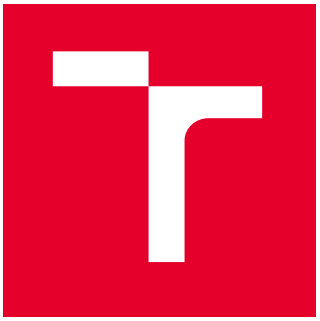


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV RADIOELEKTRONIKY

DEPARTMENT OF RADIO ELECTRONICS

ROZPOZNÁVÁNÍ OBJEKTŮ PŘI SNÍŽENÉ VIDITELNOSTI POMOCÍ AI

OBJECT RECOGNITION IN REDUCED VISIBILITY USING AI

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **David Bartoň**

AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Petr Mlýnek, Ph.D.**

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Elektronika a komunikační technologie**

Ústav radioelektroniky

Student: David Bartoň ***ID:*** 239408

Ročník: 3 ***Akademický rok:*** 2024/25

NÁZEV TÉMATU:

Rozpoznávání objektů při snížené viditelnosti pomocí AI

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s problematikou neuronových sítí a rozpoznání osob pomocí analýzy obrazu. Zároveň proveďte rešerši na problémy rozpoznání osob při snížené viditelnosti, seznamte se s dostupnými řešení těchto problémů a vyhledejte již dostupné datasety vhodné pro rozšíření vlastními daty. Vytipujte vhodné komponenty a zařízení potřebné pro detekci osob a zpracování obrazu s přihlédnutím k náročnosti neuronových sítí a potřebného výkonu k jejich zpracovávání. Vybrané řešení realizujte, ověřte jeho funkčnost a otestujte v reálném prostředí. Zařízení optimalizujte pro dosažení co nejmenší chybovosti. Zhodnoťte dosažené výsledky a využitelnost zařízení. Zařízení dále otestujte v podmínkách snížené viditelnosti. Na základě testování stanovte nutné vstupní podmínky pro dosažené požadované spolehlivosti (např. zorné pole kamery, výška kamery, parametry kamery či natočení kamery vůči povrchu země). Realizujte jednoduchou klientskou aplikaci (uživatelské prostředí) pro demonstraci rozpoznávání objektů.

DOPORUČENÁ LITERATURA:

[1] CHOLLET, F. Deep Learning with Python. Second Edition, Shelter Island, NY 11964: Manning, 2021. ISBN 9781617296864.

[2] REBALA, G.; RAVI; A.; CHURIWALA, S. An Introduction to Machine Learning. Cham, Springer, [2019]. ISBN 978-3-030-15729-6.

Termín zadání: 10.2.2025 ***Termín odevzdání:*** 2.6.2025

Vedoucí práce: doc. Ing. Petr Mlýnek, Ph.D.

doc. Ing. Lucie Hudcová, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato práce se zabývá vývojem systému pro detekci osob v podmínkách snížené viditelnosti pomocí neuronových sítí. Po prozkoumání funkce neuronových sítí byl vytvořen systém, který využívá model YOLOv5s, který byl dotrénován na datasetu obsahujícím především osoby za horších viditelnostních podmínek. Pro zpracování obrazu bylo zvoleno zařízení OAK-1 s čipem RVC2 a Raspberry Pi 5 pro zobrazení výsledků. Natrénovaný model dosahuje přesnosti detekce mAP50 $77,4\ \%$. Model byl testován jak v evaluačním prostředí Google Colab tak i v reálném prostředí s umělou mlhou, kde kamera instalovaná ve výšce 3 metrů a prokázala uspokojivé výsledky detekce při rychlosti zpracování $70 - 80$ ms na snímek. Výsledné řešení nabízí systém pro detekci osob v reálném čase s vizualizací detekovaných objektů a jejich jistotou detekce, přístupný přes webové rozhraní postavené na frameworku Flask.

KLÍČOVÁ SLOVA

neuronové sítě, rozpoznání osob, analýza obrazu, snížená viditelnost, dataset, optimalizace, detekce osob, DepthAI

ABSTRACT

This thesis deals with the development of a system for detecting people in low visibility conditions using neural networks. After investigating the function of neural networks, a system has been developed using the YOLOv5s model, which has been retrained on a dataset containing mainly people under low visibility conditions. An OAK-1 device with an RVC2 chip was chosen for image processing and a Raspberry Pi 5 was used to display the results. The trained model achieves a mAP50 detection accuracy of $77.4\ \%$. The model was tested both in the Google Colab evaluation environment and in a real artificial fog environment, where the camera was installed at a height of 3 meters and showed satisfactory detection results at a processing speed of $70 - 80$ ms per frame. The resulting solution offers a real-time person detection system with visualization of detected objects and their detection confidence, accessible through a web interface built on the Flask framework.

KEYWORDS

neural networks, person recognition, image analysis, reduced visibility, dataset, optimization, person detection, DepthAI

BARTOŇ, David. *Rozpoznávání objektů při snížené viditelnosti pomocí AI*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2025. Vedoucí práce: doc. Ing. Petr Mlýnek, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	David Bartoň
VUT ID autora:	239408
Typ práce:	Bakalářská práce
Akademický rok:	2024/25
Téma závěrečné práce:	Rozpoznávání objektů při snížené viditelnosti pomocí AI

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Ing. Petru Mlýnkovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Úvod

V této práci se zabýváme řešením problému rozpoznávání osob ve snížených viditelnostních podmínkách pomocí moderních metod počítačového vidění. Neuronové sítě představují základní stavební kameny moderních systémů počítačového vidění, inspirované funkcí biologického mozku. Konvoluční neuronové sítě pak představují specializovanou architekturu, která je optimalizovaná právě pro zpracování obrazových dat a nachází široké uplatnění v oblasti počítačového vidění.

Práce je rozdělena do dvou hlavních částí - teoretické a praktické. V teoretické části se v kapitole 1.1 věnujeme základům neuronových sítí a hlubokého učení, kde rozebíráme principy fungování jednotlivých typů vrstev a jejich propojení. Následně v kapitole 1.2 se podrobně zabýváme konvolučními neuronovými sítěmi a jejich specifiky při zpracování obrazu. Kapitola 1.3 se věnuje problematice datasetů a jejich klasifikaci, což je zásadní pro úspěšný trénink modelů. Detekční algoritmus YOLO je detailně rozebrán v kapitole 1.4, kde jsou vysvětleny jeho principy. Specifika snížené viditelnosti a její vliv na detekci osob jsou analyzovány v kapitole 1.5, kde se zaměřujeme na charakteristiky těchto podmínek.

V praktické části práce se v kapitole 2.1 soustředíme na výběr vhodných komponent pro realizaci systému, kde porovnáváme různé možnosti mikropočítačů a kamer. Kapitola 2.2 popisuje proces testování jednotlivých hardwarových komponent, včetně Jetson Nano, Orange Pi 5B a Coral USB Accelerator. V kapitole 2.3 se věnujeme trénování neuronových sítí pomocí platformy Roboflow a alternativně pomocí Pythonu a YOLOv5. Postupný vývoj systému je dokumentován v kapitolách 2.4 a 2.5, kde jsou popsány jednotlivé iterace návrhu od prvotního systému přes finální aplikaci až po testování v reálném prostředí.

Celý systém je následně podroben testování v reálném prostředí za podmínek snížené viditelnosti, kde jsou ověřeny jeho funkční parametry včetně přesnosti detekce, rychlosti zpracování a optimálního nastavení hardwaru. Na konci práce je také vývoj přehledného uživatelského rozhraní pro demonstraci funkčnosti systému a praktickou ukázku výsledků detekce.

1 Teoretická část práce

teor$_c ast$

V následujících kapitolách se blíže seznámíme s neuronovými sítěmi. Začneme obecným popisem, následně zabředneme více do historie celého vývoje umělé inteligence. Na konci tohoto dokumentu je sepsán popis konkrétního modelu neuronové sítě, který jsme zvolili a se kterým jsme následně provedli potřebné testování jednotlivých komponentů.

1.1 Neuronové sítě

Pokud chceme, aby neuronová síť pomocí detekce objektů rozpoznala, co je na obrázku, musíme si nejdříve uvědomit, co musí udělat člověk, aby pochopil, co vše se na obrázku nachází. Člověk si při pohledu na obrázek uvědomuje mnoho parametrů, na jejichž základě se pak rozhoduje. Všímá si například obrysů, barev a jejich odstínů, tvarů nebo porovnává velikost vyobrazených objektů. Zároveň ze zkušenosti ví, jak co vypadá, a pravděpodobně mu někdo oznámil, co má na obrázku hledat. Člověk má tedy přirozeně k dispozici základní sadu vstupů: velikost obrázku, barvy, hodnoty jasu barev a co má na obrázku očekávat. Poučíme-li se z takto jednoduché a známé situace, vyjde nám, že stejné vstupy musí být zadány i neuronové síti. S neuronovou sítí ale k naší smůle nemůžeme komunikovat jako s člověkem. S neuronovou sítí, stejně jako s každým jiným počítačovým komponentem, komunikujeme pomocí čísel [7]. Naše vstupní informace tedy musíme chytře převést do číselného vyjádření. K řešení nám pomohou tensory. Tensor je vícerozměrná matice reprezentující určitá data [8]. Velikost obrázku a využité barvy snadno do tensoru uložíme. Například tensor ve formě $[3, 224, 224]$ může reprezentovat trojici [počet barevných kanálů, výška, šířka]. Tento tensor určí modelu neuronové sítě formát nebo rozměr vstupních dat. Veškeré výpočty uvnitř modelu neuronové sítě jsou také prováděny v tensorech [7].

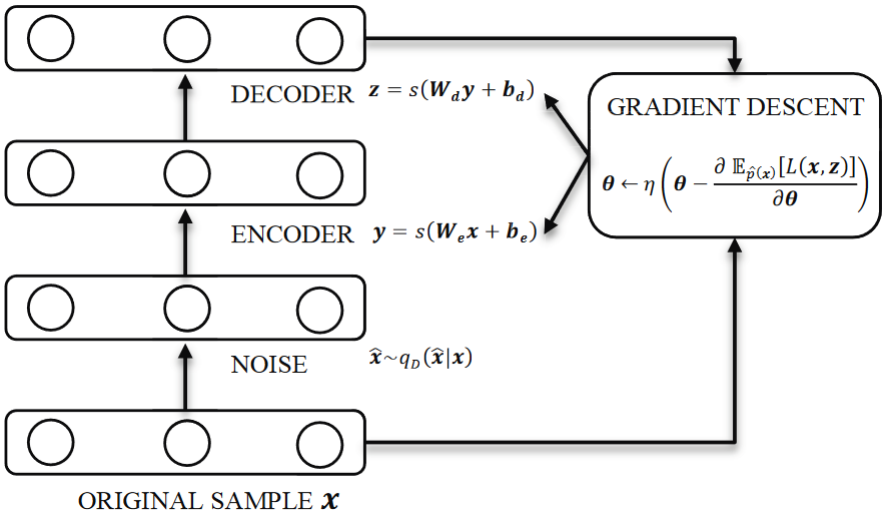
Nutností je manuálně přiřadit prvkům ve trénovacích datasetech třídu. Datasety mohou být cokoli, od sbírky audio nahrávek po sbírky obrázků. Tyto datasety se označí třídami, které jsou nebo nejsou na obrázcích a následně se na nich náš model může natrénovat (náš tréninkový dataset budou obrázky). Označené datasety budou sloužit jako trénovací materiál naší sítě. Během tréninku se neuronová síť snaží najít jednotlivé vzory, podle kterých se naučí správné klasifikaci. Tudíž stejně jako člověk porovná, co je povědomé s tím, co je naučené. Následně může ohraničit i hledaný objekt, abychom viděli jestli označila správné místo. Jelikož porovnává, co se naučila, s věcmi co "vidí", tak jde o určení pravděpodobnosti. Jak natrénovat model umělé

inteligence je popsáno v kapitole 2.3. Díky novým algoritmům, jako je YOLO (viz. 1.4), dokážeme mít rychlé a spolehlivé neuronové sítě, které mají přesnost detekce mezi 80 až 99 procenty [7].

1.1.1 Hluboké učení

Základní princip hlubokého učení je postupné předtrénování sám sebe bez dozoru a následné hierarchické vytváření reprezentací vstupních dat a nakonec trénování s dozorem. Pojem "trénování s dozorem"znamená, že model se učí na datasetu s označenými daty. Jde například o obrázky s přiřazenými třídami nebo popisky zobrazených objektů jako pes, kočka nebo auto. Model se učí spojovat vstupní data se správnými výstupy. Naopak "trénování bez dozoru"pracuje s daty bez označení. Obrázky nemají popisky ani informace o zobrazovaném obsahu. Model musí sám objevit strukturu nebo vzory v těchto datech. V tradičním přístupu se jednotlivé vrstvy trénují každá zvlášť na vlastních datech. Tento přístup se však v moderních architekturách jako YOLO (viz. 1.4) již nepoužívá. V počáteční fázi trénování se často používají autoenkodéry [1].

Autoenkodéry mají jedinou funkci a to naučit se efektivní reprezentaci dat, takovým způsobem, aby zároveň efektivně rekonstruovaly vstupní data na výstupu [1]. Celkově jde o kompresi a dekompresi dat. Při kompresi se vstupní data transformují do interní reprezentace, která zachycuje klíčové vlastnosti (prvky) dat. Následná dekomprese rekonstruuje data z interní reprezentace s minimálním rozdílem od vstupních dat celého autoenkodéru. To pomáhá odstranit šum a čistěji znázorňovat důležité rysy, které chceme ze vstupních dat získat [1]. Po předtrénování autoenkodérů se přidá klasifikační vrstva na vrchol poslední vrstvy. Pro lepší představu modelu autoenkodéru viz. Obr. 1.1.



Obr. 1.1: Grafické zobrazení trénování autoenkodéru [1]

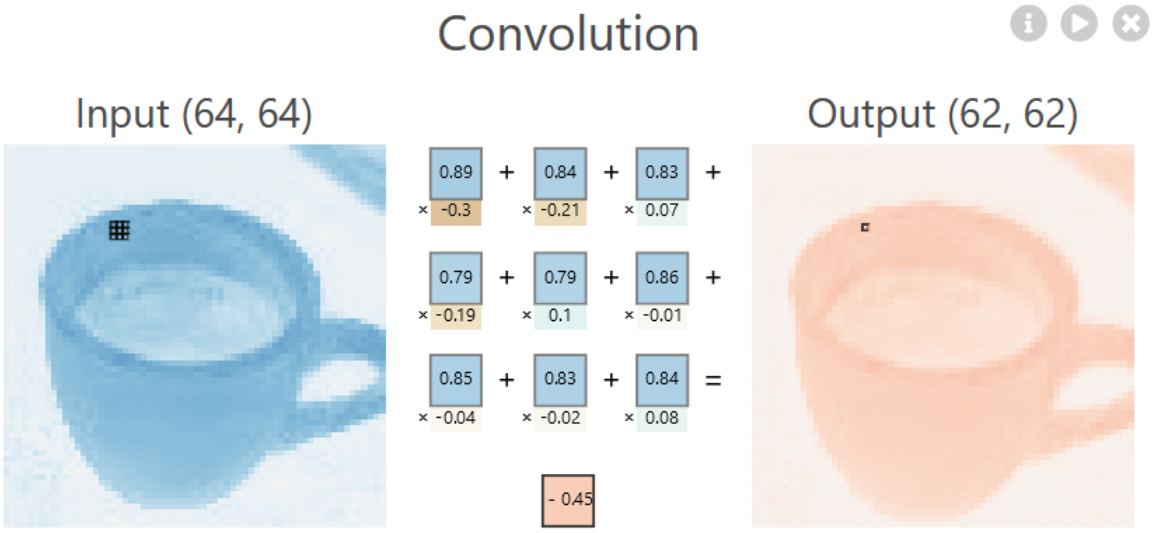
Modely neuronových sítí používají více vrstev jako například vstupní (nízké úrovně), několik tzv. skrytých (hidden) vrstev, což mohou být i naše autoenkodéry a následně vrstvy výstupní (vysoké úrovně). Proto jde o hierarchický postup. Nižší úrovně vrstev zachytí jednodušší vzory a vyšší úrovně tyto vzory kombinují. Výstupem bude složitější reprezentace dat.

Následně musíme provést trénování s dozorem, kdy předložíme označená data (vstupní datasety) a určíme modelu, že má klasifikovat vstupní data k natrénovaným třídám. Poté se vypočítá chyba klasifikace na základě rozdílu mezi predikovanou klasifikací a skutečnou třídou. Gradientním sestupem se váhy upravují tak, aby nám tato chyba vyšla co nejmenší. Gradientní sestup se počítá pomocí parciálních derivací, které nám ukazují směr největšího růstu chyby [9]. Váhy jsou parametry ovlivňující "důležitost" výstupních parametrů jednotlivých neuronů. Finální krok je optimalizace vah, aktualizace vah ve směru klesajícího gradientu a optimalizace klasifikační vrstvy na základě chyby klasifikace. Model by potom měl být schopen lépe přiřazovat správné třídy vstupním datům [10].

1.2 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou založeny na stejném principu jako klasická neuronová síť umělé inteligence (se vstupními daty provedeme matematické operace jako například skalární produkt a nelineární funkce). Každá vrstva bude fungovat úplně stejně. Nejdříve se vstupní obraz zpracuje na první vrstvě, jejímž výstupem budou váhy, v poslední vrstvě se potom vyhodnotí ztrátová funkce s klasifikací. Rozdíl je v tom, že konvoluční sítě rozeznávají spíše vzory ve vstupních datech, což je více užitečné pro rozpoznávání objektů. Tyto sítě obsahují takzvané konvoluční vrstvy,

které extrahují různé vzory. Tyto vzory mohou zahrnovat různé úrovně detailů, od základních prvků, jako jsou hrany a textury, až po složitější struktury, jako jsou části objektů nebo celé objekty. Každá konvoluční vrstva používá sadu filtrů (kernelů), které se aplikují na vstupní data. Tyto filtry jsou malé matice, které se posouvají přes vstupní obraz a provádějí konvoluci, což je operace, která kombinuje hodnoty pixelů s váhami ve filtru. Celý tento proces je velmi důležitý pro samotnou klasifikaci. Jednotlivé vrstvy jsou napojeny na konvoluční vrstvy, v tomto spojení se nachází unikátní kernel, který provede konvoluci obrazu (viz. Obr. 1.2) a vytvoří výstup konvolučního neuronu. Konvoluční neuron provede elementární bodový součin s jedinečnými váhovými maticemi a výstupem minulé vrstvy sítě daného neuronu (viz. Obr. 1.2) [2].



Obr. 1.2: Grafické zobrazení konvoluce na vstupu konvoluční vrtsvy [2]

Dostaneme stejný počet mezi výsledků jako je váhových matic. Konvoluční neuron je výsledkem součtu těchto matic a jejich naučenému bias. Přidání bias nám pomáhá lépe zachytit různé vzory pomocí počátečního posunutí aktivační úrovně nezávislé na vstupu. Tím pádem nám vstupní data neovlivňuje pouze váhový vektor (nebo matice) [2].

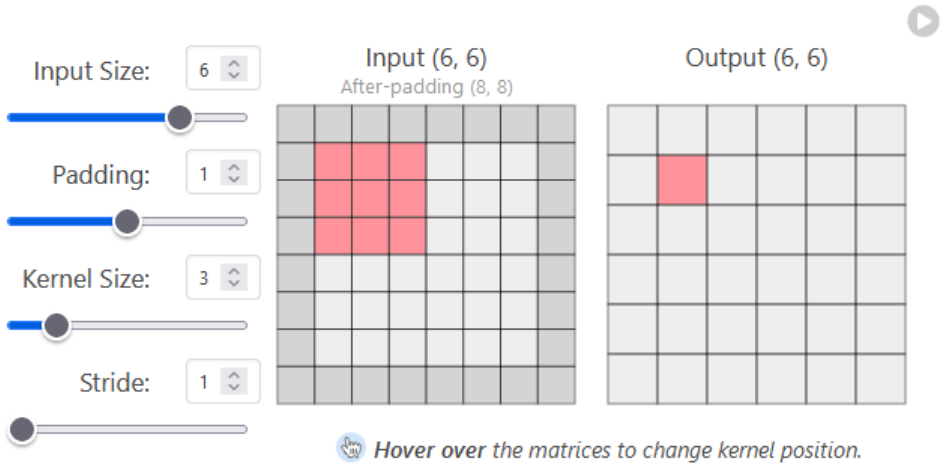
$$y = \text{aktivace}(\text{váha} \times x + \text{bias})$$

Velikost kernelů je definovaný hyper-parametr, který určil návrhář dané sítě před samotným trénováním. Hyper-parametrů je více, ovlivňují proces učení, v nějakých případech i strukturu sítě. V rámci samotných kernelů existují tyto hyper-parametry: padding ("polštářování"), velikost kernelu, krok a vstup (viz. Obr. 1.3) [2].

Padding upraví rozměr vstupního obrazu, tudíž nám přidá dodatečné prázdné nebo konstantní hodnoty okolo okrajů vstupních dat. To pomáhá zachovat data na okraji aktivačních map, což vede k lepším výsledkům, neboť to zachová počáteční rozměr vstupních data a návrhář může udělat síť hlubší. Je mnoho druhů paddingu, ale nejčastěji se používá tzv. "zero-padding", které přidá nuly okolo všech okrajů vstupu [2].

Velikost kernelu, často nazývaná i velikostí filtru, je velikost okna/filtru, které se posouvá přes vstupní data po definovaném kroku a postupně pokryje celou plochu vstupních dat. Při každém kroku provede konvoluci. Výběr správné velikosti filtru má obrovský vliv na výsledek klasifikace. Menší filtry nám dají velmi mnoho lokálních rysů za cenu delšího výpočetního času a větší dimenze výstupu kernelu. Zatímco větší kernely výrazně zmenší množství výstupních dat, nezaznamenají tolik rysů. Extrahují pouze rysy, které nejsou detailní, což může vést k horší klasifikaci. Volba správné velikosti kernelu záleží na úkolu sítě a datasetu. Nicméně menší velikost filtru bude efektivnější pro klasifikaci obrazu [2].

Krok (anglicky stride) určuje o kolik pixelů se filtr posune. Při velikosti kroku 1 a velikosti filtru 3 se provede bodový součin na okně vstupu 3×3 v pozici [0,0]. Následně se filtr posune na pozici [0,1] kde provede stejnou operaci a takto to udělá pro celý vstup. Úměra velikosti kroku k přesnosti na výstupu je obdobná jako u velikosti kernelu. Čím menší krok tím více rysů je zapamatováno a naučeno, ale opět nám roste dimenze na výstupu. Naopak větší krok sice provede operace rychleji s menší dimenzí na výstupu, ale vznikne horší výsledek především u klasifikace obrazu [2].

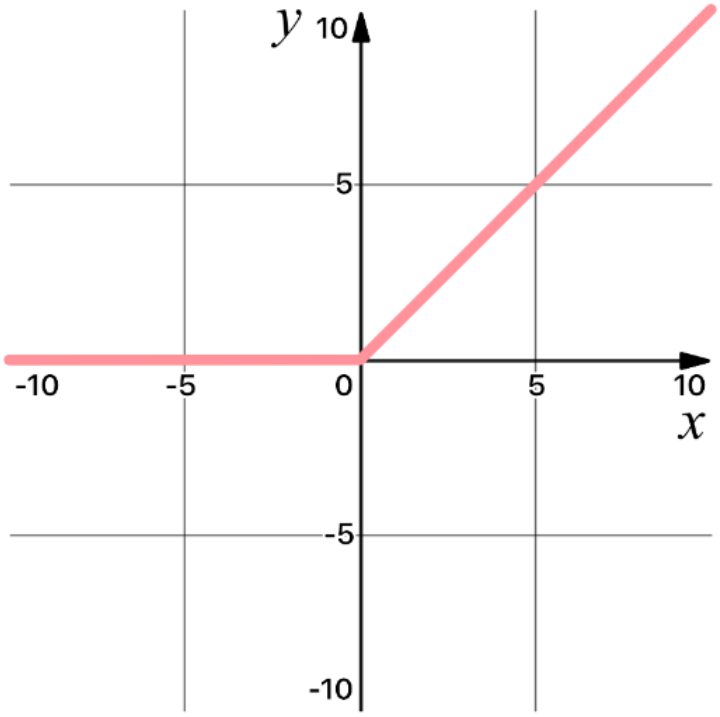


Obr. 1.3: Jednoduché zobrazení vlivu hyperparametrů na váhovou matici [2]

Výše jsme zmínili aktivační funkci této vrstvy, pojďme si ji tedy více popsat. Aktivační funkce nám přidává již výše zmíněnou nelinearitu modelu, což je důležité pro

vytvoření nelineárních rozhodovacích hranic (rozdělení vstupních dat na jednotlivé třídy). Aktivační funkci používáme, aby výstup nešel zapsat jako lineární kombinace vstupu, neboli abychom mohli zaznamenávat složitější vzory. Dvě z takových funkcí jsou ReLU (Rectified Linear Unit) a Softmax [2].
ReLU je, i přes svůj jednoduchý princip "jedna-ku-jedné", velmi efektivní při trénování. Funkce ReLU je zobrazená na Obr. 1.4.

$$\mathrm{ReLU}(x) = \max(0, x)$$

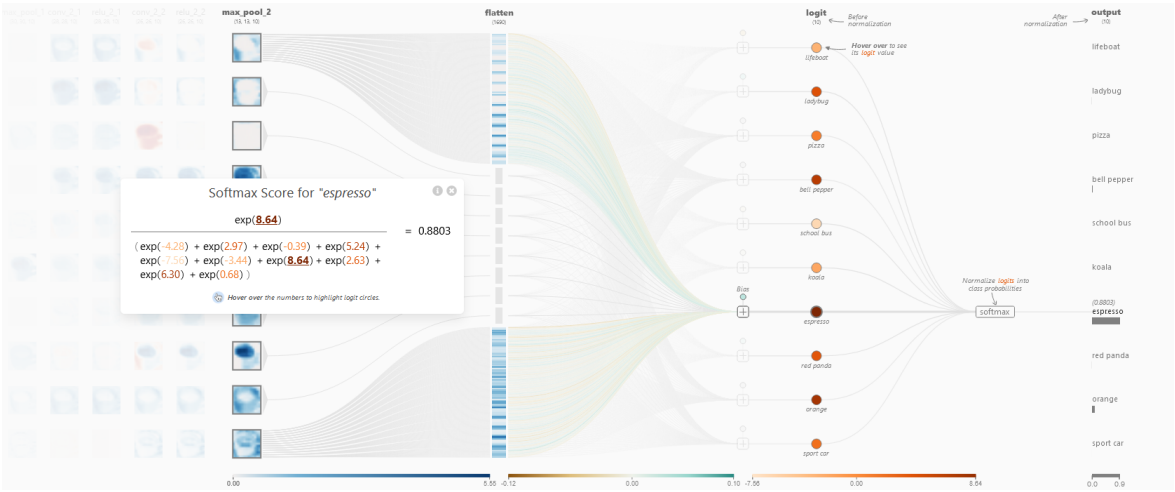


Obr. 1.4: Grafické zobrazení ReLU funkce [2]

Aktivační funkce je operace, která se provede na každé hodnotě vstupního tensoru. Pokud bychom použili funkci ReLU na hodnotu $3,56$, výsledek by byl $3,56$, jelikož tato hodnota je větší než nula. Aktivační funkce se provede po každé konvoluční vrstvě [2].

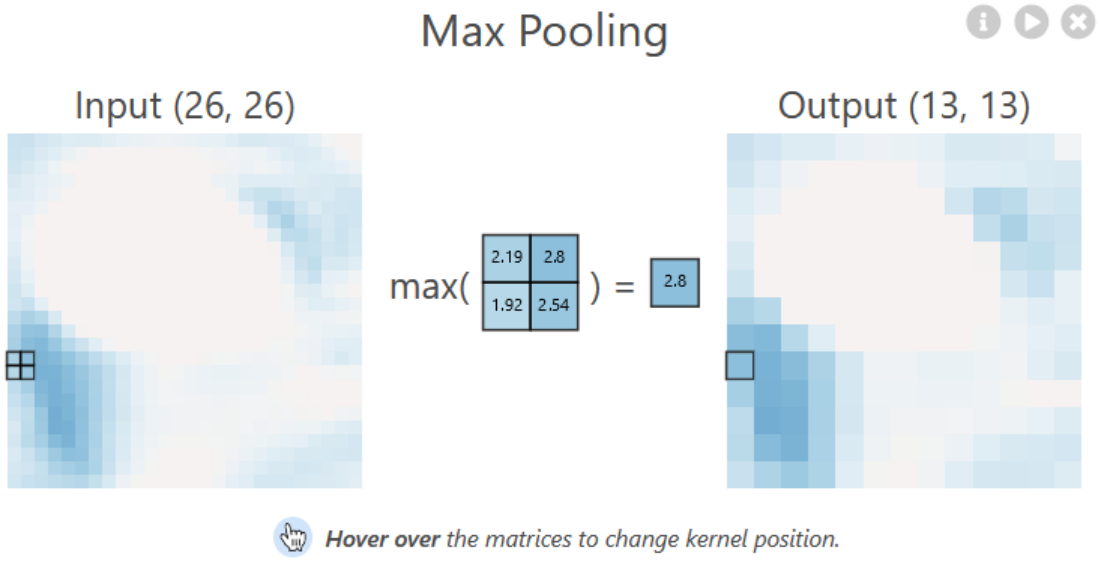
$$\mathrm{Softmax}\,(x_i) = \frac{\exp\,(x_i)}{\sum_j^K \exp\,(x_j)}$$

Softmax funkce je výhodná pro klasifikaci s více třídami, protože jejím výstupem jsou hodnoty mezi 0 a 1. Suma výstupů určujících pravděpodobnost výskytu tříd je rovna 1. Určení této pravděpodobnosti je zobrazeno na Obr. 1.5 [2].



Obr. 1.5: Zobrazení pravděpodobnosti pro jednotlivé třídy po použití Softmax funkce [2]

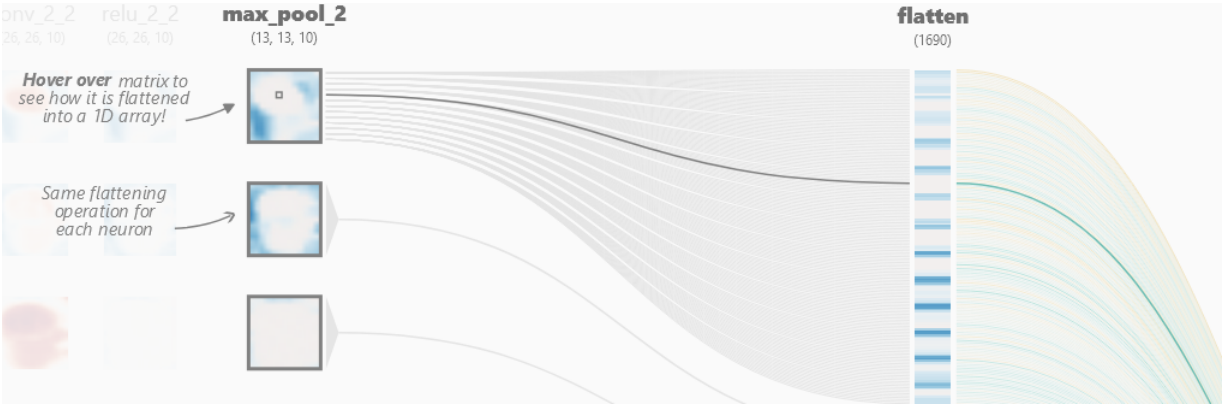
Po konvoluční a aktivační vrstvě ještě musí přijít ještě dvě vrstvy na konec celé CNN architektury. Pooling vrstva se stará o zmenšení rozměrů dimenzí prostoru se zachováním klíčových informací. Jde o celkové "zmenšení" sítě, což zmenší i dobu výpočtu. Hlavním prvkem této vrstvy je Max-pooling, který potřebuje opět velikost kernelu a kroku. Následně provede stejnou operaci, jako když jsme dělali konvoluci (aplikuje filtr na celý vstup). Akorát místo konvoluce vždy vybere největší hodnotu, kterou ve filtru najde a tuto hodnotu pošle na výstup [2]. Jednoduchý příklad najdete na obrázku 1.6.



Obr. 1.6: Grafické zobrazení pooling vrstvy [2]

Předchozí konvoluční vrstvy sítě extrahovaly rysy z vstupního obrázku, ale nyní

je čas klasifikovat tyto rysy. K tomu používáme softmax funkci klasifikace, která vyžaduje jednorozměrný vstup. Flatten ("rozmazávací") vrstva se umísťuje na výstup pooling vrstvy nebo na výstup z celé konvoluční sítě. Tato vrstva převede trojrozměrnou vrstvu v síti na jednorozměrný vektor. Převod je důležitý, protože formát vstupu klasifikační vrstvy je právě jednorozměrný vektor. Například $[5 \times 5 \times 2]$ tensor by byl převeden na vektor o velikosti 50. Proto je vrstva flatten nezbytná. Tuto vrstvu můžete zobrazit kliknutím na libovolnou výstupní třídu (viz. Obr. 1.7) [2].



Obr. 1.7: Výstup pooling vrstva do flatten vrstvy [2]

1.3 Datasety

Dataset je sada dat určená k natrénování a následnému testování algoritmů a modelů. Pro neuronovou síť může dataset znamenat jen pár pojmů, ale pro člověka hned několik. Neuronovou síť zajímají pouze vstupní data a datasety, na kterých se bude trénovat. Pro tyto datasety můžeme použít jakýkoli formát dat, ať už zvukové stopy, obrázky s definovanými objekty a ohraničením těchto objektů, nebo třeba jen text z odborných článků.
Každý dataset bude odpovídat tomu, co po specifickém modelu chceme, aby dokázal. Čím větší a kvalitnější dataset, tím lepší výsledky a menší chyba predikce. Jakýkoli dataset můžeme sami vytvořit nebo legálně obdržet z veřejně dostupných databází jako je Kaggle nebo Roboflow.

Kaggle je dceřinná firma Google LLC, která se pohybuje v oblasti datové vědy a strojového učení. Zároveň poskytuje trénink, návody a soutěže v těchto oblastech. Uživatelé zde mohou sdílet projekty, datasety a znalosti. Je to webové prostředí pro experty i amatéry v oblastech umělé inteligence a analýzy dat. [11] [5]
Roboflow je partnerská platforma Ultralytics LLC, kde mají vývojáři možnost také sdílet svoje projekty a datasety. Její hlavní funkcí je vytváření datasetů kompatibilních s YOLO modely. Tato platforma nám povolí nahrát dataset, označit objekty

pomocí tzv. bounding boxů a následně je exportovat ve formátu vhodném pro trénink dané verze YOLO modelu [12]. Roboflow nám dokáže také rozšířit dataset o několik prvků, jako např. zrcadlit obrázky přes horizontální nebo vertikální osy, rotovat obrázky o definované úhly. Tato funkce pomůže ke zvětšení datasetu a tudíž lepšímu tréninku modelu.
V našem případě musíme poskytnout obrázky osob ve zhoršené viditelnosti. Na síti Roboflow je dostupný dataset osob v mlze, který byl použil pro trénování 2.3. Náš vstupní dataset obsahuje v základu 883 fotografií osob v mlze, smogu a dešti.[13] Při hledání datasetu je důležitá různorodost dat, jako je například jiný úhel pohledu na osoby (některé pootočené o 30 stupňů apod.) nebo třeba různé vzdálenosti a velikost detekovaných objektů. Samozřejmě i jiná viditelnost na fotografiích pomůže různorodosti trénovacích dat.
Můžeme zvýšit tuto různorodost pomocí augmentace fotografií, například tím, že některé vložíme podruhé pouze pootočené o 90 stupňů. Pro rozšíření našeho datasetu by se dal také použít nástroj Albumentations, což je Python knihovna pro augmentaci obrazu. Albumentations lze integrovat se známými frameworky jako PyTorch i TensorFlow a zároveň dokáže upravit obraz mnoha způsoby.[14] V základu augmentace je pouze zrcadlení nebo pootočení fotografie, Albumentations ale dokáže i přidat stíny, dodat umělou mlhu, přidat déšť nebo sníh "na zem", rozmazat obrázek jako by byl v pohybu a mnohem více.
Po rozšíření pomocí augmentace máme 1015, následně jsme rozdělili náš dataset na tři sady: trénovací, testovací a validační. Trénovací sada fotografií bude zhruba 85 % našeho rozšířeného datasetu. Tato sada je použita na samotné trénování, tudíž je největší v obsahu. Při trénování se model naučí právě z těchto obrázků většinu parametrů objektů, které hledá, aby následně mohl určit objekty i u dat, které nikdy neviděl.
Validační sada je určena k zpřesnění parametrů při tréninku, aby náš model měl ty nejlepší výsledky. Jeho užití je v optimalizační vrstvě, kde používáme například gradient descent. Testovací sada je právě sada obrázků, které model nikdy neviděl, abychom mohli určit, jak přesný vlastně je. Hlavní účel je potvrdit, že model dokáže provést, na co byl natrénován, a následně nás ujistit, že model má dobré výsledky i na zcela nových datech. Naše testovací sada bude 5 % a validační sada bude 10 % z celého datasetu.

1.3.1 Klasifikace

Termínem klasifikační problém je myšlen problém identifikace jednotlivých kategorií. V oblasti umělé inteligence je několik klasifikačních typů: binární, multi-class (více tříd) a multi-label (více "štítků") [7].

Klasifikace binární je jednoduchou klasifikací typu ano/ne, kdy je potřeba pouze rozhodnout zda něco je nebo není pravda. Například při kontrole kvality výrobku, kde naším jediným zájmem je vědět, zda kus z výrobní linky je "přijatelný" nebo "nepřijatelný". Obsah trénovacího datasetu by byl soubor parametrů jako rozměr, velikost, hmotnost a jiné charakteristiky. Celý dataset by byl "oštítkován" pouze jako "valid" a "invalid" na základě minimální požadovaných parametrů, aby byl výrobek zařazen do provozu [7].
Datasety s Multi-class klasifikací jsou datové sety, kde na jednotlivých datech je pouze jedna klasifikační třída. Jednoduchý příklad se dá vysvětlit na klasifikaci, zda na obrázku je "savec", "pták" či "ryba". Náš trénovací dataset by obsahoval obrázky těchto kategorií a vždy by jedna kategorie odpovídala jednomu obrázku. Tudíž následně po předložení vstupních dat, která bychom chtěly kategorizovat do jednotlivých tříd, by se musel náš model rozhodovat pouze nad tím, že pes na obrázku je "savec" a není "pták" ani "ryba"[7].
Multi-label klasifikace taky rozhoduje o několika různých kategorií, ale narozdíl od multi-class je zde účelem, aby na jednom obrázku (vstupním datu) byla více než jedna třída. Tudíž kdybychom jí předložili například fotku skupiny zvířat, dokázala by určit, že na fotce je savec, pták i ryba [7].

1.4 YOLO - You Only Look Once

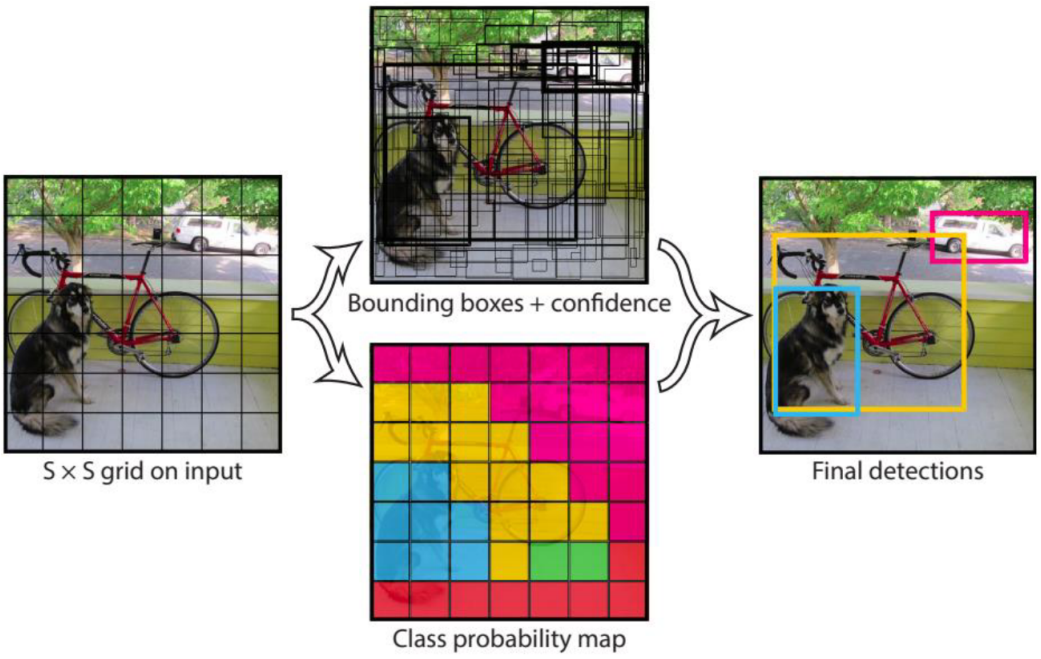
V roce 2015 publikoval Joseph Redmon s kolegy systém pro detekování objektů, který zvládne provést všechny fáze detekce objektu pomocí jedné jediné neuronové sítě. Tento model udělal detekci jednoduchou pouze tím, že z ní udělal jediný regresní problém. V jeden moment předpoví několik objektů a pravděpodobnosti jednotlivých tříd těchto objektů přímo ze surového obrázku. YOLO je už několik let špička co se týče rychlosti, korektnosti klasifikace a velikosti [3].

Od roku 2015 prošel model dohromady 8 aktualizacemi. Redmon dělal na prvních třech verzích. Po jeho ukončení kariéry vznikla verze 4, kterou pomáhal vytvořit Alexey Bochkovskiy, který se inspiroval strukturou navrhnutou Redmonem. Následného vývoje na YOLO se ujala firma Ultralytics LLC, která dělala na posledních 4 verzích tohoto modelu a přizpůsobili jej na PyTorch framework.

YOLO používá data z původního obrázku, následně je pošle přes konvoluční neuronovou síť. Jak jsme si už řekli, neuronové sítě neví, co jsou obrázky v tradičním smyslu, jak je vnímáme my lidé. Pro neuronovou síť je obrázek pouze maticí čísel, která reprezentují intenzitu pixelů. Aby YOLO algoritmus mohl efektivně detekovat objekty, rozděluje obrázek na jednotlivé sektory, což jsou malé části obrázku uspořádané do mřížky. Každý sektor je analyzován samostatně, aby se zjistilo, zda obsahuje střed nějakého objektu. Tímto způsobem může YOLO rychle a efektivně

identifikovat a lokalizovat objekty v obrázku. Rozdělení obrázku na sektory umožňuje algoritmu soustředit se na menší části obrázku a zlepšuje jeho schopnost detekovat objekty různých velikostí a tvarů. Tento přístup také pomáhá zjednodušit problém detekce na úroveň, kterou může neuronová síť efektivně řešit.

Následně si prohlédne několik ostatních pixelů kolem obrázku a usoudí, o jaký předmět z datasetu by mohlo jít. Algoritmus YOLO funguje na jednoduchém principu. První verze algoritmu se zaměřila vždy na jednotlivé části obrázku, který ji byl předložen. Obrázek byl tedy rozdělen do mřížky velikosti 7×7. Následně pokud algoritmus objeví ve středu nějakého sektoru mřížky hledaný objekt, pak tento sektor bude detekovat daný objekt s jistou pravděpodobností. Tato pravděpodobnost nám poslouží k nákresu tzv. bounding boxu neboli rámečku ohraničujícího náš objekt v obrázku. Tento rámeček určí i pozici objektu v celém původním obrázku.(viz. Obr. 1.8).



Obr. 1.8: Na vstupní snímek byl aplikován model YOLO s mřížkou 7×7 [3]

Algoritmus nám ještě pořád nedokázal říct co jsme detekovali, zatím víme , že nějaké z detekovatelných tříd jsou na obrázku. Víme i kde se zhruba nachází, takže poslední informací co chceme, je zjistit co vlastně nalezl. To se určí pomocí jednotlivých pravděpodobností v každém sektoru našeho ohraničení. Tato pravděpodobnost určí "kolik" objektu tam je nebo kolik ho tam není v rozsahu 0 až 1. Každé ohraničení by teda mohlo být reprezentováno vektorem s pouze 5 složkami, ale pokud budeme mít více tříd, tak nebudeme hodnotit pouze pravděpodobnost výskytu v rámci jedné

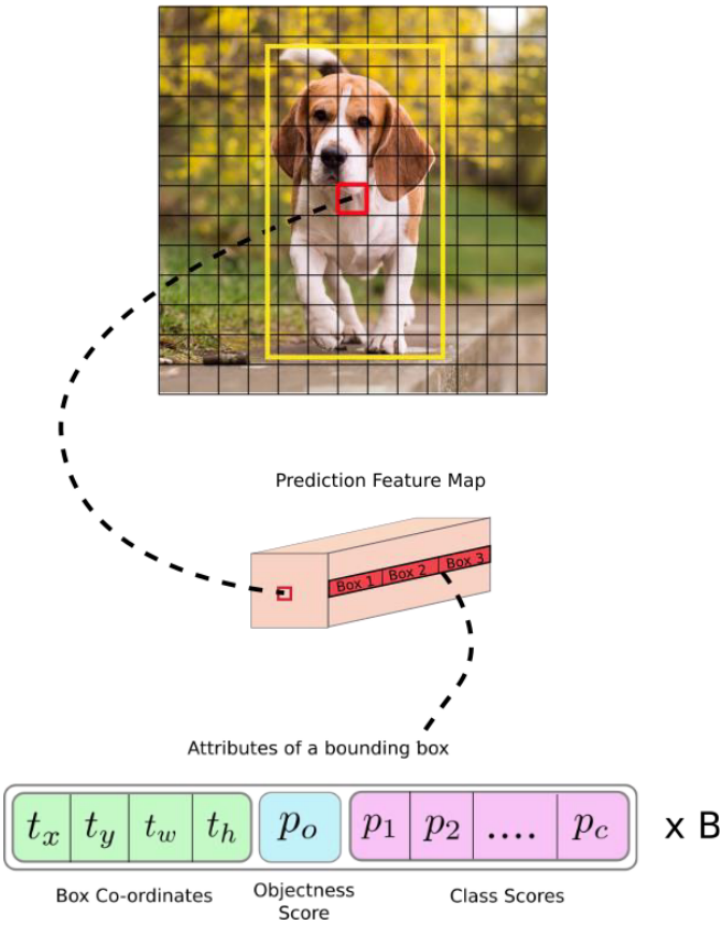
třídy. Tím nám vzniknou větší vektory o velikosti.

$$\underbrace{(x, y, w, h, a_1, ..., a_n)}_{\text{dimenze} = 4 + \text{počet tříd}}$$

Výstupní tensor jednoho sektoru je zobrazený na Obr. 1.9 [3].

Neboli

[velikost mřížky×(5·pravděpodobnost pozice ohraničení+pravděpodobnost jednotlivých tříd)] (1.1)



Obr. 1.9: Grafické znázornění celkového výstupního tensoru jednoho sektoru [3]

Jelikož rozdělíme obrázek na mřížku a v každém sektoru určujeme pravděpodobnosti, zda je zde objekt, musí algoritmus mít ztrátovou funkci. Kdyby nám chyběla ztrátová funkce, byla by přesnost predikce nestabilní, protože spousta sektorů mřížky by měla pravděpodobnosti 0 u každého objektu. Ztrátová funkce vyhodnocuje jak dobře model predikuje výstupy pro daná vstupní data ve srovnání s reálnými hodnotami. Dá se použít například suma čtverců odchylek, což by byl součet čtverců

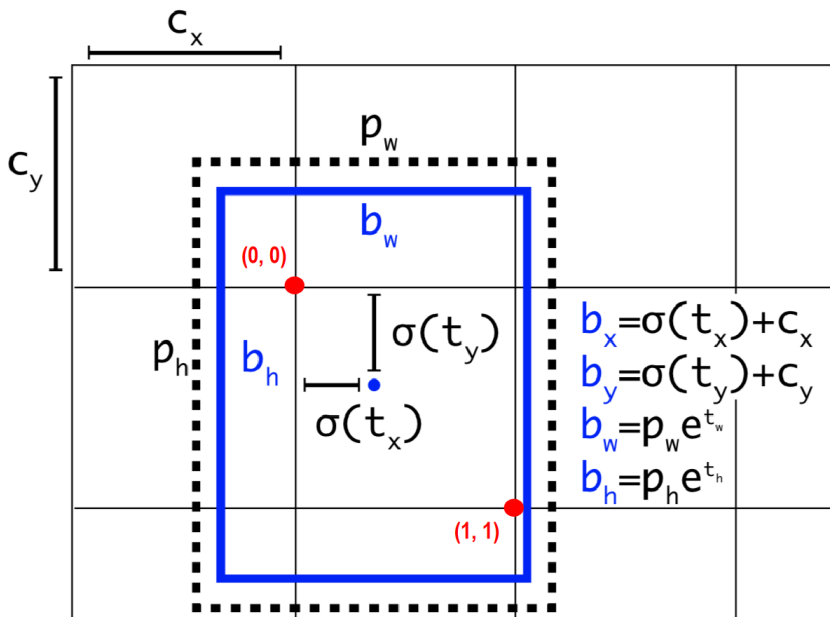
rozdílů mezi predikcemi modelu a skutečnými hodnotami. Konkrétně tato metoda zdůrazňuje velké odchylky díky druhým mocninám [3].

Ačkoliv existují i jiné modely než YOLO jako například MobileNet V2, tak YOLO je skutečně teď na špičce co se rychlosti, přesnosti a velikosti týče. Po tom, co se vývoje chytlo Ultralytics LLC a celé YOLO bylo předěláno na PyTorch framework, je daleko jednodušší implementovat tento model na naše zařízení.

YOLO prošlo od začátku velkými změnami, zde si popíšeme ty nejdůležitější, které proběhly v prvních pěti verzích.

V druhé verzi nám přibyla například normalizace dávky nebo konvoluce s kotvou. Normalizace dávky (Batch normalization) se stará o normalizaci vstupních dat jednotlivých vrstev za účelem stabilizace a urychlení konvergence. V každé dávce se vypočte průměr a směrodatná odchylka příznaků, následně se každý příznak normalizuje, normalizovaná data jsou transformována do dvou parametrů (váhy, posun), které se učí během trénování [3].

Konvoluce s kotvou se stará o přesnější určení objektů v obraze. Ve vstupních (předložených) datech jsou už pevně definované anchor boxy (kotvy) s vygenerovanými predikcemi (zda a jak je box obsazen objektem). Následně jsou predikce upraveny a použijí se boxy s nejvyšší pravděpodobností na obsahování chtěného objektu. Tento proces urychluje detekci objektů různých tvarů a velikostí (viz. Obr. 1.10) [3].

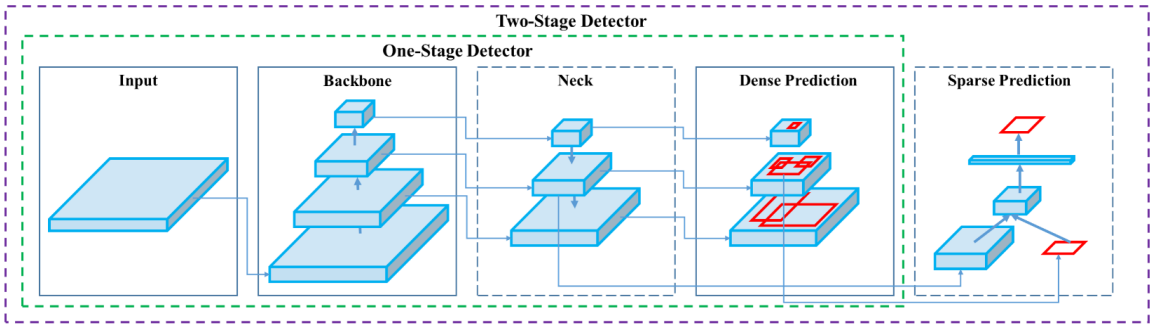


Obr. 1.10: Zobrazení predikce pozice bounding boxu [3]

Ve třetí verzi algoritmu byla změněna architektura algoritmu na ResNet (Residual Network). Vyřešil se tak problém „ztracených gradientů". Pokud byla síť moc

hluboká mizela informace o gradientu při zpětné propagaci. To výrazně ztížilo trénování. ResNet umožňuje pomocí "residuálních bloků" přeskočit několik vrstev a usnadnit tak zpětnou propagaci. Residuální bloky mají dvě větve. Jedna větev vykoná funkce na vstupu vrstvy, druhá přenese vstupy na další vrstvu [3].

Ve čtvrté verzi se změnila architektura na formu Backbone, Neck, Head (viz. Obr. 1.11). Backbone je základní struktura sloužící k extrakci obecných příznaků obrázků. Neck obsahuje operace na zvýraznění detailů v obrazu. Head je zodpovědná za konečný výstup modelu (predikce, klasifikaci, poloha objektu) [3].



Obr. 1.11: Grafické znázornění architektury s Backbone, Neck a Head, kde Dense a Sparse prediction tvoří Head [3] [4]

Pátá verze se převedla do PyTorch frameworku a byla opět mírně pozměněna architektura se zachováním backbone, neck, head strukturou [3].

1.5 Snížená viditelnost v kontextu detekce osob

Snížená viditelnost je definována jako stav prostředí, při kterém je vzdálenost viditelnosti redukována pod standardní hodnoty. Tento stav může být způsoben různými atmosférickými jevy, mezi které jsou řazeny především mlha, déšť, sněžení, prach nebo kouř [15]. V kontextu této práce je snížená viditelnost primárně reprezentována mlhou, která představuje jeden z nejčastějších a nejproblematičtějších faktorů ovlivňujících schopnost detekce objektů.

1.5.1 Charakteristika snížené viditelnosti

Snížená viditelnost je charakterizována redukcí kontrastu mezi objektem a jeho pozadím, což vede k obtížnější identifikaci objektů v zorném poli. Tento jev je způsoben rozptylem světla na částicích přítomných v atmosféře, což má za následek změnu optických vlastností prostředí [16]. Mlha je definována jako stav, kdy mikroskopické kapky vody redukují horizontální viditelnost na zemském povrchu na méně než 1 km.

Míra redukce viditelnosti závisí na struktuře mlhy, zejména na hustotě a velikostní distribuci kapiček. Při osvětlení jsou jednotlivé kapky mlhy často viditelné pouhým okem a lze pozorovat jejich mírně turbulentní pohyb. Mlha vytváří bělavý závoj pokrývající krajinu.[15]. Tyto charakteristiky mlhy představují významné výzvy pro systémy detekce osob, neboť ovlivňují optické vlastnosti prostředí a tím i kvalitu vstupních dat pro detekční algoritmy. Z fyzikálního hlediska je mlha klasifikována podle hustoty a s ní související viditelnosti. Světová meteorologická organizace definuje lehkou mlhu při viditelnosti do 500-1000 metrů, mírnou mlhu při viditelnosti do 200-500 metrů, hustou mlhu při viditelnosti do 50-200 metrů a velmi hustou mlhu při viditelnosti pod 50 metrů [17]. Tyto kategorizace jsou důležité pro stanovení podmínek, ve kterých musí detekční systémy spolehlivě fungovat.

1.5.2 Vliv snížené viditelnosti na detekci osob

Detekce osob v podmínkách snížené viditelnosti přináší specifické výzvy. Redukce kontrastu způsobená mlhou vede ke snížení rozdílů mezi objektem a pozadím, což komplikuje proces segmentace obrazu. Tento jev je dále umocněn změnou barevných charakteristik, kdy jsou barvy v mlze vnímány jako méně saturované a s posunutým barevným spektrem [18]. Dalším významným faktorem je neostrost hran, kdy obrysy objektů jsou méně definované, což představuje výzvu pro algoritmy detekce založené na hranové detekci. Variabilita podmínek snížené viditelnosti také vyžaduje adaptabilní přístup, neboť hustota mlhy se může v čase a prostoru rychle měnit [19]. Pro účely této práce byl vytvořen dataset obsahující obrazy osob v různých stupních mlhy, který byl použit pro trénování neuronové sítě YOLO.

2 Praktická část bakalářské práce

2.1 Komponenty pro realizaci

V této kapitole se podíváme na důležité komponenty pro naše zařízení. Mikropočítače budou jádrem našeho systému, poskytnou výkon na zpracování obrazu a podpůrné aplikace. Kamery nám dájí obraz pro zpracování. Také zmíníme dodatečné komponenty jako třeba radar. Následně provedeme testy těchto komponent, abychom mohli vyhodnotit, které budou nejlepší pro finální zařízení.

2.1.1 Mikropočítač

Pro zařízení budeme potřebovat dostatečný výpočetní výkon na zvládnutí detekce naší jedné třídy (lidé) pomocí modelu neuronové sítě, který si natrénujeme. Tento výkon mají většinou čipy pojmenované Grafická výpočetní jednotka (GPU), Neurální výpočetní jednotka (NPU), Tensorová výpočetní jednotka (TPU). Některé z těchto čipů jsou součástí známých mikropočítačů, viz 2.1.

Pokud si shrneme, co vše víme, máme tedy sadu parametrů, podle kterých budeme vybírat komponenty zařízení. Potřebný výpočetní výkon s možností naprogramování dodatečných subprocesů nám dokážou poskytnout i mikropočítače. Parametry mikropočítače, které nás budou zajímat, jsou operační paměť (aspoň 4 GB RAM), vnitřní paměť úložiště, počet a druh portů, cena, dostupnost (po "čipové krizi" se spousta přestala nadobro vyrábět), zda má GPU/TPU/NPU. Tyto informace jsou shrnuty v tabulce 2.1.

Z tabulky 2.1 je vidět, že můžeme použít mikropočítače jako je Jetson Nano, Orange Pi 5B, KHADAS Edge2/VIM4. Variantu od firmy KHADAS jsme nakonec nezvolili pouze kvůli ceně. Coral Dev Board se později ukázalo, že byl také vyřazen z výroby během "čipové krize".

V tabulce 2.1 jsou zaneseny i USB akcelerátory, zařízení, které se dají připojit do mikropočítače a použít je jako externí NPU/TPU jednotku. V zásadě bychom jednoduchým programem řekli mikropočítači, ať na výpočty při používání modelu neuronové sítě používá externí USB zařízení. Ovšem USB akcelerátor potřebuje velmi specifické podmínky. Při testování Coral USB akcelerátoru byl obrovský problém například s kabelem. Kabel, který přišel v balení, sice byl popsán jako USB3.0, ale nebyl, tudíž nezvládl přenést tolik dat a dával velmi špatné výsledky. Druhá zrada v USB akcelerátoru byla, když nedokázal podat dobrý výkon při YOLO modelu a nebylo jednoduché jej konvertovat na správný formát. V tomto případě byly tedy USB akcelerátory vyřazeny z dalšího testování.

Pozornost byla zaměřena na firmu Luxonis, plzeňskou firmu vyvíjející kamery se zabudovaným RCV4 čipem pro zpracování modelu neuronové sítě. Firma nabízí množství zařízení, především kamery, některé i IP67 hodnocené pro prachu/vodě-odolnost. Některé jsou vybaveny PoE, což může přispět k optimalizaci velikosti zařízení. Pro testování byla vybrána OAK-1 kamera, která dosáhla uspokojivých výsledků. Luxonis spolupracuje s Roboflow, čímž je umožněna efektivnější konverze modelu do správného formátu [20].

2.1.2 Kamery

Vybrat správný kamerový senzor a dodavatele kamery byla jedna z nejnáročnějších částí této práce. Musely být vzaty v úvahu následující parametry: kvalita obrazu, zaostření na dané vzdálenosti a propojení s mikropočítačem.

Co se propojení s mikropočítačem týče, můžeme zvolit z následujících možností: ethernetový kabel, USB kabel, CSI kabel. Přes ethernetový kabel by byl kvalitní přenos obrazu na velké vzdálenosti jednoduchý. Nicméně nastavení přijímání obrazu by bylo složitější. Bylo by zapotřebí nastavit statickou IP adresu pro Raspberry Pi, zároveň IP adresu brány (routeru) a DNS serveru. Následně sběr obrazu z kamery přes RTSP (Real Time Streaming Protocol) stream.

Kdybychom použili komunikaci přes CSI, máme kameru s malým dosahem, jelikož tyto kabely jsou maximálně 20 cm dlouhé, což by nevadilo našemu účelu. Sběr dat by se dal udělat přes oficiální Raspberry Pi zdroje. Ale tyto oficiální zdroje poskytují velmi nekvalitní obraz s pomalou obnovovací frekvencí snímků.

Jak jsme výše zmínili, použijeme kameru od firmy Luxonis, takže rozhraní bude buď ethernetový kabel, nebo USB. Jelikož nám pro první koncept zařízení stačí obyčejná kamera, která výkonově zvládne zpracování obrazu pomocí YOLO algoritmu, byla tedy zvolena OAK-1 jako testovací kamera.

Kamerový senzor by byl použit buďto IMX378 s IR osvětlením pro lepší rozpoznání ve tmě (modul je už implementován u většiny kamer od Luxonisu), nebo IMX462. IMX462 je modul ze Sony série STARVIS, jenž používá STARVIS technologii. Jde o technologii, která dokáže zachytit nejslabší světlo a toto světlo přeměnit na signály bez šumu a poskytne očekávaný obraz.

Hlavní výhoda kamer značky Luxonis je čip Myriad X VPU, který je implementován jako SoC (System-On-Chip). Myriad X je výkonný procesor pro zpracování obrazu a inferenci neuronových sítí. Tato kamera je schopna provádět detekci objektů a sledování v reálném čase díky Spatial AI, což je technologie, která kombinuje hloubkové vnímání a detekci objektů. RVC2 je součástí kamery, tudíž samotná kamera nám dokáže už dát zpracovaný obraz pomocí umělé inteligence i s detekčními boxy apod.. Samotná kamera OAK-1 nám při testování držela teplotu pod 60 °C

i se spuštěným obecným modelem YOLOv5s. OAK-1 může sloužit i jako obyčejná kamera nebo na zpracování kompletní neuronové sítě. Už bude potřeba pouze natrénovat specifický model na rozpoznání člověka i ve snížené viditelnosti.

2.2 Testování mikropočítačů

V této podkapitole se zaměříme na testování mikropočítačů, konkrétně Jetson Nano, Orange Pi 5 a Coral USB Accelerator. Porovnáváme jejich parametry, teplotu při zátěži a dobu inferencí na modelu YOLOv5s, abychom získali potřebné informace o jejich výkonu a využitelnosti pro tuto práci. V tabulce níže je vidět porovnání z testů a následně popíšeme ještě určité specifické rozdíly mezi testovanými kusy.

	Jetson Nano Dev Kit 4 GB	Orange Pi 5	Coral USB Accelerator
využití RAM na neuronku	cca 1 GB	cca 1 GB	100 MB
modely	YOLOv5su	YOLOv5s (uint8, quantized, 640×640), MobileNet V2	YOLOv5su, mobilenet V1 a V2
inference modelů (224×224 px)	$90 - 100$ ms	$25 - 50$ ms	150 ms
inference modelů (640×640 px)	$90 - 100$ ms	$35 - 60$ ms	2500 ms
teploty při testování	< 50 °C	< 70 °C	< 50 °C
FPS	cca $10 - 12$	cca $16 - 28$	cca 7

Tab. 2.1: Výsledky testů pro vybrané SBC a USB akcelerátor.

2.2.1 Jetson Nano verze 4GB RAM

Při testování Jetson Nano a při použití modelu YOLOv5su jsme zjistili následující:

1. Celkové využití operační paměti při spuštění detekce modelu YOLOv5 bylo 1 GB.
2. Používá CUDA jádra na vestavěném GPU přímo od firmy NVIDIA tzn. každý model se musí převést na správný formát. Pro Jetson Nano je doporučeno použít konverzi z formátu `.onnx` do formátu `.trt` pomocí oficiálního nástroje od výrobce. Ten by měl být nainstalován ale pro jistotu ho můžeme zkusit znovu nainstalovat pomocí příkazu [21]:

   ```
   python3 -m pip install --upgrade tensorrt
   ```

   a následná konverze se provede pomocí [21]

   ```
   trtexec --onnx=resnet50/model.onnx --saveEngine=
      resnet_engine.trt
   ```

   (samozřejmě s uvedením příslušných cest k souborům).

3. Celkový čas reakční doby pro tento model a rozlišení 640×640 byla $90 - 100$ ms, z čehož vyplývá rychlost $10 - 12$ snímků za sekundu.
4. Teplota zařízení nepřekročila 50 °C.

2.2.2 Orange Pi 5B

Při testování Orange Pi 5B s použitím modelu YOLOv5s s rozlišením 640×640 jsme zjistili následující:

1. Orange Pi 5B nám může nabídnout až 16 GB RAM, ale při maximálním zatížení využila maximálně 1 GB RAM na pipeline při zpracování videa.
2. Tato deska má vestavěnou Rockchip NPU (Neural Network Processing Unit) jednotku, která se stará o veškeré zpracování naší neuronové sítě. Musí se použít opět konverze z `.onnx` na formát `.rknn`. Na tu slouží nástroj RKNN Toolkit 2. Pro následné spuštění modelu neuronové sítě na NPU jsme použili nástroj RKNPU2 [22].
3. Reakční doba zde byla 55 ms při zpracovávání pomocí NPU, z čehož vyplývá snímková frekvence zhruba 18 snímků za vteřinu.
4. Tato deska má jak LAN port, dvě rychlé USB tak i GPIO piny a při požuití externího aktivního chlazení jsme dokázali udržet teplotu pod 50 °C.

2.2.3 Coral USB Accelerator

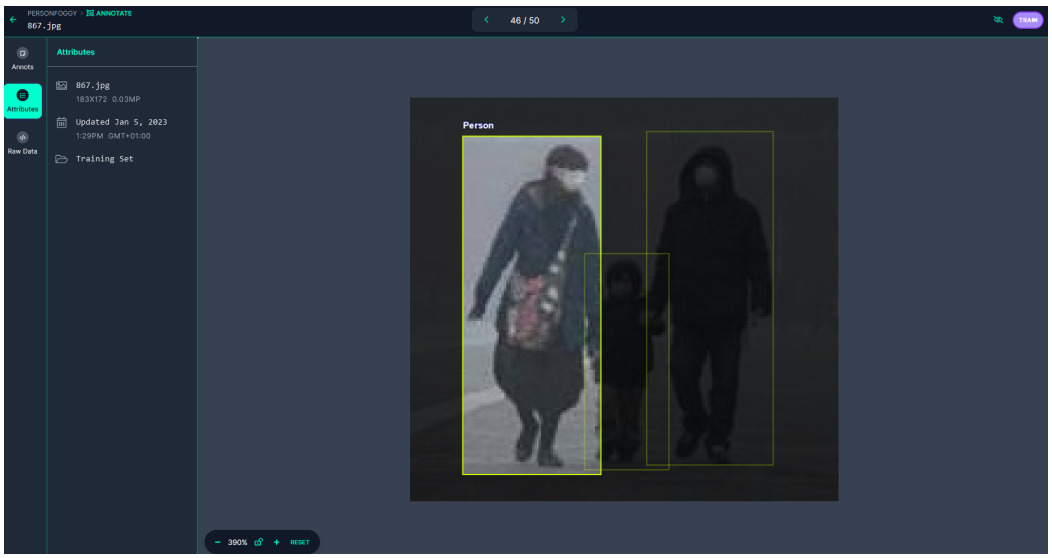
Tato deska je speciální tím, že jde pouze o akcelerátor výkonu. Samotná deska je malá a po nainstalování příslušných knihoven a stažení oficiálního frameworku od výrobce, můžete spustit na svém počítači neuronovou síť na detekci osob. Celý výkon, který NN potřebuje je v zpracován v akcelerátoru. Ovšem při prvním testování byl USB akcelerátor vyškrtnut, protože pokud jsme nepoužili model neuronové sítě od výrobce a ve formátu `.tflite` (TensorFlow Lite), byla reakční doba mezi 150 ms a 2500 ms, podle typu a rozlišení modelu. Z toho vycházejí hodnoty pod 1 snímek za vteřinu.. Tato rychlost zpracování mohla být ovlivněna použitím USB 2.0 pro přenos dat na USB akcelerátor [23].

2.3 Trénink neuronových sítí

Samotné trénování jde provádět několika způsoby, v této části si ukážeme dva z nich. Jeden nabízí už výše zmíněný Roboflow 1.3 a druhý způsob pomocí Pythonu.

2.3.1 Roboflow platforma

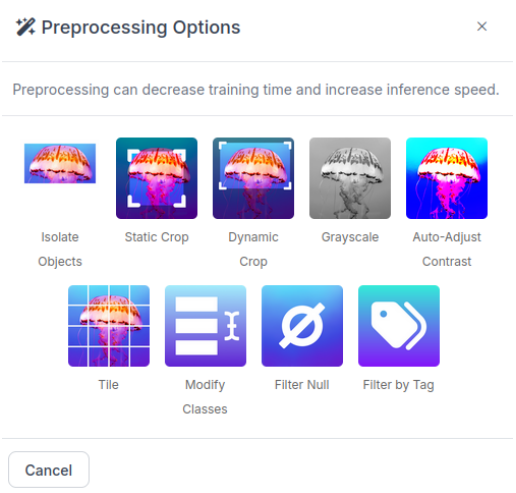
Celá platforma Roboflow nabízí totiž nejenom nástroje pro nalezení a sestrojení datasetů, ale i jeho následnou anotaci a využití při trénování až po jeho nasazení do produkce.



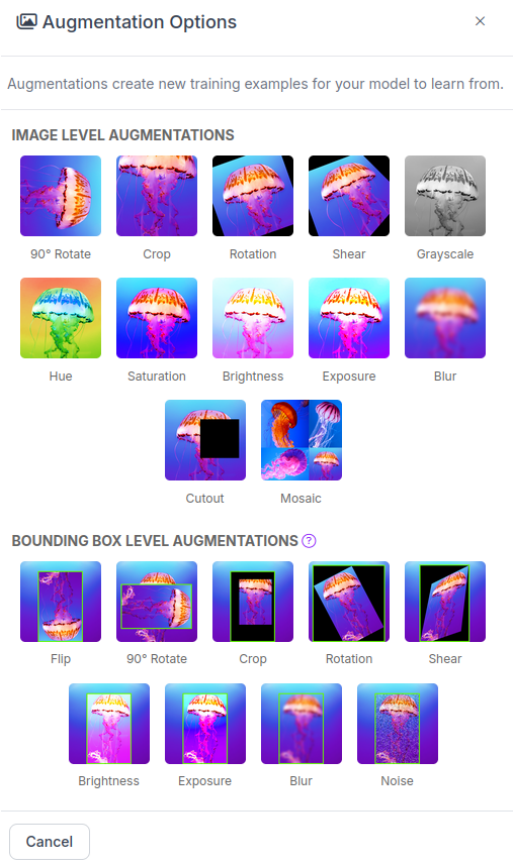
Obr. 2.1: Anotace na Roboflow platforme [5]

Dataset máme nachystaný, anotovaný (viz. Obr. 2.1) a rozdělení do tří kategorií na trénování, testování a validac (viz. Obr. 2.3d) modelu po trénování (viz. Obr. 2.3c). Roboflow před trénováním modelu nabízí jisté způsoby předzpracování obrazu v datasetu jako je automatické natočení obrazu, změna velikosti, konverze do černobílého obrazu, automatické vyvážení kontrastu a jiné (viz. Obr. 2.2a).

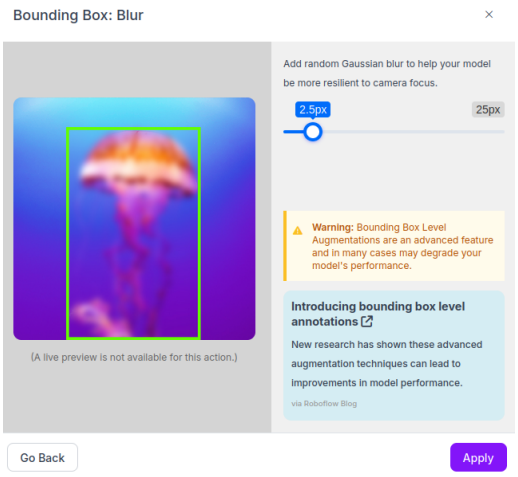
Co se augmentace týče, může být zvoleno otočení obrazu do jiných úhlů, přidání šumu nebo generování mozaiky (verze zdarma nabízí zvolit jen tři z těchto). Pro augmentaci a předzpracování byla zvolena změna rozlišení do 640×640 pixelů, pro jednotnost vstupních dat a automatickou orientaci Obr.2.2b. Pro augmentaci byly implementovány základní techniky: zrcadlení horizontálně a přidání šumu až 1.05% na pixel. Tato nastavení byla aplikována pro první verzi modelu, neboť Roboflow nabízí i přehlednou správu verzí jednotlivých modelů a jejich celkových nastavení. Předzpracování a augmentace tohoto konkrétního modelu nebyla tak významná, sloužila pouze k rozšíření datasetu. Pro druhou verzi modelu bylo implementováno přidání šumu 1% na pixel Obr. 2.2c.



(a) Předzpracování obrazu [5]



(b) Augmentace obrazu [5]



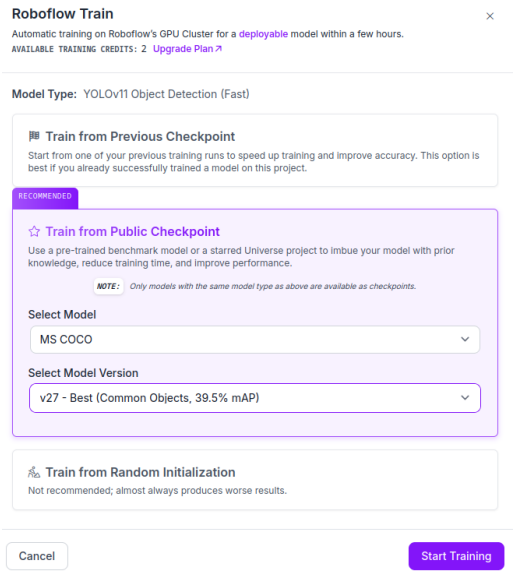
(c) Zašumění obrazu pomocí augmentace [5]

Obr. 2.2: Připravení trénovacích dat pomocí Roboflow grafického prostředí.

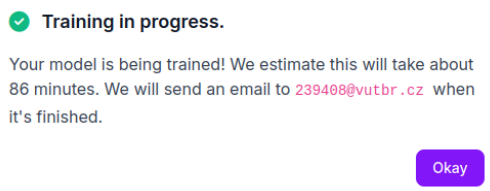
Samotné trénování je zde uděláno tzv. One-Click interface Obr. 2.3a. Po nastavení parametrů zmíněných výše si můžeme ještě nastavit nějaké hyperparametry, ale potom pouze klikneme na tlačítko trénovat a trénink samotný započne. Od počítače můžeme odejít a jakmile bude vše hotovo přijde nám email Obr. 2.3b a daný model můžeme vyzkoušet nebo stáhnout jeho váhy.

Po trénování dostáváme kompletní přehled výsledků. V přehledu výsledků najdeme základní metrik modelu jako jsou přesnost a citlivost, mAP (průměrná přesnost), zároveň nám dovolí model rovnou otestovat a vygeneruje důležité grafy z tréninku Obr. 2.3d. Přesnost určuje kolik nalezených předmětů je relevantních a citlivost určuje kolik relevantních předmětů bylo nalezeno. Průměrná přesnost (mAP) je nejdůležitější hodnotou kterou chceme získat, jelikož přesně vyjadřuje kvalitu detekce a lokalizace objektů na předložených datech. Jsou zde dvě metriky mAP50, která určuje kvalitu samotné detekce pro danou třídu (nebo několik tříd) a mAP50-95, která má posunutý IoU (Intersect over Union) mezi 50 až 95 procenty.

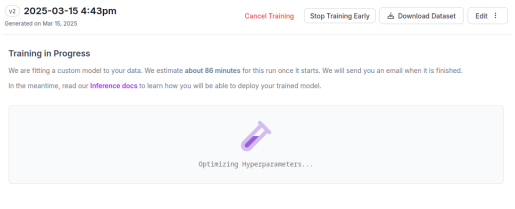
IoU (Intersection over Union) je základní metrika v počítačovém vidění, která měří přesnost detekce objektů porovnáním překryvu mezi predikovaným ohraničujícím rámečkem (bounding box) a skutečným (ground truth) ohraničujícím rámečkem. Vypočítává se jako poměr plochy průniku obou rámečků k ploše jejich sjednocení, což poskytuje hodnotu mezi 0 a 1, de hodnota blíž k 1 znamená lepší přesnost detekce.



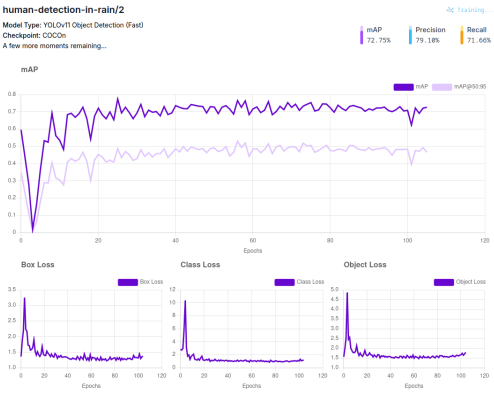
(a) One-Click interface[5]



(b) Trénování pomocí Roboflow[5]



(c) Notifikace pomocí emailu[5]



(d) Shrnutí po trénování[5]

Obr. 2.3: Zobrazení trénování pomocí Roboflow platformy

2.3.2 Trénování modelu pomocí Pythonu a YOLOv5

Samozřejmě na trénování nepotřebujeme drahé nástroje, za které by veřejná osoba musela platit, navíc Roboflow nenabízí ani zdaleka takové možnosti augmentace jako výše zmíněné Albumentations, což je python knihovna. Ukažme si tedy jak natrénovat vlastní síť pomocí Pythonu.

Použijeme YOLOv5 na natrénování nového modelu, kde využijeme náš anotovaný dataset z Roboflow. Celé trénování budeme provádět v Google Colab prostředí. Zde můžeme při vytvoření našeho sešitu zvolit jako výpočetní jednotku NVIDIA Grafickou kartu (konkrétně T4), což výrazně urychlí náš trénink.
Nejprve si naimportujeme důležité knihovny. Budeme používat PyTorch, abychom

mohli využít tensorové počty a CUDA jádra pro tyto výpočty. OpenCV pro zpracování obrazu při testech našeho modelu. Matplotlib pro vizualizaci jednotlivých výsledků při trénování a testech. Roboflow pro import datasetu. PyYAML pro lepší nastavení konfigurace našeho tréninku.

```
!pip install torch torchvision pytorch-ignite ultralytics
    roboflow

!git clone https://github.com/ultralytics/yolov5
%cd yolov5
!pip install -r requirements.txt

import torch
import torchvision
import os
import yaml
from roboflow import Roboflow
```

Připravíme si náš anotovaný dataset. Roboflow nám poskytuje API, přes kterou si jej můžeme importovat do Google Colab nebo do jiného prostředí podle našeho výběru. Dokonce můžeme rovnou extrahovat dataset ve správném formátu pro YOLOv5 bez nutnosti konverze.

```
rf = Roboflow(api_key="<YOUR_API_KEY>")
project = rf.workspace("<WORKSPACE_NAME>").project("<
   PROJECT_NAME>")
dataset = project.version("<VERSION>").download("yolov5")

data_yaml_path = os.path.join(dataset.location, "data.
   yaml")

with open(data_yaml_path, 'r') as f:
    data_config = yaml.safe_load(f)

print(f"Classes: {data_config['names']}")
print(f"Number of classes: {data_config['nc']}")
```

Specifikujeme si cesty k důležitým souborům (testovací a validační obrazy a jejich označení tříd). Následně je přetvoříme na soubor datasetu a pak načteme váhy z YOLOv5 a stanovíme ať se trénuje na GPU pokud je dostupná. Pokud GPU nebude dostupná, program sám bude trénovat přes CPU (procesor). Po stažení si dataset nakonfigurujeme pomocí YAML souboru, specifikujeme strukturu datasetu a

informaci o třídách (máme jednu). Soubor obsahuje cestu k trénovacím a validačním složkám, seznam a počet tříd.

Nemusíme si psát vlastní architekturu, protože tu nám poskytuje YOLOv5. Tudíž pro trénování použijeme pouze předpřipravený skript od Ultralytics [12] se správnými parametry.

```
!python train.py --img 640 --batch 16 --epochs 50 --data
   custom_data.yaml --weights yolov5s.pt
```

Kde:

- `-img 640`: Nastaví rozlišení vstupního obrazu na 640×640 pixelů
- `-batch 16`: Použije dávku (batch) o velikosti 16 obrazů na jednu iteraci
- `-epochs 50`: Trénování proběhne v 50 epochách, kdy jedna epocha představuje jeden průchod celým datasetem
- `-data ../custom_data.yaml`: Specifikuje cestu ke konfiguračnímu YAML souboru s informacemi o datasetu
- `-weights yolov5s.pt`: Inicializuje model předtrénovanými vahami z modelu YOLOv5s

Po 50 epochách tréningu dosáhl náš model následujících výsledku:

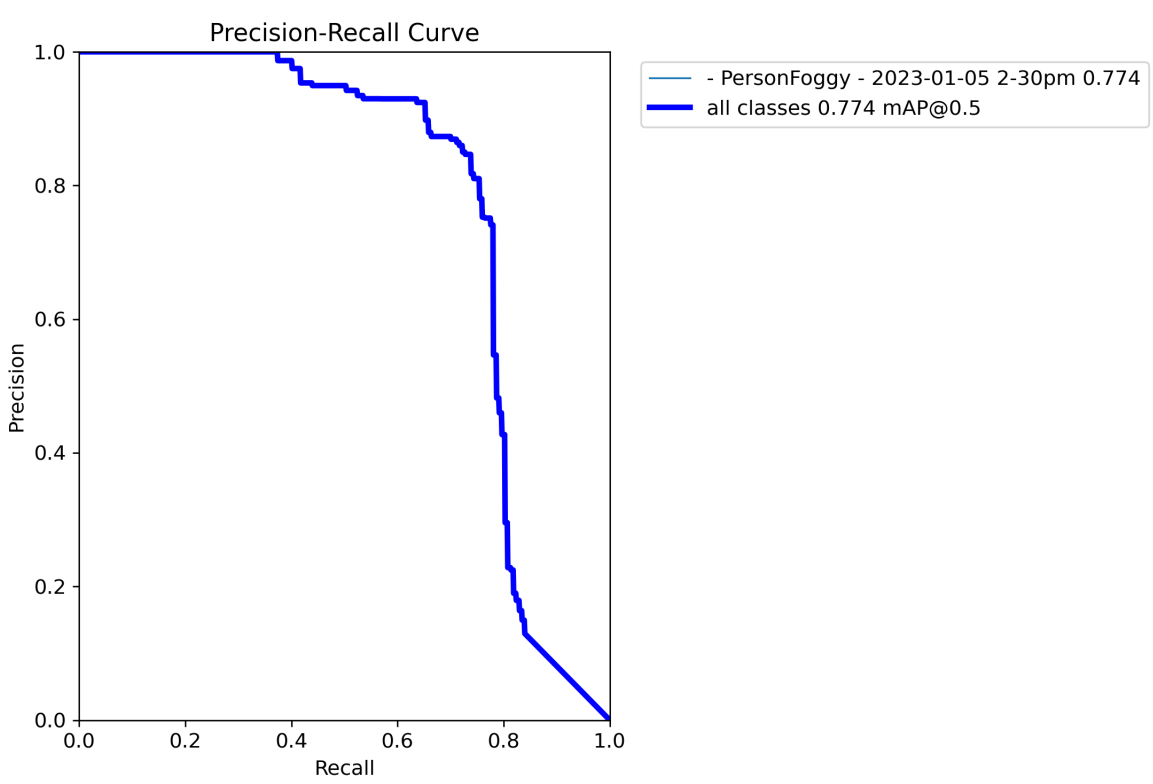
Class	Images	Instances	P	R	mAP50	mAP50-95
all	88	187	0.847	0.727	0.774	0.524

Hlavní dosažené metriky byly: Preciznost (P): 0,847 (84, 7 %), Recall (R): 0,727 (72, 7 %), mAP50: 0,774 (77, 4 %), mAP50-95: 0,524 (52, 4 %) Tyto výsledky naznačují velký výkon, zejména v oblasti mAP50, která dosáhla hodnoty 77, 4 %, což je o 5, 4 % více než 72 % dosažené modelem trénovaným na platformě Roboflow [5] na stejném datovém souboru.

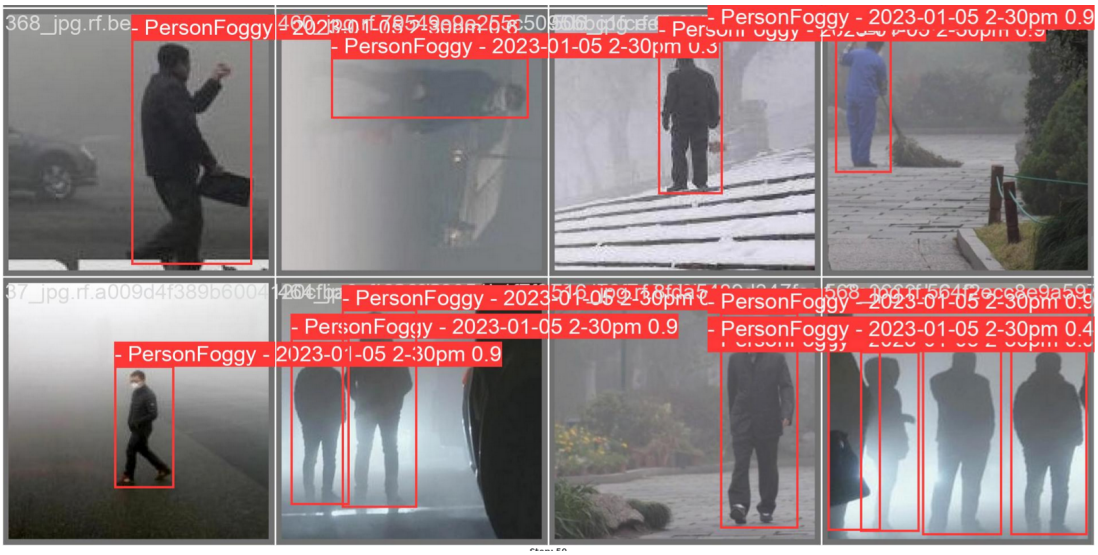
Náš přístup k vlastnímu tréninku dosáhl hodnoty mAP50 77, 4 %, což je ve srovnání se 72 % u modelu trénovaného pomocí platformy Roboflow na stejném datasetu významný rozdíl. Toto zlepšení o 5, 4 % jasně demonstruje hodnotu přizpůsobených trénovacích postupů. Metrika mAP50 by měla být směrodatnou metrikou na porovnání našeho modelu s jinými podobně natrénovanými modely, nicméně náš model nebyl trénovaný na standardním COCO datasetu s 80 třídami, ale na datasetu s jednou třídou, takže porovnání s jinými modely by mohlo být zavádějící. Hodnota 77, 4 % mAP50 nám udává, že model v průměru dosahuje 77, 4% přesnosti při detekci objektů s překryvem alespoň 50 % mezi predikovanými a skutečnými ohraničujícími rámečky objektu.

Lepší výkon lze přičíst následujícím faktorům. Především jsme použili hyperparametry jemně vyladěné specificky pro náš dataset, což umožnilo modelu lépe se přizpůsobit konkrétním charakteristikám našich dat. Dále jsme prodloužili dobu

tréninku na 50 epoch, což poskytlo modelu více času na optimalizaci a konvergenci. Významnou roli hrál také optimální výběr velikosti dávky, který zajistil efektivní využití výpočetních zdrojů a stabilitu trénovacího procesu. V neposlední řadě jsme využili transfer learning z předtrénovaných vah, což umožnilo modelu začít s již naučenými obecnými vzory a zaměřit se na specifika našeho úkolu. Toto srovnání potvrzuje správnost našeho přístupu a zdůrazňuje výhody kontroly nad trénovacím procesem, a to i v případě, že začínáme se stejným anotovaným datasetem. Zatímco GUI nástroje jako Roboflow nabízejí uživatelsky přívětivé rozhraní a rychlé nasazení, naše výsledky ukazují, že přizpůsobený přístup může přinést významné zlepšení výkonu. Tento rozdíl je obzvláště důležitý v aplikacích, kde i malé zlepšení přesnosti detekce může mít významný dopad na konečné využití systému.



Obr. 2.4: Vývoj metriky mAP (mean Average Precision) během tréninku neuronové sítě [6]



Obr. 2.5: Výsledky validace modelu YOLOv5s na testovacím datasetu [6]

2.4 Prvotní návrh systému

Prvotní systém používal obecnou verzi MobileNetSSD, kvůli své jednoduché implementaci do DepthAI pipeline. Zařízení sestávalo z Rasberry Pi 4 s 16 GB RAM s displejem a OAK-1 Kamerou od Luxonis. Kamera používá pro přenos dat USB 3.0 a z výše zmíněného testování v kontrolovaném prostředí se nepřehřívá. Vstupní videozáznam je potřeba ořezat na 300×300 pixelů kvůli MobileNetSSD síti.

Na Raspberry Pi 4 běží aplikace zpracovávající obraz (viz. Obr. 2.8), která zobrazuje detekce přímo na displeji připojeném k zařízení. Tento přístup však není pro reálné nasazení ideální, protože vyžaduje fyzický displej, který není možné napájet zároveň s OAK-1 kamerou (viz. Obr. 2.6. Kamera může odebírat z USB až 5 W a při napájení i displeje z USB na Raspberry Pi dostaneme chybovou hlášku, že zařízení nemá stabilní napájení a bylo odpojeno.

V systému jsme použili OAK-1 kameru. Pro rozpoznání osob ve snížené viditelnosti je nutné zohlednit omezení tohoto modelu kamery. V prostředí s vysokou vlhkostí by bylo vhodnější použít model OAK-1 PoE s celo-hliníkovým tělem a certifikací IP67, který poskytuje lepší ochranu proti vnějším vlivům. OAK-1 PoE také umožňuje efektivnější integraci senzoru IMX462 se STARVIS technologií, což je výhodné pro prostředí s horšími světelnými podmínkami. Alternativním řešením pro stávající OAK-1 kameru je použití IR filtru, což však vyžaduje úpravu trénovacího datasetu pomocí augmentace a pořízení nových fotografií pro zajištění spolehlivého rozpoznání osob v těchto podmínkách.

Obr. 2.6: Čelní pohled na první verzi systému, ukazuje OAK-1 a displej

Obr. 2.7: Zadní pohled na první verzi systému, ukazuje Raspberry Pi

2.4.1 Příprava systému na Raspberry Pi

V této podkapitole bude popsán proces nasazení systému DepthAI na Raspberry Pi a problémy, které byly během tohoto procesu zaznamenány. Na zařízení je nainstalován 64bitový operační systém, protože u 32bitového byly zjištěny problémy se závislostmi různých balíčků při instalaci DepthAI. Pro implementaci je využíván Python 3.11.2 a tato linuxová distribuce:

```
PRETTY_NAME= Debian GNU/Linux 12 (bookworm)
```

Pro nainstalování všech systémových balíčků pro danou distribuci nám stačí použít skript od Luxonis:

```
sudo wget -qO- https://docs.luxonis.com/
   install_dependencies.sh | bash
```

Tento skript nejprve provede aktualizaci seznamu balíčků a následně aktualizaci všech nainstalovaných balíčků na nejnovější verze pomocí příkazů apt update a apt upgrade. Poté nainstaluje základní balíčky, jako jsou python3, pip3, cmake, git a udev, pokud již nejsou na systému nainstalovány. Dále nainstaluje další závislosti, jako jsou libusb, libudev a další potřebné knihovny, které jsou nezbytné pro správnou funkci systému DepthAI. Luxonis doporučuje export proměnné:
`OPENBLAS_CORETYPE=ARMV8` , pokud selže OpenCV po instalaci přes PiPy:

```
echo "export␣OPENBLAS_CORETYPE=ARMV8" >> ~/.bashrc
source ~/.bashrc
```

Pro instalaci všech potřebných knihoven pro Python je vhodné vytvořit virtuální prostředí (`venv`). To nám umožňuje izolovat závislosti projektu a vyhnout se konfliktům s globálně nainstalovanými balíčky. Pokud bychom nepoužili virtuální prostředí, mohli bychom narazit na problémy s oprávněními při pokusu o instalaci balíčků globálně, což by vyžadovalo použití `apt` pro instalaci některých balíčků, které jsou dostupné v repozitářích systému. Navíc, knihovna `depthai` není dostupná v repozitářích `apt`.

```
python3 -m venv .venv
source .venv/bin/activate
pip install depthai Flask opencv-python
```

2.4.2 První testovací aplikace

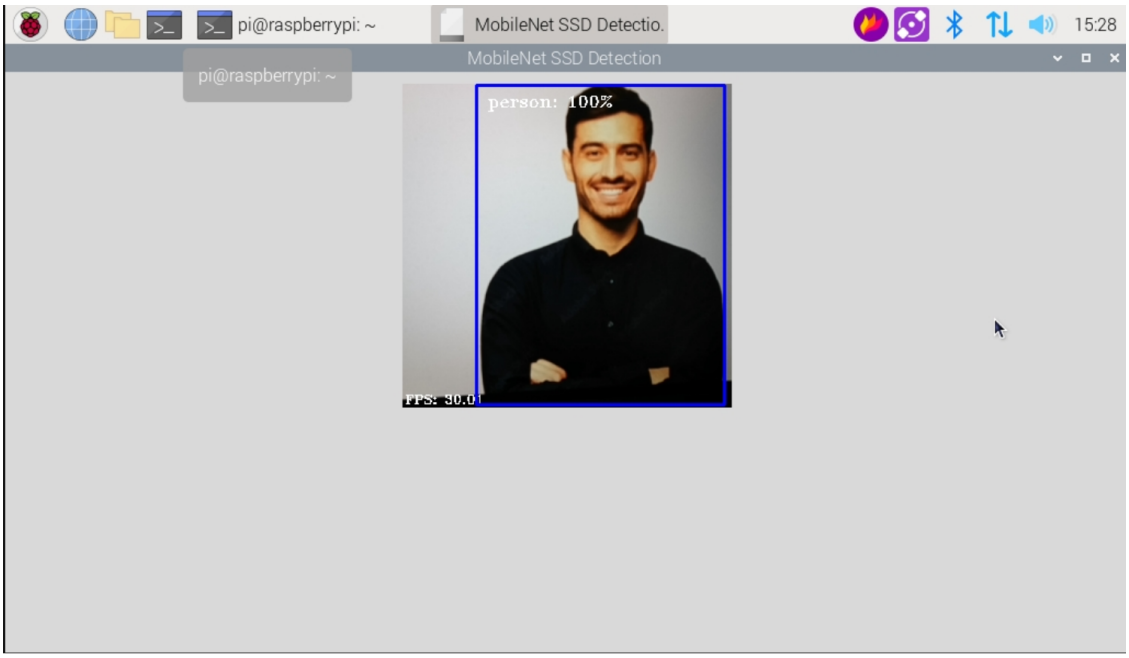
První aplikace pro testování zařízení byla navržena s důrazem na jednoduchost a ověření základních funkncí. Pro počáteční testování byl použit model MobileNetSSD, který je dobře optimalizovaný pro zařízení OAK-1.

Důležitou součástí je pipeline, což je systém propojených uzlů (nodes), který funguje jako potrubí pro zpracování dat z kamery. Jedná se o způsob, jakým nastavujeme a propojujeme různé části zpracování obrazu a detekce objektů v jednom uceleném toku. Pipeline je prakticky jediná část kódu, která se významně mění v průběhu vývoje aplikace, protože definuje, jak data proudí mezi různými komponentami systému.

V případě MobileNetSSD pipeline obsahuje několik hlavních komponent. ColorCamera node zajišťuje zachycení obrazu z kamery a jeho předání do dalších částí pipeline. MobileNetDetectionNetwork node zpracovává vstupní obraz a provádí detekci objektů pomocí modelu MobileNetSSD. XLinkOut nodes slouží k přenosu výstupních dat, jako jsou obraz a výsledky detekce, z DepthAI do hostitelského zařízení.

Samotné video s detekcemi se vykresluje na displeji v jednoduchém okně pomocí knihovny OpenCV. Tento proces slouží k ověření, že kamera funguje správně a Raspberry Pi nemá problémy s knihovnami a dokáže přijímat obraz z kamery a dále jej zpracovávat.

Pro testování s modelem YOLOv5s, bylo nutné upravit pipeline. Hlavní změny spočívaly v nahrazení MobileNetDetectionNetwork uzlu za YoloDetectionNetwork a v úpravě parametrů pro tento uzel. YOLOv5s vyžaduje specifické nastavení, jako je počet tříd, velikost koordinátů, prahová hodnota pro IoU (Intersection over Union) a další parametry. Tato pipeline zajišťuje, že obraz z kamery je předán do YOLOv5s modelu, který provede detekci objektů, a výsledky jsou poté odeslány zpět do zařízení pro zobrazení.



Obr. 2.8: Testovací aplikace pro detekci objektů pomocí MobileNet SSD neuronové sítě

2.5 Druhý návrh systému

Na základě výzkumu a testování byl navržen nový systém. Z předchozího návrhu byl odstraněn fyzický displej. Hlavním důvodem bylo zjištění, že USB rozhraní na Raspberry Pi nedokázalo současně napájet OAK-1 a displej. Pro zobrazení všech důležitých údajů a obrazu z kamery byl vytvořen pipeline skript na použití OAK-1 a streamování videa přes Flask na webový server.

Současné zařízení je sestaveno z Raspberry Pi 4, které slouží jako napájecí zdroj a server pro OAK-1 kameru. Videozáznam je sdílen v reálném čase na IP adrese Raspberry Pi na všech síťových rozhraních s portem 5000. Na streamu je vykreslován čtverec ohraničující detekované objekty s příslušným názvem objektu nebo třídy a procentuálně vyjádřenou mírou jistoty detekce.

Finální zařízení je tvořeno pouze Raspberry Pi 4 a OAK-1. Navržené řešení by mělo poskytnout vhodné prostředí pro ověření všech výše uvedených funkcí. Kamera OAK-1 se senzorem IMX378 musí být umístěna vodorovně se zemí ve výšce 3 metrů. Tato pozice odpovídá augmentaci použitého datasetu a zvyšuje přesnost detekce. Efektivní dosah detekce osob je omezen na 20 metrů v husté mlze (viz. 1.5.1).

2.5.1 Finální aplikace systému

Systém využívá výpočetní možnosti čipu RVC2 [20] v kameře OAK-1 a zároveň poskytuje uživatelské rozhraní pro zobrazení výsledků. V této podkapitole je popsána architektura aplikace, způsob zpracování obrazu, konfigurace neuronové sítě a uživatelské rozhraní.

Architektura aplikace

Aplikace je postavena na dvou klíčových technologických základech. Prvním je DepthAI SDK, který umožňuje komunikaci s kamerou OAK-1 a využití jejího výpočetního potenciálu. Druhým je webový framework Flask, který slouží k vytvoření webového rozhraní pro zobrazení výsledků detekce.

Architektura aplikace je navržena jako datová pipeline, která zajišťuje plynulý tok dat od snímání obrazu přes jeho zpracování až po zobrazení výsledků. Kamera OAK-1 snímá obraz v rozlišení 640×640 pixelů s frekvencí 40 snímků za sekundu, což poskytuje dostatečnou kvalitu pro detekci osob a zároveň umožňuje zpracování v reálném čase. Snímky jsou přímo v kameře zpracovány neuronovou sítí YOLOv5s, která běží přímo na hardwaru kamery OAK-1. Díky tomu, že detekce objektů jsou prováděny přímo v kameře, je téměř minimálně zatížené Raspberry Pi. Raspberry Pi tak slouží pouze jako webový server pro zobrazení výsledků a jako rozhraní pro uživatele, což umožňuje efektivní využití jeho výpočetních zdrojů.

Datová pipeline propojuje kameru OAK-1 s Raspberry Pi pro streamování videa. Toto propojení umožňuje přenos jak původních snímků, tak výsledků detekce z neuronové sítě do webové aplikace 2.5.1.

Zpracování obrazu

Proces zpracování obrazu začíná snímáním pomocí barevné kamery OAK-1. Kamera je nakonfigurována tak, aby poskytovala snímky v rozlišení 640×640 pixelů, což je optimální velikost pro vstup do našeho YOLOv5s modelu. Snímky jsou pořizovány v barevném formátu BGR, který je nutný pro uzel YOLO v pipeline DepthAI a je také kompatibilní s knihovnou OpenCV, která je využívána pro další zpracování obrazu.

Po získání snímku z kamery je tento snímek přímo předán neuronové síti YOLOv5s, která běží na čipu RVC2 v kameře OAK-1. Tato neuronová síť provádí detekci objektů v obraze a pro každý detekovaný objekt poskytuje informace o jeho pozici (souřadnice ohraničujícího rámečku), třídě (typ objektu) a míře jistoty detekce (confidence score). Tyto informace jsou následně odeslány zpět do Raspberry Pi.

V Raspberry Pi jsou výsledky detekce zpracovány a zobrazeny. Pro každý detekovaný objekt je vykreslen ohraničující rámeček, který označuje jeho pozici objektu v obraze. Nad rámečkem je umístěn popisek, který obsahuje název detekované třídy objektu a procentuální vyjádření míry jistoty detekce. Zobrazení poskytuje uživateli jasnou a srozumitelnou informaci o detekovaných objektech.

Zpracovaný obraz s vykreslenými detekcemi je následně převeden do formátu JPEG a streamován přes webové rozhraní Flask. Tento přístup umožňuje uživateli sledovat detekce v reálném čase prostřednictvím webového prohlížeče, což eliminuje potřebu fyzického displeje připojeného k Raspberry Pi.

Konfigurace neuronové sítě

Pro dosažení optimálních výsledků při detekci osob ve snížené viditelnosti byla síť nakonfigurována s ohledem na požadavky aplikace.

Prahová hodnota jistoty (confidence threshold) byla nastavena na $0,5$, což znamená, že detekce s mírou jistoty nižší než 50 % jsou ignorovány. Tato hodnota byla zvolena jako kompromis mezi citlivostí detekce a minimalizací falešných detekcí. Při nižší prahové hodnotě by síť detekovala více objektů, ale zvýšil by se počet falešných detekcí. Naopak při vyšší prahové hodnotě by se snížil počet falešných detekcí, ale některé objekty by nemusely být detekovány.

Důležitým parametrem je prahová hodnota IoU (Intersection Over Union), která byla nastavena na také $0,5$. Tento parametr ovlivňuje, jak jsou vyhodnocovány překrývající se detekce. Pokud se dva ohraničující rámečky překrývají více než z 50 %, je zachován pouze ten s vyšší mírou jistoty. Tato hodnota byla zvolena tak, aby se minimalizoval počet duplicitních detekcí stejného objektu.

Síť YOLOv5s provádí detekce na třech úrovních rozlišení, což jí umožňuje efektivně detekovat objekty různých velikostí. Pro malé objekty je využívána výstupní vrstva s rozlišením 80×80, pro středně velké objekty vrstva s rozlišením 40×40 a pro velké objekty vrstva s rozlišením 20×20. Tato víceúrovňová detekce významně zvyšuje schopnost sítě detekovat osoby v různých vzdálenostech od kamery.

Pro každou úroveň rozlišení jsou definovány specifické "kotvy" (anchors), které slouží jako výchozí bod pro predikci skutečných ohraničujících rámečků. Tyto kotvy jsou předem definované ohraničující rámečky různých velikostí a poměrů stran, které jsou optimalizovány pro detekci osob. Správná konfigurace těchto kotev je klíčová pro dosažení vysoké přesnosti detekce.

Uživatelská aplikace

Uživatelské rozhraní aplikace je realizováno jako webová aplikace, která je spuštěna na všech síťových rozhraních Raspberry Pi na portu 5000. Toto řešení eliminuje

potřebu fyzického displeje připojeného k Raspberry Pi, což řeší problém s nedostatečným napájením při současném připojení kamery OAK-1 a displeje přes USB. Zároveň umožňuje přístup k systému z libovolného zařízení v síti.

Webové rozhraní je velmi jednoduché. Obsahuje pouze video stream (viz. 2.9), který zobrazuje obraz z kamery s vykreslenými detekcemi v reálném čase. Pro každý detekovaný objekt je zobrazen ohraničující rámeček s popiskem obsahujícím název detekované třídy a procentuální vyjádření míry jistoty. Detekovaný objekt (třída) je i vypsán pod video streamem.

Webové rozhraní využívá framework Flask pro Python. Pro streamování videa je použit mechanismus HTTP streamování s MIME typem multipart/x-mixed-replace, který zajišťuje plynulé přenášení jednotlivých JPEG snímků v reálném čase.

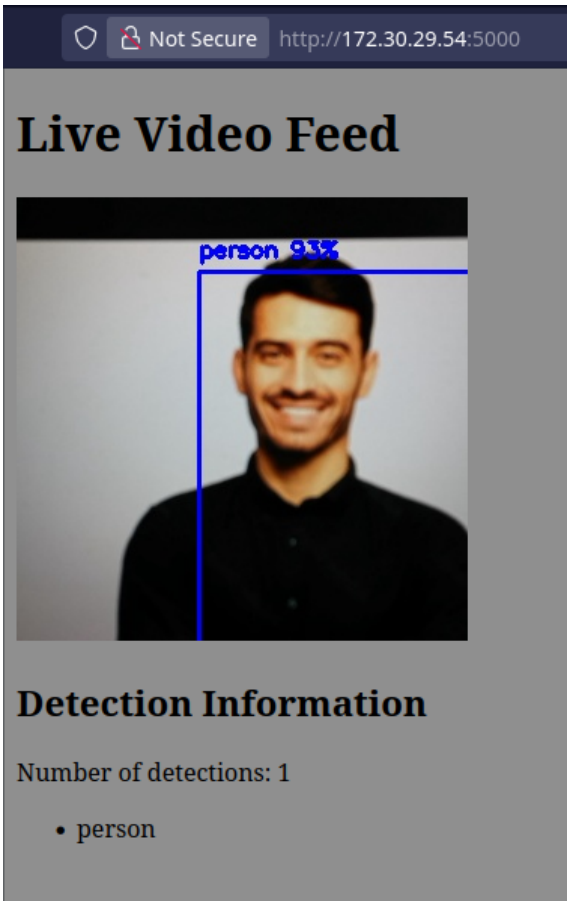
Webové rozhraní

Hlavní stránka aplikace je tvořena pomocí HTML šablony. Stránka obsahuje nadpis "Live Video Feed", pod kterým je zobrazení video streamu 2.9. Video stream poskytuje kontinuální proud JPEG snímků přes endpoint '/video_feed'. Tento přístup využívá MIME typ 'multipart/x-mixed-replace', který umožňuje webovému prohlížeči průběžně aktualizovat zobrazený obrázek bez nutnosti obnovení celé stránky.

Pod video streamem je umístěna sekce "Detection Information", která zobrazuje informace o detekovaných objektech. Tato sekce obsahuje počítadlo detekcí a seznam detekovaných tříd objektů. Tyto informace jsou aktualizovány pomocí JavaScriptu, který každou sekundu zasílá asynchronní požadavek na endpoint '/detections'. Tento endpoint vrací aktuální informace o detekovaných objektech ve formátu JSON, které jsou následně zpracovány a zobrazeny v této sekci.

Serverová část aplikace obsahuje tři hlavní endpointy:

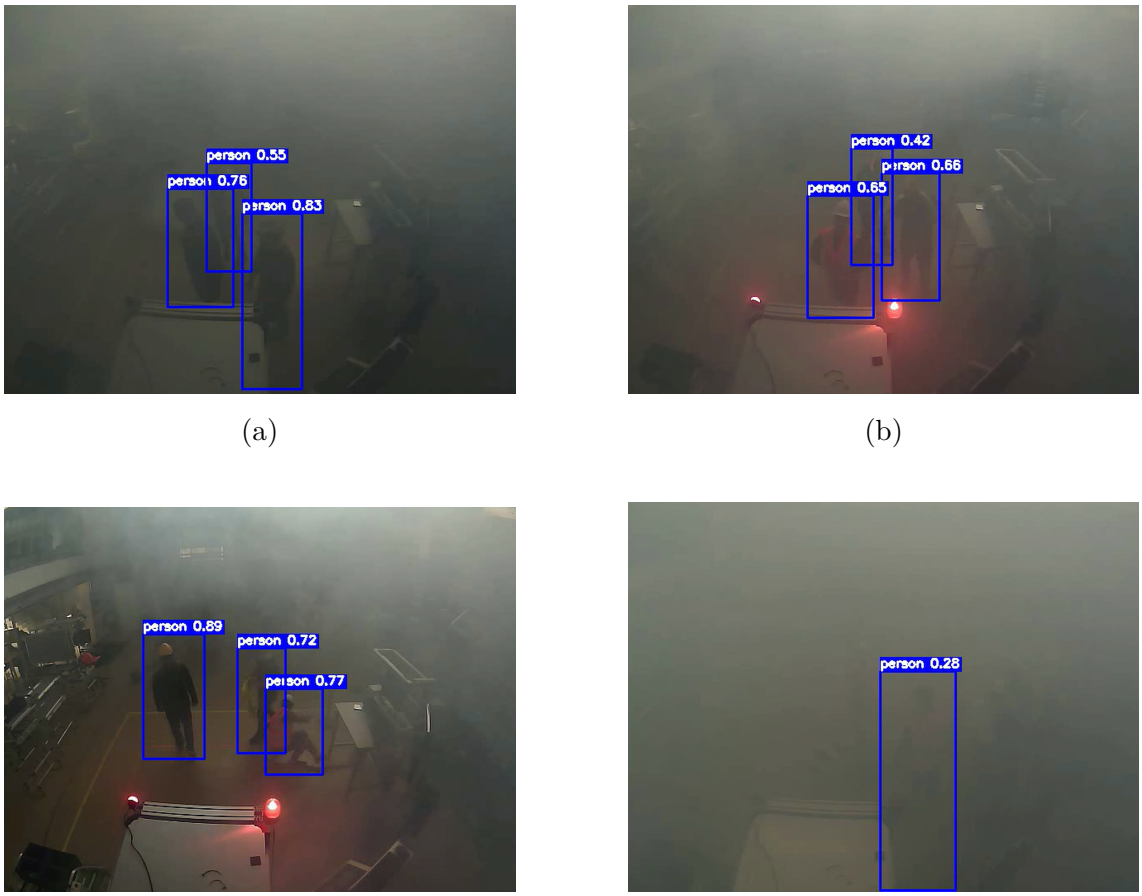
1. `/` - Hlavní endpoint, který vrací HTML šablonu pro zobrazení uživatelského rozhraní.
2. `/video_feed` - Endpoint pro stream videa využívající funkci `gen_frames()` pro generování snímků z kamery OAK-1 s vykreslenými detekcemi.
3. `/detections` - Endpoint, který poskytuje aktuální informace o detekovaných objektech ve formátu JSON.



Obr. 2.9: Webové rozhraní pro zobrazení video streamu a detekci objektů v klientské aplikaci

2.5.2 Testování v reálném prostředí

Testování probíhalo ve vnitřních prostorách osvětlených přirozeným zdrojem světla, přičemž byla s pomocí glycerinového výrobníku mlhy vytvořena umělá mlha pro simulaci reálných podmínek (viz. Obr. 2.10d. Kamera byla instalována ve výšce 3 metrů nad podlahou, což představuje kompromis mezi potřebou zachycení většího prostoru s více osobami a optimální výškou pro rozpoznávání, protože náš model byl primárně trénován na datech pořízených z úrovně hrudníku. Systém dosahoval inference v rozmezí $70 - 80$ milisekund, což při $12 - 15$ snímcích za sekundu zajišťovalo uspokojivé výsledky detekce. Během testování se teplota kamery pohybovala v průměru kolem 60 stupňů, což je dobrá hodnota pro dlouhodobý provoz systému.



(a) (b)

(c) (d)

Obr. 2.10: Srovnání detekce objektů v testovacím prostředí s různými světelnými podmínkami a úrovněmi zakouření

Závěr

V úvodu této práce jsme byli seznámeni s principem funkce neuronových sítí (viz. 1.1). Detailně bylo popsáno jakým způsobem zpracovává model neuronové sítě obraz, proč musíme pracovat s tensory a jak fungují určité vrstvy těchto sítí (viz. 1.2). Byl vybraný jeden nejrozšířenějších modelů neuronových sítí YOLOv5s (viz. 1.4).

Tento model byl dotrénován námi zvoleným datasetem, aby dosáhnul lepších výsledků v námi požadovaném prostředí (viz. 2.3). Databáze RoboFlow byla velmi nápomocná při výběru datasetu osob za horších viditelnostních podmínek (viz. 1.3). Průzkumem trhu byly vybrány jednotlivé komponenty našeho zařízení, které byly podrobeny zátěžovým testům (viz. 2.2).

Na základě těchto testů jsme zvolili Raspberry Pi 5, jako hostitele našeho webového prostředí pro zobrazení video streamu a detekci, a kameru OAK-1 na vyhodnocení samotných detekcí pomocí RVC2 (specializovaný čip pro zpracování obrazu vyvinutý společností Luxonis) a pořizování obrazu v reálném čase (viz. 2.1.

Po testu YOLOv5s natrénovaného námi zvoleným datasetem pro rozpoznání osob ve snížené viditelnosti je vidět, že mAP50 dosahuje $77,4$ %, což je nejvíce, čehož jsem byl schopný dosáhnout při změně trénovacích hyperparametrů, jako byly například dávky obrazů pro iteraci nebo počet epoch. Při větších dávkách se mAP50 nepatrně snižovala a při zvednuti počtu epoch se naopak nijak výrazně nezvětšila (viz. 2.3.2). Při trénování pomocí platformy Roboflow jsme nebyli schopni dosáhnout tak dobrého výsledku, nejlepší výsledek měl mAP50 zhruba o 5 % menší než výsledek pomocí metody v podkapitole 2.3.2. Za vinu dávám neúplnou kontrolu tréninku pomocí přednastaveného grafického prostředí na Obr. 2.3a.

Finální aplikace nám zobrazuje video stream v reálném čase s ohraničením detekovaného objektu, mírou jistoty a detekovanou třídou. Celá aplikace funguje na základě předání videostreamu a detekcí Flask rozhraní do jednoduché webové stránky, kterou si můžeme zobrazit na IP adrese zařízení s portem 5000 (viz. 2.5.1.

Testování v reálném prostředí probíhalo ve vnitřních prostorách s umělou mlhou vytvořenou glycerinovým výrobníkem mlhy pro simulaci reálných podmínek (viz. 2.5.2. Kamera byla nainstalována 3 metry nad podlahou, což představovalo kompromis mezi zachycením většího prostoru a optimální výškou pro rozpoznávání, jelikož model byl trénován na datech z úrovně hrudníku. Systém dosahoval inference $70-80$ ms na snímek a je vidět z obrázku 2.10, že dostáváme uspokojivé výsledky detekce. Teplota kamery se pohybovala pod 60 stupňů.

Literatura

[1] Francis Quintal Lauzon. An introduction to deep learning. In *2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, pages 1438–1439, 2012. doi:10.1109/ISSPA.2012.6310529.

[2] Zijie J. Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, Minsuk Kahng, and Duen Horng Polo Chau. Cnn explainer: Learning convolutional neural networks with interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1396–1406, 2021. doi:10.1109/TVCG.2020.3030418.

[3] Do Thuan. Evolution of yolo algorithm and yolov5: The state-of-the-art object detention algorithm, 2021.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. URL: https://arxiv.org/abs/2004.10934, arXiv:2004.10934.

[5] B. Dwyer, J. Nelson, T. Hansen, et al. Roboflow, 2024. computer vision. URL: https://roboflow.com.

[6] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com. URL: https://www.wandb.com/.

[7] Daniel Burke. Learn pytorch for deep learning: Zero to mastery book, Oct 2021. URL: https://www.learnpytorch.io/.

[8] Iain Rist and James McGinnigle. What is a tensor?, Aug 2008. URL: https://www.doitpoms.ac.uk/tlplib/tensors/index.php.

[9] D.P. Mandic. A generalized normalized gradient descent algorithm. *IEEE Signal Processing Letters*, 11(2):115–118, 2004. doi:10.1109/LSP.2003.821649.

[10] Ludovic Arnold, Sébastien Rebecchi, Sylvain Chevallier, and Hélène Paugam-Moisy. An Introduction to Deep Learning. In *European Symposium on Artificial Neural Networks (ESANN)*, Proceedings of the European Symposium on Artificial Neural Networks (ESANN), Bruges, Belgium, April 2011. URL: https://hal.science/hal-01352061.

[11] Wikipedia contributors. Kaggle, 2011. [Last edit on 11th December 2023]. URL: https://en.wikipedia.org/wiki/Kaggle.

[12] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics YOLO, January 2023. URL: https://github.com/ultralytics/ultralytics.

[13] Disertation project. Personfoggy dataset. https://universe.roboflow.com/disertation-project/personfoggy, jan 2023. visited on 2023-12-18. URL: https://universe.roboflow.com/disertation-project/personfoggy.

[14] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020. URL: https://www.mdpi.com/2078-2489/11/2/125, doi:10.3390/info11020125.

[15] World Meteorological Organization. *International Cloud Atlas: Manual on the Observation of Clouds and Other Meteors*, volume 1 of *WMO*. Secretariat of the World Meteorological Organization, Geneva, revised edition, 1975. Digitized October 1, 2010 from University of Michigan original.

[16] Ismail Gultepe, Robert Tardif, Silas Michaelides, Jan Cermak, Andreas Bott, M. Muller, Mariusz Pagowski, B. Hansen, Gary Ellrod, W. Jacobs, G. Toth, and S. Cober. Fog research: A review of past achievements and future perspectives. *Pure and Applied Geophysics*, 164:1121–1159, 06 2007.

[17] F. Ali, Z. H. Khan, K. S. Khattak, and T. A. Gulliver. The effect of visibility on road traffic during foggy weather conditions. *IET Intelligent Transport Systems*, 18:47–57, 2023. doi:10.1049/itr2.12432.

[18] Srinivasa Narasimhan and Shree Nayar. Vision and the atmosphere. *International Journal of Computer Vision*, 48:233–254, 07 2002. doi:10.1145/1508044.1508113.

[19] Nicolas Hautière, Jean-Philippe Tarel, and Didier Aubert. Towards fog-free in-vehicle vision systems through contrast restoration. 06 2007. doi:10.1109/CVPR.2007.383259.

[20] luxonis. DepthAI: Embedded machine learning and computer vision api, 2020. Software available from luxonis.com. URL: https://luxonis.com/.

[21] Nvidia Corp. *Jetson Nano User Guide*, December 2019. URL: https://developer.download.nvidia.com/embedded/L4T/r32-3-1_Release_v1.0/Jetson_Nano_Developer_Kit_User_Guide.pdf.

[22] Shenzhen Xunlong Software Co., Ltd. *Orange Pi 5 User Manual*, November 2022. URL: https://drive.google.com/file/d/13LJk85ueOup2HqbEhcY2UhUw5lwESth_/view.

[23] Mar 2019. URL: https://coral.ai/docs/accelerator/get-started.