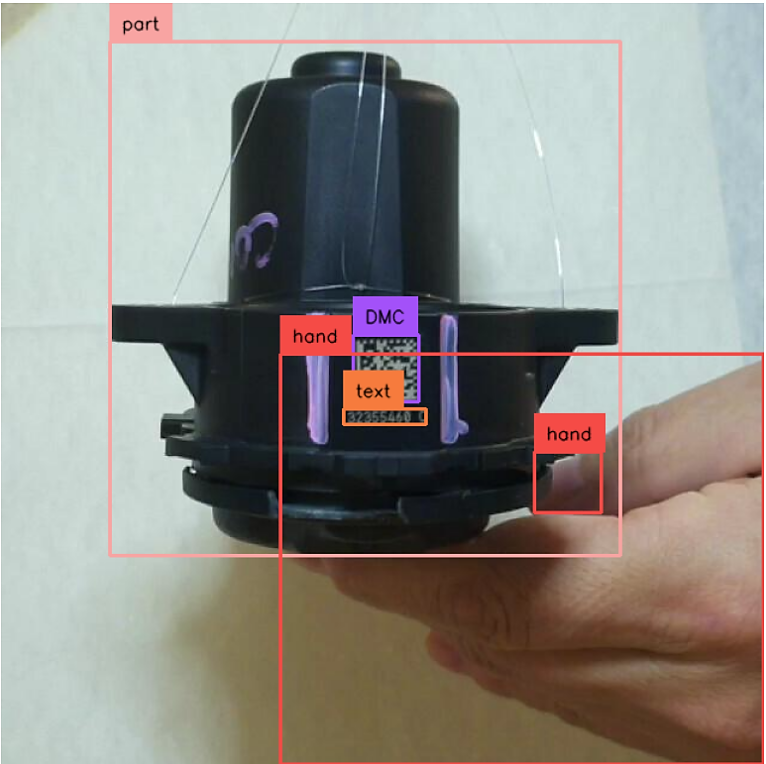
Maturitní práce

Strojní systém vidění pro detekci

2D kódu (DMC) na dílu Aktuátor

Akt. Linka EA3 ve firmě ZF Jablonec nad Nisou



| Referenční číslo | Status | Autor(é) |
| --- | --- | --- |
| 6/2/2025 | [Úvodní](http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=19663) dokument | D. Brejchová |
|  |  |  |
|  |  |  |

Obsah

Obsah iv

1 Úvod a definice cílů 1-1

1.1 Představení tématu 1-1

1.2 Význam funkce sledování ve výrobě 1-1

1.3 Zaměření a hlavní cíle maturitní práce 1-2

1.4 Vedlejší cíle práce 1-2

1.5 Hlavní kroky realizace 1-3

2 Informace k projektu 2-4

2.1 Historie revizí 2-4

2.2 Zamýšlené publikum a návrhy ke čtení 2-4

2.3 Zúčastněné strany technického projektu 2-4

2.4 Odkazy 2-4

2.5 Definice, akronymy a zkratky 2-5

3 Konfigurace HW 3-1

3.1 Raspberry PI 3-1

3.2 HW Workstation PC 3-6

3.3 HW pro napájení a komunikaci 3-6

4 Konfigurace SW 4-7

4.1 Raspberry PI 5 konfigurace 4-7

4.2 Raspberry PI 5 SW instalace 4-7

4.3 PC workstation 4-12

5 Instalace softwaru 5-1

5.1 Raspberry PI 5-1

5.2 PC pro systém učení 5-3

6 ML modely a přehled funkce 6-1

7 Work flow, učení modelu 7-2

7.1 Použití Google COLAB 7-2

8 Postup učení a konverze modelu 8-12

8.1 Převod modelu PyTorch --> ONNX 8-12

8.2 Pou6it9 Dataflow Compileru – Hailo - 8-14

8.3 ONNX model - Identifikace koncových uzlů v siti 8-16

8.4 Převod modelu ONNX --> HAR 8-18

8.5 Kontrola HAR souboru 8-20

8.6 Optimalizace HAR souboru 8-21

8.7 Optimalizace modelu do HEF souboru 8-26

8.8 Kompilace do souboru HEF 8-27

9 Raspberry Pi5 a Hailo: Základní pipeline pro zpracování dat 9-34

9.1 Instalace prostředí, základní pipeline 9-34

9.2 Hailo RPi5: Základní pipeline 9-34

9.3 Přehled pipeline a jejich využití 9-35

10 Moje aplikace 10-38

10.1 Uživatelský návod pro projekt DMC dekódování na Raspberry Pi 10-38

10.2 Přehled souborů projektu 10-38

10.3 Instalace a spuštění 10-38

10.4 Hlavní řídící skript (main.py) 10-39

10.5 GStreamer pipeline (pipeline\_code.py) 10-41

10.6 Dekódovací vlákno (decode\_thread.py) 10-43

10.7 Callback funkce (user\_callback.py) 10-44

10.8 Grafické rozhraní (gui\_display.py) 10-45

10.9 vizuální znázornění fungování systému 10-45

10.10 Závěr 10-46

11 11-47

11.1 Raspberry PI 11-1

11.2 Workastation 11-1

12 Cognex Machine Vision Systém 12-2

12.1 Hlavní součásti systému 12-2

12.2 Klíčové funkce a přínosy 12-2

12.3 Oblasti využití 12-2

12.4 Závěr 12-3

13 Porovnání modelů a závěr práce 13-4

14 APPENDIX I princip EPB 14-5

14.1 Princip elektrické parkovací brzdy a její konstrukce 14-5

15 APPENDIX II dokumentace nakup díly 15-7

# Úvod a definice cílů

## Představení tématu

V dnešní době hraje automatizace kontroly kvality klíčovou roli v průmyslové výrobě. Pomáhá dosáhnout vysoké přesnosti a efektivity, což je zásadní pro udržení konkurenceschopnosti. Mezi nejvýznamnější technologie používané v tomto procesu patří kamerové systémy, které umožňují rychlou, přesnou a spolehlivou kontrolu výrobků. Tyto systémy nacházejí uplatnění v mnoha odvětvích, od elektroniky a automobilového průmyslu až po farmaceutickou výrobu.

Díky kamerovým systémům lze snadno detekovat výrobní vady, kontrolovat montážní procesy, identifikovat jednotlivé součástky a ověřovat správnost značení. Moderní kamerové systémy využívají pokročilé algoritmy strojového vidění, které analyzují snímané obrazy a vyhodnocují je na základě předem stanovených kritérií. Tím pomáhají zajistit stabilní kvalitu výroby a eliminovat lidské chyby, které by mohly vést k vadným produktům nebo zvýšeným nákladům na opravy.

Kontrola kvality prováděná lidskými pracovníky je dnes považována za zastaralé řešení, jelikož je méně spolehlivá a nevhodná pro opakující se úkoly vyžadující vysokou přesnost. Moderní průmyslové procesy se stále více spoléhají na automatizované systémy, které eliminují lidské chyby, zvyšují efektivitu a umožňují nepřetržitou kontrolu kvality.

V automobilovém průmyslu se kamerové systémy používají například ke kontrole přítomnosti, správné montáže součástek, měření rozměrů, **čtení kódů systému sledování nakupovaných nebo hotových dílů** nebo odhalování povrchových vad.

## Význam funkce sledování ve výrobě

V automobilovém průmyslu, kde jsou nároky na spolehlivost a bezpečnost extrémně vysoké, je sledovatelnost jednotlivých součástí produktu nezbytná. Příklady důležitosti sledovatelnosti jednotlivých dílů:

### Rychlá reakce na výrobní vady a svolávací akce

Pokud se objeví vadný díl v sériové výrobě nebo u zákazníka, lze rychle dohledat jeho původ, dodavatele a konkrétní výrobní šarži. To umožňuje cílené svolávací akce místo plošných výměn, což snižuje náklady i dopady na zákazníky.

Např. při vadné brzdě lze zjistit, ve kterých vozidlech byl konkrétní kus instalován a rychle kontaktovat majitele. Případně lze dosledovat konkrétní vadnou sérii neshodné vstupní komponenty u dodavatele.

### Kvalitativní kontrola a optimalizace dodavatelského řetězce

Umožňuje porovnávat kvalitu dílů od různých dodavatelů a eliminovat nespolehlivé subdodavatele. Pomáhá odhalit opakující se vady u konkrétního výrobce a přizpůsobit požadavky na vstupní kontrolu.

### Efektivní řízení výroby a skladových zásob

Díky sledovatelnosti lze optimalizovat skladování, identifikovat starší šarže a zajistit správné pořadí použití dílů (FIFO – First In, First Out).

Minimalizuje se riziko použití nesprávných komponent nebo zastaralých verzí dílů ve výrobě.

### Ochrana proti padělkům a zajištění shody s předpisy

Umožňuje ověření pravosti dílů, což je klíčové pro bezpečnostní a certifikační normy (např. ISO 9001, IATF 16949).

Pomáhá eliminovat neoriginální nebo necertifikované komponenty, které mohou ohrozit bezpečnost vozidla.

**Systém sledování nakupovaných nebo hotových dílů** se realizuje buď štítkováním, nebo přímým značením (DMP) např. laserovým popisem Data Matrix Code (DMC) na díl, který umožňuje přesné přiřazení dílu k jeho výrobní historii.

Kontrola kvality DMC laserového značení hraje klíčovou roli pro zabezpečení čitelnosti během procesu výroby a následně během celé životnosti dílu.

## Zaměření a hlavní cíle maturitní práce

**Tato maturitní práce se zabývá návrhem a implementací kamerového systému Kontrola kvality DMC laserového značení na součásti elektrické parkovací brzdy (EPB), konkrétně na aktuátoru EPB. Identifikace a sledování této součásti v průběhu výrobního procesu je zajištěna prostřednictvím Data Matrix Code (DMC), který je na aktuátor aplikován laserovým popisem pomocí zařízení PANASONIC**

Hlavním cílem navrhovaného systému je automatická detekce DMC kódu v pracovním prostoru stanice DPM. Jakmile je kód nalezen, bude pořízený snímek analyzován z hlediska kvality čtení, což je zásadní aspekt pro sledovatelnost dílu a jeho bezchybnou identifikaci ve výrobním procesu.

## Vedlejší cíle práce

Tato maturitní práce se zaměřuje na návrh a realizaci cenově dostupného open-source řešení, které využívá strojové učení (Machine Learning – ML) pro přesnou a robustní detekci DMC kódu. Výkonnost navrženého systému bude porovnána s profesionálním řešením od společnosti COGNEX s důrazem na následující klíčové aspekty:

**Náklady na realizaci – porovnání investičních a provozních nákladů open-source řešení a profesionálního systému.**

**Náročnost integrace – posouzení technické složitosti implementace obou přístupů.**

**Spolehlivost čtení – analýza přesnosti a úspěšnosti detekce DMC kódu v reálných podmínkách výroby.**

## Hlavní kroky realizace

V rámci této práce budou provedeny následující kroky

**Výběr vhodného hardwarového open-source systému pro strojové vidění, včetně volby výpočetní jednotky, kamery, optiky a osvětlovacího systému.**

**Návrh a implementace open-source softwaru, včetně re-trénování AI modelů a úpravy softwarových algoritmů pro analýzu obrazu (ML), optimalizovaných pro detekci DMC kódu.**

**Návrh zapojení, sestavení a oživení jednotlivých komponent testovacího systému strojního vidění, včetně napájení a komunikace s řídicí jednotkou.**

**Nasazení SW a před-učeného modelu (ML) na testovací systém strojního vidění**

**Optimalizace procesu (ML) detekce DMC, včetně výběru vhodného AI modelu a jeho přizpůsobení specifickým požadavkům aplikace. Jedná se o hlavní proces, cílem je najít optimální AI model**

**Vyhodnocení experimentálního open-source řešení a jeho srovnání s profesionálním systémem COGNEX z hlediska přesnosti, efektivity a nákladů.**

# Informace k projektu

## Historie revizí

Tabulka Historie revizí zobrazuje datum, změny a autory, kteří na tomto dokumentu pracovali.

| Číslo požadavku na verzi/změnu | Datum verze | Popis změn | Autor |
| --- | --- | --- | --- |
|  | 12/23/2024 | První návrh | D.Brejchová |
|  |  |  |  |
|  |  |  |  |

## Zamýšlené publikum a návrhy ke čtení

Tento Průvodce instalací a nasazením je určen pro technické účastníky projektu, kteří budou zodpovědní za plánování, provádění nebo údržbu instalace nebo nasazení, jako je správce systému, ředitel informačních technologií (CIO), analytici nebo vývojáři.

Účelem je, aby si zúčastněné strany a pracovníci softwarové podpory mohli přečíst tento dokument a koordinovat své úsilí při instalaci/nasazení aplikace.

## Zúčastněné strany technického projektu

V této části je uveden seznam všech známých zúčastněných stran, které mají o projekt zájem.

| Jméno | E-mailová adresa | Telefon | Role |
| --- | --- | --- | --- |
| Dominika Brejchova |  |  | Hlavní vývojář |
| David Sokola |  |  | Technická podpora |

## Odkazy

Nahraďte níže uvedené ukázky relevantními odkazy pro instalační prostředí.

| Referenční číslo | Dokument | Autor(é) |
| --- | --- | --- |
| ODKAZ 1 | [Stáhnout sadu nástrojů WCF RIA Services Toolkit, květen 2010](http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=19663) | Společnost Microsoft |
| ODKAZ 2 | [Postupy: Instalace SQL Server 2008 R2 (instalační program)](http://msdn.microsoft.com/en-us/library/ms143219.aspx) | Společnost Microsoft |
| ODKAZ 3 | [Microsoft .NET Framework 4 (samostatný instalační program)](http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=17718) | Společnost Microsoft |

## Definice, akronymy a zkratky

| Semestr | Definice |
| --- | --- |
| Správce | Jedná se o kohokoli z klienta, kterému byla udělena oprávnění správce v **Chyba! Nenalezen zdroj odkazů.**. |
| IIS 7 | Microsoft Internet Information Server 7 |

# Konfigurace HW

## Raspberry PI

<https://www.raspberrypi.com/documentation/>

### Raspberry PI 5, 8Gb

Počítač Raspberry Pi 5 je vybaven 64bitovým čtyřjádrovým procesorem Arm Cortex-A76 s frekvencí 2,4 GHz. Přináší 2-3× vyšší výkon procesoru než Raspberry Pi 4 a výrazně posílenou 800MHz grafickou VideoCore VII. Zvládá dva 4Kp60 displeje připojené přes microHDMI a díky přepracovanému procesoru pro zpracování obrazu přidává i nejmodernější podporu kamer.

* Raspberry Pi 5, verze 8Gb
* MicroSD 32GB Class 10
* USB-C napájecí adaptér (5V/5A), originál příslušenství
* Klávesnice a myš (USB/Bluetooth), originál příslušenství
* Monitor s HDMI vstupem, pozor nutné micro-HDMI kabeal pro připojení
* Síťové připojení Wi-Fi nebo Ethernet kabel, pro ETHERNET kabel proběhne instalace Raspbian automaticky



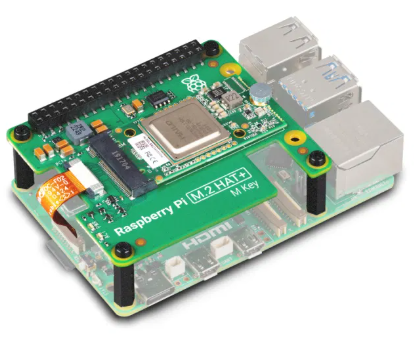
### Raspberry Pi AI HAT+ 26 TOPS

1. Odpojte Raspberry Pi od napájení před zahájením instalace.
2. Pomocí čtyř dodaných šroubů připevněte distanční sloupky. Pevně zatlačte GPIO zásuvkovou lištu na piny GPIO na Raspberry Pi. Orientace nezáleží, pokud všechny piny správně zapadnou. Odpojte páskový kabel od AI Kitu a druhý konec vložte do PCIe portu Raspberry Pi. Zvedněte držák páskového kabelu z obou stran, poté vložte kabel tak, aby měděné kontakty směřovaly dovnitř směrem k USB portům. Jakmile je páskový kabel plně a rovnoměrně zasunut do PCIe portu, zatlačte držák dolů z obou stran, aby byl kabel pevně upevněn.
3. Umístěte AI Kit na distanční sloupky. Pomocí čtyř zbývajících šroubů jej pevně připevněte.
4. Vložte páskový kabel do slotu na AI Kitu. Zvedněte držák páskového kabelu z obou stran. Vložte kabel tak, aby měděné kontakty směřovaly nahoru. Jakmile je páskový kabel plně a rovnoměrně zasunut do portu, zatlačte držák dolů z obou stran, aby byl kabel pevně upevněn.
5. Tímto je instalace AI Kitu dokončena.

Rozšiřující deska s vestavěným Hailo AI akcelerátorem pro Raspberry Pi 5.

* Akcelerátor, Neuronový, síťový inference akcelerátor 26 tera-operací za sekundu (TOPS)
* Plně integrovaný do softwarového stacku kamer Raspberry Pi
* Odpovídá specifikaci Raspberry Pi HAT+
* Dodáváno s 16 mm stohovatelným konektorem, distančními sloupky a šrouby

Komunikuje prostřednictvím rozhraní PCIe Gen 3 na Raspberry Pi 5. Když hostitelský Raspberry Pi 5 používá aktualizovaný obraz operačního systému Raspberry Pi OS, automaticky detekuje vestavěný akcelerátor Hailo a zpřístupní NPU pro úlohy umělé inteligence. Vestavěné aplikace rpicam-apps v Raspberry Pi OS nativně podporují AI modul a automaticky využívají NPU pro kompatibilní postprocessingové úkoly.



### Kamera a objektiv

1. Před připojením jakékoli kamery vypněte Raspberry Pi a odpojte jej od napájení.
2. Plochý kabel kamery se zasouvá do konektoru označeného CAMERA na Raspberry Pi, který se nachází mezi porty Ethernet a HDMI. Kabel musí být vložen tak, aby stříbrné kontakty směřovaly k HDMI portu.
3. Pro otevření konektoru zatáhněte za západky v horní části konektoru směrem nahoru a poté směrem k Ethernet portu. Plochý kabel pevně zasuňte do konektoru, přičemž dbejte na to, aby se příliš neohýbal.
4. Pro uzavření konektoru zatlačte horní část konektoru dolů a směrem od Ethernet portu, přičemž držte plochý kabel na místě.

12px oficiální Raspberry Pi kamerový modul se senzorem Sony IMX708 s vyšší světelnou citlivostí, podporou HDR a automatickým ostřením.

* Obrazový snímač CMOS s rozlišením 12 megapixelů (Sony IMX708) se zadním osvětlením a skládáním na sebe
* Vysoký odstup signálu od šumu (SNR)
* Vestavěná 2D dynamická korekce vadných pixelů (DPC)
* Automatické ostření s fázovou detekcí (PDAF) pro rychlé automatické ostření
* Funkce QBC Re-mosaic
* Režim HDR (výstup až 3 megapixely)
* Sériový datový výstup CSI-2
* Dvouvodičová sériová komunikace (podporuje rychlý režim I2C a rychlý režim plus)
* Dvouvodičové sériové ovládání zaostřovacího mechanismu

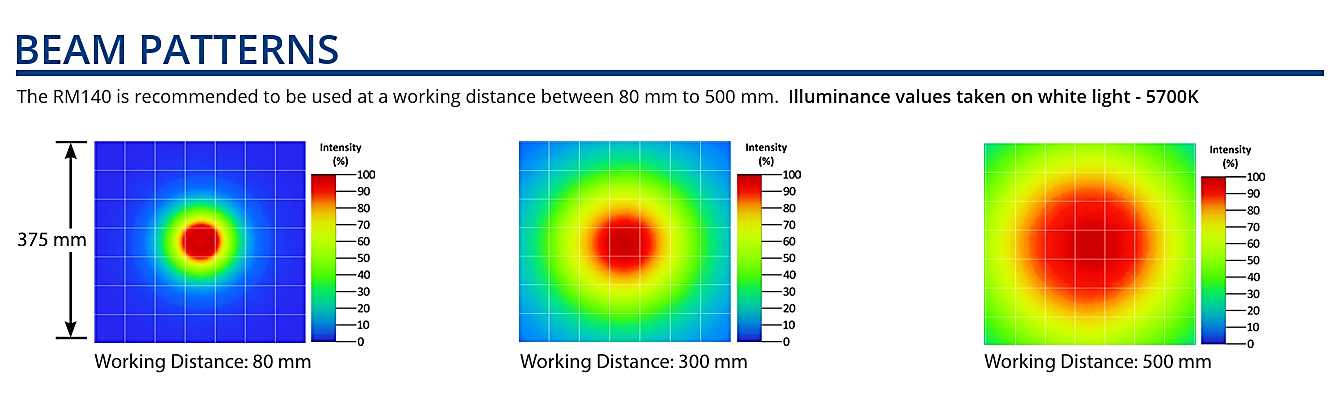
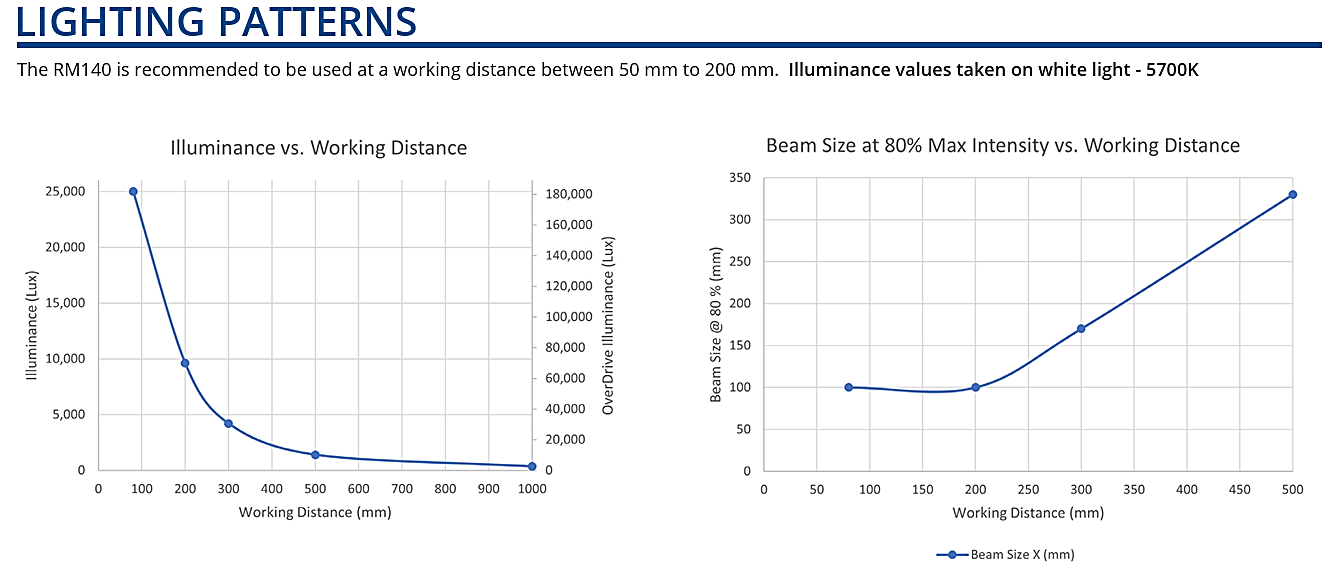


### Osvětlení pro kamerový systém

POSTUP TBD

RM140 je kruhové světlo s integrovaným ovladačem Multi-Drive™, který umožňuje provoz v kontinuálním nebo OverDrive™ režimu v závislosti na zapojení vstupů. Pro ovládání světla v pulzním (strobo) nebo kontinuálním režimu lze použít NPN nebo PNP spouštěcí signály. Intenzitu světla lze regulovat prostřednictvím analogového řídicího signálu v rozsahu 1–10 VDC.



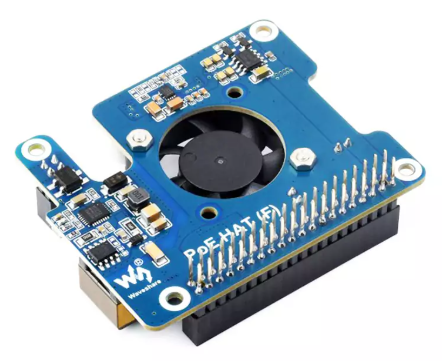


### POE Raspberry Pi HAT+

Waveshare - PoE HAT (F) pro Raspberry Pi 5

PoE HAT (F) je příslušenství pro Raspberry Pi 5 vyhovující standardu 802.3af/at pro napájení přes Ethernet. Pomocí tohoto malého HATu a odpovídajícího zařízení napájeného dle standardu 802.3af/at lze zajistit síťové připojení i napájení Raspberry Pi jediným Ethernet kabelem.

* Standardní 40PIN GPIO header pro Raspberry Pi 5
* Schopnost PoE (Power Over Ethernet), kompatibilní s IEEE 802.3af/at
* Plně izolovaný spínaný zdroj (SMPS)
* Vestavěný rychlý aktivní chladicí ventilátor s kovovým chladičem pro lepší odvod tepla

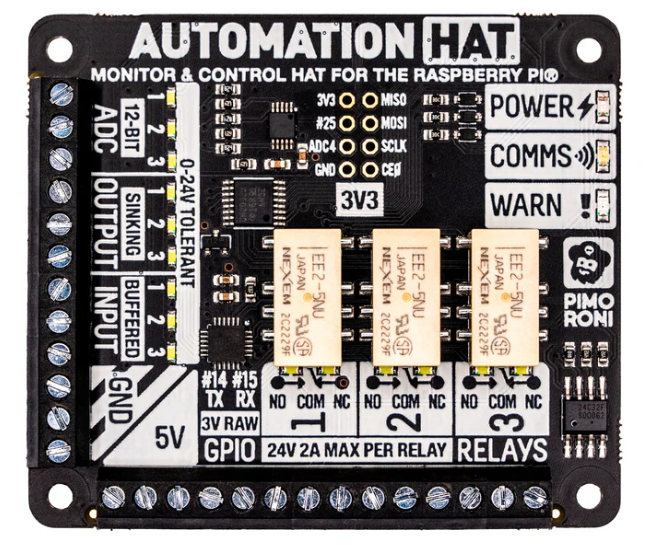
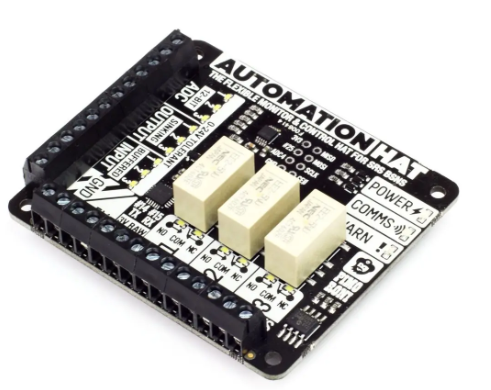


### Pimoroni Automation Raspberry Pi HAT+

Disponuje několika relé, analogovými kanály, napájenými výstupy a vstupy se zásobníkem (všechny 24V tolerantní), přes které můžete ke svému Raspberry Pi připojit celou řadu periferií.

Každý kanál má nevíc LED kontrolku, včetně analogových kanály se stmavující LED, které odrážejí hodnotu, kterou právě měří!

* 3x 24V @ 2A relays (NC and NO terminals)
* 3 x 12-bit ADC @ 0-24V
* 3 x 24V l buffered inputs
* 3 x 24V tolerant sinking outputs
* 15 x LED indikátor kanálů
* 1 x 12-bit ADC @ 0-3.3V
* 3.5mm šroubové svorky
* Napájecí, komunikační a varovné LED kontrolky
* SPI, TX (#14), RX (#15), #25 piny jsou vyvedeny



## HW Workstation PC

## HW pro napájení a komunikaci

# Konfigurace SW

## Raspberry PI 5 konfigurace

| SW | Dokument | Autor(é) |
| --- | --- | --- |
| Raspberry PI OS | Raspbian | Raspberry |
| Hailo Firmaware | [Postupy: Instalace SQL Server 2008 R2](http://msdn.microsoft.com/en-us/library/ms143219.aspx) | Společnost Microsoft |
| HailoRT Runtime |  |  |
| Hailo TAPPAS Core |  |  |
| rpicam-apps Hailo post-processing demo stages |  |  |
|  |  |  |

### SW přehled

### List knihoven a závislostí

TBD

## Raspberry PI 5 SW instalace

### Raspberry PI OS

Raspberry Pi OS je bezplatný operační systém založený na Debianu, optimalizovaný pro hardware Raspberry Pi. Raspberry Pi OS podporuje více než 35 000 balíčků Debianu. Doporučujeme jej pro většinu použití s Raspberry Pi.

Protože je Raspberry Pi OS odvozen od Debianu, následuje postupný cyklus vydávání verzí Debianu. Nové verze vycházejí přibližně každé 2 roky. Nejnovější verze Raspberry Pi OS je založena na Debianu Bookworm. Předchozí verze byla založena na Debianu Bullseye.

Obrazy Raspberry Pi OS najdete na:

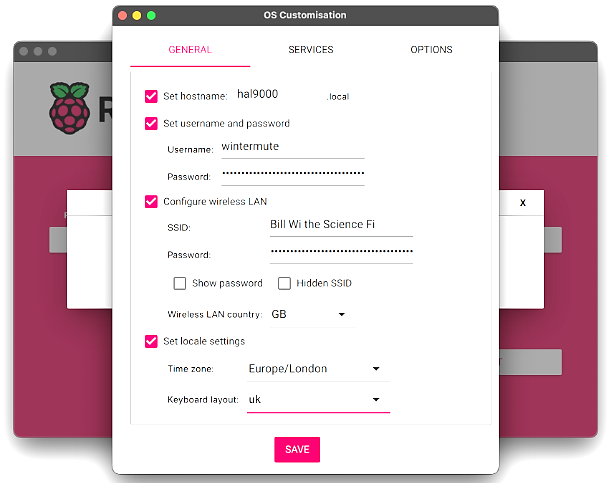
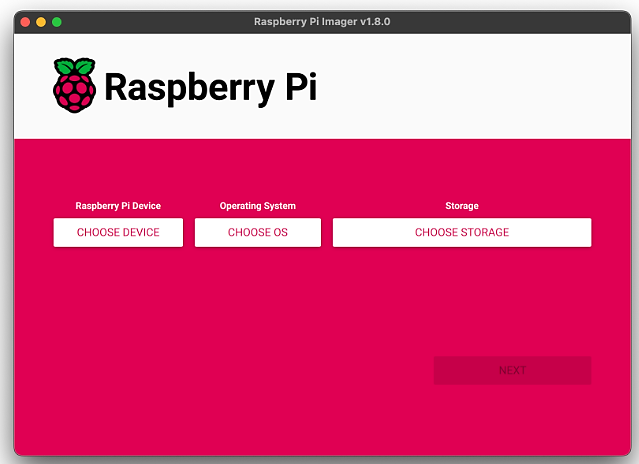
<raspberrypi.com/software/operating-systems/>

#### Úvodní nastavení a instalace systému

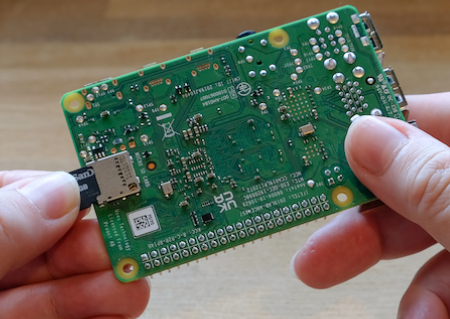
Použila jsem instalaci pomocí čtečky SD karty na učícím PC a stažení a instalace operačního SW Raspberry PI OS na SD kartu v tomto počítači

Nastavení v Imageru před instalací – Kliknutím na ⚙️ (Advanced settings) lze

* Nastavit hostname (název zařízení) → např. raspberrypi5
* Povolit SSH (pro vzdálený přístup)
* Nastavit Wi-Fi (SSID a heslo)
* Povolit výchozí uživatelské jméno a heslo



1. Stáhnout Raspberry Pi Imager pro Linux.
2. Vložit microSD kartu do počítače pomocí čtečky.
3. Zápis OS na SD kartu
4. Otevři Raspberry Pi Imager
5. Klikni na "Choose OS" → **"Raspberry Pi OS (64-bit)"**
6. Klikni na "Choose Storage" → Vyber microSD kartu
7. Volitelné: Klikni na ozubené kolečko ⚙️ pro pokročilá nastavení
8. Klikni na "Write" a počkej na dokončení
9. Po dokončení zápisu jsem vysunula microSD kartu a vložila ji do Raspberry Pi 5.



Nastavení v Raspberry PI

1. Zasuň do slotu microSD kartu do Raspberry Pi 5
2. Připoj monitor, klávesnici a myš
3. Zapoj napájení USB-C (5V/5A) → Raspberry Pi se automaticky zapne
4. Proběhne první spuštění a inicializace systému
5. Postupuj podle úvodního průvodce, kde se nastaví:

* Jazyk, časovou zónu a klávesnici
* Připojení k Wi-Fi (pokud není Ethernet)
* Vytvoření uživatelského účtu

#### Základní konfigurace systému

Otevřela jsem terminál a spustila tyto příkazy

1. Aktualizace systému

sudo apt update

1. Upgrade systemu

sudo apt full-upgrade

sudo reboot

### Kamerový systém instalace

### SW Hailo Software Ai Hailo KIT

#### Installed Components:

* **Hailo Firmware** – Essential for AI inference hardware operation.
* **HailoRT Runtime** – See the [HailoRT GitHub Repo](https://github.com/hailo-ai/hailort) for more details.
* **Hailo TAPPAS Core** – Contains GStreamer elements, post-processing functions, and tools. See [Hailo TAPPAS GitHub](https://github.com/hailo-ai/tappas).
* **rpicam-apps Hailo post-processing demo stages** – Examples from [Raspberry Pi Official Examples](https://github.com/raspberrypi/rpicam-apps).

#### Set PCIe to Gen3 for Optimal Performance

**Steps to Enable PCIe Gen3**

Open the Raspberry Pi configuration tool:

sudo raspi-config

1. Select **"6 Advanced Options"** > **"A8 PCIe Speed"**.
2. Choose **"Yes"** to enable **PCIe Gen3 mode**.
3. Exit and reboot the Raspberry Pi:

sudo reboot

#### Verify Installation

#### Check Hailo Device Recognition

Run the following command:

hailortcli fw-control identify

Expected output:

Executing on device: 0000:01:00.0

Identifying board

Control Protocol Version: 2

Firmware Version: 4.17.0 (release,app,extended context switch buffer)

Logger Version: 0

Board Name: Hailo-8

Device Architecture: HAILO8L

Serial Number: N/A

Part Number: N/A

Product Name: N/A

(If you see N/A values for Serial Number, Part Number, and Product Name, it's normal for the AI HAT.)

If the output is missing, check the **PCIe troubleshooting section**.

#### Test TAPPAS Core Installation

#### Test HailoTools (GStreamer Elements)

gst-inspect-1.0 hailotools

Expected output (partial):

Plugin Details:

Name hailotools

Description hailo tools plugin

Filename /lib/aarch64-linux-gnu/gstreamer-1.0/libgsthailotools.so

Version 3.28.2

License unknown

Source module gst-hailo-tools

#### Test HailoNet (Inference GStreamer Element)

gst-inspect-1.0 hailo

Expected output (partial):

Plugin Details:

Name hailo

Description hailo gstreamer plugin

Filename /lib/aarch64-linux-gnu/gstreamer-1.0/libgsthailo.so

If hailo or hailotools are missing, try resetting the GStreamer registry:

rm ~/.cache/gstreamer-1.0/registry.aarch64.bin

| Informace | Hodnota |
| --- | --- |
| Název serveru |  |
| Název instance SQL |  |
| Název účtu správce SQL Serveru |  |
| Heslo účtu správce SQL Serveru |  |

## PC workstation

Instalace tohoto produktu je podporována v následujících operačních systémech a verzích:

* Ubuntu

### Role, funkce a balíčky

**Role**

Před instalací softwaru musí být v operačním systému povoleny následující role serveru:

**Balíky**

Před instalací softwaru musí být v operačním systému nainstalovány následující softwarové balíčky:

### Konfigurované hodnoty

V následující tabulce si poznamenejte hodnoty pro vaše instalační prostředí pro budoucí použití.

| Informace | Hodnota |
| --- | --- |
| Název |  |

# Instalace softwaru

Nahraďte příklady v této části podrobnými pokyny k instalaci nebo nasazení softwaru.

## Raspberry PI

### Připojení a práce s Raspberry Pi ve Visual Studio Code – Můj postup

Nejprve jsem povolil SSH přístup, aby bylo možné se k Raspberry Pi připojit vzdáleně. Otevřel jsem terminál a spustil příkaz:

sudo systemctl enable ssh

sudo systemctl start ssh

Poté jsem zjistil IP adresu Raspberry Pi pomocí:

hostname -I

ip config

Následně jsem zkontroloval, zda mám na Raspberry Pi nainstalované všechny potřebné vývojové nástroje, jako Python nebo Node.js, a případně je doinstaloval.

2. Instalace Visual Studio Code

Na svém počítači jsem si stáhl a nainstaloval Visual Studio Code z oficiální stránky.

3. Instalace rozšíření Remote-SSH ve VS Code

Otevřel jsem VS Code a přešel do sekce Rozšíření (Ctrl+Shift+X). Vyhledal jsem Remote - SSH a nainstaloval ho.

4. Nastavení SSH konfigurace

Abych mohl Raspberry Pi připojit přímo ve VS Code, otevřel jsem příkazovou paletu (Ctrl+Shift+P) a zvolil možnost Remote-SSH: Add New SSH Host.

Poté jsem zadal připojovací řetězec SSH ve formátu:

pi@<IP\_adresa\_Raspberry\_Pi>

(Samozřejmě jsem nahradil <IP\_adresa\_Raspberry\_Pi> skutečnou IP adresou svého Raspberry Pi.)

Vybral jsem soubor pro uložení konfigurace (~/.ssh/config) a při prvním připojení jsem zadal heslo.

5. Připojení k Raspberry Pi

V příkazové paletě (Ctrl+Shift+P) jsem zvolil Remote-SSH: Connect to Host a vybral Raspberry Pi ze seznamu nakonfigurovaných zařízení.

VS Code se připojil a otevřel nové okno, ve kterém jsem mohl pracovat, jako bych byl přímo na Raspberry Pi.

6. Nastavení a testování vývojového prostředí

Na Raspberry Pi jsem otevřel složku s projektem nebo vytvořil novou.

Abych mohl efektivně pracovat, nainstaloval jsem potřebná rozšíření pro svůj programovací jazyk (Python, C++ atd.).

Když jsem otevřel nový projekt, VS Code mě vyzval k instalaci potřebných serverových rozšíření, která jsem potvrdil.

7. Psaní a spouštění programů

Přes integrovaný terminál ve VS Code (`Ctrl+```) jsem mohl přímo spouštět příkazy na Raspberry Pi.

Například Python skript jsem spustil takto:

python3 my\_script.py

8. Vzdálené ladění (volitelné)

Ve VS Code jsem si mohl nastavit breakpointy a spustit ladicí režim.

Předtím jsem se ujistil, že mám na Raspberry Pi nainstalovány ladicí nástroje, např. gdb pro C++ nebo pdb pro Python.

Další tipy, které mi usnadnily práci:

Přenos souborů: Jednoduše jsem mohl přetahovat soubory mezi počítačem a Raspberry Pi pomocí File Exploreru ve VS Code.

Zrychlení připojení: Nastavil jsem si SSH klíčovou autentizaci, takže jsem nemusel při každém připojení zadávat heslo. Udělal jsem to pomocí:

bash

Zkopírovat

Upravit

ssh-keygen

ssh-copy-id pi@<IP\_adresa\_Raspberry\_Pi>

Díky této konfiguraci jsem mohl programovat, testovat a ladit aplikace na Raspberry Pi přímo z VS Code bez nutnosti přímého připojení monitoru a klávesnice. 🚀

### Konfigurované hodnoty

V následující tabulce si poznamenejte hodnoty pro vaše instalační prostředí pro budoucí použití.

| Informace | Hodnota |
| --- | --- |
| Název účtu domény počátečního správce aplikace (doména\účet) |  |
| dbo.aspnet\_Users ID uživatele |  |
| application\_owner heslo k účtu |  |
| applicationproviders\_owner heslo k účtu |  |

## PC pro systém učení

Nahraďte příklady v této části podrobnými pokyny k instalaci nebo nasazení webové aplikace.

### Požadavky

### Kroky instalace

1. webových částí pro přístup k aplikaci prostřednictvím uživatelského rozhraní služby SharePoint.

### Konfigurované hodnoty

V následující tabulce si poznamenejte hodnoty pro vaše instalační prostředí pro budoucí použití.

| Informace | Hodnota |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# ML modely a přehled funkce

The models are divided to:

* Public models - which were trained on publicly available datasets.
  + For Hailo-8 - [Classification](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO8/HAILO8_classification.rst), [Object Detection](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO8/HAILO8_object_detection.rst), [Segmentation](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO8/HAILO8_semantic_segmentation.rst), [other tasks](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/PUBLIC_MODELS.rst)
  + For Hailo-8L - [Classification](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO8L/HAILO8L_classification.rst), [Object Detection](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO8L/HAILO8L_object_detection.rst), [Segmentation](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO8L/HAILO8L_semantic_segmentation.rst), [other tasks](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/PUBLIC_MODELS.rst)
  + For Hailo-15H - [Classification](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO15H/HAILO15H_classification.rst), [Object Detection](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO15H/HAILO15H_object_detection.rst), [Segmentation](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO15H/HAILO15H_semantic_segmentation.rst), [other tasks](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/PUBLIC_MODELS.rst)
  + For Hailo-15M - [Classification](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO15M/HAILO15M_classification.rst), [Object Detection](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO15M/HAILO15M_object_detection.rst), [Segmentation](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO15M/HAILO15M_semantic_segmentation.rst), [other tasks](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/PUBLIC_MODELS.rst)
  + For Hailo-10 - [Classification](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO10/HAILO10_classification.rst), [Object Detection](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO10/HAILO10_object_detection.rst), [Segmentation](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/public_models/HAILO10/HAILO10_semantic_segmentation.rst), [other tasks](https://github.com/hailo-ai/hailo_model_zoo/blob/master/docs/PUBLIC_MODELS.rst)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Network Name** | **mAP** | **HW Accuracy** | **FPS (Batch Size=1)** | **FPS (Batch Size=8)** | **Input Resolution (HxWxC)** | **Params (M)** | **OPS (G)** |
| yolov5m | 42.59 | 41.48 | 156 | 156 | 640x640x3 | 21.78 | 52.17 |
| yolov5m6\_6.1 | 50.68 | 49.31 | 26 | 38 | 1280x1280x3 | 35.70 | 200.04 |
| yolov5m\_6.1 | 44.74 | 43.49 | 78 | 146 | 640x640x3 | 21.17 | 48.96 |
| yolov5m\_wo\_spp | 43.06 | 41.59 | 242 | 242 | 640x640x3 | 22.67 | 52.88 |
| yolov5s | 35.33 | 34.1 | 542 | 542 | 640x640x3 | 7.46 | 17.44 |
| yolov11l | 52.8 | 52.22 | 22 | 42 | 640x640x3 | 25.3 | 87.17 |
| **yolov11m** | **51.1** | **49.89** | **49** | **102** | **640x640x3** | **20.1** | **68.1** |
| yolov11n | 39.0 | 37.49 | 156 | 398 | 640x640x3 | 2.6 | 6.55 |
| yolov11s | 46.3 | 45.12 | 99 | 226 | 640x640x3 | 9.4 | 21.6 |
| yolov11x | 54.1 | 53.04 | 11 | 21 | 640x640x3 | 56.9 | 195.29 |

# Work flow, učení modelu

## Použití Google COLAB

Colaboratory neboli „Colab“ umožňuje psát a spouštět v prohlížeči skripty v Pythonu:

* Není potřeba nic konfigurovat
* Přístup k GPU bez dalších poplatků

Colab dokáže usnadnit práci **studentům**, **datovým vědcům** i **výzkumníkům zabývajícím se umělou inteligencí**. Podívejte se na [úvod do služby Colab](https://www.youtube.com/watch?v=inN8seMm7UI), kde se dozvíte více, nebo se níže hned pusťte do experimentování.

### Úvod do tréninku modelu

Jak trénovat detekci objektů YOLO11 na vlastní datové sadě

YOLO11 staví na vylepšeních představených v YOLOv9 a YOLOv10 na začátku tohoto roku, včetně vylepšených architektonických návrhů, vylepšených technik extrakce funkcí a optimalizovaných metod trénování.

YOLO11m dosahuje vyššího průměrného skóre mAP na datové sadě COCO při použití o 22 % méně parametrů než YOLOv8m, díky čemuž je výpočetně jednodušší bez obětování výkonu.

YOLOv11 je k dispozici v 5 různých velikostech, v rozmezí od 2,6 M do 56,9 M parametrů, a je schopen dosáhnout od 39,5 do 54,7 mAP na datové sadě **COCO** (Common Objects in Contex)

### Nastavení a konfigurace

Konfigurace klíčů rozhraní API

Chcete-li doladit YOLO11, musíte poskytnout klíč Roboflow API. Postupujte následovně:

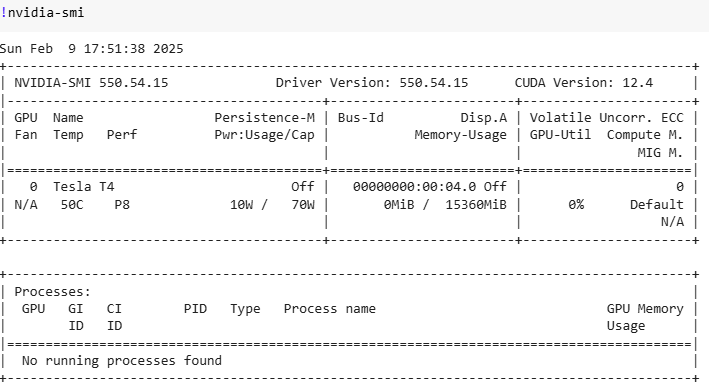
Přejděte na stránku [Nastavení Roboflow](https://www.google.com/url?q=https%3A%2F%2Fapp.roboflow.com%2Fsettings%2Fapi). Klikněte na tlačítko Kopírovat. Tím se váš soukromý klíč umístí do schránky.

### Predispozice

Ujistěme se, že máme přístup k GPU. K tomu můžeme použít příkaz nvidia-smi. V případě jakýchkoli problémů přejděte do Úpravy -> Nastavení poznámkového bloku -> Hardwarový akcelerátor, nastavte ji na GPU a poté klikněte na Uložit.

1. Ověření přítomnosti GPU v projektu

!nvidia-smi



1. Namountujeme Google disk pro jednodušší spravu souborů

from google.colab import drive

drive.mount("/content/drive")

Mounted at /content/drive

### Uložení custom DATASETu do virtuálního stroje v Google colab pro učeni

1. Abychom si usnadnili správu datových sad, obrázků a modelů, vytváříme konstantu HOME.

import os

HOME = os.getcwd()

print(HOME)

/content

1. Zkopirujeme pomocí codu připravený dataset na Roboflow a stáhneme do virtuálního stroje

!mkdir {HOME}/datasets

%cd {HOME}/datasets

from google.colab import userdata

from roboflow import Roboflow

rf = Roboflow(api\_key="secret\_API\_code")

workspace = rf.workspace("dmc-mysnl")

project = rf.workspace("dmc-mysnl").project("dmc")

version = project.version(4)

dataset = version.download("yolov11")

mkdir: cannot create directory ‘/content/datasets’: File exists

/content/datasets

loading Roboflow workspace...

loading Roboflow workspace...

loading Roboflow project...

Downloading Dataset Version Zip in DMC-4 to yolov11:: 100%|██████████| 10377/10377 [00:01<00:00, 5295.13it/s]

Extracting Dataset Version Zip to DMC-4 in yolov11:: 100%|██████████| 842/842 [00:00<00:00, 8716.59it/s]

Obsah složky DATASET musí být uložen vždy ve specifické hierarchii vyžadované modelem YOLOV11. Pokud není dodržená proces přeučení neproběhne.

#### Struktura DATASET, organizace složek:

Obrázky: Uložte všechny obrázky do složky images. V rámci této složky vytvořte podsložky pro trénovací (train), validační (val) a volitelně testovací (test) dataset.

Štítky (anotace): Odpovídající anotace by měly být umístěny ve složce labels, která kopíruje strukturu složky images.

Obrázky by měly být ve formátu JPEG nebo PNG.

Ujistěte se, že každý obrázek má jedinečný název.

#### Struktura složek by měla vypadat takto:

dataset/

├── train/

│ ├── images/

│ │ ├── image1.jpg

│ │ ├── image2.jpg

│ │ ├── ...

│ ├── labels/

│ ├── image1.txt

│ ├── image2.txt

│ ├── ...

│

├── valid/

│ ├── images/

│ │ ├── image1.jpg

│ │ ├── image2.jpg

│ │ ├── ...

│ ├── labels/

│ ├── image1.txt

│ ├── image2.txt

│ ├── ...

│

├── test/

│ ├── images/

│ │ ├── image1.jpg

│ │ ├── image2.jpg

│ │ ├── ...

│ ├── labels/

│ ├── image1.txt

│ ├── image2.txt

│ ├── ...

│

└── data.yaml

**Explanation:**

* **train/** → Contains training images and their corresponding labels.
* **valid/** → Contains validation images and labels.
* **test/** → Contains testing images and labels.

#### Konfigurační soubor Datasetu

v našem případě data1.yaml vypadá takto

train: ../train/images

val: ../valid/images

test: ../test/images

nc: 4

names: ['DMC', 'hand', 'part', 'text']

roboflow:

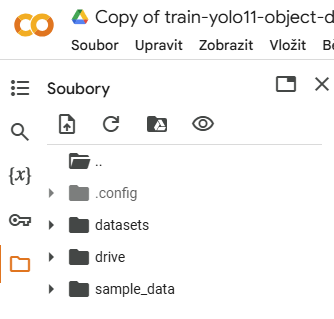
workspace: dmc-mysnl

project: dmc

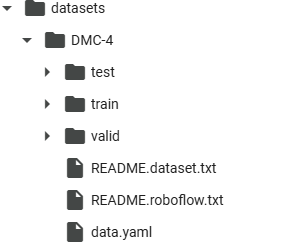
version: 4

license: CC BY 4.0

url: https://universe.roboflow.com/dmc-mysnl/dmc/dataset/4



Vytvořil se adresář „dataset“



Nakopírovány obrazy a štítky pro učení

1. Namountujeme Google disk pro jednodušší spravu souborů

from google.colab import drive

drive.mount("/content/drive")

Mounted at /content/drive

### Vlastní proces učení

1. Abychom si usnadnili správu datových sad, obrázků a modelů, vytváříme konstantu HOME.

import os

HOME = os.getcwd()

print(HOME)

/content

1. Namountujeme Google disk pro jednodušší spravu souborů

from google.colab import drive

drive.mount("/content/drive")

Mounted at /content/drive

1. Install YOLO11 via Ultralytics

%pip install "ultralytics<=8.3.40" supervision roboflow

import ultralytics

ultralytics.checks()

*Ultralytics 8.3.40 🚀 Python-3.11.11 torch-2.5.1+cu124 CUDA:0 (Tesla T4, 15095MiB)Setup complete ✅ (2 CPUs, 12.7 GB RAM, 33.4/112.6 GB disk) ✅*

1. Custom Training model podle DATASETu

!yolo task=detect mode=train model=yolo11s.pt data={dataset.location}/data1.yaml epochs=150 imgsz=640 plots=True

The results of the completed training are saved in {HOME}/runs/detect/train/. Let's examine them.

!ls -R /content/runs/detect/

/content/runs/detect/train3:

args.yaml PR\_curve.png val\_batch0\_pred.jpg

confusion\_matrix\_normalized.png R\_curve.png val\_batch1\_labels.jpg

confusion\_matrix.png results.csv val\_batch1\_pred.jpg

events.out.tfevents.1739124541.f0cfee4982aa.16344.0 results.png val\_batch2\_labels.jpg

F1\_curve.png train\_batch0.jpg val\_batch2\_pred.jpg

labels\_correlogram.jpg train\_batch1.jpg weights

labels.jpg train\_batch2.jpg

P\_curve.png val\_batch0\_labels.jpg

Přeučený model ve formátu PyTorch je k dispozici {HOME}/runs/detect/train7/weights/best.pt

1. Validate fine-tuned model

!yolo task=detect mode=val model={HOME}/runs/detect/train7/weights/best.pt data={dataset.location}/data1.yaml

Ultralytics 8.3.40 🚀 Python-3.11.11 torch-2.5.1+cu124 CUDA:0 (Tesla T4, 15095MiB)

YOLO11s summary (fused): 238 layers, 9,414,348 parameters, 0 gradients, 21.3 GFLOPs

**val:** Scanning /content/datasets/DMC-4/valid/labels.cache... 84 images, 0 backgrounds, 0 corrupt: 100% 84/84 [00:00<?, ?it/s]

WARNING ⚠️ Box and segment counts should be equal, but got len(segments) = 21, len(boxes) = 332. To resolve this only boxes will be used and all segments will be removed. To avoid this please supply either a detect or segment dataset, not a detect-segment mixed dataset.

Class Images Instances Box(P R mAP50 mAP50-95): 100% 6/6 [00:02<00:00, 2.36it/s]

all 84 332 0.906 0.81 0.821 0.722

DMC 84 89 0.913 0.865 0.88 0.827

hand 38 49 0.918 0.755 0.804 0.638

part 79 79 0.914 0.975 0.921 0.91

text 79 115 0.88 0.643 0.678 0.513

Speed: 2.1ms preprocess, 11.6ms inference, 0.0ms loss, 3.4ms postprocess per image

Results saved to **runs/detect/val**

💡 Learn more at <https://docs.ultralytics.com/modes/val>

1. Inference with custom model

!yolo task=detect mode=predict model={HOME}/runs/detect/train7/weights/best.pt conf=0.25 source={dataset.location}/test/images save=True

Ultralytics 8.3.40 🚀 Python-3.11.11 torch-2.5.1+cu124 CUDA:0 (Tesla T4, 15095MiB)

YOLO11s summary (fused): 238 layers, 9,414,348 parameters, 0 gradients, 21.3 GFLOPs

image 1/42 /content/datasets/DMC-4/test/images/111\_jpg.rf.a5c81673f7b1a03e8489ba5c53849322.jpg: 640x640 1 DMC, 2 hands, 1 part, 1 text, 15.6ms

image 2/42 /content/datasets/DMC-4/test/images/125\_jpg.rf.e96a6d2992506b7f085150f050fa7580.jpg: 640x640 1 DMC, 1 hand, 1 part, 1 text, 15.6ms

image 3/42 /content/datasets/DMC-4/test/images/137\_jpg.rf.a8ee9bc1c74e7a461b5cb50fa448daf9.jpg: 640x640 1 DMC, 1 hand, 1 part, 1 text, 15.6ms

image 4/42 /content/datasets/DMC-4/test/images/157\_jpg.rf.60ca921813261375197f1d13fbd33851.jpg: 640x640 1 DMC, 1 hand, 1 part, 1 text, 15.6ms

image 5/42 /content/datasets/DMC-4/test/images/19\_jpg.rf.8fc75d1d687df87f06c80fa31e5795f6.jpg: 640x640 1 DMC, 1 part, 1 text, 15.6ms

.

.

Kráceno, dalších 36 řádku přeskočeno

.

.

image 42/42 /content/datasets/DMC-4/test/images/91\_jpg.rf.6860f31a1e1fbce98eccee7f46c626d5.jpg: 640x640 1 DMC, 2 hands, 1 part, 1 text, 8.4ms

Speed: 1.7ms preprocess, 10.5ms inference, 4.7ms postprocess per image at shape (1, 3, 640, 640)

Results saved to runs/detect/predict3

💡 Learn more at <https://docs.ultralytics.com/modes/predict>

1. Zobrazení vysledků pro revizi posledních 3 výstupů

import glob

import os

from IPython.display import Image as IPyImage, display

latest\_folder = max(glob.glob(f'{HOME}/runs/detect/predict\*/'), key=os.path.getmtime)

for img in glob.glob(f'{latest\_folder}/\*.jpg')[:3]: display(IPyImage(filename=img, width=600))print("\n")

Obsah obrázku interiér

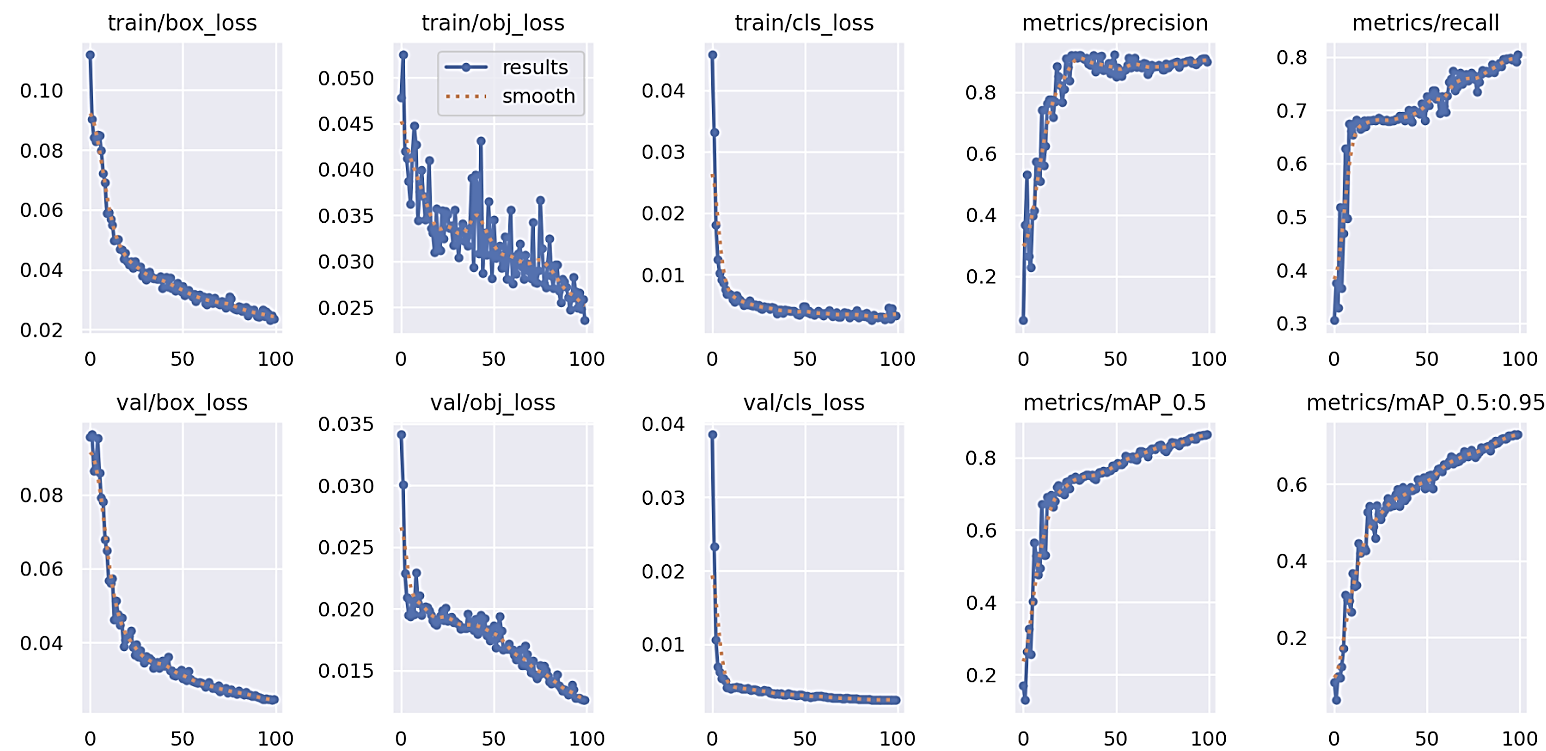
Popis byl vytvořen automaticky

Obsah obrázku interiér, design

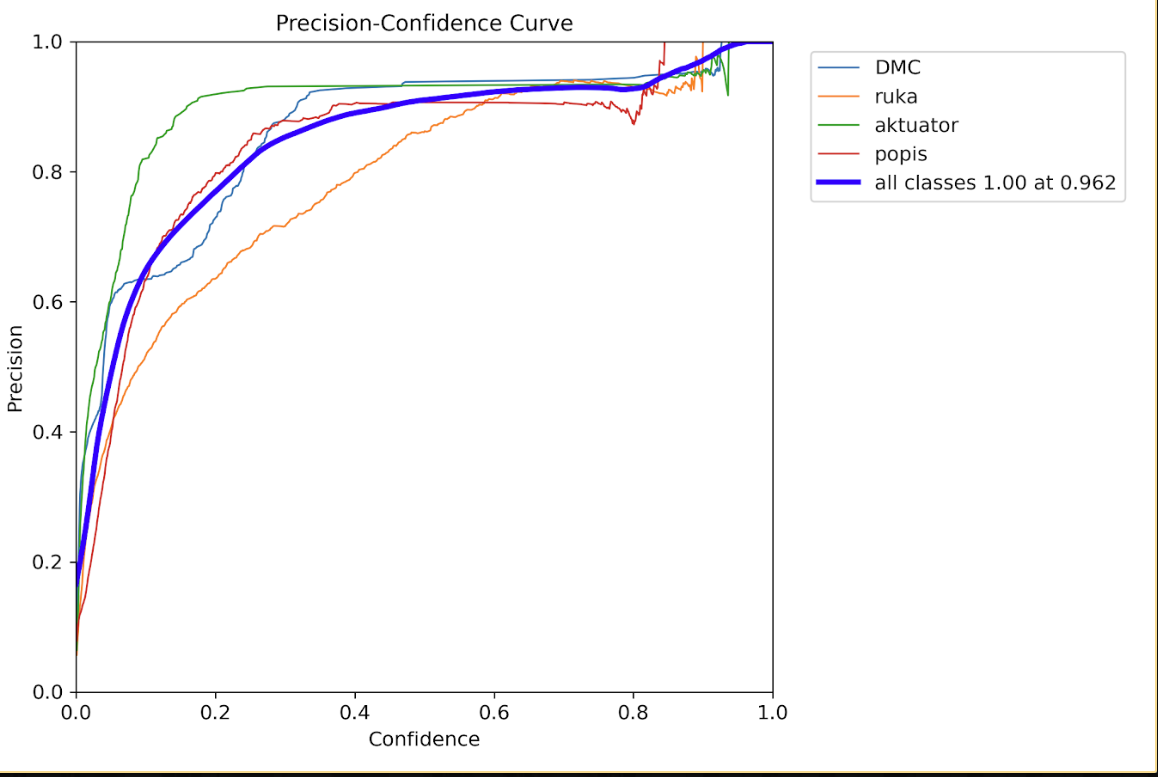
Popis byl vytvořen automaticky se střední mírou spolehlivosti Obsah obrázku interiér

Popis byl vytvořen automaticky

### Posouzení modelu ve na základě dat učení - format PyTorch



| Parametr | Vyhodnocení |
| --- | --- |
|  |  |
| *Box Loss (train/box\_loss)* | Plynule klesá v průběhu epoch, což ukazuje, že model se zlepšuje v lokalizaci ohraničujících boxů.  Stabilizuje se kolem 70. epochy. |
| *Objectness Loss (train/obj\_loss)* | Ukazuje fluktuace, ale obecně má klesající trend a ke konci se stabilizuje.  Naznačuje, že model se efektivně učí identifikovat objekty v obrázcích. |
| *Classification Loss (train/cls\_loss)* | Prudký pokles v prvních epochách, rychle dosahuje nízké stabilní hodnoty.  Naznačuje, že model se efektivně učí klasifikovat třídy objektů. |
| Precision (metrics/precision) | Rychlé zlepšení během prvních epoch, stabilizuje se kolem hodnoty 0,85.  Naznačuje, že model predikuje méně falešně pozitivních případů s postupujícím tréninkem. |
| **Recall (metrics/recall)** | Zlepšuje se pomaleji než precision, ale dosahuje ~0,8 ke konci tréninku.  Naznačuje, že model identifikuje vyšší podíl skutečných pozitivních případů. |
| **mAP při IoU 0.5 (metrics/mAP\_0.5)** | Stabilní zlepšení až na hodnotu ~0,9, což ukazuje výborný výkon při správné detekci ohraničujících boxů s prahovou hodnotou IoU 0.5. |
| ***mAP při IoU 0.5:0.95 (metrics/mAP\_0.5:0.95)*** | Postupné zlepšení na ~0,6, což je běžné při přísnějších prahových hodnotách IoU.  Ukazuje, že model si vede dobře i při vyšších nárocích na přesnost. |



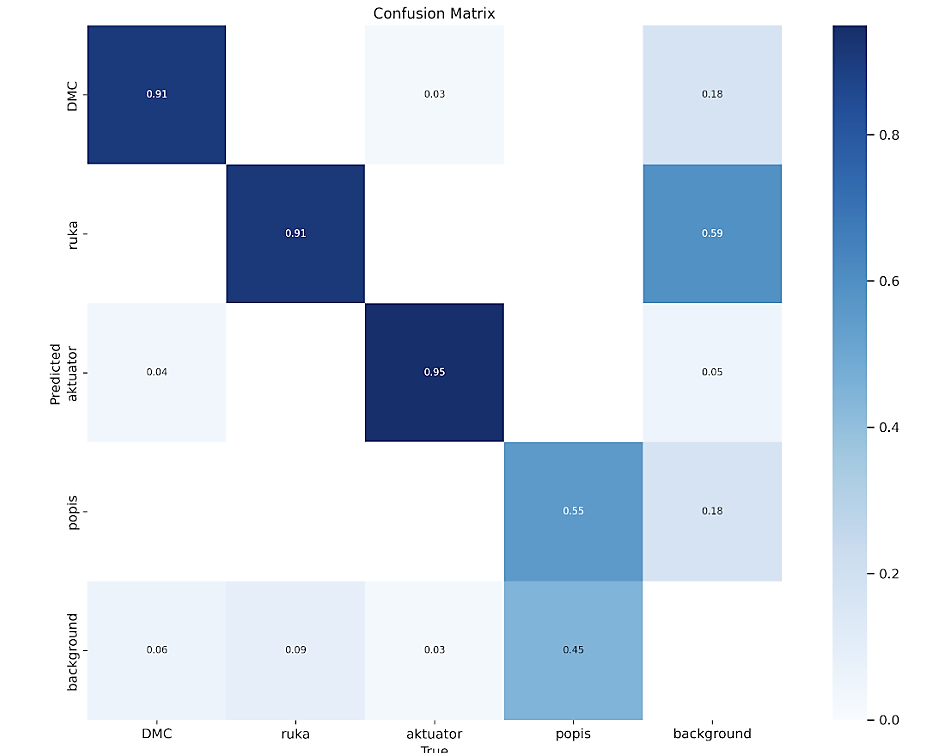
Třída "Ruka":

* Významně zaostává za ostatními třídami v přesnosti.
* Model pravděpodobně generuje více falešně pozitivních případů pro tuto třídu.

Optimalizace prahové hodnoty (Confidence Threshold):

* Optimální prahová hodnota pro maximální kombinovanou přesnost je 0.962, což minimalizuje falešně pozitivní výsledky napříč všemi třídami.
* **Výborný výkon u tříd "DMC" a "Aktuator":**

**Tyto třídy dosahují vysoké přesnosti i při nižších prahových hodnotách.**



1. Třída DMC:

**Správně klasifikováno: 91 %.**

Chybně klasifikováno jako "background": 18 %.

Velmi dobrá přesnost, ale záměny s třídou "background" naznačují, že některé vzorky mají rysy podobné pozadí.

1. Třída Ruka:

Správně klasifikováno: 91 %.

Chybně klasifikováno jako "popis": 59 %.

Významná část vzorků třídy "ruka" je zaměňována za "popis". Model může mít potíže odlišit tyto dvě třídy.

1. Třída Aktuator:

Správně klasifikováno: 95 %.

Minimální chyby (např. chybně jako "popis" nebo "background").

Model má nejvyšší výkon u této třídy.

1. Třída Popis:

Správně klasifikováno: 55 %.

Chybně klasifikováno jako "background": 45 %.

Třída "popis" má výrazně nižší přesnost a je často zaměňována s "backgroundem".

1. Třída Background:

Správně klasifikováno: 45 %.

Chybně klasifikováno jako "popis" nebo "ruka".

Zaměny naznačují, že pozadí obsahuje rysy podobné jiným třídám.

# Postup učení a konverze modelu

## Převod modelu PyTorch --> ONNX

Nejprve musíme převést náš soubor PyTorch (.pt) do formátu ONNX, aby byl připraven k překladu pro náš hardwarový akcelerátor Raspberry Pi AI Hat+. Toto zařízení obsahuje modul s výkonem 13/26 TOPS, který lze kombinovat s Raspberry Pi 5.

!pip install ultralytics

!pip install onnx

!pip install onnxruntime

### Importování předtrénovaného modelu YOLO11s

Převod a export našeho souboru PyTorch do formátu ONNX.

import torch

from ultralytics import YOLO

import cv2

import numpy as np

from IPython.display import display, Image

import os

#Načtení předtrénovaného modelu

model = YOLO('yolo11s.pt')

#Načtení modelu do prostředí

checkpoint = torch.load('/content/yolo11s.pt')

model = checkpoint['model']

model = model.float()

model.eval()

#Vytvoření testovacího vstupu ve formátu FP32

dummy\_input = torch.randn(16, 3, 640, 640, dtype=torch.float)

#Export do formátu ONNX

torch.onnx.export(

model,

dummy\_input,

"/content/best.onnx",

export\_params=True,

opset\_version=11, # Upravte verzi opset podle potřeby

do\_constant\_folding=True,

input\_names=['input'],

output\_names=['output']

)

print("Model ONNX byl úspěšně exportován!")

Creating new Ultralytics Settings v0.0.6 file ✅

View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'

Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs\_dir=path/to/dir'. For help see https://docs.ultralytics.com/quickstart/#ultralytics-settings.

Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11s.pt to 'yolo11s.pt'...

100%|██████████| 18.4M/18.4M [00:00<00:00, 146MB/s]

/usr/local/lib/python3.11/dist-packages/ultralytics/nn/modules/head.py:105: TracerWarning: Converting a tensor to a Python boolean might cause the trace to be incorrect. We can't record the data flow of Python values, so this value will be treated as a constant in the future. This means that the trace might not generalize to other inputs!

if self.format != "imx" and (self.dynamic or self.shape != shape):

/usr/local/lib/python3.11/dist-packages/ultralytics/utils/tal.py:338: TracerWarning: Iterating over a tensor might cause the trace to be incorrect. Passing a tensor of different shape won't change the number of iterations executed (and might lead to errors or silently give incorrect results).

for i, stride in enumerate(strides):

ONNX model exported successfully!

### Ověření platnosti našeho modelu, abychom zajistili, že převod proběhl správně

import onnx

import onnxruntime as ort

import torch

#Načtení modelu ONNX

onnx\_model = onnx.load("/content/best.onnx")

onnx.checker.check\_model(onnx\_model)

print("ONNX model is valid!")

#Testování modelu ONNX pomocí ONNX Runtime

dummy\_input = torch.randn(16, 3, 640, 640).numpy()

ort\_session = ort.InferenceSession("/content/best.onnx")

outputs = ort\_session.run(None, {"input": dummy\_input})

print(outputs[0])

ONNX model is valid!

[[[ 12.095 12.486 23.357 ... 521.49 565.71 596.38]

[ 4.091 3.6364 4.2469 ... 604.15 600.88 599.14]

[ 23.631 23.393 44.825 ... 291.76 263.39 197.45]

...

[ 9.8348e-07 3.8743e-07 4.1723e-07 ... 2.7716e-06 2.8014e-06 2.9504e-06]

[ 3.5763e-07 3.5763e-07 3.2783e-07 ... 2.2054e-06 2.1458e-06 2.116e-06]

[ 5.0664e-07 3.8743e-07 3.8743e-07 ... 2.1458e-06 1.9073e-06 1.967e-06]]

[[ 20.045 21.065 25.373 ... 515.73 562.16 591.04]

[ 5.3167 4.5354 4.7803 ... 602.77 598.5 596.43]

[ 40.215 42.004 49.516 ... 301.65 280.25 206.51]

...

[ 5.3644e-07 2.9802e-07 2.9802e-07 ... 2.8014e-06 2.861e-06 3.01e-06]

[ 2.9802e-07 2.3842e-07 2.3842e-07 ... 2.2054e-06 2.116e-06 2.1756e-06]

[ 2.9802e-07 2.6822e-07 1.7881e-07 ... 2.0862e-06 1.8179e-06 1.9073e-06]]

[[ 6.8863 9.8793 35.349 ... 518.88 563.27 594.17]

[ 5.115 4.5443 5.8287 ... 600.71 599.92 601.59]

[ 13.031 19.128 69.464 ... 297.26 283.63 215.4]

...

[ 1.4007e-06 5.0664e-07 5.6624e-07 ... 2.6524e-06 2.5034e-06 2.8312e-06]

[ 5.9605e-07 4.1723e-07 3.5763e-07 ... 2.0862e-06 1.9968e-06 1.9968e-06]

[ 1.1027e-06 5.6624e-07 5.0664e-07 ... 1.9968e-06 1.7285e-06 1.8775e-06]]

...

[[ 5.1775 25.394 33.051 ... 527.99 574.08 596.84]

[ 4.0751 4.1466 4.7932 ... 605.15 603.52 603.76]

[ 9.9311 50.498 64.379 ... 297.41 286.03 218.02]

...

[ 6.2585e-07 2.0862e-07 2.3842e-07 ... 2.6524e-06 2.563e-06 2.7418e-06]

[ 2.6822e-07 2.0862e-07 2.3842e-07 ... 2.1458e-06 2.0862e-06 1.967e-06]

[ 3.2783e-07 1.1921e-07 1.4901e-07 ... 1.967e-06 1.7583e-06 1.8477e-06]]

[[ 10.498 20.855 29.133 ... 523.53 564.76 593.45]

[ 4.1542 3.8135 4.1099 ... 601.97 600.65 601.08]

[ 20.394 40.316 56.35 ... 295.12 283.26 212.55]

...

[ 1.1027e-06 5.3644e-07 5.6624e-07 ... 2.8312e-06 2.7716e-06 2.9504e-06]

[ 3.5763e-07 3.5763e-07 4.7684e-07 ... 2.2352e-06 2.0862e-06 2.116e-06]

[ 5.6624e-07 2.6822e-07 2.0862e-07 ... 2.0862e-06 1.7881e-06 1.8775e-06]]

[[ 6.2057 17.939 29.649 ... 519.61 561.74 587.09]

[ 4.6514 3.6254 4.2958 ... 601.04 597.93 597.04]

[ 11.811 34.234 57.269 ... 297.08 295.39 222.99]

...

[ 8.6427e-07 3.8743e-07 2.9802e-07 ... 2.6226e-06 2.6226e-06 2.8014e-06]

[ 2.6822e-07 2.3842e-07 2.0862e-07 ... 2.0266e-06 1.9372e-06 1.967e-06]

[ 4.7684e-07 2.3842e-07 1.4901e-07 ... 1.8477e-06 1.6093e-06 1.7881e-06]]]

## Pou6it9 Dataflow Compileru – Hailo -

Tento nástroj slouží k převodu modelu ONNX do jeho finálního formátu, který lze použít na Hailo 8.

Existují tři způsoby, jak to provést:

1. Použití Dataflow Compileru (DFC)

2. Použití Python integrace Model Zoo

3. Použití AI SW Docker Suite

V nasem p59pade použijeme Dataflow Compiler (DFC), který umožňuje převádět modely, které nejsou předkompilované v Model Zoo, a také přeškolit či převést vlastní modely s architekturami podobnými těm v Model Zoo.

Abychom jej mohli použít, musíme náš ONNX soubor převést do formátu HEF (Hailo Executable File), který je kompatibilní s Hailo 8.

Postup:

1. Analýza modelu

2. Optimalizace

3. Kompilace

4. Spuštění inference

5. Použití více modelů současně (Dokumentace Hailo)

### Instalace

Nejprve nainstalujeme všechny potřebné balíčky:

!sudo apt-get update

!sudo apt-get install -y python3-dev python3-distutils python3-tk libfuse2 graphviz libgraphviz-dev

Instalace balíčku virtualenv

Nejprve aktualizujeme pip a nainstalujeme virtualenv:

!pip install --upgrade pip virtualenv

Vytvoření virtuálního prostředí (venv)

!pip install virtualenv

!virtualenv my\_env --python=python3.10

Při instalaci Dataflow Compileru (DFC) do našeho virtuálního prostředí (venv) je důležité použít správnou verzi Pythonu uvnitř tohoto prostředí.

!my\_env/bin/pip install numpy==1.23.3

Stažení a instalace Hailo Dataflow Compileru

Instalace gdown pro stahování souborů

!pip install gdown

Stažení souborů .whl

ID souboru pro Hailo Dataflow Compiler (verze 3.30.0):

hailo\_dataflow\_id = "1CZ\_ApOClYLGw57F7ps-rsDMnq1IcD0s2"

Stažení .whl souboru:

!gdown --id {hailo\_dataflow\_id} -O hailo\_dataflow\_compiler-3.30.0-py3-none-linux\_x86\_64.whl

Instalace Hailo Dataflow Compileru do venv

!my\_env/bin/pip install /content/hailo\_dataflow\_compiler-3.30.0-py3-none-linux\_x86\_64.whl

Ověření instalace

!pip list | grep hailo

### Ověření správné instalace

Po instalaci Hailo Dataflow Compileru je důležité ověřit, zda je vše správně nastaveno.

Spusťte následující příkaz pro kontrolu verze:

!my\_env/bin/hailo --version

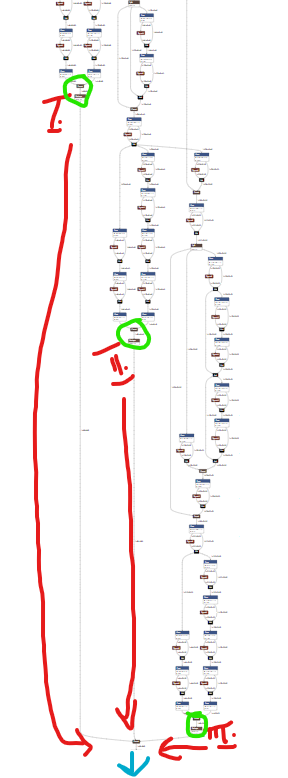
Pokud instalace proběhla úspěšně, zobrazí se verze nainstalovaného Hailo Dataflow Compileru. Pokud se zobrazí chyba, zkontrolujte, zda byl balíček správně stažen a nainstalován do virtuálního prostředí (venv). 2a. Analýza našeho ONNX souboru

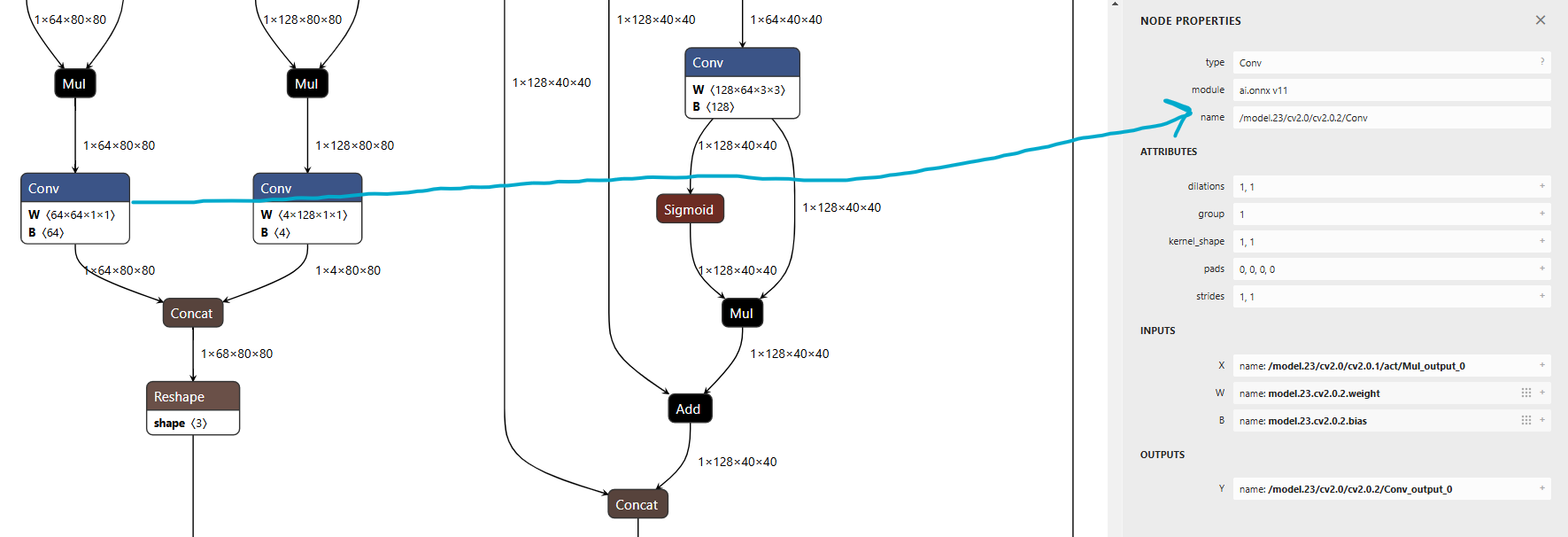
## ONNX model - Identifikace koncových uzlů v siti

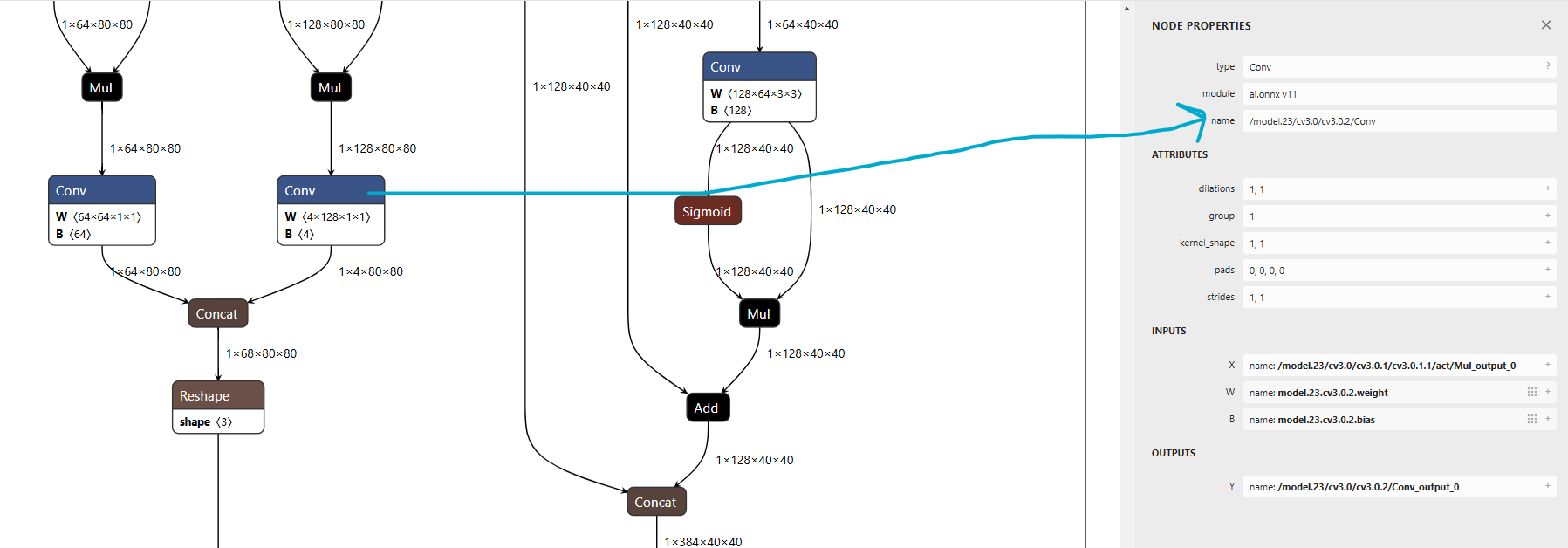
Prvním krokem analýzy je sdělit nástroji, ze kterých vrstev/uzlů má očekávat výstup, tedy takzvané koncové uzly. Jakmile správně identifikujeme koncové uzly pomocí Netron, můžeme pokračovat k první fázi kompilace – analýze.

V našem případě používají všechny verze od YoloV8 po YoloV11 stejnou architekturu, kde pro každý výstup mapy prvků existují dva koncové uzly. TO UNAMENA CELKEM 6 UZLU.

Koncové uzly modelu Yolo lze identifikovat jako uzly těsně před operacemi postprocessingu v dolní části modelu, jak je znázorněno níže:







Jakmile máme identifikované koncové uzly (můžete na ně kliknout v Netron a zjistit jejich názvy – použijte název uzlu), napíšeme jejich output jmena

/model.23/cv2.0/cv2.0.2/Conv",

/model.23/cv3.0/cv3.0.2/Conv",

/model.23/cv2.1/cv2.1.2/Conv",

/model.23/cv3.1/cv3.1.2/Conv",

/model.23/cv2.2/cv2.2.2/Conv",

/model.23/cv3.2/cv3.2.2/Conv"

Jakmile máme identifikované koncové uzly, napíšeme náš skript a uložíme jej lokálně. V google colab vztvo59 python soubor a uovrau9meho pomoci funkce code content apote uložíme. Kod je snaysi pro cteni když jsou uobreueny i python soubory.

## Převod modelu ONNX --> HAR

code\_content = '''

from hailo\_sdk\_client import ClientRunner

import onnx

import os

from typing import List, Dict, Optional

class ModelConverter:

def \_\_init\_\_(self,

onnx\_path: str,

model\_name: str = "yolov11\_model",

hw\_arch: str = "hailo8",

batch\_size: int = 16):

self.onnx\_path = onnx\_path

self.model\_name = model\_name

self.hw\_arch = hw\_arch

self.batch\_size = batch\_size

self.end\_node\_names = [

"/model.23/cv2.0/cv2.0.2/Conv",

"/model.23/cv3.0/cv3.0.2/Conv",

"/model.23/cv2.1/cv2.1.2/Conv",

"/model.23/cv3.1/cv3.1.2/Conv",

"/model.23/cv2.2/cv2.2.2/Conv",

"/model.23/cv3.2/cv3.2.2/Conv"

]

def check\_opset\_version(self, model) -> bool:

for opset in model.opset\_import:

if opset.domain == "" and opset.version > 17:

print(f"⚠️ Advertencia: El modelo usa Opset {opset.version}")

return False

return True

def convert\_ir\_version(self, model) -> str:

"""Convierte la versión IR del modelo si es necesario."""

if model.ir\_version > 8:

print(f"📝 Convirtiendo modelo de versión IR {model.ir\_version} a versión 8")

model.ir\_version = 8

converted\_path = self.onnx\_path.replace('.onnx', '\_converted.onnx')

onnx.save(model, converted\_path)

return converted\_path

return self.onnx\_path

def verify\_end\_nodes(self, model) -> None:

"""Verifica la existencia de nodos finales."""

available\_nodes = {node.name for node in model.graph.node}

missing = [node for node in self.end\_node\_names if node not in available\_nodes]

if missing:

raise ValueError(f"❌ Nodos finales faltantes:\n" + "\n".join(missing))

def convert(self) -> str:

"""Realiza la conversión completa del modelo."""

if not os.path.exists(self.onnx\_path):

raise FileNotFoundError(f"❌ No se encontró el archivo ONNX: {self.onnx\_path}")

try:

# Carga y validación inicial

print("🔄 Cargando modelo ONNX...")

model = onnx.load(self.onnx\_path)

# Verificaciones

self.check\_opset\_version(model)

self.verify\_end\_nodes(model)

# Conversión IR si es necesario

self.onnx\_path = self.convert\_ir\_version(model)

# Configuración de entrada

input\_node = model.graph.input[0]

input\_node\_name = input\_node.name

input\_shapes = [dim.dim\_value for dim in input\_node.type.tensor\_type.shape.dim]

print(f"📊 Dimensiones de entrada: {input\_shapes}")

# Configuración del runner

runner = ClientRunner(hw\_arch=self.hw\_arch)

net\_input\_shapes = {

input\_node\_name: [self.batch\_size, 3, 640, 640]

}

# Traducción del modelo

print("🚀 Iniciando traducción del modelo...")

hn, npz = runner.translate\_onnx\_model(

self.onnx\_path,

self.model\_name,

end\_node\_names=self.end\_node\_names,

net\_input\_shapes=net\_input\_shapes

)

# Guardado del modelo

har\_path = f"{self.model\_name}\_hailo\_model.har"

runner.save\_har(har\_path)

print(f"✅ Modelo HAR guardado como: {har\_path}")

return har\_path

except Exception as e:

print(f"❌ Error durante la conversión: {str(e)}")

raise

if \_\_name\_\_ == "\_\_main\_\_":

try:

converter = ModelConverter("/content/best.onnx")

har\_file = converter.convert()

print(f"✨ Proceso completado exitosamente. Archivo HAR: {har\_file}")

except Exception as e:

print(f"❌ Error en el proceso principal: {str(e)}")

'''

with open("translate\_model.py", "w") as f:

f.write(code\_content)

Spuštění skriptu v prostředí venv

!my\_env/bin/python translate\_model.py

🔄 Cargando modelo ONNX...

📊 Dimensiones de entrada: [16, 3, 640, 640]

🚀 Iniciando traducción del modelo...

[info] Translation started on ONNX model yolov11\_model

[info] Restored ONNX model yolov11\_model (completion time: 00:00:00.24)

[info] Extracted ONNXRuntime meta-data for Hailo model (completion time: 00:00:02.49)

[info] NMS structure of yolov8 (or equivalent architecture) was detected.

[info] In order to use HailoRT post-processing capabilities, these end node names should be used: /model.23/cv2.0/cv2.0.2/Conv /model.23/cv3.0/cv3.0.2/Conv /model.23/cv3.1/cv3.1.2/Conv /model.23/cv2.1/cv2.1.2/Conv /model.23/cv3.2/cv3.2.2/Conv /model.23/cv2.2/cv2.2.2/Conv.

[info] Start nodes mapped from original model: 'input': 'yolov11\_model/input\_layer1'.

[info] End nodes mapped from original model: '/model.23/cv2.0/cv2.0.2/Conv', '/model.23/cv3.0/cv3.0.2/Conv', '/model.23/cv2.1/cv2.1.2/Conv', '/model.23/cv3.1/cv3.1.2/Conv', '/model.23/cv2.2/cv2.2.2/Conv', '/model.23/cv3.2/cv3.2.2/Conv'.

[info] Translation completed on ONNX model yolov11\_model (completion time: 00:00:38.21)

[info] Saved HAR to: /content/yolov11\_model\_hailo\_model.har

✅ Modelo HAR guardado como: yolov11\_model\_hailo\_model.har

✨ Proceso completado exitosamente. Archivo HAR: yolov11\_model\_hailo\_model.har

Tento skript načetl ONNX model, ověřil správnost koncových uzlů, případně provedl potřebné úpravy a přeložil model do HAR formátu pro Hailo hardware. 🎯

Nyní vygenerujeme kalibrační data k optimalizaci našeho modelu, což pomáhá DFC optimalizovat síťový model a jeho parametry pomocí předzpracované kalibrační sady. Tento proces je nutné provést pouze jednou.

## Kontrola HAR souboru

kontrola souboru modelu HAR

with open("inspect\_dict.py", "w") as f:

f.write("""

from hailo\_sdk\_client import ClientRunner

# Load the HAR file

har\_path = "/content/yolov11\_model\_hailo\_model.har"

runner = ClientRunner(har=har\_path)

from pprint import pprint

try:

# Access the HailoNet as an OrderedDict

hn\_dict = runner.get\_hn() # Or use runner.\_hn if get\_hn() is unavailable

print("Inspecting layers from HailoNet (OrderedDict):")

# Pretty-print each layer

for key, value in hn\_dict.items():

print(f"Key: {key}")

pprint(value)

print("\n" + "="\*80 + "\n") # Add a separator between layers for clarity

except Exception as e:

print(f"Error while inspecting hn\_dict: {e}")

""")

Spu3t2n9 kontroly souboru HAR

!my\_env/bin/python inspect\_dict.py

Inspecting layers from HailoNet (OrderedDict):

Key: name

'yolov11\_model' --- KRACENO --

## Optimalizace HAR souboru

Následně generujeme kalibrační data pro optimalizaci našeho modelu, což pomáhá DFC optimalizovat model sítě a parametry pomocí předzpracované kalibrační sady. Tento proces by měl být proveden pouze jednou.

Tento soubor obsahuje obrázky ve formátu JPG pro trénování,

Importování potřebných knihoven

import os # Pro operace se souborovým systémem

import shutil # Pro pokročilé operace se soubory

from pathlib import Path # Pro moderní správu cest k souborům

# Nastavení stahování z Google Drive

file\_id = "1RJzzDzpL1p1kneOUTDrIczPokJUj48xe" # Unikátní ID souboru v Google Drive

destination = "/content/dataset.tar.gz" # Cesta pro uložení archivu

# Instalace a spuštění gdown pro stažení z Google Drive

!pip install gdown

!gdown --id {file\_id}

# Nastavení adresáře pro extrakci

output\_dir = "/content"

Path(output\_dir).mkdir(parents=True, exist\_ok=True) # Vytvoření složky, pokud neexistuje

# Rozbalení archivu

!tar -xzf /content/dataset.tar.gz -C /content/img

!tar -xzf /content/dataset.tar.gz -C /content

Requirement already satisfied: gdown in /usr/local/lib/python3.11/dist-packages (5.2.0)

Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.11/dist-packages (from gdown) (4.13.3)

Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from gdown) (3.17.0)

Requirement already satisfied: requests[socks] in /usr/local/lib/python3.11/dist-packages (from gdown) (2.32.3)

Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from gdown) (4.67.1)

Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4->gdown) (2.6)

Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4->gdown) (4.12.2)

Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests[socks]->gdown) (3.4.1)

Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests[socks]->gdown) (3.10)

Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests[socks]->gdown) (2.3.0)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests[socks]->gdown) (2025.1.31)

Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.11/dist-packages (from requests[socks]->gdown) (1.7.1)

/usr/local/lib/python3.11/dist-packages/gdown/\_\_main\_\_.py:140: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will be removed in 5.0. You don't need to pass it anymore to use a file ID.

warnings.warn(

Downloading...

From (original): https://drive.google.com/uc?id=1RJzzDzpL1p1kneOUTDrIczPokJUj48xe

From (redirected): https://drive.google.com/uc?id=1RJzzDzpL1p1kneOUTDrIczPokJUj48xe&confirm=t&uuid=b70d36e1-3246-43b3-9cde-fc4277bc030d

To: /content/dataset.tar.gz

100% 29.2M/29.2M [00:00<00:00, 57.9MB/s]

tar: /content/img: Cannot open: No such file or directory

tar: Error is not recoverable: exiting now

# Extracción del archivo RAR

!tar -xzf /content/dataset.tar.gz -C /content # 'x' extrae con rutas completas

### Generování kalibračních dat

import numpy as np

from PIL import Image

import os

from google.colab import drive

# Cesty k adresářům a souborům

image\_dir = '/content/img'

output\_dir = '/content'

os.makedirs(output\_dir, exist\_ok=True) # Vytvoření adresáře, pokud neexistuje

# Cesty pro uložení kalibračních dat

calibration\_data\_path = os.path.join(output\_dir, "calibration\_data.npy")

processed\_data\_path = os.path.join(output\_dir, "processed\_calibration\_data.npy")

# Inicializace seznamu pro kalibrační data

calib\_data = []

# Zpracování všech obrázků v adresáři

for img\_name in os.listdir(image\_dir):

img\_path = os.path.join(image\_dir, img\_name)

if img\_name.lower().endswith(('.jpg', '.jpeg', '.png')):

try:

img = Image.open(img\_path).resize((640, 640)) # Změna velikosti

img\_array = np.array(img) / 255.0 # Normalizace na rozsah [0,1]

# Kontrola, zda má obrázek správný tvar

if img\_array.shape == (640, 640, 3):

calib\_data.append(img\_array)

else:

print(f"Přeskakuji obrázek s nesprávným rozměrem: {img\_path}, shape: {img\_array.shape}")

except AttributeError:

print(f"Přeskakuji poškozený obrázek: {img\_path}")

continue

# Konverze seznamu na NumPy pole

calib\_data = np.array(calib\_data)

# Uložení normalizovaných dat

np.save(calibration\_data\_path, calib\_data)

print(f"Normalizovaná kalibrační sada uložena s tvarem: {calib\_data.shape} do {calibration\_data\_path}")

# Převod dat zpět na rozsah [0,255]

processed\_calibration\_data = calib\_data \* 255.0

# Uložení upravených kalibračních dat

np.save(processed\_data\_path, processed\_calibration\_data)

print(f"Zpracovaná kalibrační sada uložena s tvarem: {processed\_calibration\_data.shape} do {processed\_data\_path}")

Výstup:

Normalizovaná kalibrační sada uložena s tvarem: (471, 640, 640, 3) do /content/calibration\_data.npy

Zpracovaná kalibrační sada uložena s tvarem: (471, 640, 640, 3) do /content/processed\_calibration\_data.npy

Normalized calibration dataset saved with shape: (471, 640, 640, 3) to /content/calibration\_data.npy

Processed calibration dataset saved with shape: (471, 640, 640, 3) to /content/processed\_calibration\_data.npy

### Vylepšený kód s efektivním zpracováním po dávkách

import numpy as np

from PIL import Image

import os

from google.colab import drive

import gc

from tqdm import tqdm

class CalibrationDataProcessor:

def \_\_init\_\_(self,

image\_dir: str = '/content/img',

output\_dir: str = '/content',

batch\_size: int = 16,

target\_size: tuple = (640, 640)):

self.image\_dir = image\_dir

self.output\_dir = output\_dir

self.batch\_size = batch\_size

self.target\_size = target\_size

# Vytvoření výstupního adresáře

os.makedirs(output\_dir, exist\_ok=True)

# Cesty k výstupním souborům

self.calibration\_data\_path = os.path.join(output\_dir, "calibration\_data.npy")

self.processed\_data\_path = os.path.join(output\_dir, "processed\_calibration\_data.npy")

def get\_valid\_image\_files(self):

"""Získá seznam platných obrázkových souborů."""

return [f for f in os.listdir(self.image\_dir)

if f.lower().endswith(('.jpg', '.jpeg', '.png'))]

def process\_image(self, img\_path: str) -> np.ndarray:

"""Zpracuje jednotlivý obrázek."""

try:

img = Image.open(img\_path).resize(self.target\_size)

img\_array = np.array(img) / 255.0

if img\_array.shape == (\*self.target\_size, 3):

return img\_array

else:

print(f"Obrázek s nesprávnými rozměry: {img\_path}, shape: {img\_array.shape}")

return None

except Exception as e:

print(f"Chyba při zpracování obrázku {img\_path}: {str(e)}")

return None

def process\_batch(self, image\_files: list, start\_idx: int) -> np.ndarray:

"""Zpracuje dávku obrázků."""

batch\_data = []

for img\_name in image\_files:

img\_path = os.path.join(self.image\_dir, img\_name)

img\_array = self.process\_image(img\_path)

if img\_array is not None:

batch\_data.append(img\_array)

return np.array(batch\_data) if batch\_data else None

def save\_batch(self, batch\_data: np.ndarray, file\_path: str, mode: str = 'wb'):

"""Uloží dávku dat."""

if mode == 'wb':

np.save(file\_path, batch\_data)

else:

with open(file\_path, 'ab') as f:

np.save(f, batch\_data)

def process\_dataset(self):

"""Zpracuje celý dataset po dávkách."""

image\_files = self.get\_valid\_image\_files()

total\_images = len(image\_files)

if total\_images == 0:

print("Nebyly nalezeny žádné platné obrázky.")

return

# Inicializace souborů

if os.path.exists(self.calibration\_data\_path):

os.remove(self.calibration\_data\_path)

if os.path.exists(self.processed\_data\_path):

os.remove(self.processed\_data\_path)

total\_processed = 0

# Zpracování po dávkách

for batch\_start in tqdm(range(0, total\_images, self.batch\_size),

desc="Zpracovávají se obrázky"):

batch\_end = min(batch\_start + self.batch\_size, total\_images)

batch\_files = image\_files[batch\_start:batch\_end]

# Zpracování dávky

batch\_data = self.process\_batch(batch\_files, batch\_start)

if batch\_data is not None:

# Uložení normalizovaných dat

self.save\_batch(batch\_data, self.calibration\_data\_path, 'ab' if batch\_start > 0 else 'wb')

# Uložení zpracovaných dat (škálovaných na 255)

self.save\_batch(batch\_data \* 255.0, self.processed\_data\_path, 'ab' if batch\_start > 0 else 'wb')

total\_processed += len(batch\_data)

# Uvolnění paměti

del batch\_data

gc.collect()

print(f"\nProces dokončen. Celkem zpracovaných obrázků: {total\_processed}")

if \_\_name\_\_ == "\_\_main\_\_":

try:

processor = CalibrationDataProcessor(batch\_size=16)

processor.process\_dataset()

except Exception as e:

print(f"Chyba během zpracování: {str(e)}")

Procesando imágenes: 100%|██████████| 30/30 [00:46<00:00, 1.54s/it]

Proceso completado. Total de imágenes procesadas: 471

Creación de un nivel de gestión de servicios para ajustarse correctamente a nuestra arquitectura de modelo modificada.

### Vytvoření JSON souboru pro řízení modelu

Vytvoření vrstvy pro správu služeb tak, aby správně odpovídala naší upravené modelové architektuře.

import json

import os

from google.colab import drive

drive.mount('/content/drive/', force\_remount=True)

# Updated NMS layer configuration dictionary

nms\_layer\_config = {

    "nms\_scores\_th": 0.2,

    "nms\_iou\_th": 0.7,

    "image\_dims": [

        640,

        640

    ],

    "max\_proposals\_per\_class": 100,

    "classes": 80,

    "regression\_length": 16,

    "background\_removal": False,

    "background\_removal\_index": 0,

    "bbox\_decoders": [

        {

            "name": "bbox\_decoder1",

            "stride": 8,

            "reg\_layer": "conv51",

            "cls\_layer": "conv54"

        },

        {

            "name": "bbox\_decoder2",

            "stride": 16,

            "reg\_layer": "conv62",

            "cls\_layer": "conv65"

        },

        {

            "name": "bbox\_decoder3",

            "stride": 32,

            "reg\_layer": "conv77",

            "cls\_layer": "conv80"

        }

    ]

}

# Path to save the updated JSON configuration

output\_dir = "/content"

os.makedirs(output\_dir, exist\_ok=True)  # Create the directory if it doesn't exist

output\_path = os.path.join(output\_dir, "nms\_layer\_config.json")

# Save the updated configuration as a JSON file

with open(output\_path, "w") as json\_file:

    json.dump(nms\_layer\_config, json\_file, indent=4)

print(f"NMS layer configuration saved to {output\_path}")

Mounted at /content/drive/

NMS layer configuration saved to /content/nms\_layer\_config.json

Soubor bude použit dále

## Optimalizace modelu do HEF souboru

with open("optimize\_model.py", "w") as f:

f.write("""\

import os

from hailo\_sdk\_client import ClientRunner

model\_name = "yolov11\_model"

hailo\_model\_har\_name = f"{model\_name}\_hailo\_model.har"

assert os.path.isfile(hailo\_model\_har\_name), "Please provide a valid path for the HAR file"

runner = ClientRunner(har=hailo\_model\_har\_name)

# Sugerencia del último post Trieut415... https://community.hailo.ai/t/guide-to-using-the-dfc-to-convert-a-modified-yolov11-on-google-colab/7131/14?u=ahmedcr\_net

alls = \"\"\"

normalization1 = normalization([0.0, 0.0, 0.0], [255.0, 255.0, 255.0])

change\_output\_activation(conv54, sigmoid)

change\_output\_activation(conv65, sigmoid)

change\_output\_activation(conv80, sigmoid)

nms\_postprocess("/content/nms\_layer\_config.json", meta\_arch=yolov8, engine=cpu)

performance\_param(compiler\_optimization\_level=2)

\"\"\"

runner.load\_model\_script(alls)

calib\_dataset = "/content/processed\_calibration\_data.npy"

assert os.path.isfile(calib\_dataset), "Please provide a valid path for the calibration dataset"

runner.optimize(calib\_dataset)

quantized\_model\_har\_path = f"{model\_name}\_quantized\_model.har"

runner.save\_har(quantized\_model\_har\_path)

print(f"Quantized HAR file saved to: {quantized\_model\_har\_path}")

""")

Spouští se pomocí venv Pythonu.

!my\_env/bin/pip uninstall -y numpy scipy

!my\_env/bin/pip install numpy==1.23.3 scipy==1.10.1

Found existing installation: numpy 1.23.3

Uninstalling numpy-1.23.3:

Successfully uninstalled numpy-1.23.3

Found existing installation: scipy 1.13.1

Uninstalling scipy-1.13.1:

Successfully uninstalled scipy-1.13.1

Collecting numpy==1.23.3

Using cached numpy-1.23.3-cp310-cp310-manylinux\_2\_17\_x86\_64.manylinux2014\_x86\_64.whl.metadata (2.3 kB)

Collecting scipy==1.10.1

Downloading scipy-1.10.1-cp310-cp310-manylinux\_2\_17\_x86\_64.manylinux2014\_x86\_64.whl.metadata (58 kB)

Using cached numpy-1.23.3-cp310-cp310-manylinux\_2\_17\_x86\_64.manylinux2014\_x86\_64.whl (17.1 MB)

Downloading scipy-1.10.1-cp310-cp310-manylinux\_2\_17\_x86\_64.manylinux2014\_x86\_64.whl (34.4 MB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34.4/34.4 MB 101.7 MB/s eta 0:00:00

Installing collected packages: numpy, scipy

Successfully installed numpy-1.23.3 scipy-1.10.1

!my\_env/bin/python optimize\_model.py

[info] Loading model script commands to yolov11\_model from string

[info] **Starting Model Optimization**

[**warning**] Reducing optimization level to 0 (the accuracy won't be optimized and compression won't be used) because there's less data than the recommended amount (1024), and there's no available GPU

[**warning**] Running model optimization with zero level of optimization is not recommended for production use and might lead to suboptimal accuracy results

[info] Model received quantization params from the hn

[info] MatmulDecompose skipped

[info] Starting Mixed Precision

[info] Model Optimization Algorithm Mixed Precision is done (completion time is 00:00:01.01)

[info] LayerNorm Decomposition skipped

[info] Starting Statistics Collector

[info] Using dataset with 16 entries for calibration

Calibration: 100% 16/16 [01:19<00:00, 4.96s/entries]

[info] Model Optimization Algorithm Statistics Collector is done (completion time is 00:01:22.99)

[info] Starting Fix zp\_comp Encoding

[info] Model Optimization Algorithm Fix zp\_comp Encoding is done (completion time is 00:00:00.00)

[info] Starting Matmul Equalization

[info] Model Optimization Algorithm Matmul Equalization is done (completion time is 00:00:00.03)

[info] activation fitting started for yolov11\_model/reduce\_sum\_softmax1/act\_op

[**warning**] Reducing output bits of yolov11\_model/conv54 by 5.0 bits (More than half)

[info] Finetune encoding skipped

[info] Bias Correction skipped

[info] Adaround skipped

[info] Quantization-Aware Fine-Tuning skipped

[info] Layer Noise Analysis skipped

[info] **Model Optimization is done**

[info] Saved HAR to: /content/yolov11\_model\_quantized\_model.har

Quantized HAR file saved to: yolov11\_model\_quantized\_model.har

## Kompilace do souboru HEF

with open("compile\_model.py", "w") as f:

    f.write("""

from hailo\_sdk\_client import ClientRunner

import os

# Define the quantized model HAR file

model\_name = "yolov11\_model"

quantized\_model\_har\_path = f"{model\_name}\_quantized\_model.har"

output\_directory = "/content/drive/MyDrive/HAILO8"

os.makedirs(output\_directory, exist\_ok=True)

# Initialize the ClientRunner with the HAR file

runner = ClientRunner(har=quantized\_model\_har\_path)

print("[info] ClientRunner initialized successfully.")

# Compile the model

try:

    hef = runner.compile()

    print("[info] Compilation completed successfully.")

except Exception as e:

    print(f"[error] Failed to compile the model: {e}")

    raise

# Save the compiled model to the specified directory

output\_file\_path = os.path.join(output\_directory, f"{model\_name}.hef")

with open(output\_file\_path, "wb") as f:

    f.write(hef)

print(f"[info] Compiled model saved successfully to {output\_file\_path}")

""")

Kompilace modelu trvá dle použitého modelu mezi 2-8hodiny v závislosti na použitém HW a knihoven NVIDIA CUDA etc. V mém, případě při použití prostředí Google colab a specializované karty na TPU (tensor procesor unit) compilace trvala cca 4 hodiny.

Je pro nutné mít k dispozici viualiuace a aktuální status kompilace, pro je použit cod pro viualiuace.

(Tensor Processing Units (TPUs)   
Tensor Processing Units (TPUs) jsou speciálně vyvinuté aplikačně specifické integrované obvody (ASIC) od společnosti Google, které slouží k akceleraci úloh strojového učení.

Google Colab Pro / Pro+

TPU v3-8 (8 jádrová TPU)

Vyšší priorita a delší doba běhu

Výpočetní výkon přibližně 420 TFLOPS (bfloat16)

import os

import sys

import logging

import psutil

import time

import datetime

import signal

import subprocess

import queue

import shutil

import glob

from threading import Thread, Event

from pathlib import Path

from typing import List, Optional, Dict, Any

from collections import deque

from contextlib import contextmanager

from IPython.display import clear\_output

class Config:

    """

    Configuración del sistema de monitoreo.

    """

    LOG\_DIR = Path('logs')

    DATE\_FORMAT = "%Y%m%d\_%H%M%S"

    PLOT\_PREFIX = 'plots'

    LOG\_PREFIX = 'compilation\_log'

    LOG\_FORMAT = '%(asctime)s - %(levelname)s - %(message)s'

    def \_\_init\_\_(self):

        """Inicializa la configuración y crea directorios necesarios."""

        self.timestamp = datetime.datetime.now().strftime(self.DATE\_FORMAT)

        self.plots\_dir = self.LOG\_DIR / f'{self.PLOT\_PREFIX}\_{self.timestamp}'

        self.log\_file = self.LOG\_DIR / f'{self.LOG\_PREFIX}\_{self.timestamp}.txt'

        self.\_create\_directories()

        self.\_setup\_logging()

    def \_create\_directories(self) -> None:

        """Crea los directorios necesarios para logs y gráficos."""

        self.LOG\_DIR.mkdir(exist\_ok=True)

        self.plots\_dir.mkdir(exist\_ok=True)

    def \_setup\_logging(self) -> None:

        """Configura el sistema de logging."""

        logging.basicConfig(

            level=logging.INFO,

            format=self.LOG\_FORMAT,

            handlers=[

                logging.FileHandler(self.log\_file),

                logging.StreamHandler()

            ]

        )

class ResourceMonitor:

    """

    Monitor de recursos del sistema.

    """

    PROGRESS\_BAR\_LENGTH = 25

    MEMORY\_UNITS = ['B', 'KB', 'MB', 'GB', 'TB']

    MAX\_OUTPUT\_LINES = 100

    MONITOR\_INTERVAL = 2

    def \_\_init\_\_(self):

        """Inicializa el monitor de recursos."""

        self.config = Config()

        self.compile\_output = deque(maxlen=self.MAX\_OUTPUT\_LINES)

        self.running = True

        self.stop\_event = Event()

        self.\_setup\_signal\_handlers()

    def \_setup\_signal\_handlers(self) -> None:

        """Configura los manejadores de señales del sistema."""

        signal.signal(signal.SIGINT, self.\_signal\_handler)

        signal.signal(signal.SIGTERM, self.\_signal\_handler)

    def \_signal\_handler(self, signum: int, frame: Any) -> None:

        """Maneja las señales del sistema."""

        self.running = False

        self.stop\_event.set()

        logging.info(f"Señal {signum} recibida. Iniciando apagado graceful...")

    def format\_bytes(self, bytes\_value: float) -> str:

        """Formatea bytes a una representación legible."""

        for unit in self.MEMORY\_UNITS:

            if bytes\_value < 1024:

                return f"{bytes\_value:.2f}{unit}"

            bytes\_value /= 1024

        return f"{bytes\_value:.2f}TB"

    def log\_output(self, line: str) -> None:

        """Registra una línea de salida en el archivo de log."""

        try:

            with open(self.config.log\_file, 'a', encoding='utf-8') as f:

                timestamp = datetime.datetime.now().strftime(Config.DATE\_FORMAT)

                f.write(f"[{timestamp}] {line}\n")

        except IOError as e:

            logging.error(f"Error escribiendo al archivo de log: {e}")

    def create\_progress\_bar(self, percentage: float) -> str:

        """Crea una barra de progreso visual."""

        try:

            percentage = max(0, min(100, float(percentage)))

            filled = int(self.PROGRESS\_BAR\_LENGTH \* percentage / 100)

            return f"{'█' \* filled}{'░' \* (self.PROGRESS\_BAR\_LENGTH - filled)}"

        except (ValueError, TypeError):

            return "░" \* self.PROGRESS\_BAR\_LENGTH

    def print\_status(self, cpu: float, mem: Any, disk: Any) -> None:

        """Imprime el estado actual de los recursos del sistema."""

        try:

            clear\_output(wait=True)

            current\_time = datetime.datetime.now().strftime(Config.DATE\_FORMAT)

            print(f"\n{'═' \* 50}")

            print(f"  MONITOR DE RECURSOS - {current\_time}")

            print(f"{'═' \* 50}\n")

            if self.compile\_output:

                print("ESTADO DE COMPILACIÓN (últimas 20 líneas):")

                print(f"{'─' \* 50}")

                for line in list(self.compile\_output)[-10:]:

                    print(f"  {line}")

                print(f"{'─' \* 50}\n")

            print("USO DE RECURSOS:")

            print(f"CPU:  {self.create\_progress\_bar(cpu)} {cpu:>5.1f}%")

            print(f"RAM:  {self.create\_progress\_bar(mem.percent)} {mem.percent:>5.1f}%")

            print(f"DISK: {self.create\_progress\_bar(disk.percent)} {disk.percent:>5.1f}%")

            print("\nDETALLES DE MEMORIA:")

            print(f"  Total: {self.format\_bytes(mem.total):>10}")

            print(f"  Usado: {self.format\_bytes(mem.used):>10}")

            print(f"  Libre: {self.format\_bytes(mem.available):>10}")

        except Exception as e:

            logging.error(f"Error al imprimir estado: {e}")

    def monitor\_resources(self) -> None:

        """Monitorea continuamente los recursos del sistema."""

        while self.running and not self.stop\_event.is\_set():

            try:

                cpu = psutil.cpu\_percent(interval=1)

                mem = psutil.virtual\_memory()

                disk = psutil.disk\_usage('/')

                self.print\_status(cpu, mem, disk)

            except Exception as e:

                logging.error(f"Error en monitoreo: {e}")

            finally:

                time.sleep(self.MONITOR\_INTERVAL)

    def read\_output(self, pipe: Any, output\_queue: queue.Queue) -> None:

        """Lee la salida de un pipe y la almacena en una cola."""

        try:

            for line in pipe:

                if self.stop\_event.is\_set():

                    break

                line = line.strip()

                if line:

                    output\_queue.put(line)

                    self.log\_output(line)

                    self.compile\_output.append(line)

        except Exception as e:

            logging.error(f"Error leyendo output: {e}")

        finally:

            if pipe and hasattr(pipe, 'close'):

                pipe.close()

    def run(self) -> None:

        """Ejecuta el monitor de recursos y el proceso de compilación."""

        try:

            print("\n=== INICIANDO MONITOREO DE SISTEMA ===")

            print(f"Logs: {self.config.log\_file}")

            print(f"Plots: {self.config.plots\_dir}\n")

            output\_queue = queue.Queue()

            monitor\_thread = Thread(target=self.monitor\_resources, daemon=True)

            monitor\_thread.start()

            # Verificar paths

            env\_path = Path('my\_env/bin/python')

            compile\_script = Path('compile\_model.py')

            if not env\_path.exists() or not compile\_script.exists():

                raise FileNotFoundError("No se encontró el entorno virtual o el script de compilación")

            # Iniciar proceso de compilación

            process = subprocess.Popen(

                [str(env\_path), str(compile\_script)],

                stdout=subprocess.PIPE,

                stderr=subprocess.PIPE,

                text=True,

                bufsize=1

            )

            # Iniciar threads para stdout y stderr

            output\_threads = []

            for pipe in [process.stdout, process.stderr]:

                thread = Thread(

                    target=self.read\_output,

                    args=(pipe, output\_queue),

                    daemon=True

                )

                thread.start()

                output\_threads.append(thread)

            # Esperar a que termine el proceso

            return\_code = process.wait()

            if return\_code != 0:

                logging.error(f"Error en la compilación (código {return\_code})")

            else:

                logging.info("Compilación completada exitosamente")

        except KeyboardInterrupt:

            logging.info("Detención manual del monitoreo")

            self.stop\_event.set()

            if 'process' in locals():

                process.terminate()

        except Exception as e:

            logging.error(f"Error en la ejecución: {e}")

            raise

        finally:

            self.cleanup()

    def cleanup(self) -> None:

        """Realiza la limpieza final y muestra información de cierre."""

        self.running = False

        self.stop\_event.set()

        print(f"\nLogs guardados en: {self.config.log\_file}")

        print(f"Gráficos guardados en: {self.config.plots\_dir}")

def main():

    """Función principal que ejecuta el monitor de recursos."""

    try:

        monitor = ResourceMonitor()

        monitor.run()

    except Exception as e:

        logging.critical(f"Error crítico: {e}")

        sys.exit(1)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

══════════════════════════════════════════════════

MONITOR DE RECURSOS - 20250117\_045608

══════════════════════════════════════════════════

ESTADO DE COMPILACIÓN (últimas 5 líneas):

──────────────────────────────────────────────────

[info] Iteration #39 - Contexts: 2

[info] Iteration #39 - Contexts: 2,

[info] Iteration #40 - Contexts: 2

[info] Iteration #40 - Contexts: 2,

[info] Iteration #41 - Contexts: 2

[info] Iteration #41 - Contexts: 2,

[info] Iteration #42 - Contexts: 2

[info] Iteration #42 - Contexts: 2,

[info] Iteration #43 - Contexts: 2

[info] Iteration #43 - Contexts: 2,

──────────────────────────────────────────────────

USO DE RECURSOS:

CPU: █░░░░░░░░░░░░░░░░░░░░░░░░ 4.4%

RAM: ██░░░░░░░░░░░░░░░░░░░░░░░ 11.0%

DISK: ████░░░░░░░░░░░░░░░░░░░░░ 17.7%

DETALLES DE MEMORIA:

Total: 334.56GB

Usado: 34.45GB

Libre: 297.84GB

Uložení kopie HEF modelu do Google drive,

!cp yolov11\_model.hef /content/drive/MyDrive/HAILO8/

!cp hailo\_sdk.core.log /content/drive/MyDrive/HAILO8/

# Raspberry Pi5 a Hailo: Základní pipeline pro zpracování dat

Pro snadnější aplikaci strojního učení jsme použili již předdefinovanou pipeline s uživatelsky modifikovanými user\_callback, což nám umožnilo výrazně zkrátit čas integrace a snadněji porozumět funkci pipelines

## Instalace prostředí, základní pipeline

Pro využití kompilovaného modelu HEF v prostředí Hailo 8 na akcelerátoru a základní desce Raspberry Pi 5 je nutné pracovat se základními datovými pipeline, které využívají akcelerátory Hailo H8 a H8L.

Tyto modely demonstrují klíčové technologie v oblasti **detekce objektů, analýzy lidského pohybu a segmentace objektů**. Poskytují tak solidní základ pro vývoj vlastních aplikací v oblasti počítačového vidění.

Pro správné fungování pipeline tento projekt využívá knihovnu Hailo Apps Infra, která slouží jako klíčová závislost. Podrobné informace o jejím použití naleznete v oficiální vývojářské dokumentaci.

Přesné pokyny k instalaci jsou uvedeny v hlavním souboru [README](https://github.com/hailo-ai/hailo-rpi5-examples/blob/main/README.md#installation). Je nutné správně nakonfigurovat prostředí, aby bylo možné efektivně pracovat s akcelerátory Hailo a s jejich pipeline.

## Hailo RPi5: Základní pipeline

Ukázkové pipeline demonstrují detekci objektů, odhad lidské pozice a segmentaci instancí, čímž poskytují pevný základ pro vlastní projekty.

Tento repozitář využívá jako závislost nový balíček **Hailo Apps Infra**. Další informace o jeho použití naleznete v **Development Guide**.

1. Instalace klonování repozitáře

Pro stažení repozitáře spusťte následující příkaz:

git clone https://github.com/hailo-ai/hailo-rpi5-examples.git

Přejděte do složky s projektem:

cd hailo-rpi5-examples

Spusťte instalační skript, který automatizuje proces nastavení:

./install.sh

1. Dokumentace

Podrobnější informace o využití pipeline k tvorbě vlastních aplikací naleznete v sekci [Basic Pipelines Documentation.](https://github.com/hailo-ai/hailo-rpi5-examples/blob/main/doc/basic-pipelines.md)

1. Spuštění ukázkových příkladů

Při otevření nového terminálu se ujistěte, že jste správně nastavili prostředí:

source setup\_env.sh

1. Příklad detekce objektů

Podrobnosti o této pipeline naleznete v **Detection Example Documentation**.

Spuštění příkladu detekce objektů:

python basic\_pipelines/detection.py

Pro ukončení aplikace stiskněte **Ctrl+C**.

Spuštění s využitím **kamery Raspberry Pi:**

python basic\_pipelines/detection.py --input rpi

--input /dev/video<X>

Pro zobrazení dalších možností spusťte:

python basic\_pipelines/detection.py --help

1. Podpora pře-učených modelů

Tato aplikace umožňuje použití pře-učených detekčních modelů. Více informací naleznete v sekci **Using Retrained Models**.

1. Příklad odhadu lidské pozice

Další informace najdete v **Pose Estimation Example Documentation**.

Spuštění odhadu lidské pozice:

python basic\_pipelines/pose\_estimation.py

Pro ukončení aplikace stiskněte **Ctrl+C**. Možnosti vstupu odpovídají příkladu detekce objektů.

1. Příklad segmentace instancí

Podrobnosti naleznete v **Instance Segmentation Example Documentation**.

Spuštění segmentace instancí:

python basic\_pipelines/instance\_segmentation.py

## Přehled pipeline a jejich využití

Tento dokument poskytuje základní přehled o tom, jak lze vyvíjet vlastní aplikace s využitím pipeline dostupných v tomto repozitáři. Ukázkové scénáře zahrnují:

* **Detekci objektů** – identifikace různých předmětů v obraze.
* **Odhad lidské pozice** – rozpoznání a analýza postavení lidského těla.
* **Instance segmentaci** – oddělení jednotlivých objektů v obraze od pozadí.

Systém využívá výkonné akcelerátory Hailo H8 a H8L, které umožňují efektivní zpracování obrazových dat v reálném čase.

### Metoda zpětného volání (Callback)

Každý příklad implementovaný v této pipeline používá metodu **callback**, což je mechanismus umožňující zpracování dat získaných z GStreamer pipeline. Tato metoda je nastavena jako funkce, která se spustí ve chvíli, kdy je dostupný nový obrazový rámec. Jejím úkolem je analyzovat data, extrahovat relevantní informace a následně provést požadované operace, jako je vykreslení výsledků nebo výpis informací do terminálu.

### Uživatelská třída Callback

V rámci pipeline existuje speciální třída **user\_app\_callback\_class**, která rozšiřuje funkcionalitu základní **app\_callback\_class** dostupné v balíčku **hailo\_apps\_infra**. Tato třída umožňuje spravovat uživatelská data a uchovávat stav mezi jednotlivými snímky. Mezi její hlavní funkce patří například:

* počítání snímků,
* správa dat jednotlivých rámců,
* implementace specifických funkcí požadovaných uživatelem.

### Důležité poznámky k callback funkcím

Callback funkce musí být navržena tak, aby její zpracování bylo co nejrychlejší. Pokud by došlo k jejímu zablokování na příliš dlouhou dobu, pipeline se může zastavit. V případě, že je nutné provádět náročné výpočty, doporučuje se odeslat data do samostatného procesu na pozadí. Tento přístup je implementován například ve třídě **WLEDDisplay**, která běží nezávisle a umožňuje plynulý chod pipeline.

### Přehled dostupných pipeline

Pipeline obsažené v tomto repozitáři využívají knihovnu **hailo-apps-infra**, která poskytuje nástroje pro práci s daty. Mezi dostupné pipeline patří například:

**Detekce objektů**

Tato pipeline umožňuje detekci objektů s využitím modelu YOLOv8s (pro Hailo-8L s výkonem 13 TOPS) a modelu YOLOv8m (pro Hailo-8 s výkonem 26 TOPS). Systém podporuje také další modely kompatibilní s postprocesingem **HailoRT NMS**, což znamená, že všechny detekční sítě využívající tento algoritmus mohou fungovat bez nutnosti úprav.

**Klíčové funkce:**

* Implementace vlastní callback třídy umožňující odesílání uživatelských dat.
* Analýza metadat **HAILO\_DETECTION** v každém snímku.
* Výpočet souřadnic, důvěryhodnosti detekovaných objektů a jejich identifikátorů.
* Možnost použití vlastních detekčních modelů pomocí přepínače **--hef-path**.

### Ladění pipeline a výkonu systému

1. Debugging pomocí výpisů

Nejjednodušší způsob ladění pipeline je využití **print()** funkcí pro kontrolu proměnných, počtu detekovaných objektů či souřadnic v jednotlivých snímcích.

2. Interaktivní ladění s ipdb

Pro detailnější inspekci kódu lze využít debugger **ipdb**, který umožňuje krokování kódu, prohlížení proměnných a interaktivní testování funkcí. Aktivace debuggeru se provede přidáním řádku:

import ipdb; ipdb.set\_trace()

Při dosažení tohoto bodu se vykonávání kódu zastaví a uživatel může interaktivně prozkoumávat stav aplikace.

3. Řešení problémů s přehráváním videa

Pokud dochází k trhanému přehrávání videa, příčinou může být nadměrná výpočetní zátěž v callback funkci nebo v jiných částech pipeline. Možná řešení zahrnují:

* minimalizaci zpracování dat v callback funkci,
* přesun výpočetně náročných operací do samostatného procesu,
* snížení rozlišení nebo snímkové frekvence videa,
* monitorování výkonu pomocí **htop**.

Pokud je zatížení procesoru na hranici 100 %, může být vhodné přejít na méně náročný model nebo upravit dávkování dat.

### Monitorování výkonu Hailo akcelerátorů

Pro sledování vytížení akcelerátoru lze využít nástroj **Hailo Monitor**, který se aktivuje následujícím způsobem:

bash

ZkopírovatUpravit

export HAILO\_MONITOR=1

hailortcli monitor

Tento nástroj poskytuje přehled o aktuálním zatížení modelu a umožňuje optimalizaci pipeline.

### Konfigurace prostředí

Při každém spuštění nové relace je třeba správně nastavit prostředí pomocí skriptu:

bash

ZkopírovatUpravit

source setup\_env.sh

Následně je nutné doinstalovat požadované balíčky:

bash

ZkopírovatUpravit

pip install -r requirements.txt

V případě chybějící knihovny **rapidjson-dev** ji lze nainstalovat pomocí:

bash

ZkopírovatUpravit

sudo apt install -y rapidjson-dev

### Závěr

Základní pipeline pro Hailo RPi5 poskytují robustní řešení pro implementaci algoritmů počítačového vidění. Díky efektivnímu využití hardwarových akcelerátorů a optimalizovaným pipeline je možné dosáhnout vysokého výkonu při analýze obrazových dat. Systém je navržen tak, aby umožňoval snadnou modifikaci a rozšiřování, což jej činí vhodným nástrojem pro široké spektrum průmyslových aplikací.

# Moje aplikace

## Uživatelský návod pro projekt DMC dekódování na Raspberry Pi

Tento dokument poskytuje uživatelský návod k projektu pro detekci a dekódování Data Matrix kódů (DMC) na Raspberry Pi pomocí Hailo AI akcelerátoru. Projekt využívá GStreamer pro zpracování videa, knihovnu pylibdmtx pro dekódování DMC a knihovnu tkinter pro zobrazování výsledků v GUI.

## Přehled souborů projektu

Projekt obsahuje následující soubory

| SW | Dokument |
| --- | --- |
| start\_here.sh | Spouštěcí skript pro inicializaci prostředí a spuštění aplikace. |
| run\_app.sh | Skript pro aktivaci virtuálního prostředí a spuštění hlavního skriptu main.py |
| main.py | Hlavní skript aplikace, který řídí celý proces, od inicializace GStreamer pipeline až po spuštění dekódovacího vlákna a GUI. |
| pipeline\_code.py | Obsahuje kód pro zpracování videa, použití Hailo AI akcelerátoru a vyhledávání DMC kódů. |
| decode\_thread.py | Dekódovací vlákno, přijímá výřez obrázku s DMC a pokouší se jej přečíst. |
| user\_callback.py | Uživatelská callback třída pro interakci s pipeline. |
| gui\_display.py | Zajišťuje grafické zobrazení dekódovaného kódu v jednoduchém GUI. |
|  |  |

## Instalace a spuštění

#### **Požadavky**

Raspberry Pi 5 nebo kompatibilní SBC s podporou GStreamer a AI akcelerátoru Hailo.

Virtuální prostředí Python s potřebnými knihovnami (hailo, pylibdmtx, opencv, tkinter).

Kamera kompatibilní s Raspberry Pi (např. Raspberry Pi HQ kamera).

#### Spuštění

Aktivace virtuálního prostředí a spuštění aplikace

./start\_here.sh

nebo ručně:

source /home/david/DMC\_app/venv\_hailo\_rpi5\_examples/bin/activate

python main.py

#### Zajištění správného fungování kamery

* + Zkontrolujte připojení a funkčnost kamery pomocí:

rpicam-hello

Pokud je kamera připojena, aplikace automaticky spustí okno s náhledem kamery

Zkontrolujte připojení a funkčnost kamery Hailo 8 akcelerátor:

lspci

0000:00:00.0 PCI bridge: Broadcom Inc. and subsidiaries BCM2712 PCIe Bridge (rev 21)

0000:01:00.0 Co-processor: Hailo Technologies Ltd. Hailo-8 AI Processor (rev 01)

0001:00:00.0 PCI bridge: Broadcom Inc. and subsidiaries BCM2712 PCIe Bridge (rev 21)

0001:01:00.0 Ethernet controller: Raspberry Pi Ltd RP1 PCIe 2.0 South Bridge

**Hailo-8 AI Processor (rev 01)**

Toto potvrzuje, že váš Hailo-8 akcelerátor je úspěšně detekován a připojen k PCIe sběrnici.

Pokud plánujete spustit aplikace využívající Hailo SDK, měli byste ověřit, zda jsou správně nainstalovány ovladače a knihovny.

**RP1 South Bridge (Ethernet controller)**

Čip RP1 slouží k rozšíření možností PCIe, v tomto případě zajišťuje ethernetovou konektivitu.

**Broadcom PCIe Bridges (BCM2712)**

Tyto dva PCI bridge čipy umožňují komunikaci mezi Raspberry Pi 5 a připojenými zařízeními (např. Hailo-8).

## Hlavní řídící skript (main.py)

Tento skript funguje jako hlavní kontrolní bod aplikace. Jeho úkoly zahrnují:

* **Inicializaci GStreamer frameworku** – pro zpracování videostreamu.
* **Spuštění dekódovacího vlákna (DMCDecoderThread)** – které přijímá výřezy snímků a pokouší se dekódovat DMC.
* **Inicializaci uživatelského callbacku (user\_app\_callback\_class)** – který propojuje pipeline a dekodér.
* **Spuštění GStreamer pipeline (MyDetectionApp)** – což zahrnuje detekci DMC pomocí AI.
* **Spuštění GUI (DecodedGUI)** – jednoduché okno zobrazující dekódované kódy.
* **Zachytávání signálů (SIGINT, SIGTERM)** – umožňuje korektní ukončení programu při stisku Ctrl+C.

Struktura kódu

# main.py

import sys

import signal

import gi

gi.require\_version('Gst', '1.0')

from gi.repository import Gst

from queue import Queue

from decode\_thread import DMCDecoderThread

from user\_callback import user\_app\_callback\_class

from pipeline\_code import MyDetectionApp, app\_callback

from gui\_display import DecodedGUI

QUEUE\_MAXSIZE = 50

decoder\_thread = None

app = None

def signal\_handler(sig, frame):

print(f"[MAIN] Caught signal {sig}, shutting down...")

if decoder\_thread:

decoder\_thread.stop()

if app:

app.stop()

sys.exit(0)

def main():

# Handle Ctrl+C, kill signals

signal.signal(signal.SIGINT, signal\_handler)

signal.signal(signal.SIGTERM, signal\_handler)

# Initialize GStreamer

print("[MAIN] Initializing GStreamer...")

Gst.init(None)

# Create a queue for bounding box images

roi\_queue = Queue(maxsize=QUEUE\_MAXSIZE)

# Start the decode thread

global decoder\_thread

decoder\_thread = DMCDecoderThread(roi\_queue)

decoder\_thread.start()

# Create user callback object with reference to roi\_queue

user\_data = user\_app\_callback\_class(roi\_queue)

user\_data.use\_frame = True

# Build the pipeline app

global app

app = MyDetectionApp(app\_callback, user\_data)

# Start the GUI in a separate thread so GStreamer doesn't block it

# Alternatively, we can just call app.run() and not show GUI, but let's show both

gui = DecodedGUI()

# We'll run the pipeline in its own thread so the GUI can remain responsive

import threading

def run\_pipeline():

print("[MAIN] Starting pipeline...")

app.run() # blocks until pipeline stops

print("[MAIN] Pipeline ended.")

pipeline\_thread = threading.Thread(target=run\_pipeline, daemon=True)

pipeline\_thread.start()

# Now run the GUI main loop in the main thread

print("[MAIN] Running GUI. Press Ctrl+C to exit.")

gui.run()

# If GUI closes, we stop everything

decoder\_thread.stop()

pipeline\_thread.join()

print("[MAIN] Done.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

## GStreamer pipeline (pipeline\_code.py)

Pipeline je klíčovou součástí systému, která se stará o:

* **Zachytávání videa** – využívá GStreamer k získávání obrazu z kamery v reálném čase.
* **Předzpracování snímků** – provádí konverzi obrazu do požadovaného formátu.
* **Detekci DMC kódu** – využívá AI model běžící na Hailo akcelerátoru.
* **Výřez oblasti zájmu (ROI)** – pokud je nalezen DMC, jeho výřez je přidán do fronty pro dekódování.

GStreamer pipeline se konfiguruje pomocí následujícího řetězce:

# pipeline\_code.py

import cv2

import time

from datetime import datetime

from gi.repository import Gst

import hailo

from hailo\_apps\_infra.hailo\_rpi\_common import get\_caps\_from\_pad, get\_numpy\_from\_buffer

from hailo\_apps\_infra.detection\_pipeline import GStreamerDetectionApp

# Confidence threshold

DMC\_CONF\_THRESH = 0.8

def get\_pipeline\_string():

"""

Return a GStreamer pipeline string that captures from the RPi camera (or HQ camera),

at 640x480, 60 FPS, and passes raw frames to an appsink for further processing.

Adjust as needed for your hardware setup (libcamerasrc vs v4l2src, etc.).

"""

pipeline = (

"libcamerasrc sensor-id=0 exposure-time=3000 ! "

# 640 x 480 to 480 to 480

"video/x-raw,width=480,height=480,framerate=60/1 ! "

# Ensure we only produce 1 frame per second from the pipeline

"videorate ! video/x-raw,framerate=1/1 ! "

"queue max-size-buffers=1 leaky=downstream ! "

"videoconvert ! "

"appsink"

)

print(f"[DEBUG] GStreamer pipeline: {pipeline}")

return pipeline

def app\_callback(pad, info, user\_data):

"""

This is the GStreamer pad probe callback:

- Takes the GStreamer buffer

- Extracts Hailo detections for label 'DMC'

- For each detection, cut out the bounding box from the frame

- Push the ROI image onto the decode queue

"""

buffer = info.get\_buffer()

if not buffer:

return Gst.PadProbeReturn.OK

user\_data.increment() # track how many frames we've processed

frame\_idx = user\_data.get\_count()

# Grab all the HAILO\_DETECTION objects from the buffer

roi = hailo.get\_roi\_from\_buffer(buffer)

detections = roi.get\_objects\_typed(hailo.HAILO\_DETECTION)

# Check if we have a valid frame

gst\_format, width, height = get\_caps\_from\_pad(pad)

if not (user\_data.use\_frame and gst\_format and width and height):

return Gst.PadProbeReturn.OK

# Convert GStreamer buffer to an RGB NumPy array, then BGR for OpenCV

frame\_rgb = get\_numpy\_from\_buffer(buffer, gst\_format, width, height)

frame\_bgr = cv2.cvtColor(frame\_rgb, cv2.COLOR\_RGB2BGR)

# Loop over all detections

for det in detections:

label = det.get\_label()

conf = det.get\_confidence()

bbox = det.get\_bbox()

if label == "DMC" and conf >= DMC\_CONF\_THRESH:

# Add some padding

x0 = int(bbox.xmin() \* width) - 10

y0 = int(bbox.ymin() \* height) - 10

x1 = int(bbox.xmax() \* width) + 10

y1 = int(bbox.ymax() \* height) + 10

# Ensure ROI remains valid

x0 = max(0, x0)

y0 = max(0, y0)

x1 = min(width, x1)

y1 = min(height, y1)

# If bounding box is invalid (e.g., out of frame), skip

if x1 <= x0 or y1 <= y0:

continue

# 1) Extract the ROI from the frame

roi\_img = frame\_bgr[y0:y1, x0:x1].copy()

# 2) Flip horizontally

roi\_img = cv2.flip(roi\_img, 1)

# 3) debug print

print(f"[PIPELINE] Enqueuing ROI for frame={frame\_idx}")

# 4) Enqueue for decoding

timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S\_%f")[:-3]

user\_data.roi\_queue.put((frame\_idx, roi\_img, timestamp))

return Gst.PadProbeReturn.OK

class MyDetectionApp(GStreamerDetectionApp):

"""

A minimal class for running the GStreamer pipeline with a custom callback.

This inherits from hailo\_apps\_infra.detection\_pipeline.GStreamerDetectionApp,

which sets up the GStreamer loop, bus, etc.

"""

def \_\_init\_\_(self, callback, user\_data):

super().\_\_init\_\_(callback, user\_data)

print("[DEBUG] MyDetectionApp initialized.")

## Dekódovací vlákno (decode\_thread.py)

Toto vlákno pracuje na pozadí a zajišťuje dekódování Data Matrix kódů z výřezů obrazu (ROI). Jeho hlavní úkoly:

1. **Přijímá obrázky z fronty roi\_queue**, které pocházejí z detekční pipeline.
2. **Používá knihovnu pylibdmtx k dekódování DMC kódu**.
3. **Ukládá dekódovaný text a obraz do souboru** pro pozdější analýzu.
4. **Aktualizuje globální proměnnou last\_decoded**, která slouží k zobrazení v GUI.

# decode\_thread.py

import os

import time

import cv2

from queue import Empty

from threading import Thread

from pylibdmtx.pylibdmtx import decode as dmtx\_decode

from PIL import Image

# This global variable is read by gui\_display.py to show the latest code

last\_decoded = ""

class DMCDecoderThread(Thread):

"""

A background thread that receives (frame\_idx, roi\_img, timestamp)

from a queue, decodes it using pylibdmtx, and:

- Saves each ROI to 'DMC\_<timestamp>.jpg'

- Writes the decoded text (if found) to 'DMC\_<timestamp>.txt'

- Updates last\_decoded so the GUI can display it.

"""

def \_\_init\_\_(self, roi\_queue):

super().\_\_init\_\_()

self.roi\_queue = roi\_queue

self.\_stop\_flag = False

# Change this to the path where your USB drive is mounted

self.output\_dir = "/media/pi/USB\_DRIVE"

# Make sure the output directory exists (will create if it doesn't)

os.makedirs(self.output\_dir, exist\_ok=True)

print("[DEBUG] DMCDecoderThread initialized with USB output directory")

def run(self):

global last\_decoded

print("[DEBUG] DMCDecoderThread started")

while not self.\_stop\_flag:

try:

# The pipeline callback puts (frame\_idx, roi\_img, timestamp)

frame\_idx, roi\_img, stamp = self.roi\_queue.get(timeout=0.5)

print(f"[DECODE\_THREAD] Got ROI frame={frame\_idx}, running decode...")

except Empty:

continue

# Convert the ROI (a NumPy BGR array) to a PIL Image in RGB for dmtx\_decode

pil\_img = cv2.cvtColor(roi\_img, cv2.COLOR\_BGR2RGB)

pil\_img = Image.fromarray(pil\_img)

# Attempt to decode the DMC

results = dmtx\_decode(pil\_img)

if results:

decoded\_str = results[0].data.decode("utf-8", errors="ignore")

print(f"[DECODE] Frame {frame\_idx} => '{decoded\_str}'")

last\_decoded = decoded\_str

else:

last\_decoded = ""

print(f"[DECODE] Frame {frame\_idx}: no DMC found")

# Save ROI image

jpg\_path = os.path.join(self.output\_dir, f"DMC\_{stamp}.jpg")

cv2.imwrite(jpg\_path, roi\_img)

print(f"[SAVE] ROI image => {jpg\_path}")

# Write decoded text (if any) to .txt

txt\_path = os.path.join(self.output\_dir, f"DMC\_{stamp}.txt")

with open(txt\_path, "w", encoding="utf-8") as f:

if last\_decoded:

f.write(last\_decoded + "\n")

else:

f.write("NO CODE FOUND\n")

print(f"[SAVE] Wrote text => {txt\_path}")

self.roi\_queue.task\_done()

print("[DEBUG] DMCDecoderThread stopping")

def stop(self):

"""Tell the thread to exit its loop."""

self.\_stop\_flag = True

**Mount Point**  
Make sure your USB drive is actually mounted under /media/pi/USB\_DRIVE (or the path you specify). If it’s mounted elsewhere, update the string in self.output\_dir accordingly.

**Permissions**  
Ensure that the user running the Python script has write permissions for the USB drive mount point.

**Consistency**  
If you move the output directory to a USB drive, confirm that other parts of your code (e.g., logging, file referencing) still work correctly with the new path.

## Callback funkce (user\_callback.py)

Callback funkce propojuje GStreamer pipeline s dekódovacím vláknem. Pokud pipeline detekuje DMC kód, tato třída přidá jeho výřez do dekódovací fronty.

# user\_callback.py

from hailo\_apps\_infra.hailo\_rpi\_common import app\_callback\_class

class user\_app\_callback\_class(app\_callback\_class):

"""

A custom callback class that inherits from hailo\_apps\_infra.app\_callback\_class.

It holds a reference to our ROI queue, so the pipeline callback can push

bounding-box images for decoding.

"""

def \_\_init\_\_(self, roi\_queue):

super().\_\_init\_\_()

self.roi\_queue = roi\_queue

self.use\_frame = True # This can be toggled

print("[DEBUG] user\_app\_callback\_class initialized.")

## Grafické rozhraní (gui\_display.py)

GUI je vytvořeno pomocí tkinter a umožňuje zobrazit poslední dekódovaný kód. Obnovuje se každých 500 ms.

# gui\_display.py

import tkinter as tk

from decode\_thread import last\_decoded

REFRESH\_MS = 500 # GUI refresh interval (milliseconds)

class DecodedGUI:

"""

A simple Tkinter-based GUI to display the last decoded Data Matrix code.

"""

def \_\_init\_\_(self):

self.root = tk.Tk()

self.root.title("DMC Decoder Display")

self.root.geometry("600x300+50+50")

self.root.configure(bg="black")

self.label = tk.Label(

self.root,

text="Waiting for codes...",

font=("Arial", 36, "bold"),

fg="white",

bg="black",

wraplength=550

)

self.label.pack(expand=True, fill=tk.BOTH, padx=20, pady=20)

# Repeatedly update label from decode\_thread.last\_decoded

self.update\_label()

def update\_label(self):

global last\_decoded

if last\_decoded:

self.label.config(text=f"Latest Code:\n{last\_decoded}")

else:

self.label.config(text="Waiting for codes...")

self.root.after(REFRESH\_MS, self.update\_label)

def run(self):

self.root.mainloop()

## vizuální znázornění fungování systému

+----------------+ +----------------+ +-----------------+

| Kamera | ---> | GStreamer | ---> | AI detekce DMC |

+----------------+ +----------------+ +-----------------+

|

v

+------------------------------------------------+

| Callback funkce přidá ROI do dekódovací fronty|

+------------------------------------------------+

|

v

+----------------+ +-----------------+ +-----------------+

| Dekódovací vlákno | --> | pylibdmtx | --> | Uložení výsledků |

+----------------+ +-----------------+ +-----------------+

|

v

+----------------+

| Zobrazení v GUI|

+----------------+

## Závěr

Tento systém efektivně kombinuje AI model pro detekci DMC, vícevláknové zpracování obrazu a jednoduché GUI. Architektura je robustní a umožňuje snadné rozšíření o nové funkce, například podporu více kamer nebo export dat do databáze. 🚀

# 

## Raspberry PI

## Workastation

# Cognex Machine Vision Systém

Cognex Machine Vision System je komplexní řešení určené k automatizované vizuální kontrole, měření a identifikaci výrobků v reálném čase. Využívá pokročilé technologie, jako jsou algoritmy pro zpracování obrazu, umělá inteligence a hluboké učení. Tyto nástroje umožňují odhalování vad, přesné třídění produktů a spolehlivé čtení kódů nebo znaků.

## Hlavní součásti systému

Strojové vidění od společnosti Cognex se skládá z několika klíčových komponent:

**Průmyslové kamery** – zaznamenávají snímky výrobků ve vysokém rozlišení.

**Optické prvky** – zahrnují čočky a osvětlovací systémy, které zajišťují optimální viditelnost objektů.

**Výkonné procesory a analytické algoritmy** – zpracovávají získané snímky a provádějí jejich detailní analýzu.

**Specializovaný software** – Cognex poskytuje nástroje jako VisionPro nebo In-Sight, které umožňují konfiguraci, vyhodnocení a optimalizaci systému podle konkrétních potřeb.

## Klíčové funkce a přínosy

Systém Cognex přináší řadu výhod, které zvyšují efektivitu výrobních procesů:

**Kontrola kvality** – rozpoznávání vad, povrchových defektů, nesprávného sestavení nebo chybějících částí.

**Přesné měření a zarovnání** – ověřování rozměrů součástek a jejich správného umístění.

**Optické rozpoznávání znaků** (OCR) – čtení číselných a alfanumerických kódů.

**Identifikace pomocí čárových a datových** kódů – automatizovaná správa produktů díky 1D a 2D kódům.

**Sledování výroby a dohledatelnost** – zajištění kontroly nad celým výrobním procesem.

## Oblasti využití

Díky své univerzálnosti nacházejí systémy Cognex uplatnění v mnoha průmyslových odvětvích:

**Automobilový průmysl** – kontrola montáže a kvality dílů.

**Elektrotechnika** – inspekce plošných spojů a přesnost osazování součástek.

**Farmacie** – ověřování správného označení léčiv a kontrola balení.

**Potravinářství** – inspekce obalů a sledování výrobní šarže pomocí datových kódů.

## Závěr

Technologie strojového vidění od společnosti Cognex přináší spolehlivý způsob, jak zautomatizovat kontrolu kvality ve výrobě. Díky neustálému vývoji v oblasti umělé inteligence a strojového učení se tyto systémy stále zdokonalují a nacházejí širší uplatnění napříč průmyslovými odvětvími. Jejich zavedení pomáhá firmám minimalizovat chyby, zvýšit přesnost a optimalizovat výrobu, což v konečném důsledku vede ke zvýšení produktivity a snížení nákladů.

# Porovnání modelů a závěr práce

# APPENDIX I princip EPB

## Princip elektrické parkovací brzdy a její konstrukce

Elektrická parkovací brzda (EPB) je moderní systém, který nahrazuje klasickou mechanickou ruční brzdu. Její konstrukce zahrnuje několik klíčových komponent, které společně umožňují efektivní zabrzdění vozidla.

### Konstrukce elektrické parkovací brzdy

1. Brzdový třmen (brake caliper):

* Základní část brzdového systému, která působí tlakem na brzdové destičky a působí tlakem proti brzdovému kotouči.
* U elektrické parkovací brzdy je třmen modifikován tak, aby obsahoval elektrický aktuátor pro ovládání parkovací funkce.
* Brzdový třmen je vybaven držákem, který drží brzdové destičky.

### Aktuátor (elektromotor s převodem):

* Aktuátor je malý elektromotor namontovaný na brzdový třmen, popřípadě k lanku.
* Jeho úkolem je přeměnit elektrický signál na mechanickou sílu, která posune brzdové destičky proti kotouči.
* V prvních systémech aktuátor nepůsobí přímo na třmen, ale napíná brzdové lanko podobně jako u klasické brzdy.

### Brzdové destičky a kotouč

* Brzdové destičky působí na brzdový kotouč, čímž dochází k zabrzdění vozidla.
* Síla vyvinutá aktuátorem je přenášena na pístky v brzdovém třmenu, které zatlačí destičky na kotouč.

### Řídicí jednotka EPB (ECU):

* Centrální elektronická jednotka, která ovládá funkci elektrické parkovací brzdy.
* Zpracovává signály z:
* Ovládacího tlačítka EPB,
* Senzorů rychlosti,
* sklonového senzoru (v případě automatického zabrzdění na kopci),
* senzorů polohy brzdového třmenu.
* Řídicí jednotka také monitoruje stav brzdy a v případě poruchy aktivuje nouzové režimy.

### Senzory

* Senzor opotřebení destiček – kontroluje stav brzdových destiček.
* Senzor sklonu vozidla – umožňuje automatické zabrzdění vozidla na svahu.
* Senzory rychlosti – detekují pohyb vozidla pro aktivaci či deaktivaci EPB.

### Ovládací tlačítko

* Nahrazuje klasickou páku ruční brzdy.
* Jediným stisknutím tlačítka vyšle řidič pokyn k aktivaci nebo deaktivaci parkovací brzdy.

### Napájecí a kabelážní systém

* Poskytuje elektrickou energii aktuátorům a senzorům.
* Zahrnuje pojistky, relé a vodiče pro propojení s hlavní řídicí jednotkou vozu

### Jak elektrická parkovací brzda funguje?

* Pro aktivaci řidič stiskne tlačítko pro EPB.
* Řídicí jednotka pošle signál k aktuátoru v brzdovém třmenu.
* Elektromotor v aktuátoru zatlačí na pístky třmenu, které přitlačí brzdové destičky k brzdovému kotouči.
* Pro deaktivaci se po stisknutí tlačítka pro uvolnění EPB, nebo při automatickém rozjezdu, řídicí jednotka vyšle signál k uvolnění aktuátoru.
* Elektromotor stáhne pístky zpět a destičky se uvolní od kotouče.
* V případě nouze lze zastavit vozidlo stiskem tlačítka EPB a řídicí jednotka aktivuje EPB i za jízdy.
* Systém použije plynulé brzdění na všechna kola pro bezpečné zpomalení.

### Shrnutí konstrukce

• Třmen s aktuátorem: Obsahuje elektromotor, převodovku a pístky.

• Brzdové destičky a kotouč: Zajišťují fyzické zabrzdění.

• Řídicí jednotka: Mozek systému, který vše řídí a monitoruje.

• Senzory: Zajišťují zpětnou vazbu o poloze, sklonu a stavu brzd.

• Ovládací tlačítko: Jednoduché rozhraní pro řidiče.

Elektrická parkovací brzda je tedy komplexní systém, který kombinuje mechanické části (třmeny, aktuátory) s moderní elektronikou (řídicí jednotka, senzory) pro dosažení spolehlivého, efektivního a bezpečného zabrzdění vozidla.

# APPENDIX II dokumentace nakup díly