Detekční forenzní nástroj s využitím hlubokých neuronových sítí pro odhalování specifických dat dle škály COPINE

Bc. Filip Šedivý

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Filip Šedivý**
Osobní číslo:	**A22311**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Detekční forenzní nástroj s využitím hlubokých neuronových sítí pro odhalování specifických dat dle škály COPINE**
Téma práce anglicky:	**A Detection Forensic Tool Using Deep Neural Networks for Detecting Specific Data According to the COPINE Scale**

Zásady pro vypracování

1. Proveďte literární rešerši metod v oblasti detekce objektů.
2. Vyberte, doplňte či vytvořte vhodný anotovaný dataset pro detekci specifických objektů dle škály COPINE.
3. Zvolte vhodné předtrénované modely pro detekci specifických objektů, dotrénujte je, případně natrénujte i vlastní model pro detekci objektů dle škály COPINE.
4. Srovnejte výsledné modely dle standardních metrik.
5. Zhodnoťte dosažené výsledky, navrhněte doporučení a výsledný vhodný model pro praktické nasazení.

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. LAKSHMANAN, Valliappa; GÖRNER, Martin a GILLARD, Ryan, 2021. Practical Machine Learning for Computer Vision: End-to-End Machine Learning for Images. O'Reilly Media. ISBN 9781098102364.
2. GÉRON, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems. Second edition. Beijing: O'Reilly, 2019. ISBN 978-1-4920-3264-9.
3. JIANG, Xiaoyue, et al. (ed.). Deep learning in object detection and recognition. Singapur: Springer, [2019]. ISBN 978--981-10-5151-7.
4. DAVIES, E. R. Computer and machine vision: theory, algorithms, practicalities. 4th ed. Boston: Elsevier, 2012. ISBN 978-01-238-6908-1.
5. ROSEBROCK, Adrian. Practitioner Bundle. In: Deep Learning for Computer Vision with Python. PyimageSearch.com. 2017
6. CHOLLET, François. Deep learning v jazyku Python: knihovny Keras, Tensorflow . Praha: Grada Publishing, 2019, 328 s. Knihovna programátora. ISBN 978-80-247-3100-1.

Vedoucí diplomové práce: **prof. Ing. Zuzana Komínková Oplatková, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **5. listopadu 2023**
Termín odevzdání diplomové práce: **13. května 2024**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 13. 5. 2024

Filip Šedivý, v. r.
podpis studenta

ABSTRAKT

Diplomová práce je zaměřena na vývoj modelů neuronových sítí pro detekci odhalených osob dle škály COPINE. Natrénované modely, implementované v rámci webové aplikace, umožňují zpracování obrázků jako vstupních dat s cílem detekce zájmových oblastí. Modely, založené na rozdílných architekturách neuronových sítí, vykazují mAP 58,75 % na testovací datové sadě, přičemž v určitých specifických oblastech dosahuje nejlepší model AP 70 %.

Klíčová slova: Umělá inteligence, počítačové vidění, PyTorch, YOLOv8, Detectron2, Faster R-CNN

ABSTRACT

The thesis focuses on the development of neural network models for detection of detected persons according to the COPINE scale. The trained models, implemented within a web application, allow the processing of images as input data in order to detect regions of interest. The models, based on different neural network architectures, show a mAP of 58.75% on the test dataset, with the best model achieving an AP of 70% in certain specific regions.

Keywords: Artificial Intelligence, Computer Vision, PyTorch, YOLOv8, Detectron2, Faster R-CNN

Tímto bych rád poděkoval své vedoucí diplomové práce prof. Ing. Zuzaně Komínkové Oplatkové, Ph.D. za odborné rady, pomoc a čas, které mi věnovala při zpracování této práce. Dále také patří poděkovaní mé rodině a přátelům za podporu po dobu studia a při tvorbě této práce.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Prohlašuji, že při tvorbě této práce jsem použil nástroj generativního modelu AI GPT-4 Turbo za účelem asistence při úpravách textu, opravy gramatiky a zlepšení stylistiky. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

OBSAH

ÚVOD **9**
I TEORETICKÁ ČÁST **10**
1 ÚVOD DO POČÍTAČOVÉHO VIDĚNÍ **11**
1.1 HISTORIE POČÍTAČOVÉHO VIDĚNÍ 12
1.2 ZÁKLADNÍ PRINCIPY A DEFINICE 13
2 ZPRACOVÁNÍ OBRAZU **15**
2.1 PERCEPCE OBRAZU 15
2.2 BAREVNÉ MODELY 16
2.2.1 RGB 17
2.2.2 BGR 17
2.2.3 HSV a HSL 18
2.3 PŘEDZPRACOVÁNÍ OBRAZU 19
2.3.1 Změna velikosti 19
2.3.2 Normalizace 20
2.3.3 Změna pořadí kanálů 20
3 NEURONOVÉ SÍTĚ **22**
3.1 TYPY NEURONOVÝCH SÍTÍ 22
3.1.1 Dopředné neuronové sítě 23
3.1.2 Konvoluční neuronové sítě 24
3.1.3 Rekurentní neuronové sítě 25
3.2 ŠÍŘENÍ CHYBY 26
3.3 PROCES UČENÍ 27
4 DETEKCE OBJEKTŮ **30**
4.1 R-CNNs 31
4.1.1 R-CNN 31
4.1.2 Fast R-CNN 32
4.1.3 Faster R-CNN 33
4.2 SSD 34
4.3 YOLO 35
5 HODNOCENÍ VÝKONNOSTI DETEKČNÍCH MODELŮ **39**
5.1 IOU 39
5.2 MAP 40
5.3 PRECISION A RECALL 41
5.4 F1 SKÓRE 42
6 MODERNÍ PŘÍSTUP K ROZPOZNÁVÁNÍ SPECIFICKÝCH DAT **43**
II PRAKTICKÁ ČÁST **44**
7 POŽADAVKY NA SOFTWARE **45**

7.1 VIRTUÁLNÍ SERVER 45
7.2 DOCKER 46
7.3 LABEL STUDIO 46
7.4 LABEL STUDIO BACKEND ML 47
7.5 YOLO 47
7.6 DETECTRON2 47
7.7 GRADIO 47
8 INSTALACE SOFTWARU 48
8.1 INSTALACE DOCKERU 48
8.2 INSTALACE NGINX 50
8.3 INSTALACE LABEL STUDIO 51
9 PŘÍPRAVA DATOVÉ SADY 52
9.1 DEFINOVÁNÍ TŘÍD 53
9.2 ANOTOVÁNÍ DATOVÉ SADY 54
9.3 AUTOMATICKÉ ANOTOVÁNÍ DATOVÉ SADY 54
10 TRÉNOVÁNÍ MODELU 58
10.1 TRÉNOVÁNÍ MODELU YOLO 58
10.2 TRÉNOVÁNÍ MODELU DETECTRON2 61
11 APLIKACE PRO TESTOVÁNÍ MODELŮ 64
12 VYHODNOCENÍ DETEKČNÍCH MODELŮ 69
ZÁVĚR 74
SEZNAM POUŽITÉ LITERATURY 76
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 82
SEZNAM OBRÁZKŮ 83
SEZNAM UKÁZEK KÓDU 85
SEZNAM PŘÍLOH 86

ÚVOD

Bezpečnost na internetu představuje v současném digitálním prostředí klíčovou oblast, která vyžaduje odbornou pozornost. Úzce je spojena s ochranou osobní identity, kde je zásadní předcházet zneužívání a rozšiřování nevhodných obrazových materiálů a fotografií, aby se zabránilo jejich šíření na internetu. Šíření těchto materiálů může mít závažné důsledky, sahající od kybernetické šikany po právní konsekvence. Velké technologické společnosti se specializují na vývoj algoritmů, které umí proti takovému obsahu bojovat, nicméně tyto pokročilé technologie často zůstávají neveřejné. Veřejnosti se tyto technologie představují pouze v médiích.

Diplomová práce se úzce zabývá problematikou detekcí odhalených postav (dětí) s využitím umělé inteligence; od samotného získávání dat, přes jejich anotování, vývoj a trénování umělých neuronových sítí spadajících do oblasti umělé inteligence až po konečné nasazení modelu do ukázkové aplikace. Pro samotný vývoj umělé inteligence se využívají přední nástroje od společnosti Facebook a pro trénování modelů je využitá infrastruktura od sdružení CESNET, která poskytuje velmi výkonné grafické karty určené pro trénování neuronových sítí.

Téma této diplomové práce je zaměřeno na počítačové vidění, což je jedna z oblastí umělé inteligence. V teoretické části je nejprve představena historie počítačového vidění. Následně se práce zabývá zpracováním obrazu, které je pro počítačové vidění zásadní. Další kapitoly jsou věnovány neuronovým sítím, jejich funkcionalitě a procesu šíření chyby v rámci těchto sítí. Práce také obsahuje detailní rozbor procesu trénování neuronových sítí. V závěru teoretické části jsou prezentovány architektury nejčastěji používaných neuronových sítí, které se v současnosti využívají při detekci objektů v obrazech. Poslední kapitola je věnována metrikám pro hodnocení účinnosti jednotlivých modelů.

Praktická část práce se zaměřuje na konfiguraci infrastruktury tak, aby všechny nástroje byly provozovány interně na vlastních virtuálních serverech. Trénování modelů probíhá výhradně na serverech, které jsou součástí sdružení CESNET. Výstupy diplomové práce zahrnují modely schopné detekovat postavy dětského věku na základě obrazového materiálu, klasifikovaného podle škály COPINE až do úrovně 2, a to s dostatečnou úspěšností.

I. TEORETICKÁ ČÁST

1 ÚVOD DO POČÍTAČOVÉHO VIDĚNÍ

Počítačové vidění je multidisciplinární oblast, která spadá do širšího spektra umělé inteligence. Jeho hlavním cílem je rozvíjet metody a technologie, které umožňují počítačům interpretovat obsah digitálních médií, jako jsou fotografie a videa. Na první pohled se může zdát, že pochopení vizuálního obsahu je pro lidský mozek jednoduchý úkol, ale pro umělou inteligenci představuje značnou výzvu. Tato složitost pramení z omezeného porozumění procesům biologického vidění a z komplexnosti vizuálních scén, kde objekty mohou být prezentovány v nekonečném počtu orientací a světelných podmínek.

Skutečné systémy počítačového vidění musí být schopny rozpoznat a interpretovat scény v libovolných podmínkách a extrahovat z nich relevantní informace. Avšak, v současné době jsou tyto systémy efektivní pouze v omezených a specificky definovaných problémech, nikoli v obecné úloze vizuálního vnímání. Úkolem počítačového vidění je tedy napodobit a rekonstruovat schopnosti lidského zraku.

Porozumění digitálním obrazům není omezeno pouze na identifikaci objektů, ale také zahrnuje extrakci textu, třírozměrných modelů a dalších elementů z obrazového materiálu.

Počítačové vidění obnáší široký rozsah úkolů a výzev, které byly řešeny s využitím různých druhů znalostí a technik. Tato disciplína se věnuje automatizaci a integraci procesů a reprezentací, které jsou klíčové pro vizuální vnímání. Zahrnuje množství technik užitečných samy o sobě, jako je zpracování obrazu pro transformaci, kódování a přenos obrazových dat, a statistickou klasifikaci vzorů, aplikovanou na vizuální nebo jiné typy vzorů. Kromě toho se věnuje kognitivnímu zpracování, což je oblast, která je pro pochopení a aplikaci počítačového vidění zásadní.

Pochopení a využití počítačového vidění by však nebylo možné bez zásadních pokroků v oblasti výpočetní techniky a algoritmických inovací. Výkonný hardware přizpůsobený přímo pro potřeby náročným výpočtům je jedním ze základních pilířů, na kterých stojí současný rozvoj a široká škála aplikací zaměřených na počítačové vidění [1].

1.1 Historie počítačového vidění

První kroky v oblasti počítačového vidění jako vědecké disciplíny byly učiněny v 60. letech 20. století, současně s rozvojem průzkumu umělé inteligence. První úlohy byly zaměřené na identifikaci předmětů v obraze z kamery, která byla připojená k tehdejšímu počítači. Tyto prvotní úlohy byly největším impulzem k důkladnějšímu zkoumání a rozvoji oblasti počítačového vidění [1].

Přestože počáteční experimenty s počítačovým viděním často narážely na technologická omezení a nedosahovaly požadovaných výsledků, 70. a 80. léta 20. století přinesla pokrok díky pokročilým matematickým analýzám obrazu a inovacím v oblasti optických senzorů a metod zpracování obrazu. Tento výzkum umožnil v 80. letech 20. století vývoj systémů, které byly aplikovatelné v průmyslové praxi, jako například automatizované systémy pro kontrolu svárů ve výrobních procesech. Tyto rané průmyslové aplikace, přestože byly nákladné a specializované na specifické úkoly, představovaly důležitý milník v praktickém využití v oblasti počítačového vidění [1].

Devadesátá léta přinesla významný pokrok díky snižujícím se nákladům na výrobní komponenty a současně rostoucímu výpočetnímu výkonu počítačů, což vedlo ke zmenšování hardwaru. Tento trend umožnil širší implementaci počítačového vidění do praxe, zahrnující například systémy pro čtení čárových kódů nebo převod psaného textu do digitální podoby, kdy tyto systémy známe pod názvem optické rozpoznávání znaků (OCR) [1].

Další expanze počítačového vidění zaznamenala průnik do oblastí, jako jsou biometrické systémy pro rozpoznávání obličejů a otisků prstů. Velký milník v oblasti analýzy dat lze zaznamenat zejména v medicíně, kde významně výzkum zpracování obrazu přispěl k analýze obrazových dat. Tento průlom umožnil zvýšení přesnosti diagnostiky a otevřel cestu k vývoji nových metod léčby [3].

Na počátku 21. století dochází k významnému pokroku v metodách umělé inteligence a strojového učení, což umožňuje jejich širší aplikaci v běžném životě. Integrace počítačového vidění do mobilních a počítačových operačních systémů je jedním z příkladů tohoto trendu. Díky technologiím počítačového vidění dochází k efektivnějšímu řízení dopravy, pokročilému statistickému zpracování dat o pohybu osob v obchodních a veřejných prostorách a k inovacím v oblasti bezpečnosti dopravy, včetně vývoje autonomního řízení vozidel [3].

Paralelně s těmito pokroky se obor potýká s výzvami, jako je zajištění adekvátních trénovacích dat, optimalizace algoritmů pro lepší přenosnost a rychlost, a řešení etických dilemat spojených s ochranou osobních údajů a právních aspektů odpovědnosti. Tyto výzvy vyžadují pečlivou pozornost a systematický přístup k nalezení udržitelných řešení [3].

1.2 Základní principy a definice

Oblast počítačové vidění je založena na principech kognitivní vědy, která se zabývá extrahováním, analýzou a porozuměním informací z multimediálního obsahu, který je reprezentován formou obrazu nebo videa. Systémy počítačového vidění tak umožňují porozumění obsahu a reprezentování uživateli pochopitelnou formou. V tomto procesu probíhá řada kroků, od předzpracování obrazu, přes detekci nebo segmentaci objektů, až po klasifikaci a rozpoznávání vzorů, kdy jsou tyto kroky přípravou pro pozdější využití v umělých neuronových sítích, patřících k technikám umělé inteligence.

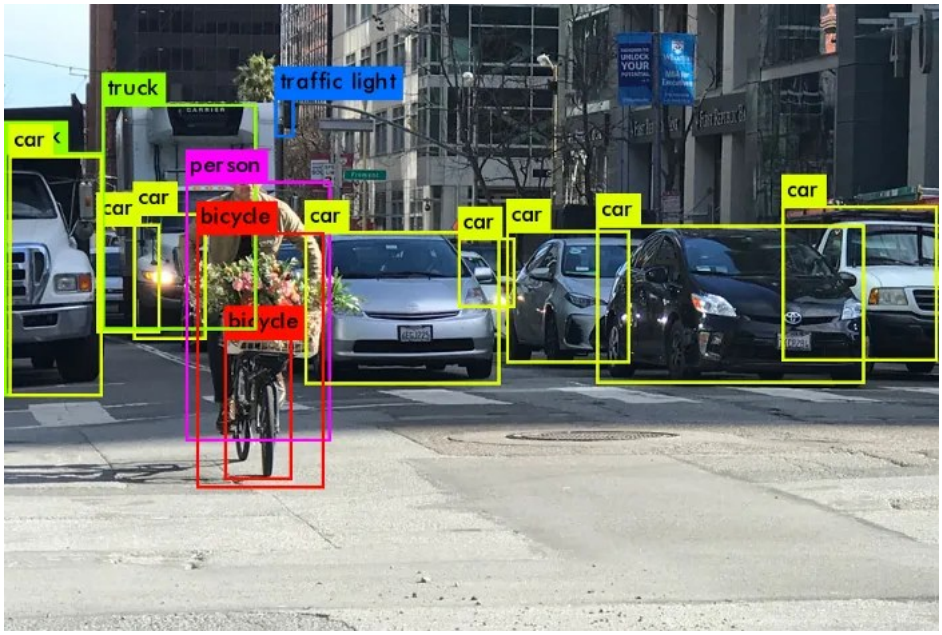
Jako prvotním krokem, před samotným trénováním neuronové sítě, je třeba získat vhodnou datovou sadu. Jednou z možností, jak takovou datovou sadu sestavit, je pořízení obrazu prostřednictvím kamer a snímačů. Pořízení kvalitní datové sady je pro úspěšné trénování modelu klíčové, protože kvalita pořízených dat ovlivňuje úspěšnost detekce dosud neviděných dat.

Dalším krokem je definovat si zájmovou oblast. To zahrnuje především jasně definovaných tříd. Tyto třídy jsou klíčové, jelikož představují kategorie, které jsou přiřazeny jednotlivým objektům v datové sadě. Pokud se například vyvíjí systém, který má detekovat dopravní značky, mohou být jako třídy konkrétní názvy těchto značek, které se na silnici vyskytují. Toto třídění do jednotlivých tříd později umožňuje v procesu trénování kategorizovat data podle naučených charakteristik. Správné definování tříd, a zařazení správných dat do správné třídy je nezbytné pro efektivní učení, protože každá třída musí obsahovat dostatečné množství trénovacích dat, aby mohlo být extrahováno dostatečné množství informací.

Po sestavení datové sady je třeba data vhodně anotovat. Anotování datové sady se liší na základě toho, jakým způsobem chceme k výsledné klasifikaci nebo detekci přistupovat. Pokud probíhá klasifikace obrazových dat, tak anotace probíhá tak, že je celý obsah obrazu přiřazený k jedné konkrétní třídě. Tento typ anotace je velmi jednoduchý, nicméně jeho největší nevýhodou je, že velmi snadno je možné se dostat do určité konfliktu ve zvolení třídy. Například pokud máme klasifikátor pro detekci psů a koček, a v datové sadě budeme mít

fotografii, která obsahuje obě zvířata, není možné k takové fotografii přiřadit dva štítky naráz, jelikož by v trénovacím procesu mohl vzniknout problém v extrakci příznaků.

Druhou možností, vedle uvedené klasifikace, je detekce obrazových dat. Jedná se o princip, kdy při anotování dat jsou přiřazovány štítky, ke konkrétnímu objektu, který je zvolený formou ohraničujících rámečků (*bounding box*). Jedná se o vizuální metodu, kde je kolem objektu v obraze nakreslený obdélník, který slouží k jeho identifikaci a lokalizaci. Tento obdélník nese informace v podobě souřadnice, kde se v prostoru nachází a také nese informaci o názvu třídy. Tento obdélník umožňuje algoritmu pochopit a naučit se, kde přesně se objekt nachází, jak velký je a o jakou třídu se jedná. Výhodou této metody je možnost přiřazování více štítků k různým objektům na jednom obrázku, což umožňuje vyšší přesnost a efektivitu v rozpoznávacích úlohách [3].



Obrázek 1.1 Ukázka ohraničujících rámečků, převzato z [2]

2 ZPRACOVÁNÍ OBRAZU

Ačkoliv je digitální zpracování obrazu postavené na matematických základech, při výběru techniky často hraje lidská intuice a analýza, která je často založena na subjektivních vizuálních úsudcích. Při zpracování obrazu, ale v některých klasifikačních a detekčních metodách na bázi umělých neuronových sítí, je důležitá základní mechanika toho, jak jsou obrazy tvořeny a vnímány [4].

2.1 Percepce obrazu

V roce 1666 Isaac Newton zjistil, že když paprsek slunečního světla prochází skleněným hranolem, není vystupující paprsek bílý, ale skládá se ze souvislého spektra barev od fialové na jednom konci po červenou na konci druhém. Škálu barev, kterou vnímáme ve viditelném spektru, je jen malou částí elektromagnetického spektra. Na jednom konci spektra jsou rádiové vlny s vlnovou délkou miliardkrát delšími, než je vlnová délka viditelného světla. Na druhém konci spektra jsou gama paprsky s vlnovými délkami milionkrát kratšími, než je u viditelného světla.

Barvy vnímané v objektu jsou určeny charakterem světla odráženého objektem. Těleso, které odráží světlo relativně vyvážené ve všech viditelných vlnových délkách, se pozorovateli jeví jako bílé. Avšak těleso, které upřednostňuje odraz v omezeném rozsahu viditelného spektra, vykazuje některé barevné odstíny. Například zelené předměty odrážejí světlo s vlnovými délkami nejčastěji v rozsahu 500 až 570 nm, zatímco většinu energie ostatních vlnových délek pohlcují. Světlo, které nemá žádnou barvu, se nazývá monochromatické světlo. Jedinou vlastností monochromatického světla je jeho intenzita. Vzhledem k tomu že intenzita monochromatického světla je vnímána jako proměnlivá od černé přes šedou až bílou, používá se pro označení intenzity monochromatického světla běžně pojem úroveň šedi. Chromatické (barvené) světlo pokrývá elektromagnetické spektrum přibližně od 0,43 do 0,79 μm. Pro popis chromatického světla se používají tři veličiny: zářivost, svítivost a jas. Zářivost je celkové množství energie, které proudí ze světelného zdroje, a obvykle se měří ve wattech. Svítivost, udávaná v lumenech, vyjadřuje množství energie, které pozorovatel vnímá ze světelného zdroje. Poslední veličinou je jas, který je subjektivním ukazatelem vnímání světla a prakticky je nemožné jej změřit [4].

2.2 Barevné modely

Barevný model lze matematicky popsat, jakým způsobem lze barvy reprezentovat. Různé barevné modely zachycují vlastnosti barevného spektra. Jedním z důležitých aspektů barevných modelů je technika míchání barev.

Aditivní míchání barev pracuje na principu sčítání jednotlivých barevných složek. Tento typ míchání využívá tři základní barevné složky: červenou, zelenou a modrou. Když se tyto barvy kombinují, výsledná světelná intenzita je součtem intenzit jednotlivých složek.

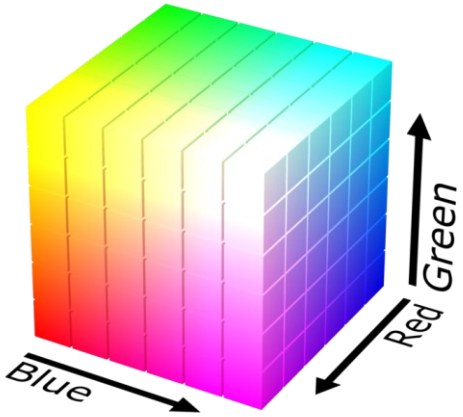
Subtraktivní míchání barev je základem pro tiskové technologie. V tomto principu míchání barev se jako základní barvy využívá žlutá, azurová a purpurová, které jsou komplementární k základním barvám aditivního způsobu míchání barev. Při subtraktivním míchání barev, dochází s každou další přidanou barvou ubrání původního světla. To znamená že pokud dojde ke smíchání všech tří základních barev dosáhne se černé barvy.

Různé barevné modely zachycují jedinečné vlastnosti barevného spektra, a mohou tak být přizpůsobeny konkrétním podmínkám a požadavkům. Příkladem mohou být barevné modely, které oddělují informace o barvě od informací o intenzitě, jako je HSV, a umožňují tak robustnější výkon v úlohách, kde se světelné podmínky značně liší.

Pokročilé aplikace počítačového vidění navíc využívají barevné modely nejen pro statickou analýzu obrazu, ale také pro zpracování v reálném čase, rozšířené realitě a dalších aplikací. Správné volba barevného modelu může zvyšovat výslednou přesnost detekce nebo klasifikace objektů [4].

2.2.1 RGB

RGB barevný model (z anglických slov *Red*, *Green*, *Blue* – červená, zelená, modrá), je aditivní způsob míchání barev. Intenzita výsledného světla odpovídá součtu intenzit těchto složek, což umožňuje různými kombinacemi vytvářet širokou škálu barev. V 8bitovém formátu má každá barva rozsah od 0 do 255, což dává možnost vytvořit přes 16 milionů různých odstínů. Takový rozsah barev může být klíčový v oblastech, jako je zpracování lékařských dat, kde jemné rozdíly v barvách na snímku mohou odpovídat významným rozdílům. Příkladem může být detekce různých příznaků onemocnění kůže, kde správné vyhodnocení barvy je zásadní pro diagnózu a případnou pozdější léčbu [4].



Obrázek 2.1 RGB model, převzato z [5]

2.2.2 BGR

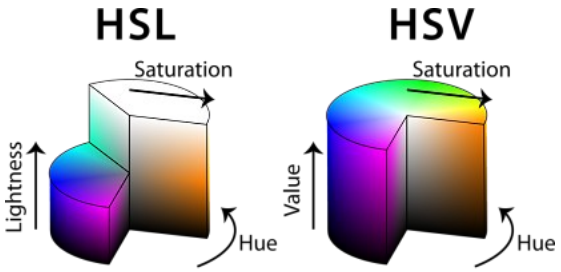
Barevný model BGR (*Blue, Green, Red*) není skutečně barevným modelem. Jedná se pouze o jiné uspořádání barevných kanálů, které vychází od klasické prezentace barevného modelu RGB. V oblasti počítačového vidění je možné se s tímto přístupem setkat, a některé algoritmy mohou využívat pro zpracování obrazu takto uspořádané kanály. Oproti barevnému modelu RGB, z kterého vychází BGR nepřináší žádné výhody, ale ani nevýhody. Využívání tohoto barevného modelu je do značné míry historickým důvodem. V některých knihovnách, které jsou využívány pro zpracování obrazu, jsou implementovány funkce, které obraz zpracovávají v tomto barevném modelu. BGR byl původně přijat kvůli preferencím výrobcům hardwaru. Ačkoliv se jedná již o historický standard, obecně platí, že pokud dojde k

takovému zavedení standardu, mají tendenci přetrvávat z důvodu zpětné kompatibility, i když již nenabízí žádnou technologickou výhodu.

Navzdory různému uspořádání kanálů v barevných modelech RGB a BGR, různé pokusy prokázaly, že oba barevné modely dosahují srovnatelných výsledků při řešení úloh, jako je klasifikace, detekce nebo segmentace obrazu. Z tohoto důvodu novější modely, které pracují s obrazem mají implementované postupy, které při předzpracování obrazu všechny datové soubory převádějí na shodný barevný model, kterým ve většině případů je RGB [6].

2.2.3 HSV a HSL

Barevné modely HSV (*Hue, Saturation, Value* – odstín, sytost, hodnota) a HSL (*Hue, Saturation, Lightness* – odstín, sytost, světlost) jsou založeny na lidském vnímání barev a nabízí intuitivní metody pro popisování barevných odstínů. Barevný model HSV rozděluje barvy na odstín, sytost a hodnotu, kde hodnota popisuje světlost barvy. Na druhé straně, barevný model HSL zaměňuje hodnotu za světlost, což odráží míru světla od čistě bílé po čistě černou, přičemž odstín se mění po obvodu válce.



Obrázek 2.2 HSL a HSV model, převzato z [7]

Využití barevného modelu HSV může být výhodné, protože odděluje barvu (odstín) od světlosti (hodnoty). Toto oddělení může pomoci při procesu trénování modelu lépe zobecnit různá osvětlení [8].

2.3 Předzpracování obrazu

Cílem předzpracování obrazu před samotným trénováním dat, je zvýšení výkonu a kvality modelů neuronových sítí. Tyto operace jsou nezbytné, protože pomáhají potlačovat nežádoucí zkreslení nebo vylepšit určité vlastnosti, které jsou důležité pro další zpracování a analýzu datových souborů [9].

V této kapitole jsou zahrnuté obecné kroky, které se využívají při předzpracování obrazu v oblasti počítačového vidění.

2.3.1 Změna velikosti

Jedním z důležitých kroků v počítačovém vidění je změna velikosti všech vstupní obrazů, na společnou velikost. Tento krok zajišťuje jednotnost všech vstupů. Nutnost jednotného rozměru vstupních dat je zásadní, jelikož většina modelů počítačového vidění vyžaduje vstupní data, která mají konzistentní velikost, aby byla zachována jednotnost výpočtů. Tento požadavek vyplývá z architektonického návrhu konvolučních neuronových sítí (CNN), které se běžně používají při zpracování obrazu. Detailně jsou konvoluční neuronové sítě rozebrány v kapitole 3.1.2. Tyto sítě využívají filtry nebo jádra, která konvolují nad obrazem v definované velikosti okna. Pokud by se velikost vstupních obrazů lišila, výstupní rozměry konvolučních vrstev by se nesrovnaly, což by vedlo k výpočetním problémům a neefektivitě ve fázích trénování a odvozování.

Změna velikosti obrázků na pevný rozměr, například 416x416 pixelů, je běžnou praxí, která nejen pomáhá dosáhnout souladu modelu, ale také zvyšuje výpočetní efektivitu. Menší obrázky snižují výpočetní zátěž, což umožňuje rychlejší zpracování.

Jednotná velikost obrazu navíc zajišťuje, že extrahované rysy jsou měřítkově invariantní, což znamená, že schopnost modelu rozpoznávat objekty nezávisí na jejich velikosti ve vstupním obraze. To má zásadní význam pro zachování přesnosti a spolehlivosti modelu.

Je však důležité zvážit důsledky změny velikosti obrazu na kvalitu předpovědí modelu. Přílišné zmenšení velikosti může vést ke ztrátě detailů, což může zakrýt důležité rysy nezbytné pro přesnou detekci nebo klasifikaci objektů. Naopak velmi velké vstupní velikosti mohou zvýšit detailnost, ale za cenu zvýšení výpočetních nároků a doby zpracování [4].

2.3.2 Normalizace

Normalizace obrazových dat je dalším důležitým krokem v rámci procesu předzpracování obrazu před samotným trénováním. Tento postup zahrnuje úpravu rozsahu hodnot intenzity pixelů na standardní stupnici, obvykle mezi 0 a 1. Taková normalizace je nezbytná, protože usnadňuje rychlejší konvergenci během procesu trénování tím, že poskytuje konzistentní stupnici pro všechny vstupní hodnoty, a tím pomáhá při optimalizaci sestupu gradientu.

Důvod normalizace spočívá v povaze početních operací v neuronových sítích, zejména ve způsobu zpracování dat ve vrstvách. Sítě fungují optimálně, když mají prvky vstupních dat podobná měřítka, protože to podporuje stabilní konvergenci a zabraňuje převaze určitých prvků nad jinými jen kvůli jejich měřítku. Normalizace vstupů může urychlit fázi učení hlubokých sítí tím, že zajistí, aby gradienty nebyly ani příliš malé, ani příliš velké a aby se kroky učení pohybovaly ve vhodném rozsahu.

Normalizaci je možné provést vydělením původních hodnot pixelů maximální možnou hodnotou (často 255 u 8bitových obrázků), čímž se všechny hodnoty dostanou do požadovaného rozsahu 0 až 1. Tato transformace nejen pomáhá při trénování neuronových sítí, ale také odpovídá požadavkům na předzpracování různých architektur, které jsou citlivé na měřítko a rozložení vstupních dat [10].

2.3.3 Změna pořadí kanálů

Moderní architektury neuronových modelů, které kromě samotné sítě zahrnují i postupy pro trénování neuronových modelů mají v procesu předzpracování zahrnutý krok, který upravuje pořadí kanálů. Tento krok zahrnuje úpravu pořadí barevných kanálů v obraze, obvykle převod z barevného modelu BGR na barevný model RGB. Tato změna pořadí je nutná, protože různé obrazové formáty a softwarové knihovny používají různé konvence pro ukládání barevných informací (Kapitola 2.2.2).

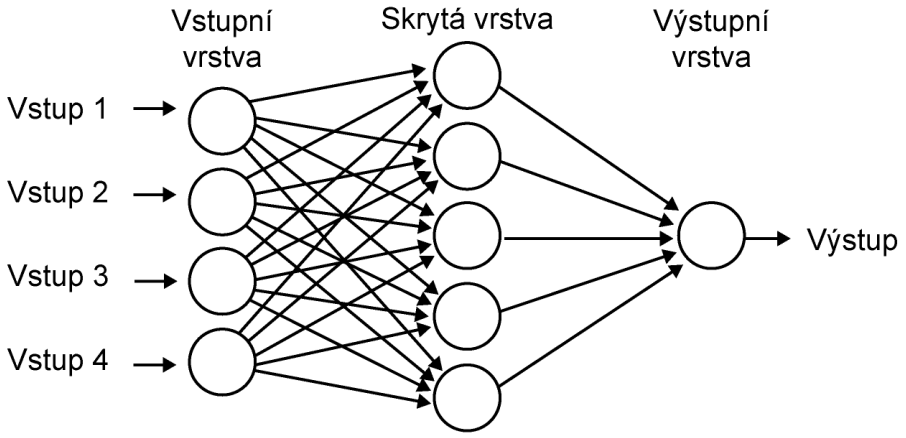
Význam změny pořadí kanálů vyplývá z požadavků stanovených architekturou většiny předtrénovaných modelů, které jsou obvykle trénovány na sadách dat, kde jsou obrázky formátovány v barevném modelu RGB. Široké využití barevného modelu RGB je z toho důvodu, jelikož odpovídá lidskému zraku, který vnímá barvy především v podobě červené, zelené a modré složky. Formát BGR naopak běžně používají některé softwarové knihovny pro zpracování obrazu, například OpenCV, který vychází z jiné počítačové tradice.

Nesprávná změna pořadí kanálů může vést k nesprávné interpretaci barevných dat modelem, což může vést ke zhoršení výkonu modelu a jeho schopnosti zobecnit tréninková data na reálné aplikace [11].

3 NEURONOVÉ SÍTĚ

Neuronové sítě jsou inspirovány biologickými neuronovými sítěmi, které jsou součástí mozku každého živočicha. Umělá neuronová síť (ANN) je systém vzájemně propojených uzlů neboli „neuronů“, které spolupracují při plnění konkrétních úkolů. Ve své podstatě neuronová síť zpracovává vstupy prostřednictvím vrstev neuronů, z nichž každá je schopna provádět jednoduché výpočty [12].

Architektura neuronové sítě obvykle zahrnuje tři hlavní vrstvy: vstupní vrstvu, skryté vrstvy a výstupní vrstvu. Každý neuron v jedné vrstvě se spojuje s neurony v další vrstvě a vytváří síť, která dokáže modelovat složité vzory v datech. Síla těchto spojení, známá jako váhy, se během procesu trénování upravuje tak, aby se minimalizoval rozdíl mezi předpovědí sítě a skutečnými výsledky dat [13].



Obrázek 3.1 Uspořádání neuronů do vrstev, převzato z [14]

Neuronové sítě jsou proslulé zejména svou schopností předpovídat a rozpoznávat vzory, díky čemuž se staly neocenitelnými v různých oblastech, jako jsou finance, zdravotnictví, robotika a další. V oblasti počítačového vidění neuronové sítě dramaticky zlepšily schopnost strojů rozpoznávat a interpretovat obrázky a videa a často překonávají výkony na úrovni člověka v úlohách, jako je rozpoznávání objektů [15].

3.1 Typy neuronových sítí

Neuronové sítě, se vyvíjejí v různých architekturách, z nichž každá je přizpůsobena k efektivnímu řešení konkrétních typů problémů. Každý typ sítě, ať už se jedná o síť dopřednou,

konvoluční nebo rekurentní, nabízí jedinečné vlastnosti, díky nimž je vhodný pro různé druhy úloh zpracování dat a rozpoznávání vzorů [12].

3.1.1 Dopředné neuronové sítě

Dopředné neuronové sítě, známé také jako vícevrstvé perceptrony (MLP), jsou nejjednodušším typem architektury umělých neuronových sítí. V těchto sítích se informace pohybují pouze jedním směrem – dopředu – ze vstupních uzlů, přes skryté uzly (pokud existují) a nakonec do výstupních uzlů. V síti nejsou žádné cykly ani smyčky, což je odlišuje od rekurentních neuronových sítí.

Obrázek 3.2 Schéma dopředné neuronové sítě s jednou skrytou vrstvou, převzato z [16]

Základní princip fungování dopředných neuronových sítí spočívá ve zpracování vstupních dat prostřednictvím řady vrstev. Každá vrstva je tvořena řadou vzájemně propojených uzlů neboli neuronů, kde každé spojení představuje váhu. Tyto váhy jsou během tréninku upravovány tak, aby se minimalizovala chyba při předpovídání výstupu ze vstupních dat. Uzly v každé vrstvě aplikují nelineární aktivační funkci na vážený součet svých vstupů, což zavádí schopnost modelovat nelineární vztahy mezi vstupními a výstupními daty [12].

Dopředné neuronové sítě se obvykle trénují pomocí metody známé jako zpětné šíření, která efektivně počítá gradienty ztrátové funkce vzhledem k vahám sítě. Tento proces iterativně upravuje váhy tak, aby minimalizoval ztrátovou funkci, která je obvykle mírou rozdílu mezi skutečným výstupem a předpovídaným výstupem sítě. Tento proces učení vyžaduje datovou sadu obsahující dvojice vstupů a výstupů, což umožňuje síti naučit se mapování ze vstupů na výstupy pod dohledem [17].

Dopředné sítě se dobře hodí pro různé úlohy, kde je vztah mezi vstupem a výstupem statický a kde není důležitá posloupnost nebo časový průběh jednotlivých datových bodů. Před nástupem specializovanějších sítí, jako jsou konvoluční neuronové sítě, byly úspěšně používány v aplikacích od jednoduché binární klasifikace až po složitější problémy, jako je rozpoznávání obrazu. Obecně však nejsou vhodné pro úlohy, kde je důležitý kontext nebo posloupnost, jako je například predikce časových řad nebo zpracování přirozeného jazyka, kde rekurentní neuronové sítě nebo jiné modely obvykle fungují lépe [18].

Přes svou jednoduchost mohou dopředné neuronové sítě modelovat složité rozhodovací hranice a byly nápomocny při demonstraci potenciálu hlubokého učení. Jejich úspěšnost však může být omezena potřebou velkého množství označených dat. Kromě toho mohou být náchylné k nadměrnému přizpůsobení, zejména když se architektura sítě prohlubuje nebo stává složitější [19].

3.1.2 Konvoluční neuronové sítě

Konvoluční neuronové sítě (CNN) představují specializovaný druh neuronových sítí, které jsou obzvláště výkonné pro zpracování dat, která mají mřížkovou topologii, jako jsou například obrázky. Konvoluční neuronové sítě se vyznačují schopností automaticky a adaptivně se učit od hran a textur nízké úrovně až po vzory a třídy objektů vysoké úrovně, a to pomocí konvolučních filtrů [20].

Architektura konvolučních neuronových sítí obvykle zahrnuje několik vrstev, které se skládají z konvolučních vrstev, aktivační funkce, sdružovací vrstvy a plně propojené vrstvy. Konvoluční vrstva aplikuje na vstup sérii naučitelných filtrů. Každý filtr aktivuje určité funkce na určitých prostorových pozicích. Vrstvy pro sdružování pak zmenšují prostorovou velikost reprezentace, takže detekce rysů je invariantní vůči malým posunům a zkreslením vstupního obrazu. Aktivační funkce zavádějí do sítě nelinearity, které jí umožňují učit se složitější vzory [12].

Obrázek 3.3 Ukázka konvoluční neuronové sítě, převzato z [21]

Konvoluční neuronové sítě jsou vysoce efektivní pro rozpoznávání obrázků a videí, analýzu lékařských snímků a jakýkoli jiný typ analýz, které zahrnují prostorová data. Tato efektivita vychází z jejich schopnosti učit se reprezentace příznaků přímo ze zdrojových dat, čímž se minimalizuje potřeba manuální extrakce rysů. Jejich vícevrstvá architektura pomáhá rozpoznávat rysy na různých úrovních abstrakce, čímž se stávají mimořádně univerzálními a výkonnými pro úlohy počítačového vidění [22].

Konvoluční neuronové sítě však nejsou bez omezení. Při trénování na malých souborech dat, mohou být náchylné k nadměrnému přizpůsobování. Při trénování také vyžadují značný výpočetní výkon a paměť, zejména s rostoucí hloubkou a rozsahem sítě. To je může činit méně vhodnými pro zařízení s omezenými zdroji. Kromě toho, zatímco jsou vynikající pro

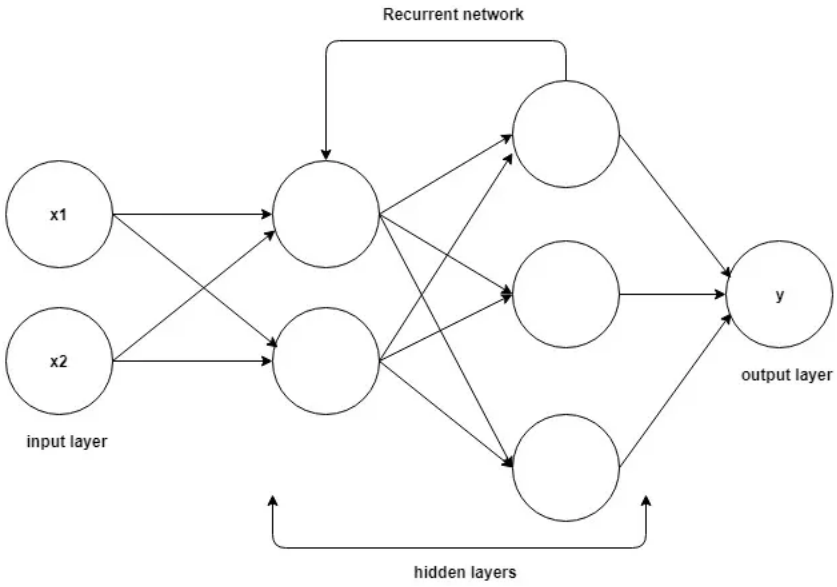
mřížkovou topologii dat, konvoluční neuronové sítě jsou méně efektivní pro zpracování sekvenčních dat, například při analýze časových řad nebo zpracování přirozeného jazyka. Pro tyto typy dat mohou být vhodnější rekurentní neuronové sítě [23].

3.1.3 Rekurentní neuronové sítě

Rekurentní neuronové sítě (RNN) je typem neuronových sítí, které jsou klíčové při modelování sekvenčních dat. Na rozdíl od dopředných neuronových sítí se rekurentní neuronové sítě vyznačují tím, že si udržují "paměť" předchozích vstupů pomocí svého vnitřního stavu (skryté vrstvy), což jim umožňuje vykazovat časově dynamické chování. Díky této schopnosti jsou rekurentní neuronové sítě zvláště vhodné pro aplikace, kde vstupem není pevný vektor, ale posloupnost dat, jako jsou data časových řad, řeč, text nebo jakákoli jiná forma sekvenčních informací [18].

Základní myšlenkou rekurentních neuronových sítí je využití sekvenčních informací. V klasické neuronové síti jsou všechny vstupy a výstupy na sobě nezávislé. Rekurentní neuronové sítě tento problém řeší tím, že v nich existují smyčky, které umožňují, aby informace přetrvávaly z jednoho kroku úlohy do dalšího. Tento mechanismus smyček jim propůjčuje schopnost zpracovávat nejen jednotlivé datové body, ale celé sekvence dat.

Trénování standardních rekurentních neuronových sítí však může být náročné kvůli problémům, jako jsou mizející gradienty, kdy se gradienty zmenšují a velmi ztěžují učení závislostí, a na základě toho mohou způsobit nestabilitu celého procesu učení [24].



Obrázek 3.4 Ukázka rekurentní neuronové sítě, převzato z [25]

3.2 Šíření chyby

Mezi nejznámější způsoby šíření chyby je zpětné šíření chyby (backpropagation), který je základním principem při trénování neuronových sítí, zejména v procesu učení vícevrstvých sítí. Tato metoda se používá k efektivnímu výpočtu gradientu ztrátové funkce sítě ve vztahu k jejím vahám. Tato technika hraje klíčovou roli ve fázi učení, kde pomáhá optimalizovat neuronovou síť minimalizací ztrátové funkce.

Zpětné šíření chyby bylo zpopularizováno v 80. letech 20. století jako praktický způsob trénování hlubokých neuronových sítí, což byl významný pokrok v oblasti strojového učení [26].

Při zpětném průchodu se chyba šíří zpět sítí od výstupní vrstvy směrem ke vstupní vrstvě, což umožňuje výpočet gradientu ztrátové funkce vzhledem ke každé váze pomocí řetězového pravidla výpočtu.

Hlavní výhodou šíření chyb je jeho účinnost při zpracování složitých sítí s mnoha vrstvami. Umožňuje síti učit se z dat úpravou vah tak, aby se minimalizovala chyba, a tím se v průběhu iterací zlepšovala přesnost předpovědí. Tato metoda je také velmi flexibilní a lze ji aplikovat na téměř jakoukoli architekturu neuronové sítě, což z ní činí univerzální nástroj pro úlohy hlubokého učení [15].

Šíření chyby má i své nevýhody. Jedním z hlavních problémů je problém mizejících gradientů, kdy se gradienty ztrátové funkce stávají velmi malými, což efektivně brání změnám hodnot vah, což může zastavit proces učení. Tento problém se projevuje zejména u hlubokých sítí s mnoha vrstvami. Může se také vyskytnout problém explodujícího gradientu, kdy gradienty rostou exponenciálně, což vede k nestabilitě procesu učení [24].

3.3 Proces učení

Proces učení v neuronových sítích, často označovaný jako trénování, je zásadní pro to, aby tyto výpočetní modely mohly provádět úlohy, jako je klasifikace, regrese a predikce. Tento proces zahrnuje úpravu vnitřních parametrů sítě – především vah spojení mezi neurony – v reakci na data. Cílem je minimalizovat rozdíl mezi předpovídaným výstupem sítě a skutečným výstupem v trénovacích datech, čehož se obvykle dosahuje metodou známou jako zpětné šíření chyby [26].

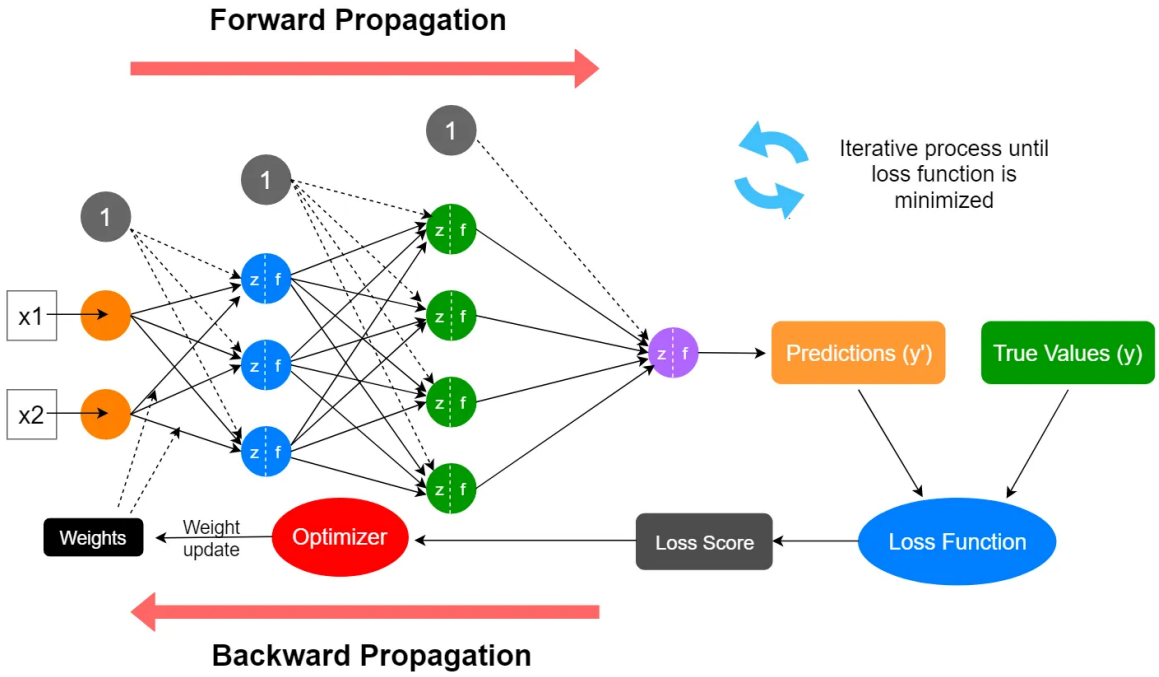
Základem trénování neuronových sítí je optimalizace ztrátové funkce, známé také jako *cost function*, která měří chybu mezi předpovídaným výstupem a skutečným výstupem. Mezi běžné příklady ztrátových funkcí patří střední kvadratická chyba pro regresní úlohy a křížová entropie pro klasifikační úlohy. Optimalizace se obvykle provádí pomocí gradientního sestupu nebo sofistikovanějších variant, jako je stochastický gradientní sestup (SGD) nebo Adam. Tyto optimalizační algoritmy iterativně upravují váhy s cílem snížit ztrátu [27].

Během každé iterace trénování síť zpracuje dávky dat, pro každou z nich vypočítá výstup, porovná jej se skutečným výstupem a vypočítá chybu. Algoritmus zpětného šíření pak vypočítá gradient ztrátové funkce vzhledem ke každé váze v síti tak, že použije řetězové pravidlo, základní techniku z kalkulu, k šíření chyby zpětně sítí. Tato informace se použije k aktualizaci vah s cílem minimalizovat ztrátu, přičemž se obvykle upravují váhy v opačném směru, než je gradient [15].

Jedním z kritických problémů při trénování neuronových sítí je vyhnout se nadměrnému přizpůsobení, kdy se model příliš dobře naučí trénovací data, na úkor své schopnosti zobecnit se na nová data. Techniky, jako je regularizace (např. L1 a L2), dropout a předčasné zastavení, jsou běžně používány k zabránění nadměrnému přeučení. Regularizační metody přidávají penalizaci, předčasné zastavení zastaví proces trénování, jakmile se výkonnost modelu přestane zlepšovat na validačním souboru dat [19].

Kromě toho při trénování neuronových sítí je kritickým parametrem rychlost učení, která určuje velikost kroku v každé iteraci. Příliš vysoká rychlost učení může způsobit, že proces trénování bude příliš rychle konvergovat k neoptimálnímu řešení, zatímco příliš nízká rychlost může způsobit, že proces trénování bude příliš pomalý a může se zaseknout v lokálních minimech.

Proces učení (Obrázek 3.5) je iterativní a může být výpočetně náročný, zejména u velkých sítí a datových sadách. Vývoj specializovaného hardwaru, jako jsou grafické karty a zařízení přímo určené pro zpracování neuronových sítí, však tento proces výrazně urychlil, což umožňuje trénovat složité modely na velkých datových sadách v proveditelném časovém rámci [28].



Obrázek 3.5 Proces učení, převzato z [29]

V prvotní fázi je proces učení zahájen dopředným šířením (*forward propagation*), kde vstupní data (*x1, x2*) procházejí od vstupní vrstvy přes skryté vrstvy až do výstupní vrstvy. Každý neuron v těchto vrstvách používá váhy (*weight*), práh (*bias*) a aktivační funkce k transformaci vstupů na výstupy [29].

Výstupy (*predictions y'*) jsou následně porovnány se skutečnými hodnotami (*true values y*) pomocí ztrátové funkce (*loss function*). Tato funkce vyhodnocuje chybu mezi predikovanými a reálnými hodnotami, což je zohledněno ve ztrátovém skóre (*loss score*). Cílem učícího procesu je minimalizace ztrátového skóre, což následně zvyšuje přesnost modelu [29].

Následuje fáze zpětného šíření (*backward propagation*), během níž se vypočtená ztráta šíří zpět skrze síť. Během tohoto procesu se vypočítají gradienty ztrátové funkce vzhledem k jednotlivým váhám, což ukazuje, jakým způsobem by měly být váhy upraveny pro snížení celkové ztráty [29].

Pro aktualizaci vah se využívá optimalizační funkce (*optimizer*), jako například SGD nebo Adam. Tyto optimalizační algoritmy upravují váhy tak, aby byla minimalizována ztrátová funkce a tím i celková chyba modelu [29].

Po každém zpětném průchodu následuje nový dopředný průchod s aktualizovanými váhami, a celý cyklus (dopředné šíření, výpočet ztráty, zpětné šíření) se opakuje. S každou iterací dochází k postupnému zlepšování vah, což by mělo vést ke snížení ztrátového skóre a ke zlepšení přesnosti modelu [29].

Trénink obvykle probíhá v epochách, kde jedna epocha znamená jeden kompletní průchod celou trénovací datovou sadou. Data mohou být rozdělena do menších dávek, které se zpracovávají postupně. V průběhu jedné epochy tedy dojde k několika aktualizacím parametrů modelu [29].

4 DETEKCE OBJEKTŮ

V oblasti počítačového vidění jsou nejvíce zastoupeny dvě základní úlohy: klasifikace a detekce objektů. Tyto úlohy jsou klíčové v širokém spektru aplikací, včetně medicíny, autonomního řízení nebo v automatizovaném průmyslu.

Klasifikace obrazu je tou nejzákladnější formou úlohy v počítačovém vidění. Jedná se o přiřazení třídy celému obrazu na základě jeho obsahu. Hlavním cílem této úlohy je zařadit obraz do předem nastavených tříd.

Detekce objektů je proces identifikace a lokalizace předem definovaných tříd v digitálním obraze. Když systém identifikuje přítomnost třídy objektů v digitálním obraze, výstupem jsou souřadnice tzv. „bounding boxu," který tento objekt ohraničuje. Na základě toho je možné přesněji lokalizovat a rozlišovat více objektů různých tříd v obraze na základě specifických souřadnic [30], [31].

Algoritmy pro detekci objektů můžeme rozdělit na dvě základní metodiky založené na jejich mechanismu fungování a předpokládaných výsledcích: jednostupňové a dvoustupňové.

Jednostupňové algoritmy se vyhýbají kroku návrhu oblasti, který je nedílnou součástí dvoustupňových metod. Tyto algoritmy přímo předpovídají třídy objektů a souřadnice ohraničujícího rámečku v jediném průchodu sítí vpřed, a tím upřednostňují rychlost a schopnost zpracování v reálném čase. Význam jednostupňových modelů spočívá v jejich zjednodušené architektuře, která umožňuje rychlou detekci, ale někdy může být na úkor přesnosti detekce, zejména v hustých scénách objektů nebo u malých objektů. Jedním z populárních jednostupňových algoritmů je You Only Look Once (YOLO) [32], který funguje na principu toho, že obraz rozdělí na mřížku a pro každou buňku mřížky předpovídá ohraničující boxy a pravděpodobnost tříd. Je velmi známý svou rychlosti, tedy vhodný na aplikace v reálném čase, i když může mít problémy s malými objekty a těsně u sebe rozmístěnými objekty. Jako další jednostupňový algoritmus je Single Shot Detector (SSD) [62], který funguje tak, že předpovídá ohraničení objektů a skóre tříd v několika měřítkách pomocí jedné hluboké neuronové sítě. Vyvažuje mezi rychlostí a přesností lépe než YOLO a díky mapám příznaků ve více měřítkách je schopna detekovat objekty různých velikostí.

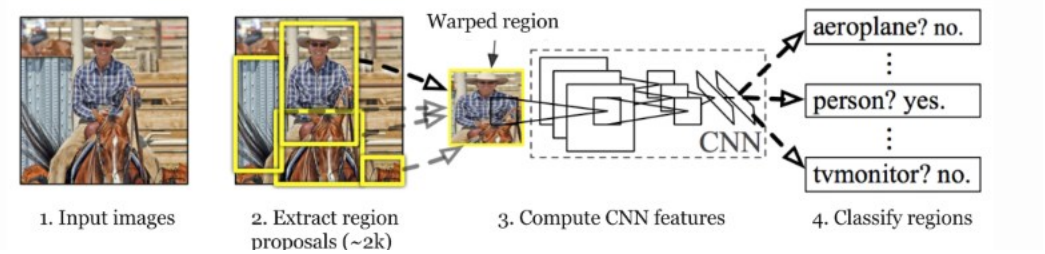
Dvoustupňové algoritmy představují jednu z metodik v oblasti detekce objektů. Patří mezi ně například algoritmy z rodiny R-CNNs [36]. Jejich počáteční krok zahrnuje generování návrhu oblastí. Tyto oblasti pak pravděpodobně obsahují objekty určené k detekci. Následuje

klasifikace každého z těchto návrhů do příslušných kategorií objektů. Součástí procesu je i upřesnění ohraničujících boxů objektů.

Tento přístup obecně přináší vyšší přesnost a preciznost, protože pečlivě zkoumá každý návrh oblasti na potenciální objekty, i když za cenu zvýšené výpočetní zátěže a nižší rychlost zpracování ve srovnání s jednostupňovými detektory [33].

4.1 R-CNNs

Region-based Convolutional Networks (R-CNNs) představují průlom v technologii detekce objektů. Ve své podstatě využívá model R-CNN vysokokapacitní konvoluční neuronové sítě (CNN) k vytváření návrhu oblastí zdola nahoru pro lokalizaci a segmentaci objektů v obrazech. Tento proces začíná použitím selektivního vyhledávání k identifikaci mnoha kandidátů na ohraničené oblasti objektů, známých také jako „oblasti zájmu." Po této identifikaci jsou z každé oblasti nezávisle extrahovány vlastnosti pro účely klasifikace [34], [35].



Obrázek 4.1 R-CNN architektura, převzato z [36]

4.1.1 R-CNN

V roce 2014 byla představena výzkumnou skupinou z Kalifornské univerzity konvoluční neuronová síť známá pod zkratkou R-CNN [36]. Tento výzkum přinesl revoluci v detekci objektů ale také v sémantické segmentaci obrazu. Výzkum se zabývá implementací vysokokapacitních konvolučních neuronových sítí k analýze potencionálních oblastí v obraze, které mohou obsahovat objekty, ke kterým může být přiřazená třída. Tyto modely, které využívají velký počet parametrů a vrstev, umožňují zachycovat složité vzory ve velkém množství dat. Tyto sítě se skládají z několika vrstev konvolučních filtrů, po niž následují plně propojené vrstvy. Aspekt vysoké kapacity označuje schopnost učit se z velkého množství dat, což umožňuje extrakci rozsáhlých a strukturovaných rysů. Tato vlastnost je činí obzvláště

efektivními pro úlohy vyžadující porozumění složitým detailům v obrazech, jako je detekce objektů, kde identifikace a rozlišení různých objektů v obraze vyžaduje analýzu jak prvků na nižší úrovni (hrany, textury), tak i atributů na vyšších úrovních (části objektů, tvary). Tento přístup umožňuje přesnou lokalizaci a segmentaci objektů v různorodých a složitějších obrazech.

Celý proces detekce začíná u prohlížení obrázku a získání odhadu, kde se mohou objekty nacházet. To se provádí pomocí algoritmů, který vytvoří přibližně 2 000 malých výřezů obrázku, kde se pravděpodobně nacházejí zájmové objekty. Při dalším kroku dochází k extrakci prvků. Pro každou z navrhovaných oblastí, používá konvoluční neuronovou síť, která dokáže automaticky určit a identifikovat různé vlastnosti objektů, jako jsou hrany, textury nebo vzory, což pomáhá při rozlišování jednoho objektu od druhého. Tyto neuronové sítě jsou z principu návrhu hluboké a na základě toho může být taková síť naučena na velmi mnoho detailů, z kterého potom vzejde velké množství parametrů. Při dalším kroku dochází ke klasifikaci objektů. Jakmile konvoluční neuronová síť zpracuje tyto části obrazu, dalším úkolem je zjistit, co každá z nich představuje. To se provádí pomocí klasifikátoru, který nahlíží na rysy extrahované konvoluční neuronovou sítí a rozhodne s určitou mírou, o jakou třídu se jedná. Každý typ objektu, který systém dokáže rozpoznat, má svůj vlastní klasifikátor vycvičený přímo speciálně pro identifikaci. Na konec systém upřesní své původní odhady, kde se objekty v obraze nacházejí. Upravuje hranice oblastí tak, aby lépe odpovídaly identifikovaným objektům. Tento krok pomáhá při přesném obkreslení objektů a zajišťuje, že konečný výstup systému nejen ví, jaké objekty se v obraze nacházejí, ale také dokáže určit kde se přesně nacházejí [37].

4.1.2 Fast R-CNN

V roce 2015 představil Ross Girshick z výzkumného týmu Microsoft Research odlišný přístup, který výrazně zefektivnil proces. Jedním z klíčových vylepšení bylo zjednodušení tréninkového procesu, který na rozdíl od vícestupňového trénovacího procesu, který je typický pro R-CNN používá pouze jednostupňový trénovací proces. Tento přístup společně optimalizuje úlohy klasifikace objektů a regrese ohraničení pomocí jednotné víceúlohové ztrátové funkce. Tato kombinace proces trénování nejen zjednodušuje, ale také výrazně urychluje.

Další zásadním vylepšením je vrstva pro sdružování oblasti zájmu, která je známá pod zkratkou RoI (Region of Interest). Tato vrstva extrahuje pro každý návrh objektu z mapy rysů vektor rysů s pevnou velikostí, čímž je zajištěna jednotnost velikosti extrahovaných rysů.

Tato jednotnost je klíčová pro následné vrstvy klasifikace a regrese ohraničení, protože jim umožňuje zpracovávat rysy stejných rozměrů.

Fast R-CNN oproti původnímu R-CNN doznal i vyšší rychlosti a přesnosti. Při trénování sítě VGG-16, která je jedním z modelů používaných pro hluboké učení, vykazuje Fast R-CNN přibližně dvakrát vyšší rychlost trénování než R-CNN a nabízí přibližně 213krát rychlejší zpracování. Na srovnávacích testech, kdy síť byla natrénovaná na datové sadě PASCAL VOC 2012 dosahovala vyšší průměrné přesnosti (mAP), což dokazuje v porovnání její vyšší výkonosti.

Poslední změnou algoritmu, který měl největší dopad na rychlost a přesnost bylo umožnění trénování celé sítě, včetně konvolučních vrstev, prostřednictvím zpětného šíření stochastického gradientního sestupu (SGD). Tím že umožňuje aktualizace ve všech vrstvách, zajišťuje Fast R-CNN důkladnější učení a přizpůsobení sítě úlohám detekce objektů [37].

4.1.3 Faster R-CNN

Jelikož selektivní vyhledávání, které jsou používané v předchozích R-CNN architekturách jsou z hlediska výpočtů velmi náročné, z toho důvodu přišla úprava v podobě nové architektury sítě, která účinně nahrazuje algoritmus selektivního vyhledávání. Tato modernizace umožňuje, aby generování návrhů regionu bylo součástí neuronové sítě. Tato úprava přináší významné zlepšení rychlosti ale i přesnosti detekce objektů. Region Proposal Network (RPN), na kterém je založená Faster R-CNN [38], je sama plně konvoluční vrstvou, prochází poslední společnou konvoluční mapou příznaků a vytváří návrhy oblastí spolu s bodovým ohodnocením, které udává pravděpodobnost přítomnosti objektu.

Faster R-CNN spojuje detektor RPN a Fast R-CNN do jedné sítě tím, že sdílí jejich konvoluční funkce. Tento sjednocený přístup umožňuje systému využívat výhod komplexního učení, při kterém si úlohy návrhu oblastní i detekce objektů mohou vzájemně zlepšovat přesnost. RPN navíc zavádí koncept kotev neboli referenčních polí, které jsou nezbytné pro efektivní předpovědní návrhů v různých měřítkách a poměrech stran. Jedná se o předem definované referenční boxy s různými měřítky a poměry stran. Tato konstrukce umožňuje síti efektivně předpovídat návrhy oblastí různých velikostí, aniž by bylo nutné vytvářet obrazovou pyramidu. Jelikož kotvy, tak i referenční pole jsou translačně invariantní, což znamená, že pokud se objekt v obraze pohne, měl by se odpovídajícím způsobem pohnout i návrh [38].

4.2 SSD

Single Shot Detector (SSD) [62] představuje pokročilou metodu detekce objektů v obrazech, využívající jednostupňový algoritmus založený na hlubokých neuronových sítích. Jeho inovativní přístup spočívá ve zjednodušení procesu detekce objektů odstraněním nutnosti generovat návrhy objektů a přímou predikcí tříd objektů a souřadnic ohraničujících boxů v jednom kroku. Tato architektura je navržena tak, aby zvyšovala rychlost a efektivitu detekce objektů. Na základě tohoto návrhu nachází v aplikacích vyžadující rychlé zpracování v reálném čase, při zachování vysoké úrovní přesnosti.

Jedním z klíčových vylepšení SSD je použití výchozích ohraničujících boxů různých poměrů stran a měřítek napříč různými mapami rysů obrazu. Tento přístup umožňuje efektivně zpracovat objekty různých velikostí a tvarů pomocí jediného průchodu sítí. Síť SSD rozšiřuje základní architekturu sítě, jako je VGG-16, přidáním pomocných konvolučních vrstev, které postupně zmenšují velikost, což umožňuje detekci objektů v různých měřítkách. Pro každé místo v mapách rysů síť SSD předpovídá posuny vzhledem k výchozím ohraničujícím boxům a pravděpodobnost pro každou třídu objektů.

Trénování SSD zahrnuje několik klíčových postupů. Jedním z prvních kroků je přiřazení každého základního boxu k nejvhodnějšímu objektu v trénovacím datasetu. SSD využívá předem definované boxy různých poměrů a velikostí, které se pokouší přiřadit k odpovídajícím objektům v obrázku. Tento proces zahrnuje výpočet míry překryvu (Jaccardův index nebo IOU – Intersection Over Union) mezi každým výchozím boxem a skutečnými ohraničujícími boxy. Box s nejvyšším překryvem je poté přiřazen k danému objektu. Dalším klíčovým aspektem je výpočet ztrátové funkce, která kombinuje lokalizační ztrátu, tedy jak přesně jsou polohy ohraničujících boxů predikovány a ztrátu spolehlivosti, která hodnotí přesnost klasifikace objektů. Lokalizační ztráta obvykle využívá Smooth L1 loss [63], zatímco pro ztrátu spolehlivosti se často používá cross-entropy loss [64]. Tato kombinace pomáhá modelu naučit se nejen rozpoznávat, kde objekty jsou, ale také co objekty jsou.

Pro zvýšení robustnosti a schopnosti generalizace modelu je důležitá augmentace dat. To zahrnuje různé techniky pro umělé rozšíření trénovací datové sady, jako je náhodné oříznutí, převrácení obrazu, změny osvětlení, a další transformace. Tyto techniky pomáhají modelu naučit se rozpoznávat objekty v různých polohách, měřítcích, a pod různými světelnými podmínkami, což výrazně zlepšuje jeho schopnost generalizace na nových, neviděných datech. Dalším aspektem trénování je správné nakládání s negativními příklady (tj. výchozími

boxy, které neodpovídají žádnému objektu). SSD implementuje metodu hard negative mining, která vybírá negativní příklady s vysokou ztrátou spolehlivosti pro zahrnutí do ztrátové funkce, čímž se zajistí, že model se bude soustředit na těžší, nesprávně klasifikované příklady. Tento přístup pomáhá udržet rovnováhu mezi pozitivními a negativními příklady a zabránit modelu, aby byl příliš zkreslený směrem k snadněji klasifikovatelným negativním příkladům.

Společně tyto postupy vytváří síť, která umožňuje vyšší rychlosti detekce při zachování vysoké míry přesnosti v reálném čase [34].

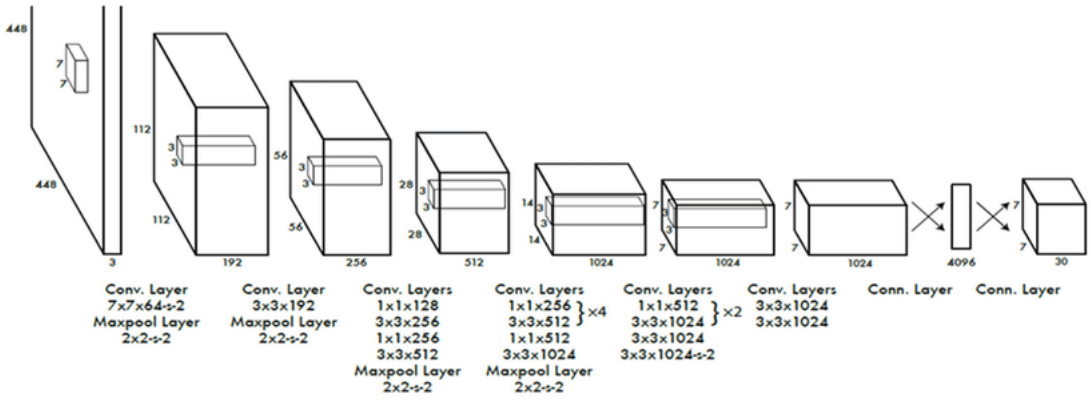
4.3 YOLO

Model YOLO (You Only Look Once) [32], který v roce 2016 byl představen Josephem Redmonem, zásadně mění přístup z tradičního dvoustupňového procesu detekce objektů na jednostupňový, který současně předpovídá více ohraničujících boxů a pravděpodobnost tříd přímo z obrazových dat. Tato inovace nejen zjednodušuje postup detekce objektů, ale také významně zvyšuje rychlost detekce a umožňuje dosáhnout výkonu v reálném čase, který byl dříve u starších modelů, jako je rodina detektorů R-CNN, nedosažitelný [32].

Základní myšlenka YOLO spočívá že detekce objektů se považuje za jediný regresní problém. Na rozdíl od metod, které generují návrhy oblastí zájmů, které jsou následně klasifikovány, YOLO rozdělí obraz na mřížku a předpovídá ohraničující boxy a pravděpodobnost pro každou buňku mřížky. Skóre důvěryhodnosti pro každý ohraničující box odráží jak pravděpodobnost, že box obsahuje určitý objekt, tak přesnost předpovězeného boxu. Tato metoda umožňuje přímé a simultánní předpovědi, které výrazně zkracují dobu zpracování, čímž se YOLO odlišuje od jiných systémů detekce objektů, které pro detekční úlohy používají více krokové metody.

Architektura systému YOLO se vyznačuje konvoluční neuronovou sítí, která využívá vlastnosti celého obrazu k předpovědi ohraničujících boxů a pravděpodobností tříd v jednom dopředném průchodu. Tato konstrukce umožňuje systému YOLO "vidět" celý obraz během trénování a odvozování, což podporuje globálnější chápání kontextu, které je pro přesnou detekci objektů klíčové. Ztrátová funkce modelu je pečlivě navržena tak, aby penalizovala chybu klasifikace, chybu lokalizace souřadnic ohraničujících boxů a skóre důvěryhodnosti boxů, čímž je zajištěna komplexní optimalizace výkonu detekce.

Od svého uvedení na trh prošel model YOLO několika vývojovými iteracemi, přičemž každá verze přinesla oproti předchozí výrazné zlepšení. Tato vylepšení zahrnují vylepšení architektury sítě pro lepší extrakci prvků, integraci predikcí ve více měřítkách pro efektivnější zpracování objektů různých velikostí a optimalizace ztrátové funkce pro zlepšení kompromisu mezi rychlostí a přesností. Následující verze YOLO také využily pokroky v hlubokém učení, jako je dávková normalizace, vynechávání spojení a používání kotevních boxů pro zlepšení přesnosti lokalizace.



Obrázek 4.2 Architektura YOLO, převzato z [35]

Navzdory svým silným stránkám není YOLO bez omezení. Zpočátku měl YOLO problémy s detekcí malých objektů a trpěl nižší přesností ve srovnání se složitějšími dvoustupňovými detektory. Neustálý pokrok v architektuře modelu a procesu trénování však tyto problémy podstatně zmírnil. Nejnovější iterace YOLO nabízejí konkurenceschopnou, ne-li vyšší přesnost než tradiční detekční systémy při zachování vysoké rychlosti zpracování.

Základem, na kterém je model YOLO postaven, je síť Darknet [65], zejména na její nejpokročilejší vývojové iteraci Darknet-53, která je vyvinutá s cílem upřednostnit rychlost a přesnost, slouží jako základ systému YOLO. Síť Darknet se vyznačuje řadou konstrukčních rozhodnutí zaměřených na maximalizaci výpočetní účinnosti. Na rozdíl od sítě VGG-16 [66], která je známá svou jednoduchostí a hloubkou, jež přispívají k její vysoké přesnosti v klasifikačních úlohách, je síť Darknet navržena se zaměřením na snížení výpočetní režie bez výrazného snížení výkonu. Této efektivity je dosaženo kombinací menšího počtu, ale strategičtěji umístěných konvolučních vrstev, přímých spojení, která obcházejí určité vrstvy, a snížením závislosti na plně propojených vrstvách, které jsou výpočetně náročné a pro úlohu detekce objektů nejsou tak klíčové.

Síť Darknet-53, jak je použita od verze YOLOv3, využívá 53 konvolučních vrstev a zahrnuje poznatky ze svého předchůdce Darknet-19 i inovace z oblasti zbytkových sítí. Tento hybridní přístup umožňuje síti Darknet-53 dosáhnout vysoké přesnosti s menším počtem operací ve srovnání s architekturami, jako je VGG-16. Konkrétně Darknet-53 využívá kombinaci konvolučních filtrů 3x3 a 1x1, které jsou prokládány spojeními, jež usnadňují tok gradientů sítí, čímž zvyšují efektivitu učení a umožňují vytvářet hlubší architektury bez problému mizejícího gradientu.

Architektura sítě Darknet je navíc navržena tak, aby byla ze své podstaty vhodnější pro úlohy detekce objektů. Zatímco VGG-16 slouží jako výkonný extraktor rysů pro různé úlohy vidění, struktura Darknet-53 je optimalizována pro rychlost a efektivitu, což je pro detekci v reálném čase klíčové. Tato optimalizace je patrná ve schopnosti YOLO pracovat výrazně rychleji než srovnatelné metody detekce při zachování konkurenceschopné přesnosti.

Jako jedním z dalších inovačních přístupů k detekci objektů, je považována integrace prostorového pyramidového sdružování (Spatial Pyramid Pooling, SPP) [67] do modelu YOLO, který představuje významný pokrok ve schopnosti modelu zpracovávat objekty různých velikostí a poměrů stran. Původně navržený SPP v kontextu hlubokého učení pro dosažení robustnosti vůči měřítku objektu, byl v architektuře YOLO upraven tak, aby zlepšil jeho detekční schopnosti, zejména z hlediska prostorové invariance.

Spatial Pyramid Pooling umožňuje neuronové síti zpracovávat vstupy různých velikostí a rozměrů a vytvářet výstupy s pevnou velikostí, které lze přivádět do následujících vrstev bez ztráty prostorové hierarchie nebo vztahů mezi prvky. Implementace zvyšuje odolnost vůči rozdílům ve velikosti objektů v rámci jednoho snímku nebo v různých snímcích. Díky agregaci prvků v různých měřítkách může YOLO efektivně detekovat malé i velké objekty s vyšší přesností.

Ačkoliv má model YOLO otevřený zdrojový kód a je neustále vyvíjený svou širokou komunitou, tak i přesto se potýká s problémy, které pramení ze samotného návrhu architektury sítě. Jednou z mnoha největších výzev spočívá pro model schopnost přesně detekovat objekty ve scénách, kde jsou hustě rozmístěny. Tento problém částečně vyplývá z přístupu založeného na mřížce, který model používá k předpovídání ohraničujících boxů. Každá buňka mřížky je zodpovědná za předpověď omezeného počtu ohraničujících boxů, což může vést k chybějícím detekcím v hustých oblastech, kde může do jedné buňky spadat více objektů. Překrývání ohraničujících boxů v hustých scénách navíc zvyšuje složitost správného

přiřazení pravděpodobnosti třídy, což může vést k nižší přesnosti. Kromě hustě rozmístěných objektů, je další výzvou detekce velmi malých objektů. Malé objekty představují pro model menší počet pixelů k analýze, který ztěžuje síti extrakci smysluplných rysů. Rysy odlišují jeden objekt od druhého. Problematický je zejména kontext složitého pozadí nebo pokud jsou objekty zájmu podobné necílovým prvkům ve scéně [39], [40], [41].

5 HODNOCENÍ VÝKONNOSTI DETEKČNÍCH MODELŮ

Jednou z mnoha důležitých vlastností správného modelu je schopnost generalizace na dosud neviděných datech. Výkonnost detekční modelů musí být dobře ohodnocena, aby bylo možné ověřit jejich schopnost správně identifikovat a lokalizovat objekty. Tato kapitola popisuje hlavní metriky a kritéria, které se používají pro hodnocení kvality a efektivity detekčních modelů.

5.1 IoU

Intersection over union (IoU), známý také jako Jaccardův index, je metrikou, která měří poměr plochy překrytí mezi předpovídaným a pravdivým ohraničením. Význam IoU spočívá v kvantitativním měření toho, jak dobře se predikovaný ohraničující rámeček shoduje s pravdivým ohraničujícím rámečkem, což lze definovat jako oblast objektu, která byla zaznamenána v rámci datové sady.

Obrázek 5.1 Vizualizace IoU, převzato z [42]

IoU je možné vypočítat jako (5.1):

$$\text{Intersection over union } = \frac{Area\ of\ Intersection}{Area\ of\ Union} \tag{5.1}$$

kde, *Area of Intersection* je společnou plochou sdílenou dvěma ohraničujícími boxy a *Area of Union* je celkovou plochou pokrytou dvěma ohraničujícími boxy.

Výpočet se provede tak, že se nejdříve spočítá *Area of Intersection,* což je společná oblast, kterou sdílí pravdivý ohraničující rámeček a předpovídaný ohraničující rámeček. Plochu překrývající oblasti se provede zjištěním souřadnic jejího levého horního a pravého dolního rohu. Následně se provede výpočet *Area of Union* což je celková oblast, kterou pokrývá základní pravdivý ohraničující rámeček a předpovídaný ohraničující rámeček. Oblast sjednocení se zjistí tak, že se sečtou plochy obou ohraničujících boxů a odečte se plocha průsečíků. Po zjištění dílčích hodnot, se tyto hodnoty vzájemně vydělí. Vyšší hodnota IoU znamená přesnější předpověď, zatímco nižší hodnota naznačuje špatné sladění předpovězených a základních ohraničujících boxů. Přijatelné hodnoty IoU jsou obvykle vyšší než 0,5,

zatímco dobré hodnoty IoU jsou vyšší než 0,7. Tyto prahové hodnoty se však mohou lišit v závislosti na aplikaci a úloze.

I přesto, že je IoU výkonnou metrikou, má své omezení. Jednou z mnoha omezení je citlivost na velikost ohraničujících rámečků. Malý posun ve velkém rámečku může mít na výsledek IoU minimální vliv, zatímco stejný posun v malém rámečku může výsledné skóre výrazně ovlivnit. Další významným omezením je problém určení správného prahu. Pro určení správnosti předpovědi se obvykle využívá binární práh (např. 0,5). V důsledku toho může být výsledek příliš zjednodušený a může přehlédnout jemné rozdíly v kvalitě [43].

5.2 mAP

Mean Average Precision (mAP), jinak známé jako střední průměrná přesnost, poskytuje ukazatel výkonnosti modelu.

Vypočet probíhá tak, že se zjistí průměrná přesnost pro každou třídu a poté průměr za více tříd. Vzorec mAP v podstatě říká, že pro danou třídu je třeba vypočítat její odpovídající průměrnou přesnost. Průměr za jednotlivé třídy je výslednou hodnotu mAP.

Výpočet střední průměrné přesnosti můžeme provést pomocí (5.2):

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k \tag{5.2}$$

kde, n je počet tříd a AP_k je průměrná přesnost třídy k.

Dokonalé skóre mAP 1,0 naznačuje, že model dosáhl bezchybné detekce ve všech třídách a prahových hodnotách *recall*. Naopak nižší hodnota skóre mAP znamená potenciální oblasti, kde je třeba zlepšit přesnost modelu.

Zásadní nevýhodou mAP je, že tato hodnotící metrika pracuje se všemi štítky se stejnou důležitostí. V reálných aplikací mohou být některé štítky kritičtější než jiné a tento faktor v mAP není zohledněn [44].

5.3 Precision a Recall

Přesnost (*precision*) je kritickou metrikou při hodnocení modelu. Slouží ke kvantifikaci přesnosti pozitivních předpovědí modelu. Konkrétně hodnotí, jak model dobře rozlišuje skutečné objekty od falešně pozitivních. Přesnost v podstatně poskytuje přehled o schopnosti modelu vytvářet pozitivní předpovědi, které jsou skutečně přesné. Vysoké skóre přesnosti naznačuje, že model se umí vyhnout falešně pozitivním předpovědím a poskytuje spolehlivé pozitivní předpovědi.

Výpočet přesnosti provedeme tak, že počet pravdivě pozitivních (*True Postive,* TP), což je počet hodnot, které byli správně detekovány vydělíme součtem počtu pravdivě pozitivních a falešně pozitivních (*False Positive*, FP), což je počet hodnot, které byli nesprávně detekovány.

$$Precision = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Positive\ (FP)} \tag{5.3}$$

Recall je míra pravdivé pozitivity, která měří schopnost modelu zachytit všechny relevantní objekty v obraze. Vysoké skóre naznačuje, že model účinně identifikuje většinu relevantních objektů v datech.

Výpočet provedeme tak, že počet pravdivě pozitivních vydělíme součtem počtu pravdivě pozitivních a počet falešně negativních (*False Negative, FN*), kdy tato hodnota nám udává počet nesprávně nezachycených objektů.

$$Recall = \frac{True\ Positive\ (TP)}{True\ Positive\ (TP) + False\ Negative\ (FN)} \tag{5.4}$$

Společná rovnováha mezi *precision a recall* je často žádoucí, ale v praxi může být těžké dosáhnout vysokých hodnot obou metrik současně. V závislosti na konkrétní aplikaci může být preferováno zvýšení jedné metriky na úkor druhé [45].

5.4 F1 skóre

F1 skóre představuje harmonický průměr *precision* a *recall*, což je metoda výpočtu průměru, která správně penalizuje extrémní hodnoty. Na rozdíl od aritmetického průměru klade harmonický průměr důraz na nejnižší čísla, čímž zajišťuje, že falešně pozitivní a falešně negativním výsledkům je věnována náležitá pozornost, a poskytuje tak ucelenější pohled na výkonost modelu.

Význam F1 skóre se projeví zejména při práci se soubory dat z nevyvážeností tříd. V takových případech může mít model vysokou přesnost, pokud jednoduše předpovídá většinovou třídu, ale ve výsledku by výsledek této předpovědi byl zavádějící, jelikož by nezachytil vzácné, ale klíčové případy.

Výpočet F1 skóre vychází z hodnot *precision* a *recall*, přičemž vzorec můžeme odvodit následovně (5.5):

$$F1\ skóre\ = \frac{2\ \times\ Precision\ \times\ Recall}{Precision\ +\ Recall} \tag{5.5}$$

Zdvojnásobením součinu *precision a recall* zajistíme, že součin *precision a recall* bude ve vzorci náležitě zastoupen. Zdvojnásobení *precision a recall* je pro harmonický průměr neodmyslitelně nutný, aby se zajistilo, že nebude převažovat ani *precision,* ale ani *recall*.

Může nastat situace, kdy F1 skóre bude nulové. To by znamenalo, že mezi předpovězenými pozitivními výsledky není žádný pravdivě pozitivní, což by způsobilo že čitatel vzorce F1 by byl nulový. V tomto důsledku to může poukazovat na úplné selhání funkčnosti modelu a lze předpokládat, že hodnoty jsou zcela chybné. V opačném případě je možné stanovit práh, kdy je možné model označit za dobrý. Takový práh je často nastavován na hodnou 0,7 nebo vyšší [46].

6 MODERNÍ PŘÍSTUP K ROZPOZNÁVÁNÍ SPECIFICKÝCH DAT

Technologičtí giganti jako Google, Facebook a další společnosti se intenzivně zabývají detekcí dětské nahoty s cílem eliminovat šíření tohoto nežádoucího materiálu. Hlavním problémem však zůstává, že nástroje vyvinuté těmito společnostmi nejsou veřejně přístupné. Často tyto společnosti spolupracují s Národním centrem pro pohřešované a zneužívané děti (NCMEC), a společně vyvíjejí nástroje jako Google Hash Matching API [59], které je založeno na percepčním hašování. Percepční hašování je technika používaná k vytváření digitálních otisků multimediálních souborů, jako jsou obrázky a videa, která odráží jejich percepční obsah namísto přímých binárních dat. Tato metoda se liší od tradičního kryptografického hašování tím, že se její výsledek minimálně mění i při změnách formátu nebo kvality, pokud základní percepční vlastnosti obsahu zůstávají konzistentní. Více se problematikou v souvislostech s percepčním hašováním zabývá literatura [58].

Kromě percepčního hašování některé společnosti implementují i metody založené na neuronových sítích, které slouží ke klasifikaci nebo detekci obsahu s nahými dětmi nebo jiného nevhodného materiálu. Tyto modely však nejsou veřejně dostupné, a to ani prostřednictvím API. Důvodem je obava, že by takový nástroj mohl být zneužit k vyhledávání a šíření nevhodného obsahu, což by mohlo vést k vážným etickým a právním problémům.

Přestože existují open-source nástroje jako nude.js [60] a NudeNet [61], které jsou určeny pro detekci nahoty, ve svých modelech neimplementují rozpoznávání nahých dětských postav. Vývoj nástrojů nude.js i NudeNet již byl ukončen.

II. PRAKTICKÁ ČÁST

7 POŽADAVKY NA SOFTWARE

Pro vývoj modelu jsou voleny takové postupy a nástroje, aby všechny výpočty a provoz softwaru byly výhradně jen na vyhrazených serverech. Pro účel provozování softwaru, datových úložišť a grafických karet, tyto výpočetní zdroje poskytl projekt e-INFRA CZ (ID:90254), podpořeného Ministerstvem školství, mládeže a tělovýchovy České republiky. V této kapitole budou uvedeny nástroje, které jsou využité pro práci s datovou sadou a využité modely, které byly použity k realizaci praktické části této práce.

7.1 Virtuální server

Mezi hlavní požadavky na praktickou část byl kladen důraz na zabezpečení dat a samotného serveru, kde běžel podpůrný software. Virtuální server je založený na principu rozdělení výpočetního výkonu na fyzickém serveru na menší celky, které jsou provozovány jako virtuální servery. Na každém virtuálním serveru je provozován takový operační systém, který je pro potřeby dané práce nutný. Pro potřeby této diplomové práce byl nainstalován operační systém Ubuntu verze 22, jehož označení je *Jammy Jellyfish*. Zvolení instalace operačního systému Ubuntu, bylo z důvodu, že se jedná o otevřený software, tedy není požadována licence, a také že autor této práce má s tímto operačním systémem největší zkušenosti, a případné problémy mohly být v krátkém čase vyřešeny a odstraněny.

Samotné zabezpečení serveru bylo zajištěno tak, že přístup k příkazovému řádku bylo zajištěno formou SSH spojení, kdy pro připojení bylo nutné znát přístupové údaje a fyzicky vlastnit přístupový klíč, což je soubor umožňují další vrstvu zabezpečení. Pokusy o útok na přihlášení k SSH spojení bylo řešeno formou blokování IP adres, případně samotné přihlášení bylo uděláno formou povolení určité geografické zóny. Vytvoření seznamu povolených IP adres, které mohou přistupovat k serveru nebylo možné vzhledem k tomu, že na místě, kde docházelo k přihlašování k serveru, byla k dispozici pouze dynamická IP adresa, tudíž v časovém intervalu docházelo k obměně IP adresy.

Zálohování samotného serveru bylo řešeno formou lokální zálohy dat, kdy k virtuálnímu serveru byl připojený virtuální externí datový disk, kde byla uložena všechna data ze systému a také lokální zálohy, které se prováděly vždy každou hodinu. Aby se předešlo k tomu, že dojde ke kolapsu serveru a tím i zničení lokálních záloh, sdružení MetaCentrum vytváří zálohu kompletního virtuálního serveru a všech připojených virtuálních externích disků formou snapshotů. Tento typ zálohy slouží v případě ztráty lokálních záloh nebo k celému

zhroucení operačního systému. Ačkoliv samotný virtuální sever byl umístěný na území České republiky, zálohování formou snapshotů bylo geograficky oddělené. Kdyby došlo ke škodě na fyzickém zařízení, zálohy by bylo možné obratem spustit na jiném funkčním serveru a pokračovat dále v práci.

7.2 Docker

Docker [47] představuje klíčový nástroj pro nasazení a správu aplikací v izolovaných kontejnerech, což zajišťuje vysokou úroveň bezpečnosti. Výběr tohoto nástroje byl především potřebou efektivní správy aplikací a snadného škálování služeb na virtuálním serveru bez ovlivnění ostatních běžících procesů. Dalším aspektem výběru byla i vysoká bezpečnost, který Docker poskytuje. Izoluje software od samotného hostitelského operačního systému a také izoluje běžící aplikaci v kontejneru od ostatních aplikací. V případě nějaké bezpečnostní chyby nebo chyby samotné běžící aplikace je ovlivněn pouze daný kontejner a nehrozí bezprostřední riziko napadení hostitelského systému ani ostatních kontejnerů.

V rámci samotného zabezpečení bylo potřeba, aby jednotlivé kontejnery běžely s omezenými právy, což minimalizuje riziko zneužití v případě bezpečnostních chyb v aplikacích. Dále byly implementovány síťové politiky, které izolují kontejnery od zbytku systému a od sebe navzájem, čímž se zabránilo neautorizovanému síťovému provozu.

7.3 Label Studio

Label Studio [48] je nástroj, který slouží pro anotování datové sady. Jeho největší předností je jeho otevřený zdrojový kód a velká podpora komunity. Label Studio je vyvíjený v jazyce Python a je připravený pro provoz v Dockeru. Díky tomu je možné tento software spustit bez výraznějších obtíží, jelikož v sobě obsahuje přednastavený Docker obraz. Kromě samotného anotování datové sady, umožňuje i automatické anotování datové sady. Automatické anotování funguje tak, že je vytvořeno aplikační rozhraní, pomocí něhož Label Studio komunikuje. Dále Label Studio přináší velké ulehčení v přípravě dat pro vstupní trénovací model. Jelikož každý model požaduje jiný formát datové sady, je v rámci exportu toto zohledněno. Při exportu dat je možné vybrat například formát pro YOLO model. Ačkoliv tento software k tomu není primárně určen, jeho největší nevýhodou je, že neumí pracovat s rozdělením datové sady na validační, testovací a trénovací množinu. Toto rozdělení se musí udělat manuálně formou nástrojů, které poskytuje YOLO model.

7.4 Label Studio Backend ML

Jedná se o nezávislou část nástroje Label Studio, který rozšiřuje jeho hlavní funkčnost o možnost automatického anotování dat. Tato část nástroje je pouhou šablonou zdrojového kódu, do kterého se musí doprogramovat hlavní funkčnost. Poskytuje tedy komunikační rozhraní mezi Label Studiem a inferencí modelu. V rámci této diplomové práce byla tato část využita pro rychlejší anotování datové sady, kdy propojení této části a samotného Label Studia bude rozebíráno v následujících kapitolách.

7.5 YOLO

YOLO [32] je jedním z modelů pro detekci objektů v reálném čase, který je zvláště ceněn pro svou rychlost a přesnost. Model YOLO analyzuje obraz jako celek a předpovídá bounding boxy a pravděpodobnosti tříd objektů ve velmi krátkém čase. Tato schopnost umožňuje modelu provádět detekci objektů s vysokou přesností. V praktické části této práce bude využíván YOLO model ve verzi 8, která je v době psaní této práce nejnovější stabilní verzí.

7.6 Detectron2

Detectron2 [54] je neuronová síť pro detekci objektů, která vychází z architektury Faster R-CNN, na rozdíl od YOLO, který analyzuje celý obraz najednou.

Hlavní výhodou Detectron2 je schopnost provádět přesné lokalizace a klasifikace objektů v obraze s využitím techniky Faster R-CNN, kde nejprve identifikuje návrhy regionů a následně extrahuje vlastnosti pro detekci a klasifikaci.

7.7 Gradio

Gradio [55] je moderní nástroj určený k demonstraci strojových učících modelů prostřednictvím webového rozhraní, které umožňuje snadný přístup a použití modelů širokou veřejností, bez ohledu na jejich technickou znalost. Gradio umožňuje uživatelům snadno nahrávat vstupní data do modelu, ať už se jedná o obrázky, texty, nebo zvukové soubory, a okamžitě zobrazovat výstupy.

8 INSTALACE SOFTWARU

Vzhledem k tomu, že celá infrastruktura je hostována na virtuálním serveru, je nezbytné provést konfiguraci a instalaci potřebného softwaru na daném serveru před zahájením anotace datové sady a trénování modelu.

Před samotnou instalací je potřeba zajistit, aby operační systém a balíčky, které jsou již nainstalovány byly aktuální. To uděláme tak, že pomocí SSH protokolu se připojíme vzdáleně k příkazové řádce serveru, a jako správce (*superuser*) spustíme aktualizaci balíčků a jejich závislostí:

```
$ sudo apt update && sudo apt upgrade
```

Kód 8.1 Aktualizace balíčků a jejich závislostí

Následně je třeba nainstalovat balíček, který umožňuje komunikovat se zabezpečeným protokolem HTTPS:

```
$ sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

Kód 8.2 Instalace balíčků, které umožní používat zabezpečený protokol HTTPS

8.1 Instalace Dockeru

Jak bylo v předchozích kapitolách zmíněno, jedním z požadavků je bezpečnost celého virtuálního serveru. To je docíleno instalací a provozováním nástroje Docker, který spouští software ve vlastním virtuálním kontejneru. Pomocí toho je docílena vzájemná izolace kontejnerů.

Před instalací samotného nástroje je potřeba přidat GPG klíč, který zajišťuje kontrolu toho, že balíčky pocházející z úložiště, kde je umístěný instalační balíček nástroje Docker, je ověřený a není pozměněný.

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Kód 8.3 Přidání GPG klíče do seznamu důvěryhodných klíčů

Po přidání klíče, již můžeme nainstalovat balíčky, které umožní virtualizovat software pomocí nástroje Docker. Samotnou instalaci provedeme pomocí balíčkovacího nástroje *aptitude* (Kód 8.4):

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Kód 8.4 Instalace nástroje Docker

Pro správné ověření funkčnosti Dockeru, je možné spustit testovací příkaz (Kód 8.5):

```
$ sudo docker run hello-world
```

Kód 8.5 Spuštění testovacího obrazu *hello-world*

Po spuštění testovacího příkazu (Kód 8.5) dojde k instalaci obrazu *hello-world* a k následnému spuštění. Pokud instalace nástroje Docker je provedena v pořádku, na obrazovku konzole se vypíše text, že vše funguje správně.

```
ubuntu@master-thesis:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Obrázek 8.1 Výstup po spuštění testovacího příkazu

8.2 Instalace Nginx

Jelikož software provozovaný v prostředí Dockeru funguje jako webová aplikace, je potřeba nainstalovat Nginx, který funguje jako webový server. V této práci však Nginx nevyužíváme primárně jako webový server, ale jako reverzní proxy server. Proxy server přijímá HTTP požadavky od klientů a přeposílá je do jednoho nebo více Docker kontejnerů. Nginx tak může skrýt identitu vnitřních systémů a poskytnout další úroveň zabezpečení. Samotnou instalaci provedeme pomocí příkazu (Kód 8.6):

```
$ sudo apt install nginx
```

Kód 8.6 Instalace Nginx

Po úspěšné instalaci je automaticky tento nástroj zaveden do systémových služeb. To znamená, že je spuštěn a nastaven tak aby byl spuštěn i po případném restartu operačního systému. Tento krok je vhodný pro případ, když je virtuální server restartován, bude Nginx automaticky spuštěn.

Pokud byl webový server nainstalován správně, tak po přístupu na IP adresu serveru, je zobrazena úvodní webová stránka (Obrázek 8.2).

Obrázek 8.2 Úvodní webová stránka

8.3 Instalace Label Studio

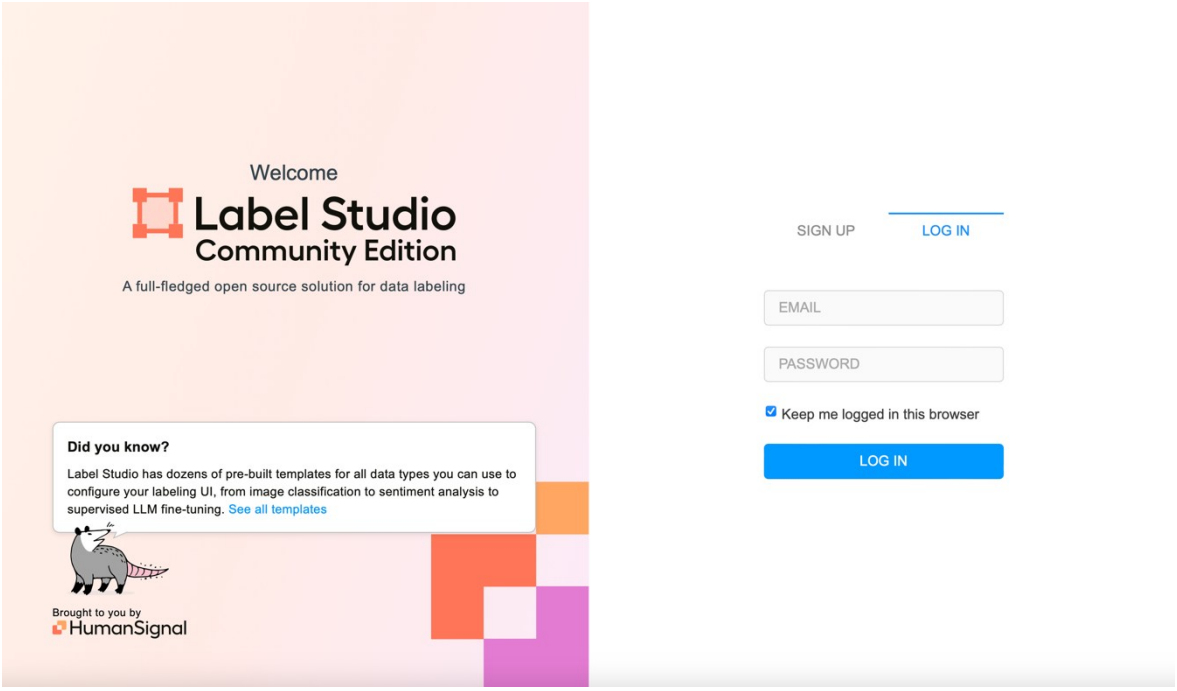
Spuštění samotného nástroje se provádí již ve virtualizačním prostředí Dockeru, pomocí příkazu (Kód 8.7):

```
$ docker run -it -p 8080:8080 -v `pwd`/mydata:/label-studio/data heartexlabs/label-studio:latest
```

Kód 8.7 Spuštění Label Studio ve virtuálním prostředí Docker

Příkaz nejprve zkontroluje, zda obraz *heartexlabs/label-studio:latest* existuje v lokální kopii na serveru. Pokud lokální kopie obrazu neexistuje, stáhne ho. Parametr *-p 8080:8080* v příkazu (Kód 8.7) nastavuje mapování portů, což umožňuje přístup k aplikaci běžící v kontejneru z webového prohlížeče na hostitelském počítači prostřednictvím portu 8080. Parametr *-v `pwd`/mydata:/label-studio/data* vytváří mapování mezi lokálním adresářem */mydata* na serveru a adresářem */label-studio/data* v kontejneru, což umožňuje přístup k datům v lokálním adresáři přímo z aplikace běžící v Dockeru.

Pokud instalace proběhla správně, tak po zadání IP adresy serveru a portu, dojde k načtení aplikace.



Obrázek 8.3 Úvodní stránka Label Studio

9 PŘÍPRAVA DATOVÉ SADY

Příprava datové sady je klíčovým krokem při trénování neuronových modelů. Pro shromáždění datové sady byly vybrány dvě webové stránky, které jsou považovány za problematické: rajce.net, největší česká sociální síť specializující se na ukládání fotografií, a imgsrc.ru, zahraniční platforma fungující na podobném principu.

Webová stránka rajce.net [50] čelí dlouhodobému problému, kdy převážně rodiče ukládají fotografie svých oblečených, polonahých a nahých dětí do veřejně přístupných alb, což umožňuje snadné získání datové sady odpovídající požadované kvalitě. Podobně imgsrc.ru [51] nabízí obsah, který může být využit pro trénování modelů v této práci. Tyto fotografie obecně odpovídají druhé úrovni škály COPINE [49], která představuje obrázky nahých či polonahých dětí, často pocházející z rodinných alb.

Pro kontext je první úroveň škály COPINE definována jako fotografie dětí v oblečení bez dalšího podtextu, jako například snímky dětí v reklamách na oblečení. Druhá úroveň zahrnuje již zmíněné fotografie nahých či polonahých dětí.

Pro ostatní třídy byly obrazové data získány z veřejných vyhledávačů jako je Bing, Google a Yandex.

9.1 Definování tříd

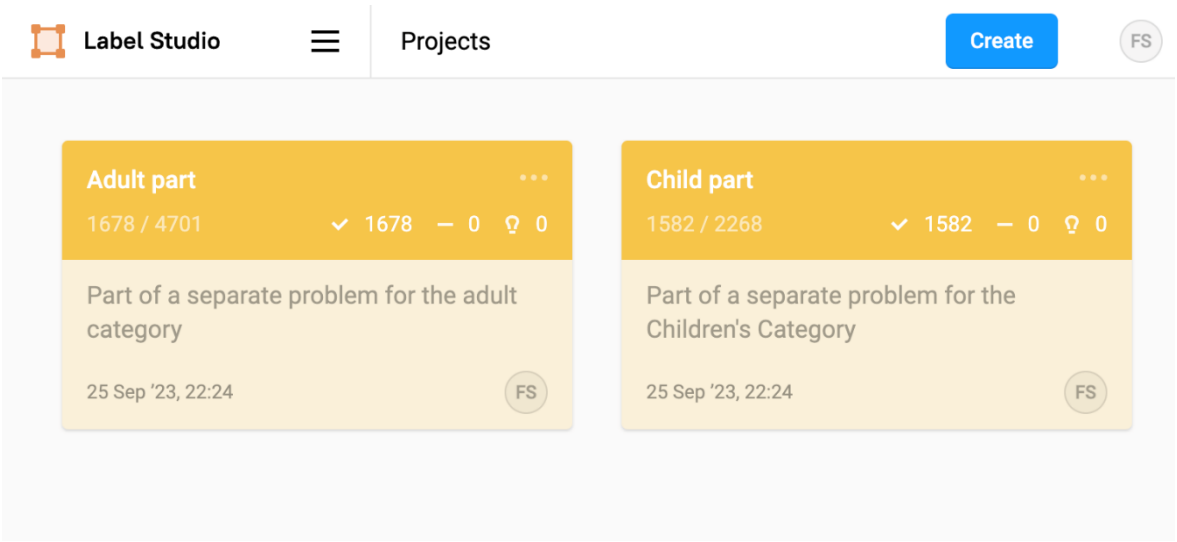
Jelikož v této práci je vyvíjen detektor, což je typ modelu schopného generovat ohraničení objektů, je nezbytné definovat třídy objektů, které budou detekovány. Pro účely této diplomové práce byly zvoleny následující třídy:

- Mužské genitálie – dospělá osoba
- Ženské genitálie – dospělá osoba
- Zadek – dospělá osoba
- Obličej – dospělá osoba
- Spodní prádlo – dospělá osoba
- Podprsenka – dospělá osoba
- Prsa ženy – dospělá osoba
- Mužské genitálie – dítě
- Ženské genitálie – dítě
- Zadek – dítě
- Obličej – dítě
- Spodní prádlo – dítě
- Podprsenka – dítě
- Prsa ženy – dítě

V rámci této práce byly definovány samostatné kategorie pro děti a dospělé osoby. Tento přístup umožňuje neuronové síti lépe rozlišit a naučit se specifické rysy, které charakterizují každou z těchto skupin. Rozdělení na tyto dvě kategorie je nezbytné, jelikož pokud by třídy nebyly takto rozdělené, síť by mohla mít problémy s detekcí rozdílných rysů, což je pro praktické využití této diplomové páce zásadní.

9.2 Anotování datové sady

V této praktické části byla datová sada rozdělena na dětskou a dospělou část pro účely anotace. Kromě hlavního nástroje Label Studio byl také využit Label Studio ML Backend, což je nástroj podporující automatickou anotaci dat. Automatická anotace dat je založena na vytrénování modelu na menším vzorku dat. Takový model může dosahovat nižší přesnosti, ale cílem tohoto postupu je zefektivnit proces anotace a ušetřit tak čas při zpracování obrazových dat.



Obrázek 9.1 Label Studio s připravenými projekty

9.3 Automatické anotování datové sady

Pro zavedení podpory automatického anotování datové sady pomocí nástroje Label Studio ML Backend, je nutné z GitHub repositáře [52] stáhnout základní zdrojový kód vytvořený v jazyce Python, který vytváří webový server a komunikaci mezi Label Studiem a samotným ML Backend nástrojem.

Po stažení repositáře je potřeba nainstalovat závislosti, což jsou nástroje třetích stran, které jsou potřeba pro samotné fungování nástroje. Instalaci závislostí se provede pomocí balíčkovacího nástroje *pip*, který je součástí jazyka Python (Kód 9.1):

```
$ pip install -e .
```

Kód 9.1 Instalace závislostí třetích stran

Po úspěšně instalaci, je potřeba vygenerovat základní kód, který poskytuje podporu webového serveru, jelikož Label Studio ML Backend funguje na základě HTTP komunikace. Vygenerování nového projektu se provede pomocí příkazu (Kód 9.2):

```
$ label-studio-ml create my_ml_backend
```

Kód 9.2 Vytvoření nového projektu

Při vytvoření nového projektu se vytvoří složka podle názvu projektu, v tomto případě složka *my_ml_backend*. Uvnitř této složky jsou soubory umožňující spouštět tento projekt pomocí nástroje Docker, což umožňuje tomuto kontejneru vyhradit potřebnou paměť samotného virtuálního serveru. Nejdůležitějším souborem je *model.py* do kterého je nutné implementovat funkcionalitu inference s modelem.

V souboru *model.py* je potřeba nejdříve upravit konstruktor třídy tak, aby se načetl model.

```
    def __init__(self, **kwargs):
        # Call base class constructor
        super(YOLOv8Model, self).__init__(**kwargs)

        # Initialize self variables
        self.from_name, self.to_name, self.value, self.classes = get_single_tag_keys(
            self.parsed_label_config, 'RectangleLabels', 'Image')
        self.labels = [
            'ass',
            'face',
            'female genitalia',
            'male genitalia',
            'underwear',
            'woman breast'
        ]

        # Load model
        from ultralytics import YOLO
        self.model = YOLO("model/child_v4.pt")
```

Kód 9.3 Načítání modelu pomocí konstruktoru

V momentě, kdy je načtený model je potřeba provést nad ním inferenci. Samotnou inferenci provádí metoda *predict*, kterou je potřeba upravit (Kód 9.4):

```
    def predict(self, tasks, **kwargs):
        task = tasks[0]

        image = Image.open(
            BytesIO(requests.get(LS_URL + task['data'][self.value]).content)
        )

        original_width, original_height = image.size
        predictions = []
        score = 0
        results = self.model.predict(image)
        boxes = 0

        # Gettings results
        for result in results:
            for i, box in enumerate(result.boxes):
                im_height, im_width = box.orig_shape
                x_min, y_min, x_max, y_max = box.xyxy.squeeze().tolist()

                x = x_min / im_width * 100
                y = y_min / im_height * 100
                width = (x_max - x_min) / im_width * 100
                height = (y_max - y_min) / im_height * 100

                cls_ids = int(box.cls.item())

                predictions.append({
                    "from_name": self.from_name,
                    "to_name": self.to_name,
                    "id": str(i),
                    "type": "rectanglelabels",
                    "score": box.conf.item(),
                    "original_width": original_width,
                    "original_height": original_height,
                    "image_rotation": 0,
                    "value": {
                        "x": x,
                        "y": y,
                        "width": width,
                        "height": height,
                        "rotation": 0,
                        "rectanglelabels": [
                            self.labels[cls_ids]
                        ]
                    }
                })

                # Calculating score
                score += box.conf.item()
                boxes += 1

        # Dict with final dicts with predictions
        final_prediction = [{
            "result": predictions,
            "score": score / boxes,
            "model_version": "yolo_v4"
        }]

        return final_prediction
```

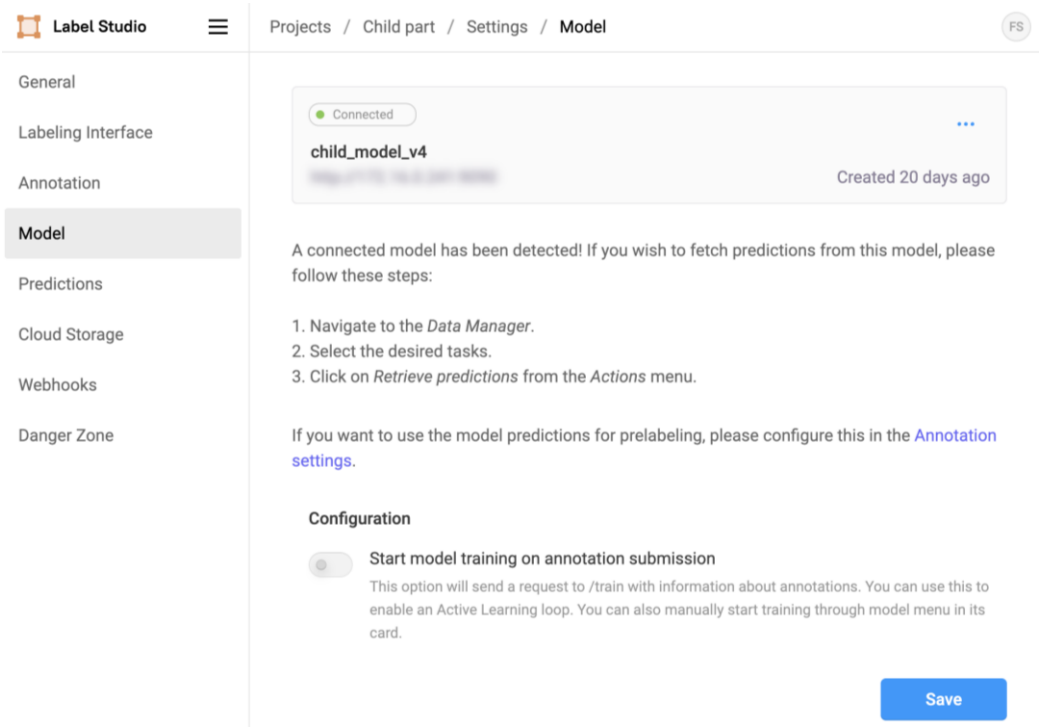
Kód 9.4 Provádění inference nad modelem

Po provedených úpravách nyní disponujeme funkčním nástrojem, který automaticky anotuje nová data v Label Studiu. Automatická anotace je založena na modelu YOLO, který byl natrénován na menším vzorku dat, jak bylo popsáno výše. Pro spuštění kontejneru a přidání nového modelu do Label Studia nám postačí použít příslušný spouštěcí příkaz (Kód 9.5):

```
$ docker compose up --detach
```

Kód 9.5 Spuštění kontejneru na pozadí

Propojení těchto nástrojů se provede tak, že v nastavení konkrétního projektu se zadá IP adresa kontejneru, kde běží auto anotační nástroj. Poté se v aplikaci otestuje spojení a funkčnost. Pokud je vše v pořádku, je uživatel informován zelenou ikonou a textem *Connected*.



Obrázek 9.2 Nastavení modelu pro automatickou anotaci

10 TRÉNOVÁNÍ MODELU

Trénování modelu probíhalo na Národní Gridové Infrastruktuře MetaCentrum, jejíž provozovatel je sdružení CESNET, z.s.p.o. Servery, na kterých trénování probíhalo jsou vybaveny grafickými čipy NVIDIA. Konkrétní typ grafického čipu nelze specifikovat, jelikož celý systém infrastruktury je založený na přidělovaní prostředků, které jsou po vyžádání volné. Nejčastěji však bylo trénování prováděno na modelech A40 a GeForce RTX 2080 Ti, kterých je v celé infrastruktuře zastoupených nejvíce. Trénování modelu lze provést i pomocí centrální procesorové jednotky (CPU), vzhledem k principu fungování by trénování bylo neefektivní.

10.1 Trénování modelu YOLO

Před zahájením trénování neuronového modelu je nezbytné nejprve exportovat anotovanou datovou sadu tak, aby odpovídala formátovým požadavkům neuronové sítě YOLO. Nástroj Label Studio umožňuje tento export provést, čímž zajišťuje, že data jsou připravena v požadovaném vstupním formátu pro trénování.

Před zahájením trénování je důležité spojit projekty do jednoho, aby byl trénován jeden společný model místo dvou odlišných. V Label Studiu se toho dosáhne vytvořením nového projektu, do kterého se naimportují již dříve exportované, rozdělené projekty. Label Studio tyto projekty spojí, což umožní projekt znovu exportovat jako jednotný celek. Tento přístup zajišťuje, že během trénovacího procesu budou extrahovány rysy z dospělých i dětských těl.

Exportovaný projekt obsahuje několik souborů. Nejdůležitějším souborem je *data/train.yaml*, bez kterého se samotný trénovací proces nespustí. Tento soubor obsahuje klíčové informace pro trénování, včetně cest k obrázkovým datům a seznamu tříd, které jsou pro trénování použity. Kromě konfiguračního souboru je datová množina rozdělena na dvě podsložky. Ve složce *images* jsou umístněné samotné obrázky. Ve složce *labels* jsou umístěné soubory, které obsahují identifikátor třídy, který musí být shodný s identifikátorem z konfiguračního souboru *data/train.yaml* a pozici ohraničujícího rámečku (Obrázek 10.1). Je nutné, aby název obrázku, umístněného ve složce *images* byl shodný s textovým souborem ze složky *labels*. Jedině tak je obrázek považovaný za anotovaný.

```
2 0.4139383036206752 0.12640801001251567 0.2350055968914354 0.20525657071339187
3 0.4302099259561736 0.7340425531914895 0.07954050432199684 0.06633291614518176
6 0.40851511874585783 0.4455569461827271 0.339843728995008 0.13516896120150165
```

Obrázek 10.1 Zápis třídy a pozic ohraničení pro jeden obrázek

Pro trénování byl využit předtrénovaný model YOLOv8l, který poskytuje optimální kompromis mezi rychlostí inferenčního procesu na procesoru a přesností. Model má velikost vstupního obrázku 640 pixelů a dosahuje na předtrénované datové sadě mAP (mean Average Precision) hodnoty 52,9 v rozmezí IoU 0,50 až 0,95. Na centrální procesorové jednotce (CPU) dosahuje model rychlosti 375,2 ms, zatímco na GPU Nvidia A100 s využitím TensorRT je rychlost inferenčního procesu 2,39 ms. Model má 43,7 milionu parametrů [53].

Před zahájením samotného procesu trénování bylo v rámci předzpracování obrazových dat použito několik metod. Velikost všech obrázků byla upravena na 640 pixelů, provedena normalizace a aplikována augmentace. Během augmentace bylo náhodně 1 % trénovacích obrázků rozmazáno a další 1 % obrázků bylo převedeno do stupně šedi. Tyto techniky augmentace přispívají k větší robustnosti modelu tím, že simulují různé možné podmínky, ve kterých může být model v praxi použit.

Před samotným trénováním byla množina rozdělena do trénovací a validační množiny v poměru 90 % dat pro trénování a 10 % dat pro validování. Toto rozdělení umožňuje efektivně validovat výkonnost modelu během a po trénování. Jelikož rozdělení nepodporuje Label Studio ani samotný model YOLO, bylo nutné napsat zdrojový kód, který bude datovou sadu rozdělovat. Spuštění programu pro rozdělení datové sady se provádí pomocí příkazu (Kód 10.1). Rozdělení nezohledňuje vyvážení tříd, ale pouze v určitém poměru rozdělí datovou sadu. Jediná prováděná kontrola při rozdělení je, aby obrázek měl anotaci. Pokud není anotovaný, tak je z výsledné datové sady odstraněn. Toto odstranění vede k optimalizaci velikosti datové sady.

```
$ python master-thesis/dataset-autosplit.py --path dataset-yolo --annotated_only
```

Kód 10.1 Rozdělení datové sady na trénovací a validační množinu

Na takto spojené datové sadě provedeme trénování modelu, o 300 epochách a vstupní velikosti obrázku 640 pixelů pomocí příkazu (Kód 10.2).

```
$ yolo detect train data=data/train.yaml model=yolov8l.pt epochs=300 imgsz=640
```

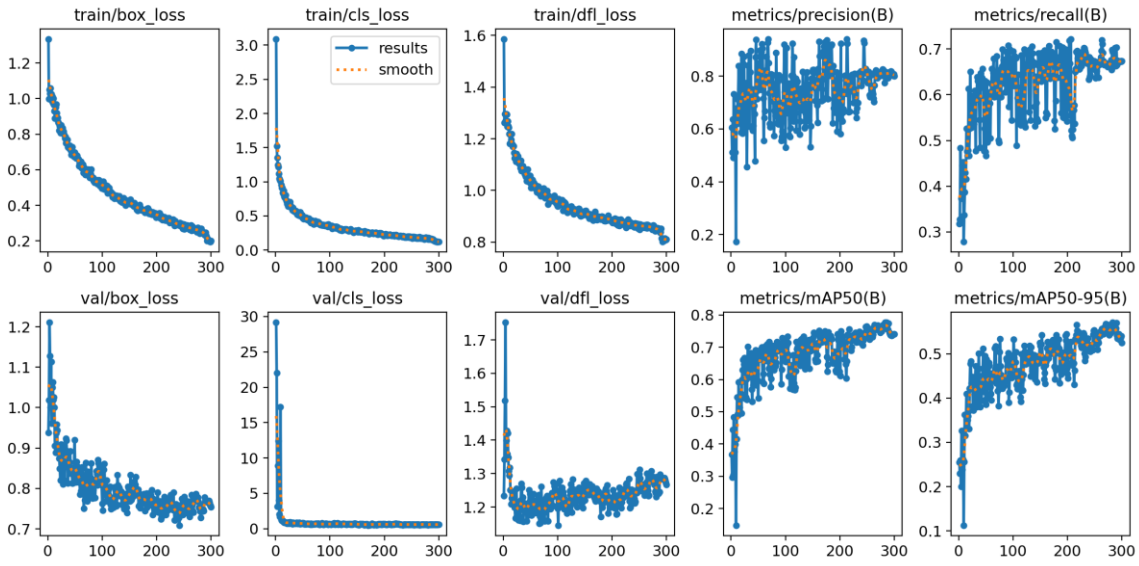
Kód 10.2 Spuštění trénování YOLO modelu

Protože konfigurační soubor také nezmiňuje žádnou specifickou optimalizační funkci, byla automaticky zvolena optimalizační funkce Adam. Tato metoda byla automaticky konfigurována s parametry: *learning rate* 0,000556 a *momentum* 0,9, což jsou standardní hodnoty pro dosažení dobré konvergence při trénování hlubokých neuronových sítí.

Po dokončení trénovacího procesu, který trval 300 epoch, byla celková doba tréninku 2 hodiny a 20 minut. Velikost výsledného modelu činila 87,7 MB. Další statistické údaje jsou uvedeny v následující tabulce (Tabulka 10.1) a obrázku (Obrázek 10.2). Dle procesu trénování a vlastností modelu lze usoudit že model se natrénoval dobře a vykazuje dobré vlastnosti.

Tabulka 10.1 Statistické vlastnosti natrénovaného modelu YOLO

mAP	0,7765
mAP50-90	0,5707
precision	0,8165
recall	0,6949
parameters	43 640 634



Obrázek 10.2 Proces učení YOLO modelu

10.2 Trénování modelu Detectron2

Pro trénování modelu Detectron2 je nutné nejprve exportovat datovou sadu ve formátu COCO (*Common Objects in Context*). Tento formát uchovává všechny anotace v jednom JSON souboru, zatímco obrázky jsou organizovány ve složce *images*. Před zahájením trénování je důležité připravit prostředí pro trénování modelu. Prvním krokem je instalace modelu Detectron2, což lze provést pomocí následujícího příkazu (Kód 10.3).

```
$ python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'
```

Kód 10.3 Instalace modelu Detectron2

Po úspěšné instalaci je důležité ověřit, zda byla instalace provedena správně. To lze provést vytvořením testovacího skriptu (Kód 10.4).

```
1. import torch, detectron2
2. !nvcc --version
3. TORCH_VERSION = ".".join(torch.__version__.split(".")[:2])
4. CUDA_VERSION = torch.__version__.split("+")[-1]
5. print("torch: ", TORCH_VERSION, "; cuda: ", CUDA_VERSION)
6. print("detectron2:", detectron2.__version__)
```

Kód 10.4 Kód pro testování úspěšné instalace Detectron2

Pokud v prostředí je k dispozici grafická karta podporující CUDA (*Compute Unified Device Architecture*) a instalace byla provedena správně, zobrazí se informace o verzi CUDA a instalované verzi Detectron2 (Obrázek 10.3). Tyto informace se vypíší automaticky, což potvrzuje, že systém je správně nakonfigurován pro využití s Detectron2.

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
torch:  2.2 ; cuda:  cu121
detectron2: 0.6
```

Obrázek 10.3 Výstup NVIDIA CUDA a verze Detectron2

Pro zahájení procesu trénování je nyní potřeba nejprve zaregistrovat datovou sadu. Po její registraci můžete spustit samotný proces trénování.

```
 1. import detectron2
 2. from detectron2.utils.logger import setup_logger
 3. setup_logger()
 4.
 5. import os
 6.
 7. # import some common detectron2 utilities
 8. from detectron2 import model_zoo
 9. from detectron2.engine import DefaultTrainer
10. from detectron2.config import get_cfg
11. from detectron2.data.datasets import register_coco_instances
12. from detectron2.config import get_cfg
13.
14. register_coco_instances("project2", {}, "./result.json", ".")
15.
16. cfg = get_cfg()
17. cfg.merge_from_file(model_zoo.get_config_file("COCO-
Detection/faster_rcnn_R_101_FPN_3x.yaml"))
18. cfg.DATASETS.TRAIN = ("project2",)
19. cfg.DATASETS.TEST = ()
20. cfg.DATALOADER.NUM_WORKERS = 2
21. cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-
Detection/faster_rcnn_R_101_FPN_3x.yaml")
22. cfg.SOLVER.IMS_PER_BATCH = 16
23. cfg.SOLVER.BASE_LR = 0.000025
24. cfg.SOLVER.MAX_ITER = 5000
25. cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
26. cfg.MODEL.ROI_HEADS.NUM_CLASSES = 14
27.
28. os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
29. trainer = DefaultTrainer(cfg)
30. trainer.resume_or_load(resume=False)
31. trainer.train()
```

Kód 10.5 Trénování Detectron2 modelu

Z konfiguračního souboru je patrné, že se využívá předtrénovaný model Faster R-CNN, integrující architekturu Feature Pyramid Network (FPN). Předtrénovaný model byl původně natrénován na 270 000 iteracích, avšak v konfiguračním souboru je nastaven limit pouze na 5 000 iterací. Trénovací sada je specifická pro tento model, načítaná z lokálního souboru *result.json*, který odpovídá formátu COCO. Tento formát obsahuje všechny potřebné anotace, definice tříd a cesty k obrázkovým souborům. V rámci trénování je v jedné dávce zpracováno 16 obrázku a model je konfigurován pro rozpoznávání celkem 14 tříd.

Po dokončení trénování jsou automaticky uloženy data z trénovacího procesu do výstupní složky *output*, kde je uložen také soubor *model_final.pth*, což je natrénovaný model na vlastní datovou sadu. Dále je potřeba uložit i konfiguraci, která byla použita u trénování. Tento uložený konfigurační soubor může poskytovat do budoucna informace o parametrech,

které při trénování byly použity a umožňuje srovnávat vstupní konfigurace s výstupními metriky. Uložení konfiguračního souboru se provede pomocí kódu (Kód 10.6):

```
1. f = open("config.yml", "w")
2. f.write(cfg.dump())
3. f.close()
```

Kód 10.6 Uložení konfigurace do souboru

Po zahájení trénovacího procesu byl model natrénován na 5 000 iterací, což trvalo 1 hodinu a 3 minuty. Doba trvání jedné iterace trvala přibližně 0,755 sekundy. Velikost výsledného modelu dosáhla 482 MB.

Následně byla provedena validace modelu, včetně vypsání statistických údajů. Tato validace byla realizována na validační datové sadě, přičemž spuštění validace proběhlo pomocí kódu uvedeného v kódu 10.7.

```
1. from detectron2.evaluation import COCOEvaluator, inference_on_dataset
2. from detectron2.data import build_detection_test_loader
3. evaluator = COCOEvaluator("project2_val", ("bbox",), False, output_dir="./output/")
4. val_loader = build_detection_test_loader(cfg, "project2_val")
5. print(inference_on_dataset(trainer.model, val_loader, evaluator))
```

Kód 10.7 Spuštění validace natrénovaného modelu

Výstup zahrnuje několik statistických metrik, které poskytují přehled o klíčových schopnostech natrénovaného modelu. K těmto metrikám patří mAP a průměrná přesnost (AP) a její variace pro různé úrovně prahu IoU (Intersection over Union). Tyto metriky jsou podrobně prezentovány v tabulce 10.2.

Tabulka 10.2 Statistické vlastnosti natrénovaného modelu Detectron2

mAP	0,5302
AP50	0,6609
AP75	0,3434
APs	0,1750
APm	0,3243
APl	0,3807

11 APLIKACE PRO TESTOVÁNÍ MODELŮ

Důležitým procesem v průběhu vývoje modelu je jeho testování, které poskytuje zpětnou vazbu pro jeho další iterativní zlepšování. Toto vyhodnocování umožňuje identifikovat slabiny v modelu, ověřit jeho schopnost generalizace na nová data. Jedním z postupů je využití různých skriptů, které se spouštění z příkazové řádky a umožňují získat základní pohled na výstupy, které natrénovaný model poskytuje. Dlouhodobě je takový typ testování neefektivní, vyžaduje určité znalosti obsluhy takových skriptů. Řešením může být vyvinout aplikaci, která bude fungovat ve webovém rozhraní. Takové řešení poskytuje výhodu, že uživatel není nucen nic instalovat a je možné aplikaci použít ihned. Velkou nevýhodu tohoto řešení je, pokud webová aplikace nemá otevřený zdrojový kód, tak existuje možnost, že aplikace může ukládat vstupní data bez vědomí uživatele. Jedním z velmi používaných nástrojů pro vývoj webových aplikací je Gradio, které umožňuje vytvořit rozhraní bez nutnosti umět programovat webové aplikace. Gradio obsahuje řadu naprogramovaných komponent, které se jen pomocí jazyka Python skládají dohromady a podle jeho použití se vygeneruje grafické rozhraní.

Tato kapitola se zabývá vývojem webové aplikace, kde jediným vstupem je obrázek, nad kterým provede detekci na všech dostupných nejnovějších modelech, které jsou k dispozici.

Gradio, které je využité pro vývoj webové aplikaci vyžaduje minimální verzi Python 3.8 nebo vyšší [56]. Samotnou instalaci se provede příkazem (Kód 11.1) pomocí balíčkovacího nástroje *pip*, který je součástí jazyka Python.

```
$ pip install gradio
```

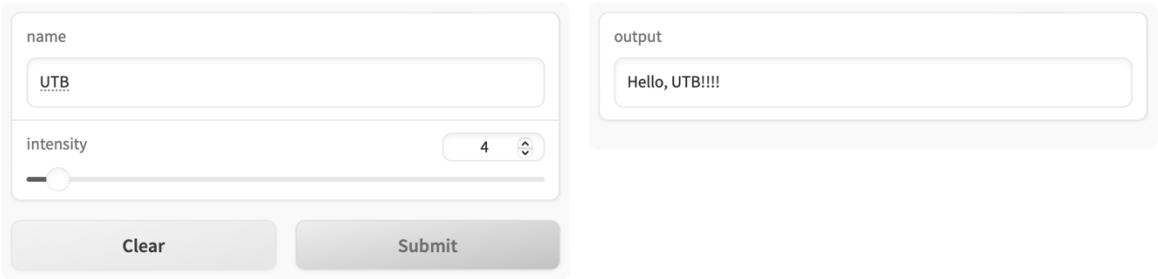
Kód 11.1 Instalace balíčku Gradio pro vývoj webové aplikace

Správnou instalaci je možné otestovat vytvořením testovacího souboru, jehož obsahem bude importována knihovna Gradio, jeden textový vstup, jeden posuvník a výstupní text. Na výstupu se je využitá komponenta textového pole. Následně je potřeba definovat funkci, která bude vstupní parametry zpracovávat. V ukázkové aplikaci (Obrázek 11.1) je ukázáno že na výstupu se vypíše text *Hello*, ke kterému se připojí vstupní text a na konec se podle zvoleného čísla na posuvníku připojí znásobený znak vykřičníku.

```
 1. import gradio as gr
 2.
 3. def greet(name, intensity):
 4.     return "Hello, " + name + "!" * int(intensity)
 5.
 6. demo = gr.Interface(
 7.     fn=greet,
 8.     inputs=["text", "slider"],
 9.     outputs=["text"],
10. )
11.
12. demo.launch()
```

Kód 11.2 Ukázkový kód práce s Gradio, převzato z [56]

Soubor obsahující kód pro naprogramované rozhraní (Kód 11.2) se uloží a spustí stejným způsobem jako kterýkoli jiný Python soubor. Po spuštění se v příkazové řádce zobrazí webová adresa, která po otevření v prohlížeči zobrazí aplikaci s naprogramovaným rozhraním (Obrázek 11.1).



Obrázek 11.1 Rozhraní aplikace ve webovém prohlížeči, převzato z [56]

Pro vývoj rozhraní určeného k testování bude nezbytné definovat jeden vstupní prvek, kterým bude obrázek, a dva výstupní obrázky. První výstupní obrázek bude použit k zobrazení výsledků z natrénovaného modelu YOLO, zatímco druhý výstupní obrázek bude sloužit k vizualizaci výsledků z natrénovaného modelu Detectron2.

Nejdříve byl vytvořen vstupní soubor, který implementuje rozhraní a zajišťuje inferenci s modely.

```
import gradio as gr
from thesis.utils import interface
from thesis import detector

# Create interface
with gr.Blocks() as app:
    gr.Markdown(f"# {interface.TITLE}")
    gr.Markdown(interface.DESCRIPTION)

    with gr.Tab("Image detection"):
        with gr.Row():
            with gr.Column():
                file_input = interface.image_input()
                conf_input = interface.conf_input()
                iou_input = interface.iou_input()
            yolo_output = gr.Image(label="YOLOv8")
            detectron2_output = gr.Image(label="Detectron2")

        btn = gr.Button("Run")
        btn.click(fn=detector.detect_image,
                  inputs=[file_input, conf_input, iou_input],
                  outputs=[yolo_output, detectron2_output])

app.launch()
```

Kód 11.3 Vstupní soubor rozhraní Gradio

Vzhledem k tomu, že některé vstupní prvky byly komplexnější a vyžadovaly specifické popisy a omezení, tyto části kódu byly přesunuté do samostatného souboru (Kód 11.4). Tento soubor nyní slouží jako místo pro načítání těchto prvků, kdy takový přístup zlepšuje přehlednost a organizaci kódu.

```
def image_input():
    return gr.Image(
        type="filepath",
        label="Image for detection",
    )


def iou_input():
    return gr.Slider(
        label="IoU threshold",
        step=0.01,
        value=0.45,
        minimum=0,
        maximum=0.95
    )


def conf_input():
    return gr.Slider(
        label="Confidence threshold",
        step=0.01,
        value=0.25,
        minimum=0,
        maximum=0.95
    )
```

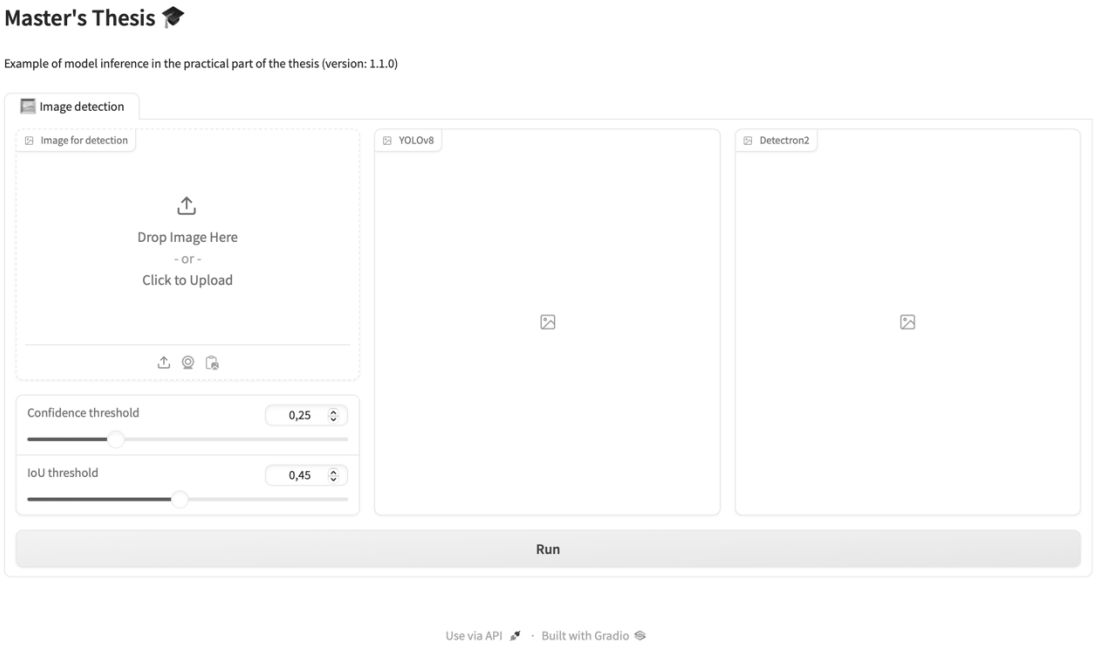
Kód 11.4 Část kódu s definovaným rozhraním

Další klíčovou funkcí je samotné zpracování vstupů. Vzhledem k tomu, že vstupním prvkem je obrázek, celý proces jeho zpracování zahrnuje uložení do dočasné složky. Po zpracování je obrázek odstraněn. Tento proces ukládání a mazání řídí Gradio samo a vývojář do něj nemá možnost zasáhnout. Funkce pro zpracování vstupů pracuje tak, že přijme parametry vstupního obrázku, nastavený práh spolehlivosti a Jaccardův index (IoU). Tyto parametry jsou následně předány jednotlivým prediktorům, přičemž dochází také k vizualizaci detekcí. Detekce jsou poté vráceny zpět do aplikace Gradio, kde jsou vykresleny.

```
 1. def detect_image(file_input, conf_input, iou_input):
 2.     # YOLO
 3.     yolo_model = YOLO(os.path.join("/mnt/storage/models/yolov8", "yolov8.pt"))
 4.     yolo_result = processor.yolo_im_box_prediction(yolo_model, file_input, conf_input,
iou_input)
 5.
 6.     # Detectron2
 7.
 8.     try:
 9.         print(os.path.join("/mnt/storage/datasets/master-thesis-coco", "result.json"))
10.         register_coco_instances("master-thesis",
11.             {},
12.             os.path.join("/mnt/storage/datasets/master-thesis-coco", "result.json"),
13.             os.path.join("/mnt/storage/datasets", "master-thesis-coco"))
14.     except AssertionError:
15.         print("COCO instance is registered")
16.
17.     cfg = get_cfg()
18.     cfg.merge_from_file(os.path.join("/mnt/storage/models/detectron2", "config.yml"))
19.     cfg.MODEL.WEIGHTS = os.path.join("/mnt/storage/models/detectron2", "model_final.pth")
20.     cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = conf_input
21.     cfg.MODEL.ROI_HEADS.NMS_THRESH_TEST = iou_input
22.     cfg.MODEL.DEVICE = "cpu"
23.
24.     predictor = DefaultPredictor(cfg)
25.     im = cv2.imread(file_input)
26.     outputs = predictor(im)
27.
28.     v = Visualizer(im[:, :, ::-1], metadata=MetadataCatalog.get("master-thesis"),
scale=0.8)
29.     v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
30.
31.     detectron2_result = cv2.cvtColor(v.get_image()[:, :, ::-1], cv2.COLOR_BGR2RGB)
32.
33.     return [yolo_result, detectron2_result]
```

Kód 11.5 Funkce pro zpracování vstupního obrázku prediktory

Po spuštění kódu je možné pomocí webového prohlížeče zobrazit grafické rozhraní aplikace (Rozhraní pro testování modelů na vstupních datech), do kterého je nyní možné vložit fotografii a následně porovnávat výstupy jednotlivých modelů.



Obrázek 11.2 Rozhraní pro testování modelů na vstupních datech

Webová aplikace umožňuje interaktivně nastavovat parametry pro výslednou detekci objektů pomocí dvou posuvníků. První posuvník slouží k nastavení prahu důvěryhodnosti (*Confidence threshold*), který určuje minimální úroveň jistoty, jež musí model dosáhnout, aby objekt byl považován za detekovaný. Druhý posuvník reguluje prahovou hodnotu Intersection over Union (*IoU threshold*), což je metrika používaná k evaluaci přesnosti detekce tím, že porovnává míru překryvu mezi predikovaným bounding boxem a skutečným bounding boxem. Umožněním dynamické úpravy těchto hodnot mohou uživatelé v reálném čase optimalizovat detekční výkon modelu v závislosti na specifických požadavcích nebo podmínkách datové sady. Aplikace je dostupná na URL adrese https://thesis-example.filipsedivy.com.

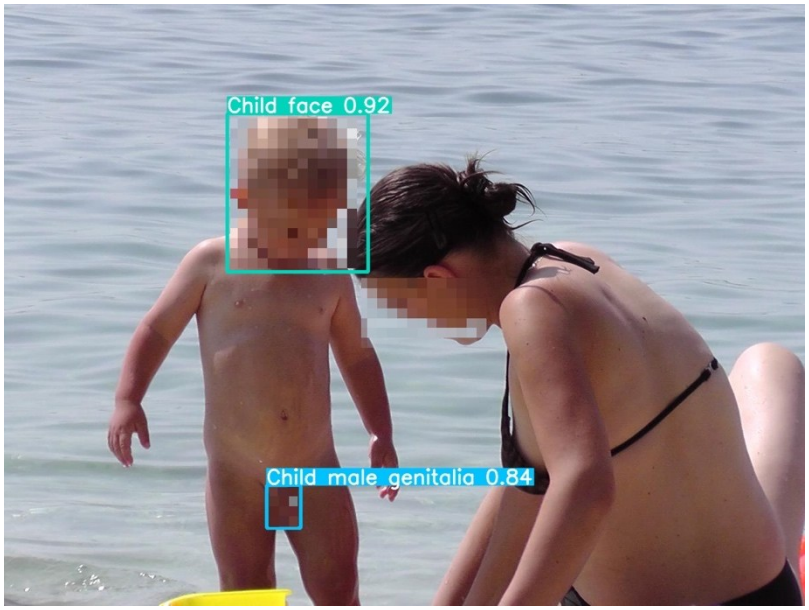
12 VYHODNOCENÍ DETEKČNÍCH MODELŮ

Modely YOLO verze 8 a Detectron2 byly natrénovány pro detekci nahých dětských osob. Díky způsobu nastavení trénovacích dat však oba modely dokážou detekovat nejen nahé dětské osoby, ale také polonahé a nahé osoby obecně, a rozlišovat mezi dětmi a dospělými. Tato schopnost rozšířené detekce vyplývá z nastavení tříd, které byly zvoleny před samotným trénovacím procesem. Pro vyhodnocení obou natrénovaných modelů byla zvolena testovací fotografie [57], která je veřejně dostupná na internetu již více než deset let. Výstup z této fotografie byl upraven s důrazem na anonymizaci osob na ní vyobrazených, přičemž citlivé části obrazu byly rozmazány za účelem zachování soukromí. Kromě této specifické fotografie bylo také provedeno širší kvalitativní vizuální srovnání schopnosti modelů pracovat s obrazovými daty různé kvality. Toto srovnání zahrnovalo datovou sadu obsahující fotografie s extrémními podmínkami, jako je přesvícení, velmi malé rozlišení nebo nestandardní barevné schéma, aby bylo možné simulovat nejhorší možné scénáře, se kterými se modely mohou setkat v praxi. Výsledky tohoto širšího srovnání byly detailně zaznamenány a jsou prezentovány v tabulce.

Při vyhodnocení obou modelů byly nastaveny ve webové aplikaci následující parametry:

- Confidence threshold: 0,6
- IoU (Intersection over Union) threshold: 0,45

Prvním natrénovaným detekčním modelem byl YOLO, který je speciálně navržen tak, aby integroval několik kroků procesu detekce do jediného kroku. Díky této integraci je model schopen provádět inferenci s významně vyšší rychlostí: na serverovém procesoru Intel Xeon Skylake dosahuje doby inferenčního zpracování 357,2 ms, zatímco na grafické kartě NVIDIA A100 s využitím TensorRT je tato doba redukována na pouhé 2,39 ms. Tato výrazná úspora času je přímo výsledkem efektivního spojení procesních kroků.



Obrázek 12.1 Vizualizovaný výstup z detektoru modelu YOLO

Vstupní testovací obrázek, zpracovaný testovací aplikací (Obrázek 12.1), odhalil jeden dětský obličej s místo jistoty 92 % a dětské genitálie s mírou jistoty 84 %.

Tabulka 12.1 Výsledek kvantitativního vizuálního srovnání modelu YOLO

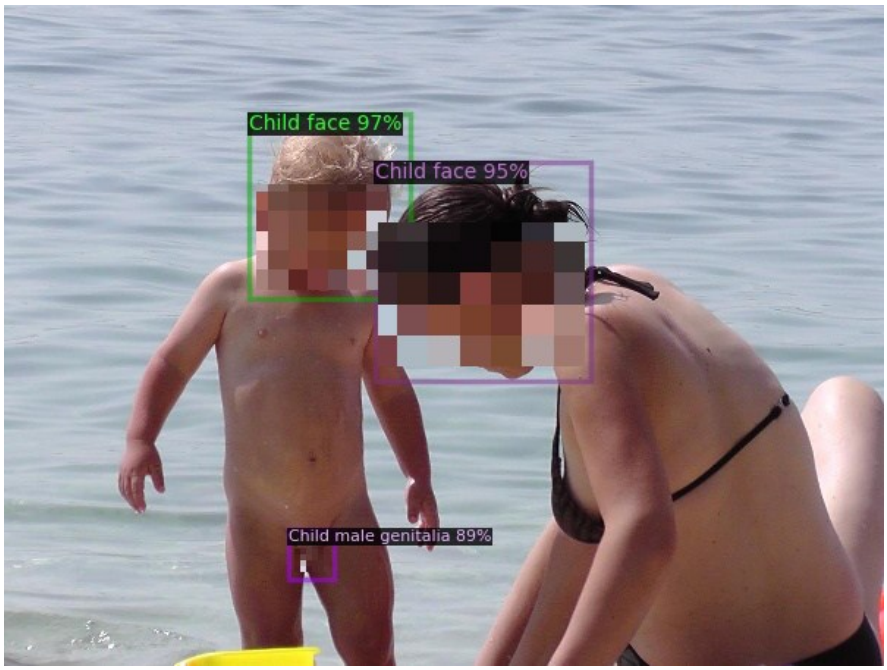
		Správně detekováno	**Chybně detekováno**	**Přesnost**
Osoba starší 18 let	Oblečená osoba	7	13	35 %
	Polonahá osoba	10	10	50 %
	Nahá osoba	4	16	20 %
Osoba mladší 18 let	Oblečená osoba	13	7	65 %
	Polonahá osoba	11	9	55 %
	Nahá osoba	14	6	70 %
	Zvíře	15	5	75 %
	Příroda	20	0	100 %

Z výsledků kvantitativního vizuálního srovnání (Tabulka 12.1) vyplývá, že model neuronové sítě má problémy s přesnou detekcí dospělých osob. Během vizuální kontroly bylo zjištěno, že mnoho obličejů dospělých bylo správně identifikováno, avšak mylně klasifikováno jako

dětské. Tento problém může pramenit z nevyváženosti datové sady a nedostatečného množství trénovacích dat. Nicméně, detekce osob mladších 18 let byla provedena s vysokou přesností. Srovnání také odhalilo, že u předložených obrázků se zvířecími obličeji byla v pěti případech nesprávně detekována přítomnost člověka. To je samozřejmě chyba, ale vysvětluje se to tím, že trénovací data neobsahovala třídu pro zvířata, což znemožnilo modelu naučit se rozlišovat mezi lidskými a zvířecími tvářemi. Nejčastěji bylo zvíře klasifikováno jako člověk, pokud se na fotografii objevilo zvíře podobné člověku, například opice nebo šimpanz. Celková přesnost modelu na testovací sadě je přibližně 58,75 %.

Z hlediska cíle detekce dětské nahoty tedy detektor splnil formální očekávání. Avšak z pohledu validace modelu byly výsledky nedostačující, jelikož detektor neidentifikoval obličej dospělé osoby, který byl součástí štítků a trénovacích dat. Pokud by tento detektor byl nasazen v produkčním prostředí, splnil by cíle této diplomové práce z hlediska detekce dětské nahoty, což by potvrzovalo jeho efektivitu pro stanovené účely.

Druhým modelem, který byl natrénován, je Detectron2.Výkon byl testován na CPU i GPU, přičemž na serverovém procesoru Intel Xeon Skylake dosáhl doby inferenčního zpracování 10,02 sekundy. I přes delší dobu zpracování, což je dáno komplexnější strukturou modelu, bylo na grafické kartě NVIDIA A100 s využitím TensorRT dosaženo výrazně rychlejší doby zpracování, která činila 3,2158 sekundy. Tato zrychlení na GPU ukazuje, jaký potenciál má Detectron2 pro nasazení v prostředích, kde je dostupná výkonná grafická karta.



Obrázek 12.2 Vizualizovaný výstup z detektoru modelu Detectron2

Výstupy z testovací aplikace ukazují (Obrázek 12.2), že dětská tvář byla detekována s vysokou mírou jistoty 97 %. Dětské mužské genitálie byly rovněž úspěšně identifikovány s mírou jistoty 89 %. Na druhou stranu, tvář patřící dospělé osobě byla detekována s vysokou, avšak nesprávnou mírou jistoty 95 %, jako dětská.

Tabulka 12.2 Výsledek kvantitativního vizuálního srovnání modelu Detectron2

		Správně detekováno	**Chybně detekováno**	**Přesnost**
Osoba starší 18 let	Oblečená osoba	10	10	50 %
	Polonahá osoba	8	12	40 %
	Nahá osoba	7	13	35 %
Osoba mladší 18 let	Oblečená osoba	14	6	70 %
	Polonahá osoba	3	17	15 %
	Nahá osoba	12	8	60 %
	Zvíře	20	0	100 %
	Příroda	20	0	100 %

Z kvantitativního vizuálního srovnání modelu Detectron2 (Tabulka 12.2) vyplývá, že detekce obličeje dospělé osoby dosahovala pouze poloviční míry přesnosti. U polonahé osoby bylo více než v polovině případů spodní prádlo nesprávně klasifikováno jako dětské, nebo nebylo detekováno vůbec. V případě nahé dospělé osoby často selhala detekce klíčových tělesných částí, což bylo způsobeno především nízkým rozlišením nebo vysokým kontrastem fotografie.

Naopak u oblečených dětských osob byla detekce dětských tváří velmi úspěšná. Největší problémy model zaznamenal při detekci dětského spodního prádla, které bylo často nesprávně přiřazováno dospělým. U nahých dětských osob, naopak, byla detekce prováděna velmi kvalitně. Největší výzvy modelu představovaly fotografie ve velmi nízkém rozlišení nebo převedené do černobílého formátu. Nejlépe byly detekovány fotografie bez jakýchkoli úprav.

V kategoriích jako zvířata a příroda byla detekce správná, což znamená, že v rámci detekce nebyly mylně identifikovány žádné objekty. V datové sadě se zvířaty, například u fotografie šimpanze, nebyla detekována tvář osoby, což je správně. Celková přesnost modelu Detectron2 na testovací datové sadě dosáhla 58,75 %.

Tento výsledek naznačuje možný nedostatek trénovacích dat pro třídu zahrnující dospělé obličeje. Ve vztahu k datové sadě pro detekci dětských osob by bylo vhodné zvýšit počet dat obsahujících snímky dětských osob ve specifických situacích, jako jsou například děti v plavkách. Celkově lze konstatovat, že model Detectron2 splňuje požadavky pro detekci dětské nahoty a je vhodný pro použití v produkčním prostředí.

V celkovém srovnání model YOLO dosáhl průměrné přesnosti (mAP) 77,65 % a Detectron2 vykázal průměrnou přesnost (mAP) 53,02 %. Lze tak konstatovat, že model YOLO má lepší schopnosti detekce oproti modelu Detectron2.

ZÁVĚR

Cílem této diplomové práce bylo získat a vhodně anotovat data dle škály COPINE, a to konkrétně do druhé úrovně. Tato data byla následně použita k trénování několika modelů umělých neuronových sítí, které byly navrženy pro detekci nahoty nezletilých osob. Nejprve byla provedena literární rešerše zaměřená na metody detekce objektů. Teoretická část práce se věnovala historii počítačového vidění, předzpracování obrazu, vlivu výběru barevných modelů na neuronové sítě a podrobně rozebírala architektury YOLO a Detectron2 založený na Faster R-CNN, na kterých byla založena praktická část této práce.

Praktická část zahrnovala vývoj datové sady s pomocí automatizované anotace a vytvoření prostředí na virtuálním serveru, který umožňuje anotovat citlivá data bez nutnosti jejich nahrávání do komerčních aplikací, čímž dochází k minimalizaci rizika úniku informací a zvýšení kontroly nad daty. Data pro anotaci nebyla dostupná ve veřejně anotovaných datových sadách specificky pro tuto problematiku, proto byla získána z webových stránek, které čelí problémům s ukládáním fotografií nezletilých osob v různých stavech oblečení – od plně oblečených po polonahé a nahé. Další data byla získána z volně přístupných internetových zdrojů. Pro trénování neuronových sítí byly zvoleny modely YOLO, konkrétně YOLO verze 8, a Detectron2 s architekturou Faster R-CNN.

Praktická část diplomové práce byla zakončena vytvořením aplikace pomocí nástroje Gradio, která umožňuje na základě vstupního obrázku porovnat výsledky obou modelů. Díky této aplikaci je možné zhodnotit efektivitu obou modelů. Webová aplikace Gradio funguje tak, že na vstupní obrázk aplikuje oba natrénované modely, YOLO a Detectron2, detekuje v nich předem natrénované třídy a následně výsledky zobrazí v uživatelském rozhraní s vykreslenými bounding boxy.

Následně byla provedena komparace modelů YOLO a Detectron2, zahrnující jak kvalitativní, tak kvantitativní srovnání. V kvantitativním srovnání byly získány následující výsledky: model YOLO dosáhl průměrné přesnosti (mAP) 77,65 %. Naproti tomu model Detectron2 vykázal průměrnou přesnost (mAP) 53,02 %. Tento rozdíl ukazuje že YOLO je lepší ve správném detekování než Detectron2.

Kvalitativní srovnání obou modelů dále ukázalo, že oba modely na testovací datové sadě dosahují přesnosti 58,75 %.

Tento výsledek naznačuje, že YOLO je robustnější při rozpoznávání objektů v různých podmínkách a má lepší schopnost než Detectron2. Detectron2 má nižší celkovou přesnost, přestože dosahuje vyšší AP50, což ukazuje na jeho efektivitu v přesné lokalizaci objektů při vyšší prahové hodnotě IoU.

Tyto modely rozšiřují možnosti pro využití jak ve státní sféře, tak v komerčním sektoru. Zejména státní orgány, jako je policie, mohou tyto modely efektivně využívat v boji proti dětské pornografii, což představuje významný krok v automatizaci detekce těchto trestných činů případně urychlují analýzu zabavených počítačů. V komerčním sektoru mohou být modely implementovány do aplikací umožňujících nahrávání souborů, kde by sloužily jako kontrolní mechanismus pro dodržování právních předpisů a podmínek užívání.

Dalším směrem rozvoje může být rozšíření detekce na video obsah. Přestože oba modely již disponují nástroji pro analýzu videí, v rámci této práce nebyla tato možnost zpracována. Rozšíření datových sad o další data by rovněž mohlo vést ke zvýšení přesnosti modelů. Tím by se základní modely mohly dotrénovat a dosáhnout lepších výsledků v detekci, což by zlepšilo celkové jejich přesnosti.

SEZNAM POUŽITÉ LITERATURY

[1] What is computer vision? Online. What is computer vision? | IBM. Dostupné z: https://www.ibm.com/topics/computer-vision. [cit. 2024-05-07].

[2] AGGARWAL, Ani, 2020. YOLO Explained. Online. Dostupné z: https://medium.com/analytics-vidhya/yolo-explained-5b6f4564f31. [cit. 2024-05-07].

[3] Computer Vision: Algorithms and Applications (Texts in Computer Science), 2022. 2nd edition. Springer. ISBN 978-3030343712.

[4] GONZALEZ, Rafael C. a WOODS, Richard E., [2018]. Digital image processing. Fourth edition. New York: Pearson. ISBN 1292223049.

[5] RGB color space. Online. Dostupné z: https://book2net.net/en/2021/11/16/rgb-color-space/. [cit. 2024-05-07].

[6] MALLICK, Satya, 2015. Why does OpenCV use BGR color format ? Online. Why does OpenCV use BGR color format ? | Learn OpenCV. Dostupné z: https://learnopencv.com/why-does-opencv-use-bgr-color-format/. [cit. 2024-05-07].

[7] HSL COLOR MODEL DECOMPOSITION IN BLENDER. Online. HSL COLOR MODEL DECOMPOSITION IN BLENDER | MeshLogic. Dostupné z: https://meshlogic.github.io/posts/blender/materials/nodes-hsl-color-model/. [cit. 2024-05-07].

[8] MONTANARI, Tobia, 2023. HSL and HSV Explained: Which Color Model Should You Use? [online]. [cit. 2024-05-07]. Dostupné z: https://www.tobiamontanari.com/hsl-and-hsv-explained-which-color-model-should-you-use/

[9] GREAT LEARNING TEAM, 2024. Introduction to Image Pre-processing | What is Image Pre-processing? [online]. [cit. 2024-05-07]. Dostupné z: https://www.mygreatlearning.com/blog/introduction-to-image-pre-processing/

[10] LECUN, Yann et al., 2000. Efficient BackProp. ResearchGate [online]. [cit. 2024-05-07]. Dostupné z: https://www.researchgate.net/publication/2811922_Efficient_BackProp

[11] KLETTE, Reinhard, 2014. Concise Computer Vision: An Introduction into Theory and Algorithms [online]. London: Springer London [cit. 2024-05-07]. Undergraduate Topics in Computer Science. ISBN 978-1-4471-6320-6. Dostupné z: doi:978-1-4471-6320-6

[12] BENGIO, Yoshua, 2016. Deep Learning. MIT Press. ISBN 978-0262035613.

[13] SCHMIDHUBER, Jürgen, 2015. Deep learning in neural networks: An overview. Neural Networks [online]. 61, 85-117 [cit. 2024-05-07]. ISSN 08936080. Dostupné z: doi:10.1016/j.neunet.2014.09.003

[14] Koncept umělé neuronové sítě. Matematická biologie učebnice: Koncept umělé neuronové sítě [online]. [cit. 2024-05-07]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--uvod-do-neuronovych-siti--koncept-umele-neuronove-site

[15] LECUN, Yann, Yoshua BENGIO a Geoffrey HINTON, 2015. Deep learning. Nature [online]. 521(7553), 436-444 [cit. 2024-05-07]. ISSN 0028-0836. Dostupné z: doi:10.1038/nature14539

[16] Simple Feed Forward Neural Network code for digital Handwritten digit recognition, 2019. WARKE, Chetan. Simple Feed Forward Neural Network code for digital Handwritten digit recognition [online]. [cit. 2024-05-07]. Dostupné z: https://medium.com/random-techpark/simple-feed-forward-neural-network-code-for-digital-handwritten-digit-recognition-a234955103d4

[17] LECUN, Y. et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE [online]. 86(11), 2278-2324 [cit. 2024-05-07]. ISSN 00189219. Dostupné z: doi:10.1109/5.726791

[18] ELMAN, J., 1990. Finding Structure in Time [online]. [cit. 2024-05-07]. Dostupné z: https://api.semanticscholar.org/CorpusID:2763403

[19] SRIVASTAVA, Nitish et al. Dropout: a simple way to prevent neural networks from overfitting. In: The Journal of Machine Learning Research [online]. s. 1929–1958 [cit. 2024-05-07]. Dostupné z: https://dl.acm.org/doi/10.5555/2627435.2670313

[20] LECUN, Y. et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE [online]. 86(11), 2278-2324 [cit. 2024-05-07]. ISSN 00189219. Dostupné z: doi:10.1109/5.726791

[21] TABIAN, Iuliana, Hailing FU a Zahra Sharif KHODAEI, 2019. A Convolutional Neural Network for Impact Detection and Characterization of Complex Composite Structures. Sensors [online]. 19(22) [cit. 2024-05-07]. ISSN 1424-8220. Dostupné z: doi:10.3390/s19224933

[22] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey E. HINTON, 2017. ImageNet classification with deep convolutional neural networks. Communications of the ACM [online]. 60(6), 84-90 [cit. 2024-05-07]. ISSN 0001-0782. Dostupné z: doi:10.1145/3065386

[23] BENGIO, Y., 2009. Learning Deep Architectures for AI. Foundations and Trends® in Machine Learning [online]. 2(1), 1-127 [cit. 2024-05-07]. ISSN 1935-8237. Dostupné z: doi:10.1561/2200000006

[24] HOCHREITER, Sepp a Jürgen SCHMIDHUBER, 1997. Long Short-Term Memory. Neural Computation [online]. 9(8), 1735-1780 [cit. 2024-05-07]. ISSN 0899-7667. Dostupné z: doi:10.1162/neco.1997.9.8.1735

[25] DE, Debarko, 2018. RNN or Recurrent Neural Network for Noobs [online]. [cit. 2024-05-07]. Dostupné z: https://medium.com/hackernoon/rnn-or-recurrent-neural-network-for-noobs-a9afbb00e860

[26] RUMELHART, David E., Geoffrey E. HINTON a Ronald J. WILLIAMS, 1986. Learning representations by back-propagating errors. Nature [online]. 323(6088), 533-536 [cit. 2024-05-07]. ISSN 0028-0836. Dostupné z: doi:10.1038/323533a0

[27] P. KINGMA, Diederik a Jimmy BA, 2017. Adam: A Method for Stochastic Optimization [online]. [cit. 2024-05-07]. Dostupné z: https://arxiv.org/abs/1412.6980

[28] RAINA, Rajat, Anand MADHAVAN a Andrew Y. NG, 2009. Large-scale deep unsupervised learning using graphics processors. In: Proceedings of the 26th Annual International Conference on Machine Learning [online]. New York, NY, USA: ACM, s. 873-880 [cit. 2024-05-07]. ISBN 9781605585161. Dostupné z: doi:10.1145/1553374.1553486

[29] PRAMODITHA, Rukshan, 2022. Overview of a Neural Network's Learning Process [online]. [cit. 2024-05-07]. Dostupné z: https://medium.com/data-science-365/overview-of-a-neural-networks-learning-process-61690a502fa

[30] IMAGE CLASSIFICATION VS. OBJECT DETECTION: EVERYTHING YOU NEED TO KNOW [online], 2024. [cit. 2024-05-07]. Dostupné z: https://mindy-support.com/news-post/image-classification-vs-object-detection-everything-you-need-to-know/

[31] Image Classification vs. Object Detection vs. Image Segmentation, 2019. SHARMA, Pulkit. Image Classification vs. Object Detection vs. Image

Segmentation [online]. [cit. 2024-05-07]. Dostupné z: https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81

[32] Ultralytics YOLOv8 Docs [online]. [cit. 2024-05-07]. Dostupné z: https://docs.ultralytics.com/

[33] DU, Lixuan, Rongyu ZHANG a Xiaotian WANG, 2020. Overview of two-stage object detection algorithms. Journal of Physics: Conference Series [online]. 1544(1) [cit. 2024-05-07]. ISSN 1742-6588. Dostupné z: doi:10.1088/1742-6596/1544/1/012033

[34] GIRSHICK, Ross et al., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation [online]. [cit. 2024-05-07]. Dostupné z: doi:10.48550/arXiv.1311.2524

[35] WU, Jiatu a Yansong WANG, 2018. Complexity and accuracy analysis of common artificial neural networks on pedestrian detection. MATEC Web of Conferences [online]. 232 [cit. 2024-05-07]. ISSN 2261-236X. Dostupné z: doi:10.1051/matecconf/201823201003

[36] R-CNN | Region Based CNNs [online], 2023. [cit. 2024-05-07]. Dostupné z: https://www.geeksforgeeks.org/r-cnn-region-based-cnns/

[37] GIRSHICK, Ross et al., 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition [online]. IEEE, s. 580-587 [cit. 2024-05-07]. ISBN 978-1-4799-5118-5. Dostupné z: doi:10.1109/CVPR.2014.81

[38] REN, Shaoqing et al., 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [online]. [cit. 2024-05-07]. Dostupné z: https://arxiv.org/abs/1506.01497

[39] TERVEN, Juan a Diana CORDOVA-ESPARZA, 2023. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS [online]. [cit. 2024-05-07]. Dostupné z: https://arxiv.org/abs/2304.00501

[40] HUSSAIN, Muhammad, 2023. YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect

Detection. Machines [online]. 11(7) [cit. 2024-05-07]. ISSN 2075-1702. Dostupné z: doi:10.3390/machines11070677

[41] REDMON, Joseph a Ali FARHADI, 2018. YOLOv3: An Incremental Improvement [online]. [cit. 2024-05-07]. Dostupné z: https://arxiv.org/abs/1804.02767v1

[42] ROSEBROCK, Adrian, 2016. Intersection over Union (IoU) for object detection [online]. [cit. 2024-05-07]. Dostupné z: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[43] BOESCH, Gaudenz. What is Intersection over Union (IoU)? [online]. [cit. 2024-05-07]. Dostupné z: https://viso.ai/computer-vision/intersection-over-union-iou/

[44] BUHL, Nikolaj, 2023. Mean Average Precision in Object Detection [online]. [cit. 2024-05-07]. Dostupné z: https://encord.com/blog/mean-average-precision-object-detection/

[45] LOWE, Jay, 2022. Precision and Recall in Machine Learning [online]. [cit. 2024-05-07]. Dostupné z: https://blog.roboflow.com/precision-and-recall/

[46] F1 Score in Machine Learning [online], 2024. [cit. 2024-05-07]. Dostupné z: https://deepgram.com/ai-glossary/f1-score-machine-learning

[47] Docker [online]. [cit. 2024-05-07]. Dostupné z: https://www.docker.com/

[48] Label Studio. Open Source Data Labeling | Label Studio [online]. [cit. 2024-05-07]. Dostupné z: https://labelstud.io/

[49] COPINE scale, 2007. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation [cit. 2024-05-07]. Dostupné z: https://en.wikipedia.org/wiki/COPINE_scale

[50] Rajče. Rajče - místo pro vaše fotky [online]. [cit. 2024-05-07]. Dostupné z: https://www.rajce.idnes.cz/

[51] Neomezený hosting zdarma. Neomezený hosting zdarma @iMGSRC.RU [online]. [cit. 2024-05-07]. Dostupné z: https://imgsrc.ru

[52] Configs and boilerplates for Label Studio's Machine Learning backend [online]. [cit. 2024-05-08]. Dostupné z: https://github.com/HumanSignal/label-studio-ml-backend

[53] Object Detection. Detect - Ultralytics YOLOv8 Docs [online]. [cit. 2024-05-08]. Dostupné z: https://docs.ultralytics.com/tasks/detect/

[54] Detectron2 [online]. [cit. 2024-05-07]. Dostupné z: https://github.com/facebookresearch/detectron2

[55] Gradio [online]. [cit. 2024-05-09]. Dostupné z: https://www.gradio.app

[56] Quickstart. Gradio [online]. [cit. 2024-05-09]. Dostupné z: https://www.gradio.app/guides/quickstart

[57] 2011 - Makarská, 2011. Album na Rajčeti [online]. [cit. 2024-05-09]. Dostupné z: https://www.rajce.idnes.cz/holekpetr/album/2011-makarska

[58] The open source perceptual hash library. PHash.org: Home of pHash, the open source perceptual hash library [online]. [cit. 2024-05-11]. Dostupné z: https://www.phash.org/

[59] NCMEC, Google a technologie hashování obrazu [online]. [cit. 2024-05-11]. Dostupné z: https://safety.google/stories/hash-matching-to-help-ncmec/

[60] Nude.js [online]. [cit. 2024-05-11]. Dostupné z: https://github.com/pa7/nude.js

[61] NudeNet [online]. [cit. 2024-05-11]. Dostupné z: https://github.com/notAI-tech/NudeNet

[62] SSD: Single Shot MultiBox Detector [online]. [cit. 2024-05-12]. Dostupné z: https://arxiv.org/abs/1512.02325

[63] FENGADE, Somesh, 2023. Understanding L1 and SmoothL1Loss [online]. [cit. 2024-05-12]. Dostupné z: https://someshfengde.medium.com/understanding-l1-and-smoothl1loss-f5af0f801c71

[64] KOECH, Kiprono Elijah, 2020. Cross-Entropy Loss Function [online]. [cit. 2024-05-12]. Dostupné z: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

[65] REDMON, Redmon, 2016. Darknet: Open Source Neural Networks in C [online]. [cit. 2024-05-12]. Dostupné z: http://pjreddie.com/darknet/

[66] SIMONYAN, Karen a Andrew ZISSERMAN, 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition [online]. [cit. 2024-05-12]. Dostupné z: https://arxiv.org/abs/1409.1556v6

[67] Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition [online], 2014. In: HE, Kaiming et al. [cit. 2024-05-12]. Dostupné z: https://arxiv.org/abs/1406.4729

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

Adam	Adaptive moment estimation
ANN	Artificial neural network
CNN	Convolutional neural network
COCO	Common Objects in Context
CUDA	Compute Unified Device Architecture
mAP	Mean average precision
MLP	Multilayer perceptron
OCR	Optické rozpoznávání znaků
R-CNN	Region-based convolutional neural network
RNN	Recurrent neural network
ROI	Region of interest
RPN	Region proposal network
SGD	Stochastic gradient descent
SPP	Spatial pyramid pooling
SSD	Single Shot Detector
SSH	Secure Shell
YOLO	You Only Look Once

SEZNAM OBRÁZKŮ

Obrázek 1.1 Ukázka ohraničujících rámečků, převzato z [2] 14
Obrázek 2.1 RGB model, převzato z [5] 17
Obrázek 2.2 HSL a HSV model, převzato z [7] 18
Obrázek 3.1 Uspořádání neuronů do vrstev, převzato z [14] 22
Obrázek 3.2 Schéma dopředné neuronové sítě s jednou skrytou vrstvou, převzato z [16] 23
Obrázek 3.3 Ukázka konvoluční neuronové sítě, převzato z [21] 24
Obrázek 3.4 Ukázka rekurentní neuronové sítě, převzato z [25] 26
Obrázek 3.5 Proces učení, převzato z [29] 28
Obrázek 4.1 R-CNN architektura, převzato z [36] 31
Obrázek 4.2 Architektura YOLO, převzato z [35] 36
Obrázek 5.1 Vizualizace IoU, převzato z [42] 39
Obrázek 8.1 Výstup po spuštění testovacího příkazu 49
Obrázek 8.2 Úvodní webová stránka 50
Obrázek 8.3 Úvodní stránka Label Studio 51
Obrázek 9.1 Label Studio s připravenými projekty 54
Obrázek 9.2 Nastavení modelu pro automatickou anotaci 57
Obrázek 10.1 Zápis třídy a pozic ohraničení pro jeden obrázek 58
Obrázek 10.2 Proces učení YOLO modelu 60
Obrázek 10.3 Výstup NVIDIA CUDA a verze Detectron2 61
Obrázek 11.1 Rozhraní aplikace ve webovém prohlížeči, převzato z [56] 65
Obrázek 11.2 Rozhraní pro testování modelů na vstupních datech 68
Obrázek 12.1 Vizualizovaný výstup z detektoru modelu YOLO 70
Obrázek 12.2 Vizualizovaný výstup z detektoru modelu Detectron2 71

SEZNAM TABULEK

Tabulka 10.1 Statistické vlastnosti natrénovaného modelu YOLO.....60
Tabulka 10.2 Statistické vlastnosti natrénovaného modelu Detectron2 63
Tabulka 12.1 Výsledek kvantitativního vizuálního srovnání modelu YOLO 70
Tabulka 12.2 Výsledek kvantitativního vizuálního srovnání modelu Detectron2.....72

SEZNAM UKÁZEK KÓDU

Kód 8.1 Aktualizace balíčků a jejich závislostí 48
Kód 8.2 Instalace balíčků, které umožní používat zabezpečený protokol HTTPS 48
Kód 8.3 Přidání GPG klíče do seznamu důvěryhodných klíčů 48
Kód 8.4 Instalace nástroje Docker 49
Kód 8.5 Spuštění testovacího obrazu *hello-world* 49
Kód 8.6 Instalace Nginx 50
Kód 8.7 Spuštění Label Studio ve virtuálním prostředí Docker 51
Kód 9.1 Instalace závislostí třetích stran 54
Kód 9.2 Vytvoření nového projektu 55
Kód 9.3 Načítání modelu pomocí konstruktoru 55
Kód 9.4 Provádění inference nad modelem 56
Kód 9.5 Spuštění kontejneru na pozadí 57
Kód 10.1 Rozdělení datové sady na trénovací a validační množinu 59
Kód 10.2 Spuštění trénování YOLO modelu 59
Kód 10.3 Instalace modelu Detectron2 61
Kód 10.4 Kód pro testování úspěšné instalace Detectron2 61
Kód 10.5 Trénování Detectron2 modelu 62
Kód 10.6 Uložení konfigurace do souboru 63
Kód 10.7 Spuštění validace natrénovaného modelu 63
Kód 11.1 Instalace balíčku Gradio pro vývoj webové aplikace 64
Kód 11.2 Ukázkový kód práce s Gradio, převzato z [56] 65
Kód 11.3 Vstupní soubor rozhraní Gradio 66
Kód 11.4 Část kódu s definovaným rozhraním 66
Kód 11.5 Funkce pro zpracování vstupního obrázku prediktory 67

SEZNAM PŘÍLOH

Příloha P I: Obsah přiloženého CD

PŘÍLOHA P I: OBSAH PŘILOŽENÉHO CD

fulltext.pdf	Textová část diplomové práce
modely/	Zdrojové kódy a natrénované modely
scripty/	Skripty pro práci s datovou sadou
webova_aplikace/	Webová aplikace