Rozpoznávání objektů v obraze na platformě NVidia Jetson Nano

Bc. Jan Veselý

Diplomová práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav automatizace a řídicí techniky

Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jan Veselý**
Osobní číslo:	**A19387**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Automatické řízení a informatika**
Forma studia:	**Kombinovaná**
Téma práce:	**Rozpoznávání objektů v obraze na platformě NVidia Jetson Nano**
Téma práce anglicky:	**Object Detection on nVidia Jetson Nano Platform**

Zásady pro vypracování

1. Proveďte rešerši metod detekce objektů pro strojové vidění.
2. Seznamte se s embedded platformou nVidia Jetson Nano a jejími možnostmi pro detekci objektů v obraze.
3. Vytvořte aplikaci s využitím hlubokého učení na platformě Jetson Nano pro detekci objektů.
4. Ověřte funkci aplikace pro detekci vad výrobku.
5. Porovnejte s průmyslovým systémem pro detekci vad.

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

Seznam doporučené literatury:

1. CHOLLET, Francois, 2019. *Deep learning v jazyku Python*. Praha: Grada. ISBN 978-8024731001.
2. SOLEM, Jan Erik, 2012. *Programming Computer Vision with Python: Tools and algorithms for analyzing images*. Massachusetts: O'Reilly Media. ISBN 978-1449316549.
3. GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE, 2016. *Deep learning*. Cambridge: MIT Press. Adaptive computation and machine learning (MIT Press). ISBN 978-0262035613.
4. ROSENBROCK, Adrian, 2017. *Deep learning for computer vision with Python*. Místo není známé: PyimageSearch. ISBN 978-1986538138.
5. FORSYTH, David a Jean PONCE, 2011. *Computer Vision: A Modern Approach*. 2nd Edition. London: Pearson. ISBN 9780273764144.
6. DAVIES, E.R., 2012. *Computer and machine vision: Theory, algorithms, practicalities*. 4th Edition. Boston: Elsevier. ISBN ISBN 978-01-2386-908-1.

Vedoucí diplomové práce: **Ing. Jakub Novák, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání diplomové práce: **15. ledna 2022**
Termín odevzdání diplomové práce: **20. května 2022**

L.S.

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Ing. Vladimír Vašek, CSc. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

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 18.5.2022

Jan Veselý, v. r.

ABSTRAKT

Cílem této práce je tvorba aplikace detekce objektů v obraze implementovaná na platformě Nvidia Jetson Nano. Je vypracován potřebný teoretický základ potřebný porozumění problematice konvolučních neuronových sítí a na příkladech popsán postup jejich učení. Na závěr je vypracováno srovnání vytvořeného řešení na platformě Nvidia Jetson Nano s komerčním systémem od firmy Cognex pro zpracování obrazu na principu neuronových sítí.

Klíčová slova:

Neuronové sítě, Počítačové vidění, Nvidia Jetson Nano, OpenCV, TensorFlow, Detekce vad, Hluboké učení

ABSTRACT

This work focuses on creation visual object detection application implemented on Nevidia Jetson Nano. There is given necessary theoretical basis for understanding the problem of convolutional neural networks and described procedure of their learning is described on examples. Finally, a comparison of the developed solution on the Nvidia Jetson Nano platform with a commercial vision system from Cognex company is made.

Keywords:

Neural networks, Computer vision, Nvidia Jetson Nano, OpenCV, TensorFlow, Defect detection, Deep learning

Rád bych poděkoval vedoucímu mé diplomové práce, panu Ing. Jakubu Novákovi, Ph.D. za odborné vedení, konzultace, a cenné rady.

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

OBSAH

ÚVOD

Počítačové vidění je jednou z technických oblastí, která v poslední době zaznamenává velký rozmach. Zvláště dynamický rozvoj pozorujeme při využití v průmyslové výrobě, autonomních systémech vozidel a biometrii. Obecně lze říci, že počítačové vidění má jako hlavní úkol především automatizaci procesu získávání požadovaných dat ze vstupních snímků tak, aby mohl proces probíhat bez zásahů člověka.

Analýza obrazu probíhá zpravidla dvěma odlišnými způsoby. První, v praxi více zaběhlou, je analýza konvenčními postupy, jež zahrnuje například filtraci, detekci hran, vyhodnocení jasu a histogramu. Zmíněné postupy jsou implementovány člověkem a probíhají dle plně popsaných matematických vzorců.

Druhý způsob analýzy obrazu, zpracování obrazu založené na umělé inteligenci, do praxe teprve proniká. Samotné zpracování probíhá procesem podobným učení lidí, kdy se za asistence člověka naučí rozpoznávat prvky a jejich vlastnosti v obrazu. Umělá inteligence používá odlišné přístupy než klasické počítačové vidění a často úspěšně řeší problémy, které bývají konvenčními postupy počítačového vidění těžko uchopitelné.

Cílem této práce je použití embedded zařízení Nvidia Jetson Nano umožňující použít metody hlubokého učení pro vytvoření aplikace vizuální detekce vad a dále porovnání tohoto zařízení s možnostmi komerčního systému optické kontroly využívající umělé inteligence.

I. TEORETICKÁ ČÁST

1 STROJOVÉ VIDĚNÍ

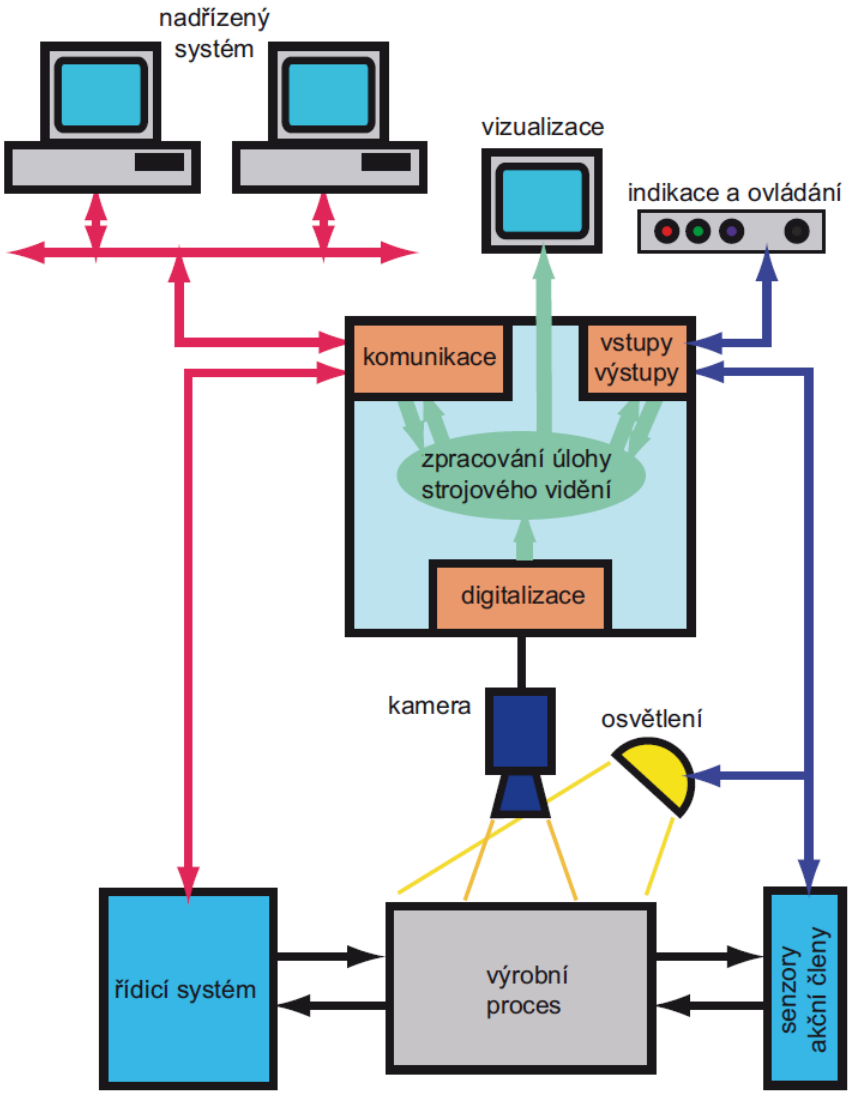
Obor počítačové vidění (v průmyslové automatizaci častěji nazývané strojové vidění), vznikl v 70. letech, kdy rozvoj výpočetní techniky začal umožňovat zpracování dostatečně velkého objemu dat [1]. Od té doby probíhá v této oblasti vývoj algoritmů pro zpracování obrazu a strojové vidění, nicméně až na přelomu tisíciletí byla běžně dostupná řešení pro přijatelnou realizaci aplikací strojového vidění v podmínkách průmyslové automatizace, v nichž se strojové vidění využívá především pro vizuální kontrolu výrobních procesů, které s moderním pojetím výroby, jenž zahrnuje včasné dodávky a nulovou nekvalitu, dostává čím dál větší uplatnění.

Společně se zvyšujícími se nároky na strojového vidění byly vyvíjeny i jeho hardwarové součásti a zároveň vznikaly speciální objektivy a osvětlovače pro použití v průmyslové automatizaci, které budou představeny v následujících kapitolách.

1.1 Princip a význam

Strojové vidění v průmyslu je charakteristické tím, že slouží jako zpětná vazba při řízení výrobního procesu. Systém získává z procesu nezbytné vstupní údaje, kterými zpravidla bývá povel k pořízení snímku v okamžiku, v němž je sledovaný objekt ve vhodné poloze a v závislosti na výsledku vyhodnocení obrazu, systém obyčejně vykoná akční zásah do procesu, nejčastěji označení nebo vyřazení vadného kusu [2].

Systém strojového vidění lze nalézt na Obrázku 1 a skládá z osvětlovačů, objektivu, kamerového senzoru a počítače pro zpracování obrazu. Dále je systém doplněn o prostředky komunikace s okolními zařízeními, například digitálními vstupy a výstupy či některým standardem průmyslového ethernetu pro komunikaci s nadřazeným systémem. Jednotlivé komponenty jsou specifikovány pro každou aplikaci strojového vidění, dle snímaného objektu, množství potřebného výpočetního výkonu pro zpracování obrazu a dalších. Podrobnější popis jednotlivých částí je uveden dále v této kapitole.



Obrázek 1: Obecné uspořádání systému strojového vidění [1]

Kamerový senzor je nejjednodušší podoba systému strojového vidění, v němž je celý systém včetně kamery, osvětlení i objektivu integrován v jednom pouzdře a také je určen pro základní optickou kontrolu a umožňuje pouze omezené možnosti programování funkcí. Podoba kamerového senzoru od firmy Keyence je na Obrázku 2.

Obrázek 2: Příklad kamerového senzoru – Keyence IV-HG500 [3]

Inteligentní kamera představuje kompaktní systém strojového vidění, obvykle s širokými možnostmi programování. Vyhodnocovací jednotka v podobě jednočipového počítače je součástí pouzdra kamery obvykle i s objektivem a osvětlením. Příklad smart kamery od firmy Cognex je na Obrázku 3.

Obrázek 3: Příklad smart kamery – Cognex In-Sight 2000 [4]

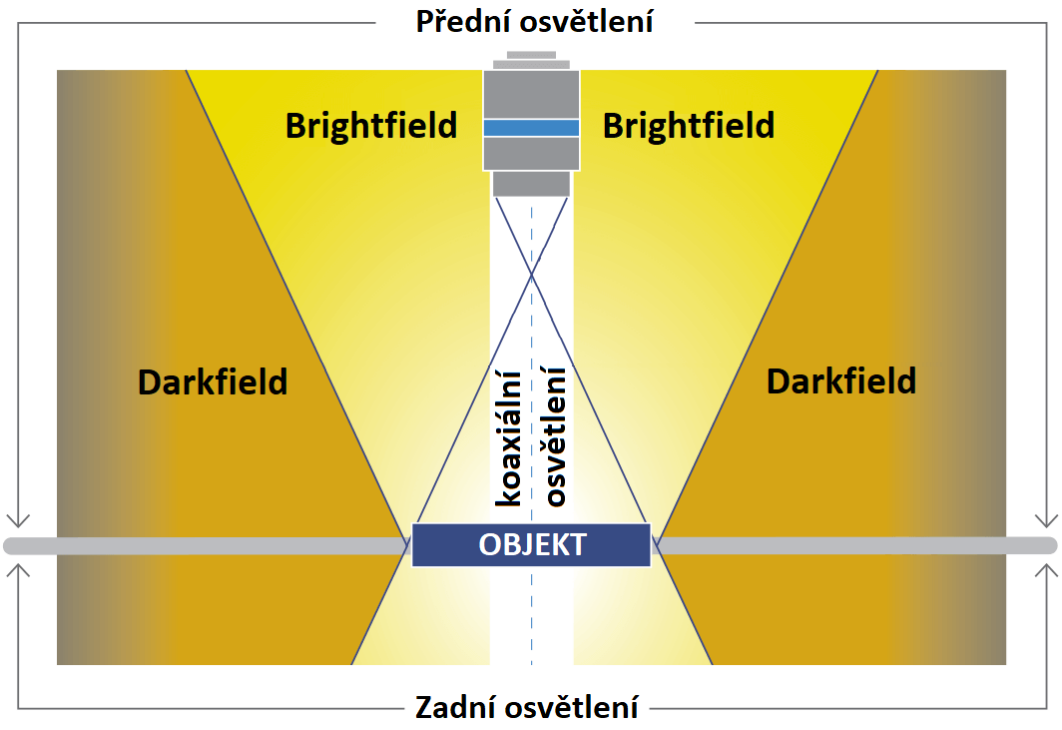
PC systém počítačového vidění již nemá podobu kompaktní kamery, ale je implementovaný jako stolní počítač vybavený periferiemi. Hlavní výhoda tohoto řešení spočívá v modularitě celého systému. Takto uspořádaný systém strojového vidění disponuje výkonem PC, může obsluhovat větší množství kamer a komunikovat pomocí rozlišných rozhraní, z čehož plyne, že v případě použití většího množství kamer je tak toto řešení obvykle také cenově nejvýhodnější. Podoba PC-based systému strojového vidění je na Obrázku 4.

Obrázek 4: Příklad PC kamerového systému – Cognex VC5 [4]

Zákaznické systémy se při návrhu naprosto přizpůsobují požadavkům zákazníka a je pro ně vyvinut speciální hardware. Většinou jsou to úlohy vyžadující vysokou rychlost zpracování nebo speciální typ kamery.

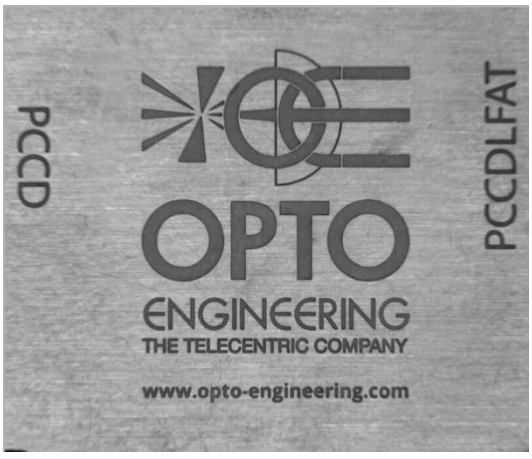
1.2 Osvětlení

Správně zvolené osvětlení je důležitým předpokladem pro správnou funkci systému strojového vidění. Účelem osvětlení ve strojovém vidění je dosáhnout maximálního kontrastu těch částí objektu, které nás zajímají proti částem, které nejsou předmětem inspekce. Využívá se k tomu různá absorpce světla nebo rozdíl v odrazu světla v závislosti na jeho směru. Návrh vhodného osvětlení spočívá v uvážení vlastností osvětlovaného objektu, jako jsou jeho struktura, průsvitnost a osvětlení z okolí. Přehled základních typů osvětlení je na Obrázku 5. V následujícím textu budou tyto typy osvětlení dále rozebrány s popisem jejich vlastností a typického uplatnění [1].



Obrázek 5: Typy osvětlení podle směru světla vůči objektu a objektivu

Jasné světelné pole (brightfield) – nejčastěji používané uspořádání osvětlovačů, kdy je objekt nasvícen z úhlu blízkého ose objektivu. Světlo se tak odráží od ploch na objektu a je zachyceno objektivem. Obrázek 6 zobrazuje příklad snímku zachyceného s osvětlením typu brightfield.



Obrázek 6: Snímek s osvětlením typu brightfield [5]

Temné světelné pole (darkfield) – objekt je zde osvětlován z velmi nízkého úhlu, světlo je směrované téměř kolmo k ose objektivu. Pouze paprsky dopadající na nerovnosti na po-

vrchu objektu se pak odráží do objektivu a při takovémto osvětlení vynikne i jemná struktura na povrchu objektu, jenž je typickým použitím při kontrole povrchových vad, například gravírování a podobně. Ukázku snímku při osvětlení typu darkfield lze nalézt na Obrázku 7.

Obrázek 7: Snímek s osvětlením typu darkfield [5]

Koaxiální světlo – druh osvětlení, kde světlo dopadá na objekt rovnoběžně s osou kamery. Použití nachází při snímání reflexních objektů. Na Obrázek 8 je zachycen snímek s použitím koaxiálního osvětlení.

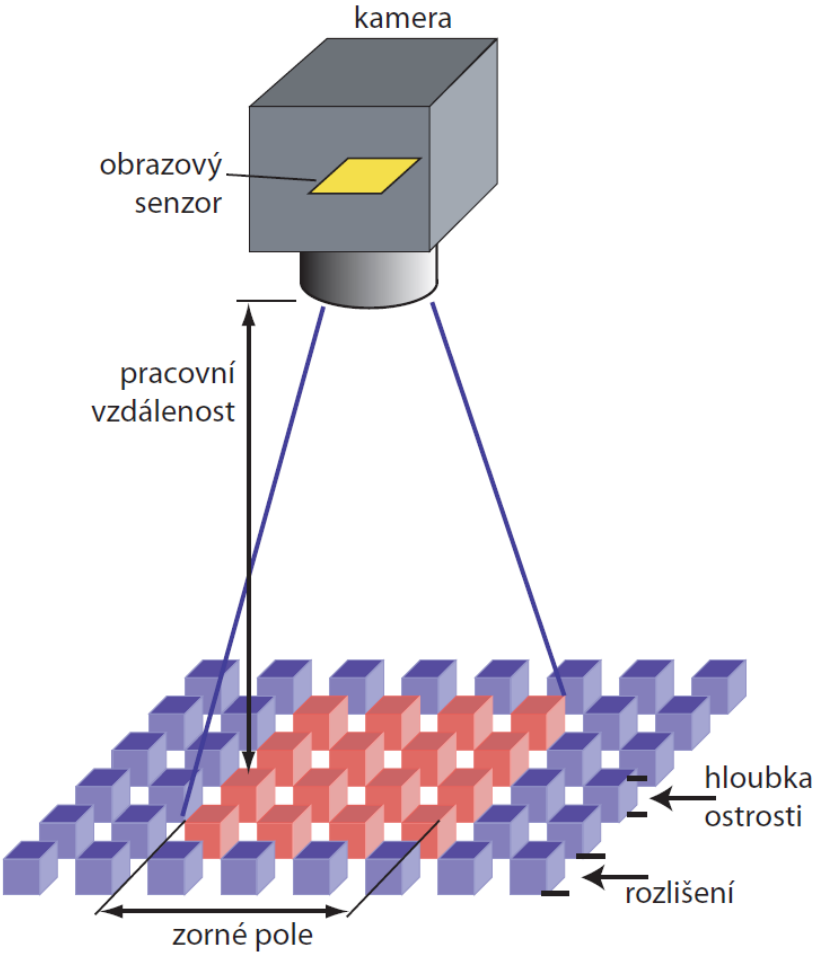
Obrázek 8: Snímek osvětlený koaxiálním osvětlením [5]

Zadní osvětlení (backlight) – světlo zde svítí zpoza objektu a obvykle je využíváno v měřících aplikacích, kde poskytuje maximální kontrast mezi objektem a pozadím. Snímek, kterého může být dosaženo s takovýmto osvětlením lze pozorovat na Obrázku 9.

Obrázek 9: Snímek se zadním osvětlením – obrys lahve [5]

1.3 Optická soustava

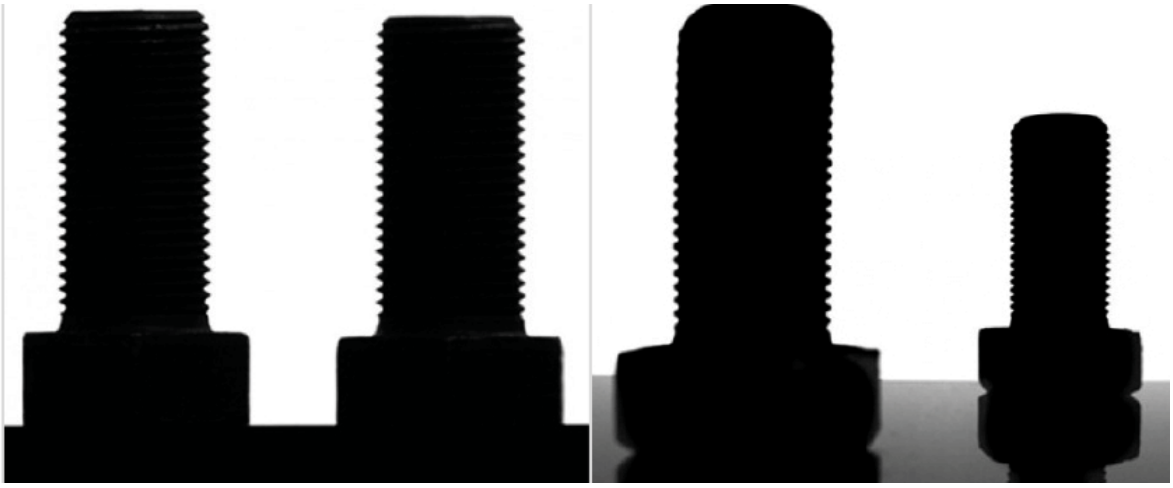
Optickou soustavou je zde myšlen především objektiv, jehož účelem je vytvořit na snímači kamery dvojrozměrný obraz třírozměrné skutečnosti, který obsahuje informaci vyhodnotitelnou pro strojové vidění [1]. Vhodná volba objektivu je naprosto stěžejní při návrhu aplikace strojového vidění. Volba objektivu vychází ze základních parametrů, jakož jsou pracovní vzdálenost, zorné pole, rozlišení a hloubka ostrosti. Zmíněné parametry jsou znázorněny na Obrázku 10.



Obrázek 10: Parametry pro volbu objektivu [1]

Dalším požadavkem při návrhu objektivu může být potlačení perspektivního zkreslení, především v precizních měřících aplikacích je perspektivní zkreslení velmi nežádoucí, proto jej kompenzují telecentrické objektivy, které propouští pouze paprsky paralelní s osou objektivu. Takovéto objektivy nemají perspektivní zkreslení a velikost objektu v obraze nezávisí na jeho fyzické vzdálenosti od objektivu. Charakteristickou vlastností telecentrického objektivu je, že průměr jeho vstupní čočky je stejný jako úhlopříčka zorného pole, z čehož vyplývá, že pro zachycení velkých objektů extrémně roste potřebná velikost objektivu, což je často zásadní limitující faktor při návrhu sestavy strojového vidění [5].

Porovnání snímku pořízeného s použitím klasického a telecentrického objektivu můžeme vidět na Obrázku 11.



Obrázek 11: Porovnání snímků pořízených telecentrickým (vlevo) a endocentrickým (vpravo) objektivem [5]

1.4 Kamera

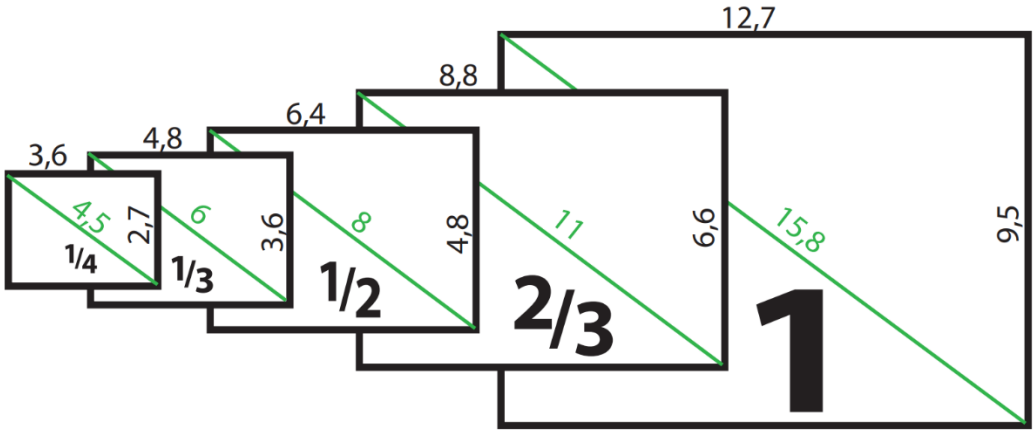
V případě strojového vidění se kamerou rozumí snímač obrazu, který dvojrozměrný obraz, promítaný objektivem na světlocitlivou plochu snímače převádí na vhodné měronosné veličiny, v našem případě nejčastěji snímek v digitální podobě. V následujícím textu jsou popsány základní parametry kamery, které mají největší vliv na její chování.

Technologie výroby – naprostá většina snímačů je vytvořena technologií CMOS (Complmentary Metal-Oxide-Semiconductor) nebo CCD (Charged-Coupled device). CMOS technologie využívá jako světlocitlivý prvek MOS tranzistory uspořádané do sloupců a řádků, kdy je lze takto přímo adresovat coby pixely v obraze. CCD využívá pro snímání světla obdobně uspořádané Schottkyho diody. Elektrický náboj uchovaný v diodách je pak vyveden a elektrický náboj je převeden na napětí [1].

Snímače CMOS ve srovnání s CCD nevyžadují složitou elektroniku potřebnou pro integraci do kamery a jejich výroba je ekonomicky výhodnější, ale naproti tomu mají CCD snímače lepší citlivost na světlo a menší šum, z čehož vyplývá, že pro použití se lépe hodí technologie CMOS, jenž se v současné době používá v naprosté většině kamer pro strojové vidění v průmyslové automatizaci.

Velikost a rozlišení senzoru – velikost senzoru na čipu je udávána v palcích, porovnání standardních velikostí senzorů ukazuje Obrázek 12. Tato velikost nevyjadřuje však skutečný rozměr snímače, ale jde o průměr skleněné trubice odpovídající ekvivalentní historické

snímací elektronce. Plocha snímače obsahuje matici polovodičových buněk citlivých na světlo. S ohledem na dodržení vzorkovacího teorému je rozlišovací schopnost kamery dána dvojnásobkem velikosti jedné buňky [5].



Obrázek 12:Standardní velikosti senzoru [1]

Funkce a periferie kamery – kamera splňující standardy pro využití v průmyslové automatizaci musí umožňovat nastavení kamery (zisk, kontrast, dobu expozice) a také podporovat vhodný standard komunikace, případně přenosu obrazu do systému strojového vidění, kupříkladu GiGe (Gigabit Ethernet) apod.

2 METODY DETEKCE OBJEKTU POMOCÍ STROJOVÉHO VIDĚNÍ

Systémy strojového vidění mají jako hlavní úkol extrahovat z dvojrozměrné obrazové interpretace trojrozměrného objektu požadovanou informaci, která může být různého charakteru, od identifikace tvaru a barvy objektu až po měření nebo OCR (Optical Character Recognition).

Každou aplikaci strojového vidění lze rozdělit na pět kroků dle Obrázku 10 [1].

Obrázek 10: Řetězec zpracování obrazu

V průmyslové praxi převládají zejména konvenční metody strojového vidění využívající exaktní matematické metody analýzy obrazu, a tudíž jsou snadno implementovány na běžné výpočetní systémy. V poslední době se však objevují komerční řešení pro průmyslové aplikace, jež využívají hlubokého učení, avšak tento nový přístup ke strojovému vidění není plnohodnotnou náhradou konvenčních metod a například pro měřící aplikace není z principu vhodný. Naopak v aplikacích, u nichž problém není jasně definován, kupříkladu při kontrole povrchových vad výrobků, přináší hluboké učení nástroje, které u exaktních metod, vzhledem k jejich striktně matematickému principu funkce, zcela chybí. Komerční řešení využívající hluboké učení (MVTec Halcon, Cognex ViDi) jsou již běžně nasazována v průmyslové automatizaci. V rámci této práce je provedeno srovnání námi vytvořených

aplikací umělých neuronových sítí. S komerčním systémem Cognex reprezentovaným softwarem Cognex ViDi Suite.

Zmiňované konvenční metody přístupu ke strojovému vidění budou dále detailněji rozebrány.

2.1 Předzpracování obrazu

Během předzpracování se s obrazem manipuluje takovým způsobem, aby informace v něm obsažená byla využitelná pro následné zpracování vybraným algoritmem, případně aby tato informace byla zdůrazněna a zároveň zpravidla nastává ztráta části jiné informace.

Na obraz se při jeho zpracování nahlíží jako na matici $n \times m$, kde n je počet řádků a m vyjadřuje počet sloupců. Každý prvek z matice poté nazýváme pixelem. Jas pixelu vyjadřujeme pomocí skalárního čísla pro šedotónový obraz, pro barevný obraz se pak každý pixel skládá zpravidla ze tří složek, jejichž význam se liší v závislosti na barevném formátu.

Následující kapitola zahrnuje úvod do problematiky manipulace s obrazem a předkládá několik příkladů jednoduchých operací.

2.1.1 Geometrické transformace

Skupina metod geometrické transformace zahrnuje postupy pro přepočet souřadnic pixelů obrazu ze stávajících pozic na jiné, což může být v některých případech výhodné. Typickým příkladem geometrické transformace je korekce zkreslení, které do systému zanesla nedokonalost a dispozice optické soustavy. Zkreslení jsou zejména v měřících aplikacích důsledně odstraňována pomocí telecentrických objektivů, pokud však zkreslení není velké, nebo pouze zbytkové, je možné jeho vliv korigovat programově, pomocí přepočtu systému souřadnic a bodů v obraze.

2.1.2 Segmentace obrazu

Hlavní účel této operace představuje rozdělení obrazu na části, přičemž je nejčastěji využívána pro separaci na objekty a pozadí. Mezi nejjednodušší metody segmentace patří prahování, při němž jsou všechny pixely obrazu porovnávány se zadanou prahovou hodnotou.

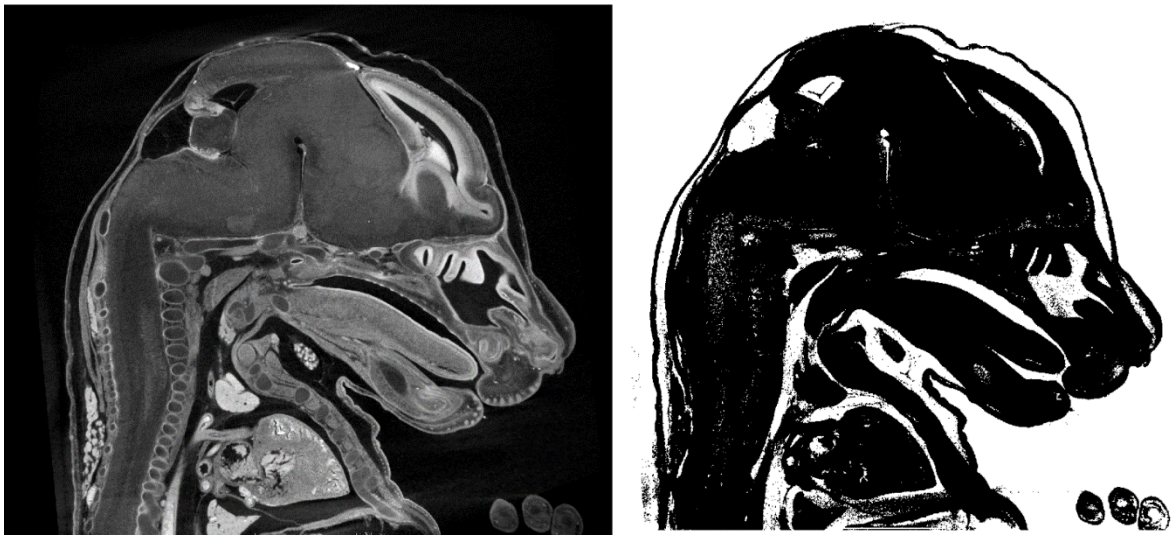
Výsledkem je poté obraz, ve kterém se vyskytuje pouze zcela černá a zcela bílá barva, tzv. binární obraz. Při vhodně zvolené prahové hodnotě jsou pak pixely objektu a pixely pozadí zobrazeny odlišnými stupni jasu [6].

Operaci prahování lze zapsat jako:

$$f(i) = \begin{cases} 0 & i \leq T \\ 1 & i > T \end{cases}$$

kde i je hodnota jasu pixelu původního obrazu, $f(i)$ je výsledná hodnota jasu pixelu obrazu po segmentaci a T je rozhodovací hodnota jasu.

Příklad segmentace obrazu prahováním lze nalézt na Obrázku 13.



Obrázek 13: Příklad použití prahování pro segmentaci obrazu [7]

2.1.3 Konvoluční filtrace

Konvoluce je jednou z nejzákladnějších operací v oblasti zpracování obrazu. Jedná se o operaci nad dvěma signály libovolné dimenze. V tomto případě je na zpracovávaný monochromatický snímek nahlíženo jako na dvourozměrný diskrétní signál, konvoluce pak bude vyjádřena vztahem: [6]

$$(f * g)(x, y) = \sum_{i=-n}^{n} \sum_{j=-m}^{m} f(x - i, y - j) g(i, j)$$

Kde n je polovina šířky konvolučního jádra, m je polovina výšky konvolučního jádra, (x, y) jsou souřadnice bodu v originálním obrazu nad kterým je prováděna konvoluce a (i, j) souřadnice bodu v konvoluční masce.

Funkce f je při operaci filtrace zpracovávaný obraz a konvoluční jádro g má funkci aplikovaného filtru. Je-li součet hodnot v konvoluční masce roven jedné, říkáme, že konvoluční jádro je normalizované. V takovém případě pak totiž aplikace filtru nemá vliv na celkovou hodnotu jasu obrazu. Výpočet hodnot pixelů takto zpracovaného obrazu lze vidět na Obrázku 22.

Příklad konvoluční filtrace obrazu při aplikaci vyhlazovacího filtru, který je používán pro odstranění šumu z obrazu má konvoluční matici g ve tvaru:

$$g = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

Výsledek po konvoluci snímku s takovouto maticí zobrazen na Obrázku 14.

Obrázek 14: Příklad filtrace obrazu – rozostření [5]

2.2 Konvenční metody strojového vidění

Konvenční metody vycházejí z analýzy obrazové informace pomocí algoritmů o matematickém základu, ve kterých se na obraz pohlíží jako na uspořádaný soubor pixelů, jenž je možné zkoumat metodami globální charakteristiky jako statistický soubor. Tímto dostáváme obecné informace o obraze, nicméně takovýmto způsobem není využita většina informace nesené v obrazu, proto abychom získali celistvou charakterizaci informace je nezbytně nutno zohlednit i vzájemné polohy pixelů.

Lokální charakteristiky zahrnují metody analyzující omezený region a jsou zpravidla výpočetně náročnější a komplexnější a z něj obdržené parametry nenesou informaci o celkových vlastnostech obrazu.

Globální a lokální charakteristiky však nemusí mít přesně vyhraněné použití při řešení komplexních úloh. Pojmem charakteristika se v tomto případě rozumí údaj, který nese určitou informaci o vlastnostech obrazu a může jím být numerický skalár, případně matice [8].

2.2.1 Jas

Charakteristika udává střední jasovou hodnotu pixelů obsažených v obraze, případně vybraném regionu. Zpravidla se používá aritmetický průměr jasových hodnot pixelů, nicméně v indikovaných případech může být vhodný také medián jasu. Obdrženou hodnotu jasu poté vyjadřujeme následujícím vzorcem:

$$L = \frac{1}{n \cdot m} \cdot \sum_{i=0}^{n} \left(\sum_{j=0}^{m} (l_{nm}) \right)$$

kde L je výsledná průměrná hodnota jasu, l_{nm} představuje hodnotu jasu pixelu na n-tém řádku a m-tém sloupci, tudíž n je počet řádků a m počet sloupců.

2.2.2 Kontrast

Kontrast vystihuje vlastnost obrazu, která vyjadřuje vzdálenost extrémních hodnot jasu od jeho typické hodnoty. Kontrast mezi dvěma pixely šedotónového obrazu lze vyjádřit jako:

$$k = \left|\frac{I_a - I_b}{I_a + I_b}\right|$$

kde k je výsledný kontrast, I_a je hodnota jasu bodu A a I_b je hodnota jasu bodu B.

V kontextu barevného obrazu se kontrast často vyjadřuje jako barevná sytost.

2.2.3 Histogram

Histogram lze chápat jako komplexní globální charakteristiku obrazu, která podává informaci o relativní četnosti všech úrovní jasu obsažených v obrazu, jež nese podobu vektoru, jehož délka odpovídá počtu zobrazitelných stupňů šedi a tím pádem pro šedotónový obraz s hloubkou 8 bitů je délka vektoru histogramu $2^8 = 256$. Tento vektor bývá zpravidla zobrazen ve sloupcovém grafu, kde na vodorovné ose jsou vyneseny hodnoty jasu jednotlivých pixelů a výška sloupce odpovídá relativní četnosti pixelů o daném jasu. Příklad histogramu lze nalézt na Obrázek 1515.

Obrázek 15: Histogram obrazu [9]

V případě barevného obrazu lze obdobným způsobem vyjádřit a zobrazit histogram pro jednotlivé barevné složky.

Histogram představuje jeden z nejvíce používaných prostředků pro hodnocení kvality fotografie a při správné analýze poskytuje obrovské množství informací. Histogram, jenž je poměrně úzký, značí málo kontrastní obraz, a naopak široký histogram značí výrazně kontrastní obraz. Dle rozložení histogramu a pozice maxima lze poté získat představu o celkovém jasu snímku [9].

2.2.4 Momentová metoda identifikace

Algoritmus momentové metody řeší úlohu identifikace podle naučeného vzoru ve formě momentových příznaků objektu, které vyjadřují změnu jasu od středu vzoru. Vyjádření pomocí momentových příznaků má své výhody především vzhledem k invariancí vůči natočení, změně velikosti a posunutí identifikovaného objektu [8].

Obecně se moment řádu $p + q$ funkce f(x,y) definuje vztahem:

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y) dx dy$$

kde x a y jsou souřadnice bodu v obraze.

Takto vytvořený vzor sám o sobě není invariantní na translaci, rotaci ani změnu měřítka, čehož se dosahuje až následnými transformacemi detekovaných momentů, například normalizací, transformací momentů do těžiště vzoru a dalších.

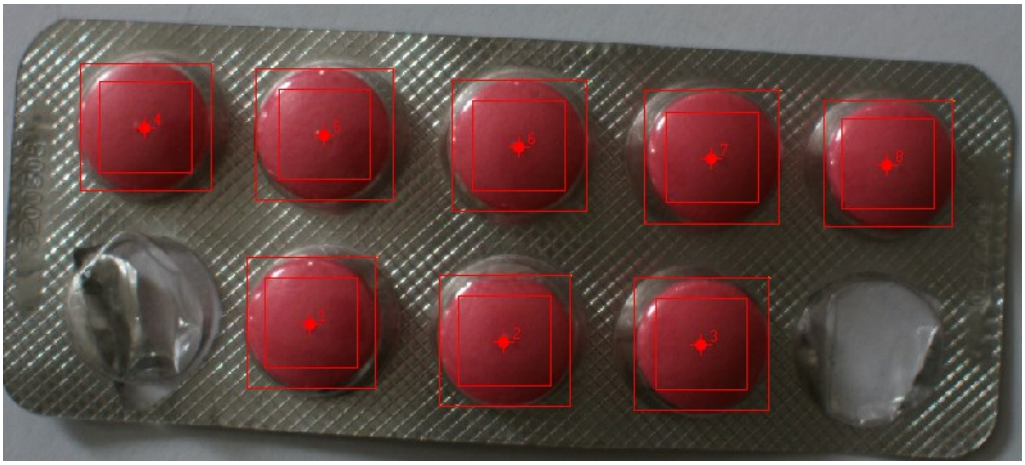
2.2.5 Konvenční metody detekce objektů

Prostá detekce přítomnosti objektů v obrazu je nejčastější aplikace strojové ho vidění v průmyslové automatizaci. Vzhledem k této skutečnosti, je problematika dobře zpracována a je vyvinuto množství nástrojů pro detekci objektů v obrazu, které se vzájemně liší jak algoritmem identifikace, tak vhodností a výpočetní náročností ve specifických případech použití.

Společnou vlastností těchto metod, je popsání hledaného prvku vhodným způsobem (podle nalezených hran, jasu apod.) a následné nalezení podobných rysů v obrazu. V následujícím textu budou popsány základní principy nejčastěji používaných metod pro detekci objektů v obrazu.

2.2.5.1 Popis obecnými příznaky

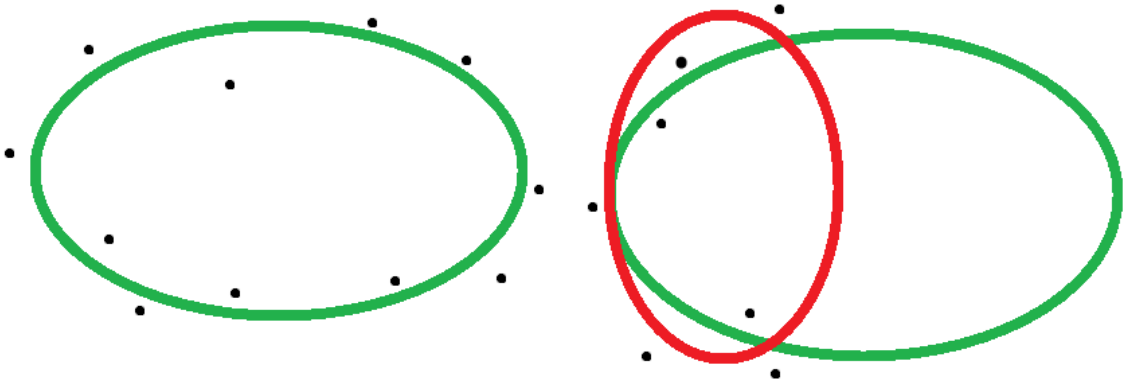
Uvedený soubor metod je nejjednodušší verzí identifikace objektu v obraze. Objekt je popsán například jeho barvou a umístěním. Tento přístup k identifikaci objektů lze nalézt například v kamerových senzorech a jednoduchých aplikacích strojového vidění. Obrázku 16 uvádí příklad aplikace pro hledání tablet léků.



Obrázek 16: Identifikace barvou objektu [10]

2.2.5.2 Geometrické příznaky

Na hranách objektu a pozadí dochází ke změně hodnoty jasu, tímto způsobem lze nalézt jednotlivé hrany objektů, nicméně pro detekci objektů různých tvarů je zapotřebí komplexnější přístup, kdy do očekávané oblasti obrazu je promítnut očekávaný tvar objektu a po jeho obvodu jsou umístěny výše popsané měřící body pro detekci hran. Výsledkem detekce je množina bodů s různou vzdáleností od promítaného tvaru. Obrázek 17 obsahuje příklad, kde byly po obvodu promítané elipsy detekovány hrany v označených bodech, které lze následně vyhodnocovat, například dle kontrastu vůči pozadí, vzdálenosti od promítané elipsy a na základě tohoto vyhodnocení bude následně možné rozhodnout, zda se předmět identifikace ve tvaru elipsy na obrazu reálně nachází nebo ne.



Obrázek 17: Příklad správného proložení detekovaných hran předpokládanou konturou objektu (vlevo) a nesprávného proložení vlivem nedostatečného množství a nesprávného rozložení prokládaných bodů

Dále je na obrázku znázorněna i situace, při které pozorujeme nedostatečný kontrast předmětu vůči pozadí, případně špatně nastavených parametrech algoritmu detekce hran. Velká část měřících bodů detekce hran selhala a tím pádem algoritmus rozhodl o přítomnosti předmětu tvaru zelené elipsy, i když byla analyzována fotografie s předmětem tvaru červené elipsy. Algoritmus tedy zaměnil identifikovaný objekt za jiný.

2.3 Metody detekce objektů využívající umělé inteligence

V minulosti bylo vyvinuto velké množství procesů využívajících umělou inteligenci, jenž jsou rozdílné v podobě vstupních dat i způsobem práce s nimi. Obecně lze umělou inteligenci nasadit na řešení rozmanitých problémů. Výběr použití konkrétní metody UI pak záleží na povaze řešeného problému. Metody využívající hluboké učení, použité v této práci budou dále podrobně popsány v následujících kapitolách.

2.3.1 Strojové učení

Obecný proces, při kterém se využívá identifikace vzorů v datech k vytvoření modelu, který pak může sloužit například k identifikaci jevů v datech nebo predikci dat budoucích. Velmi podstatná je zde velikost souboru vstupních dat použitých k vytvoření modelu a jejich rozdělení do definovaných tříd.

Strojové učení zahrnuje několik metod, do nichž patří také neuronové sítě, použité pro detekci objektů v obraze v této práci.

2.3.2 Umělé neuronové sítě

Název metody vznikl již v polovině 20. století, kdy Warren McCulloch a Walter Pitts při snaze o vytvoření umělého modelu mozku definovali matematický model neuronu pracující v diskrétní množině {-1,0,1}. Pozorováním podmíněných reflexů pak Donald Hebb navrhl pravidla pro učení synapsí neuronů, která se staly základem pro učící algoritmy neuronových sítí. V roce 1957 Frank Rosenblatt zobecnil McCullochův a Pitssův model neuronu pro reálný číselný obor parametrů a vytvořil tak perceptron včetně učícího algoritmu. V roce 1959 vyvinul Bernard Widrow společně se svými studenty model neuronu nazvaný ADELINE a jejich spojením vzniká umělá neuronová síť MADELINE [11].

Z důvodu technických nedostatků nebyla na neuronové sítě soustředěna pozornost. Průlom přišel v 80. letech, kdy byla ustavena podoba vícevrstvé neuronové sítě se zpětným šířením chyby, jež odstranila dosavadní nedostatky bránící v použití pro řešení reálných aplikací.

Tato podoba neuronových sítí je používána dodnes a umělé neuronové sítě nacházejí stále širší uplatnění. Aktuální trend vývoje výpočetní techniky navíc vede k nasazování vícejádrových a vícevláknových procesorů, jenž umožňují paralelní zpracování výpočtů, které je umělým neuronovým sítím z principu vlastní. Typickým příkladem může být právě Nvidia Jetson Nano, disponující velkým množstvím jader, jenž díky paralelnímu zpracování umožňují dosáhnout zrychlení výpočtů i rozsáhlejších sítí při zachování nízkých nákladů v porovnání s distribuovanými paralelními výpočetními systémy.

Jak je patrné již z názvu, metoda se snaží se napodobit lidskou neuronovou síť s rozhodováním podobným lidskému, díky čemuž využití nachází v úlohách klasifikace a třídění. Vzhledem ke skutečnosti, že si tato práce klade za cíl praktickou realizaci neuronové sítě pro analýzu obrazu, budou neuronové sítě dále podrobněji popsány v tomto kontextu.

Na rozdíl od konvenčních metod, které pracují na základě jasně popsaného algoritmu, neuronové sítě využívají algoritmus naučený v adaptační fázi, díky kterému je možné řešit úlohy, jejichž matematická definice by byla příliš komplikovaná pro exaktní metody analýzy obrazu.

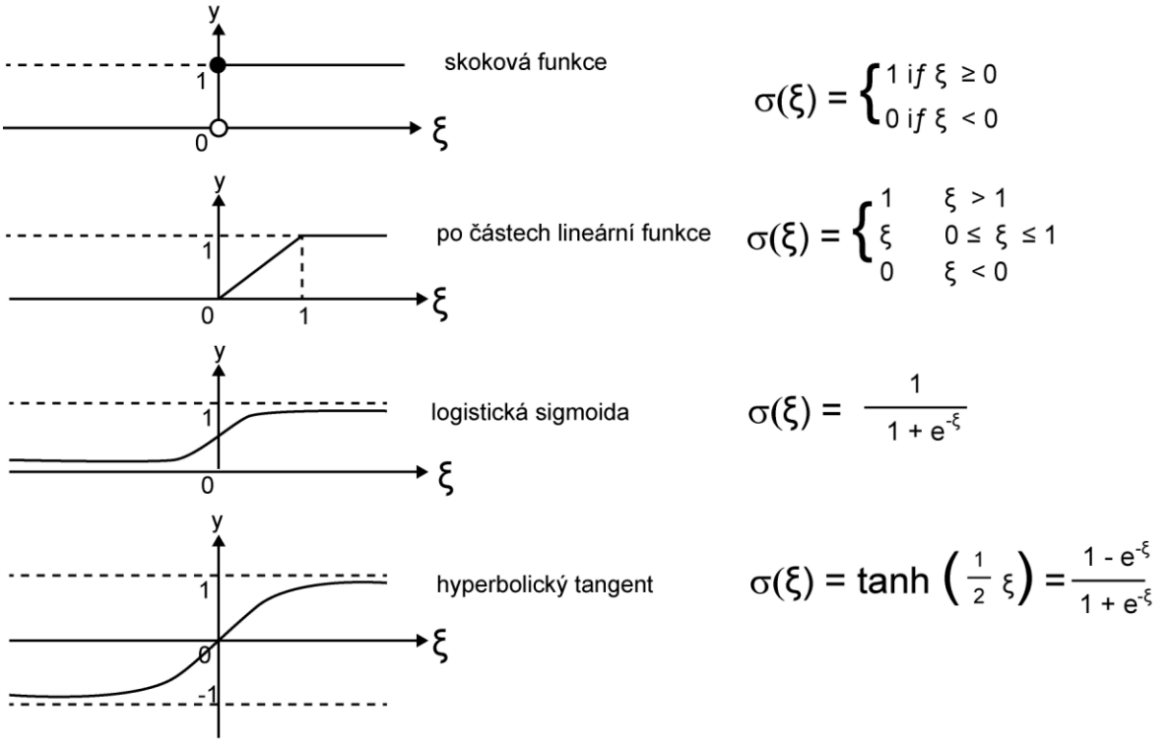
Níže je na Obrázku 18 zobrazeno obecné schéma neuronu. Zde x_1 $až$ x_3 představují vstupy neuronu a jsou přenášeny do neuronu s váhami w_1 $až$ w_3. Tyto vstupy jsou pak podle přenosové funkce neuronu převedeny na výstup y, matematicky lze vyjádřit takovýto přenos následovně: [12]

$$f(a) = TF\left(\sum_{i=1}^{n} x_i w_i + b w_b\right)$$

kde TF je aktivační funkce (transfer function) neuronu, x_i je vstup do neuronu, w_i je váha na vstupu, i je pořadové číslo vstupu, n počet vstupů neuronu, b je prahová hodnota neuronu a w_b je váha této prahové hodnoty.

Obrázek 18: Schéma neuronu [11]

Aktivační funkce neuronu je ve své podstatě přenosová funkce, přiřazující výstupní hodnotu na základě sečtených vážených vstupů. Průběh této funkce se liší dle požadovaného chování neuronu. Na Obrázku 19 jsou uvedeny příklady jednoduchých přenosových funkcí.



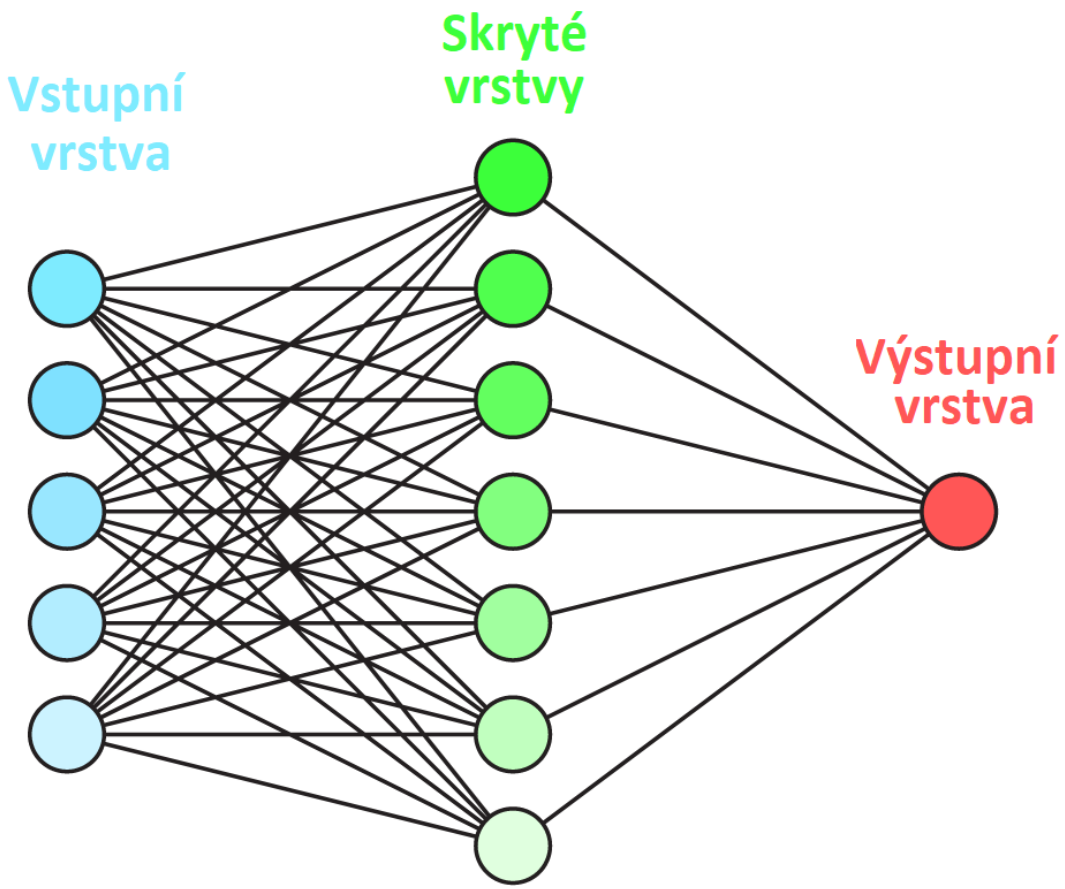
Obrázek 19: Příklady aktivačních funkcí neuronu [13]

Neuronové sítě se skládají z vrstev neuronů, kde výstupy z jedné vrstvy jsou vstupy do následující, viz. Obrázek 20. Neuronová síť pracuje ve dvou fázích – adaptační a aktivní.

V adaptační fázi se síť učí, adaptuje se na řešení daného problému. Toto učení je realizováno přenastavením vah mezi neurony. Učení může probíhat s učitelem nebo bez učitele, přičemž při učení s učitelem jsou nastavovány váhy tak, že je snižován rozdíl mezi žádaným a aktuálním výstupem

V aktivní fázi neuronová síť používá již nastavené váhy ke zpracování informace a poskytnutí výstupní informace.

Podstatnou vlastností při realizaci neuronových sítí je, že se informace neuronovou sítí šíří a je jí zpracovávána zároveň všemi neurony. I když lze tyto výpočty realizovat pomocí běžného procesoru, používají se běžné počítače především pro simulaci neuronové sítě při odladění, pro reálné nasazení jsou běžně využívány grafické karty (dále pouze GPU), které umožňují paralelní zpracování. Ve velkém měřítku jsou používány grafické procesory s velkým množstvím jader, jako v případě Nvidia Jetson Nano, kterému je věnována praktická část této práce.



Obrázek 20: Schéma neuronové sítě [14]

2.3.3 Konvoluční neuronové sítě

Postupy uvedenými v předchozích kapitolách lze učit neuronovou síť dle konkrétních trénovacích snímků. V těchto neuronových sítích jsou však jednotlivé neurony učeny izolovaně od ostatních a síť je tak ve výsledku velmi náchylná k variantnosti vstupního obrazu, například při změně jasu, natočení, zkosení apod. Abychom byli schopni naučit neuronovou síť výše uvedenými algoritmy, které jsou funkční, byla by její hloubka neúměrně velká (desítky vrstev) a také počet neuronů by musel řádově odpovídat počtu pixelů v obrazu, což by kladlo neadekvátní nároky na výpočetní prostředky a čas učení neuronové sítě [15].

Konvoluční neuronové sítě přináší vylepšení architektury, ve které jsou do sítě zařazeny speciální vrstvy, které množství parametrů neuronové sítě velmi významně snižují. Tento přístup se používá primárně při práci s obrazovými nebo zvukovými daty.

2.3.3.1 Architektura konvoluční neuronové sítě

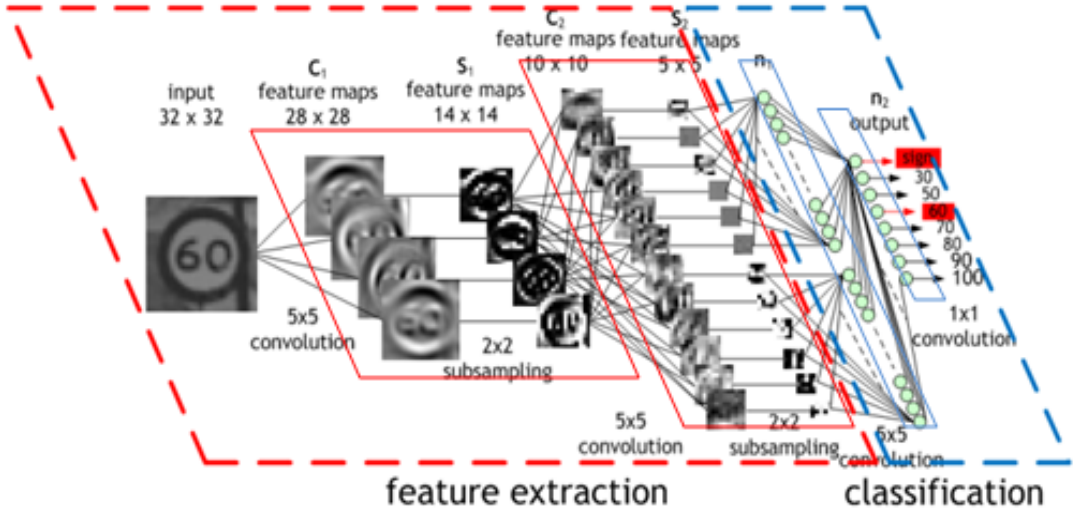
Oproti původní výše uvedené podobě neuronové sítě obsahuje konvoluční neuronová síť minimálně dvě speciální vrstvy neuronů. Jedná se o konvoluční vrstvy a subsamplingové vrstvy (někdy také označeny jako pooling layers). Úkolem těchto vrstev je získat z obrázku sadu koeficientů, které se posléze předají do několika klasických vrstev neuronů. Neurony v následných vrstvách jsou pak propojeny pouze s menší částí předchozí vrstvy místo toho, aby byly plně propojeny. Počet koeficientů vstupujících do posledních vrstev se může pohybovat od několika desítek až po stovky tisíc, dle velikosti vstupních dat (rozlišení snímku, …). Na konci konvoluční neuronové sítě je plný obraz redukován do jednoho tenzoru, který zastupuje skóre jednotlivých tříd [15].

Struktura neuronové sítě je pak zpravidla následující:

1. Vstupní vrstva
2. Konvoluční vrstva 1
3. Subsamplingová vrstva 1
4. Konvoluční vrstva 2
5. Subsamplingová vrstva 2
6. …
7. …
8. Skrytá vrstva neuronové sítě

9. Výstupní vrstva

Tuto strukturu znázorňuje i Obrázek 21.



Obrázek 21: Struktura konvoluční neuronové sítě [15]

Konvoluční neuronovou síť je tedy možné považovat za dvě sítě, přičemž první se skládá z konvolučních a subsamplingových vrstev, které z původního obrazu generují jednorozměrný tenzor koeficientů a druhá síť je již známá neuronová síť, jenž pak z těchto koeficientů určuje objekty nalezené v obrazu. Jednotlivé vrstvy budou podrobněji rozebrány dále.

2.3.3.2 Vstupní vrstva

Surový obraz vstupuje do této vrstvy, konkrétně hodnoty jeho pixelů. Vstupní vrstva tedy bude mít rozměr dle rozlišení obrazu a hloubku podle počtu barevných kanálů vstupního obrazu.

2.3.3.3 Konvoluční vrstva

Konvoluční vrstva aplikuje na vstupní obraz masku o zadané velikosti a vytváří tak lokální charakteristiku obrazu. Jednotlivé oblasti se překrývají, takže počet oblastí výsledného snímku lze vyjádřit dle vzorce:

$$m = (x - k + 1) \times (y - l + 1)$$

kde m je počet oblastí výsledného obrazu, x a y rozměry původního obrazu, k a l jsou rozměry konvoluční masky [15].

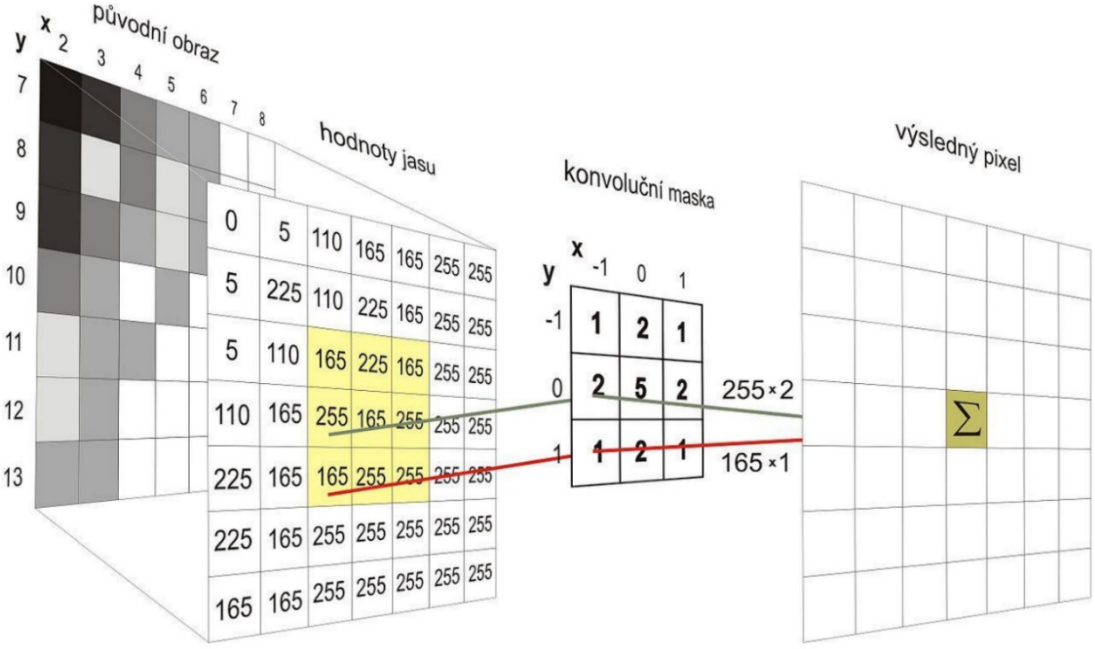
Z výše uvedeného vztahu vyplývá, že při použití konvoluční masky o velikosti větší než 1, bude výsledný obraz menší než původní. V případě, kdy je třeba obsáhnout konvolucí celý obraz, je používána funkce zero-padding, která na hrany obrazu vkládá nulové pixely, aby konvoluce mohla být korektně provedena pro všechny pixely obrazu.

Samotná operace konvoluce je pak vyjádřena vztahem obdobným jako v kapitole 2.1.3 Konvoluční filtrace:

$$(f * g)(x, y) = \sum_{i=-n}^{n} \sum_{j=-m}^{m} f(x - i, y - j) g(i, j)$$

Kde n je polovina šířky konvolučního jádra, m je polovina výšky konvolučního jádra, (x, y) jsou souřadnice bodu v originálním obrazu nad kterým je prováděna konvoluce a (i, j) souřadnice bodu v konvoluční masce.

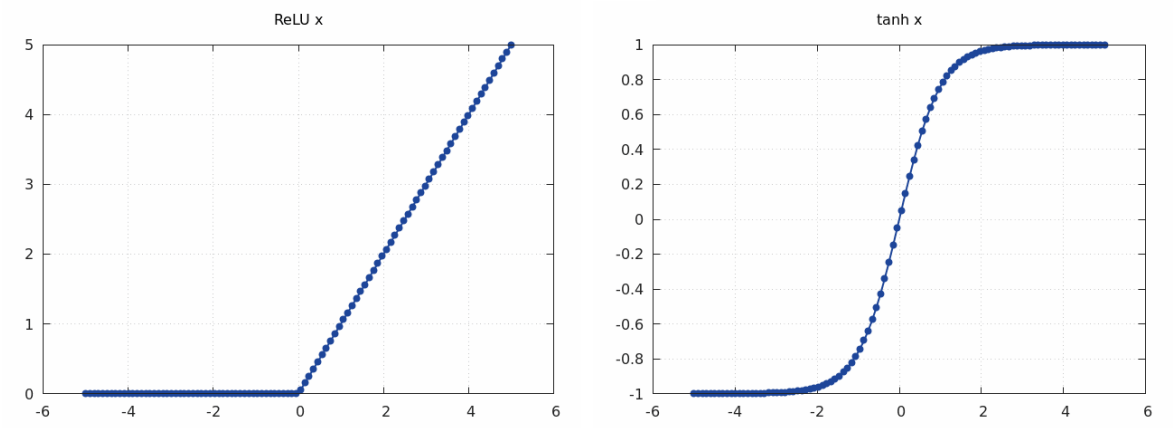
Princip konvoluce je vysvětlen také na následujícím Obrázku 22.



Obrázek 22: Konvoluce obrazových dat [15]

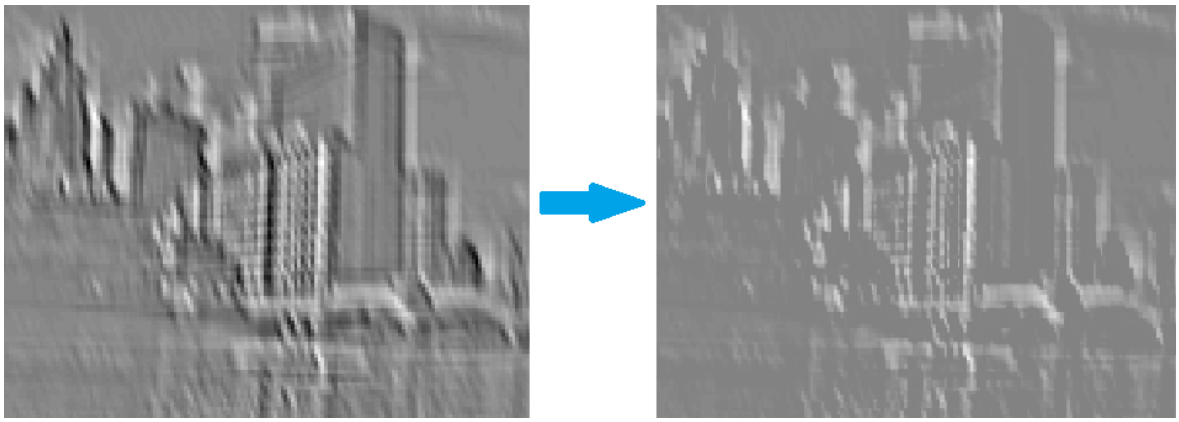
2.3.3.4 Aktivační vrstva

Aktivační vrstva aplikuje aktivační funkci na vstupní data a jejím účelem je manipulace s obrazem, jako je například zanesení nelinearit nebo segmentace objektů. Rozměr dat a jejich hloubka je přitom ponechána beze změny. Často používané jsou například funkce ReLU (Rectified Linear Unit) sigmoida a hyperbolický tangent, vykreslené na Obrázku 23.



Obrázek 23: Příklady aktivačních funkcí

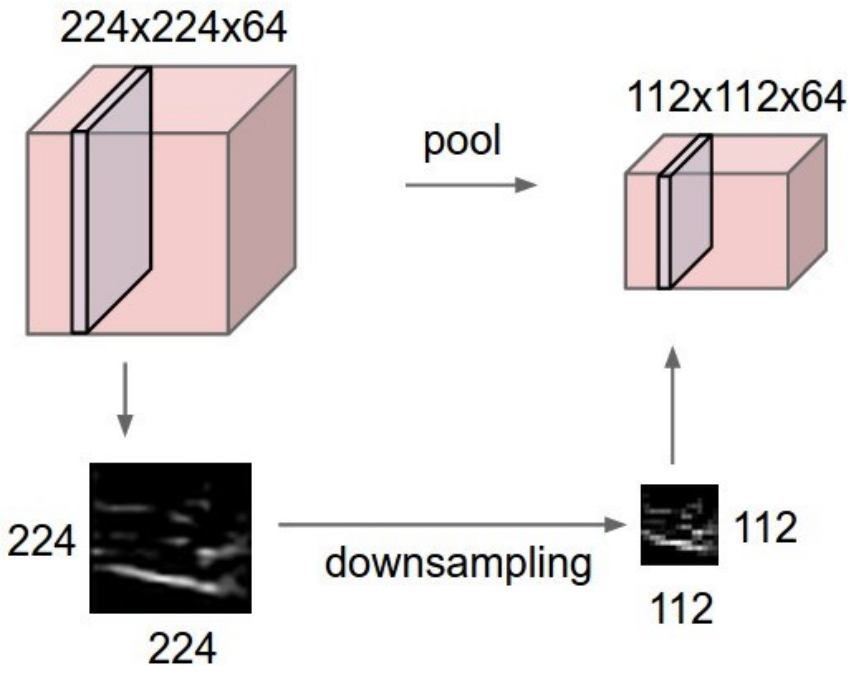
Na Obrázku 24 je možné sledovat, že provedením funkce ReLU jsou z původních dat odstraněny černé oblasti, zastupující záporné hodnoty. Takto upravený obraz pak umožňuje rychlejší trénování zbytku neuronové sítě [16].



Obrázek 24: Příklad snímku před a po průchodu ReLU vrstvou neuronové sítě [16]

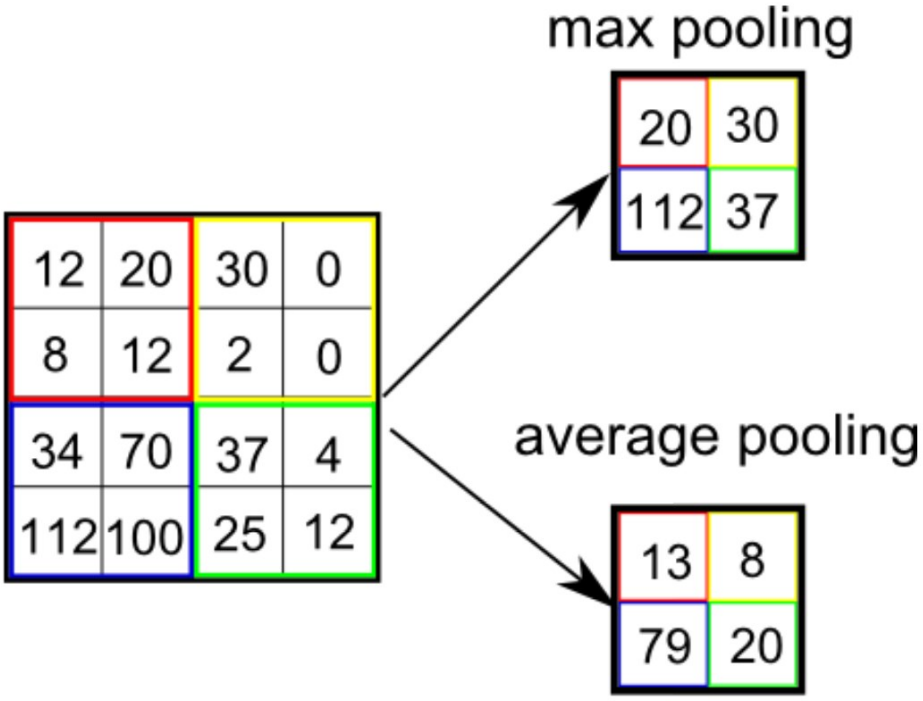
2.3.3.5 Subsamplingová vrstva

Subsampling (pooling) má za úkol snížit velikost reprezentovatelných dat a tím redukovat množství parametrů předávaných do sítě, velikost sítě i její potřebnou hloubku tak, aby bylo zachováno co nejvíce důležitých informací pro další vrstvy neuronové sítě, čímž je snížen výpočetní čas a využití paměti. Vrstva pracuje se zvolenou velikostí masky, která provádí funkci sdružování. Velikosti masky odpovídá redukce vstupních dat, kdy kupříkladu nejběžnější použití masky 2x2 s krokem 2 redukuje 75 % ze vstupních dat, viz. Obrázek 2525.



Obrázek 25: Zmenšení obrazu při poolingu [15]

Funkce poolingové masky je uzpůsobena potřebám aplikace. Nejběžnějšími maskami jsou masky vyhledávající maximum a průměr, dle Obrázek 2626.



Obrázek 26: Max pooling, Average pooling [17]

2.3.3.6 Plně propojená vrstva

Poslední, výstupní plně propojená vrstva počítá hodnocení tříd. Její rozměr odpovídá počtu klasifikovaných tříd a hodnota na výstupu každého neuronu koresponduje se skóre dané třídy. Každý neuron v této výstupní vrstvě je propojen s každým neuronem z vrstvy předchozí, kde daná spojení mají různé váhy získané při procesu učení sítě.

2.3.4 Učení neuronové sítě

Učení neuronové sítě je inspirováno učením člověka, podobně jako vznik její koncepce. Stejně jako se dítě učí od učitele, který mu poskytuje zpětnou vazbu, tak i neuronová síť dostává při učení informace o úspěšnosti své predikce či identifikace. Cílem učení neuronové sítě je optimalizace vnitřních parametrů s cílem minimalizovat chybu a probíhá zpravidla v mnoha iteracích. Typickým příkladem algoritmu pro učení neuronové sítě je algoritmus Back-propagation, jenž bude podrobněji popsán dále.

2.3.4.1 Přenesené učení

Metoda, kdy pomocí modelu jiné, už vytrénované neuronové sítě vytrénujeme naši vlastní síť, která bude například klasifikovat jiné objekty. Tento proces přeneseného učení je pak časově kratší a méně náročný na výpočetní výkon než proces trénování celé sítě znovu od začátku. K dispozici jsou kvalitní předtrénované modely např. od komunity nebo od tvůrců API TensorFlow či Caffe, kdy k trénování těchto modelů bylo využito velkých databází obrázků. Dostupné modely mají rozdílné architektury a jsou tak využitelné pro rozdílné aplikace.

Přenesené učení pak spočívá v jemném doladění již předtrénované neuronové sítě na jinou úlohu, příkladem může být převzetí některé z předních volně dostupných sítí, například ResNet, která je naučená na databázi obrázků ImageNet a přeučení poslední plně propojené vrstvy dle námi podaných obrázků.

2.3.4.2 Algoritmus Back-propagation

Metoda Back-propagation, jinými slovy také algoritmus zpětného šíření chyby je metoda pro učení vícevrstvých neuronových sítí s učitelem. Algoritmus učení probíhá v iteracích, v nichž je pozorováno, zda na vstupní vektor odpovídá neuronová síť podle trénovací množiny. V případě že síť nepodává správné požadované výsledky, jsou změněny váhové vektory a algoritmus je opakován [17].

Logické kroky algoritmu pro učení Back-propagation jsou detailněji popsány zde:

1. Inicializace

   Všechny váhy jsou nastaveny na defaultní hodnoty

2. Zpracování vzoru

   Prvek z trénovací množiny je vybrán jako vzor a je předložen pro zpracování neuronovou sítí. Vzor je postupně zpracováván neurony dle vztahu:

$$u = \sigma \cdot \left(\sum_{i=0}^{n} x_i w_i \right)$$

   kde x_i je vstup do i-tého neuronu, w_i je váha i-tého neuronu a $\sigma(\varphi) = \frac{1}{1+e^{-\varphi}}$ je aktivační funkce neuronu.

3. Srovnávání odezvy

   Výstup ze sítě je srovnáván s požadovaným výstupem a je vypočtena chyba sítě, která bude použita jako přírůstek pro všechny vzory. Chyba je vypočítána dle následujícího vztahu:

$$E = \frac{1}{2} \sum_{i=1}^{n} (u_i - d_i)^2$$

   kde n je počet neuronů ve výstupní vrstvě, u_i je skutečná odezva i-tého neuronu ve výstupní vrstvě na vstupní vektor a d_i je očekávaný výstup i-tého neuronu ve výstupní vrstvě.

   Chyba pro výstupní vrstvu je vyjádřena jako:

$$\delta_i^u = (d_i - u_i^u)(1 - u_i^u)u_i^u$$

4. Zpětné šíření chyby

   Pro všechny vrstvy, počínaje výstupní vrstvou jsou vypočteny váhy dle vztahů:

$$\Delta w_{ij}^l(t) = \eta \delta_i^l(t) u_j^{l-1}(t) + \mu \Delta w_{ij}^l(t-1)$$

$$\Delta \theta_i^l(t) = \eta \delta_i^l(t) + \mu \Delta \theta_i^l(t-1)$$

   kde η je koeficient učení a μ je koeficient vlivu změny vah z předchozího kroku.

   Tyto koeficienty nabývají hodnot <0,1> a s jejich rostoucí hodnotou roste také vliv na nové nastavení váhy. Síť se zároveň učí tím rychleji, čím je koeficient učení η větší, pokud je však koeficient příliš velký, učení se stane nestabilní a nemusí konvergovat k řešení. Výhodné je volit koeficient učení proměnný, kdy na začátku

učení je větší, síť najde přibližné hodnoty vah a následně je η snižováno, jak se vypočtené váhy zpřesňují.

Chyba se dále šíří do dalších vrstev směrem od výstupní ke vstupní podle vztahu:

$$\delta_i^{h-1} = y_i^{h-1}\left(1 - u_i^{h-1}\right)\sum_{k=1}^{n} w_{ki}^h \delta_k^h$$

A dále jsou upraveny váhy:

$$w_{ij}^l(t+1) = w_{ij}^l(t) + \Delta w_{ij}^h(t)$$

$$\theta_i^l(t+1) = \theta_i^l(t) + \Delta\theta_i^l(t)$$

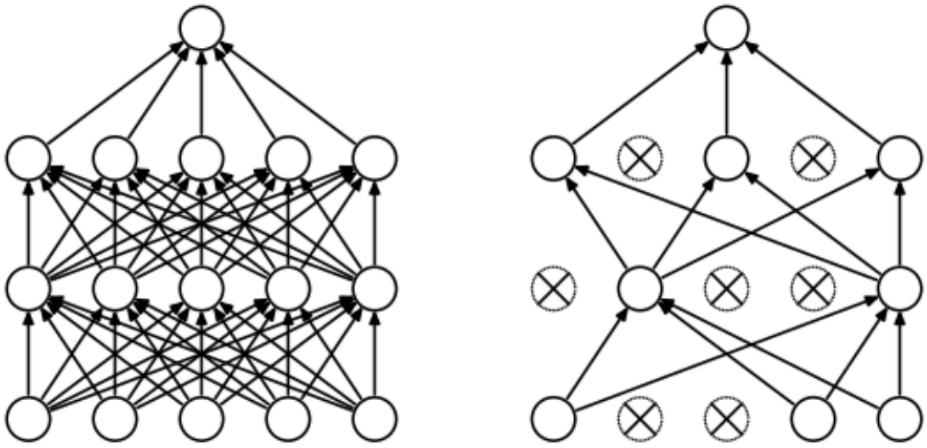
Jak již bylo uvedeno, tímto postupem jsou vypočteny nové váhy pro všechny vrstvy sítě. Pro vrstvu nejblíže vstupní vrstvě bude vztah pro výpočet pozměněn:

$$\Delta w_{ij}^l(t) = \eta\delta_i^l(t)x_j(t) + \mu\Delta w_{ij}^l(t-1)$$

5. Opakování postupu učení pro všechny prvky z trénovací množiny. Pokud je následně chyba sítě menší než povolená odchylka, učení končí.

2.3.4.3 Dropout

Operace, která slouží k tomu, aby se předešlo přeučení neuronové sítě. V každé iteraci učení je deaktivována určitá část neuronů. Tím se zajistí, že konečný výstup neuronové sítě nebude ovlivňovat pouze malá část neuronů a že ostatní neurony budou mít pouze malý vliv, ale na výsledku se budou podílet všechny neurony rovnoměrně. Přiklad operace dropout při jedné epoše učení je na Obrázku 27.



(a) Standard Neural Net (b) After applying dropout.

Obrázek 27: Dropout

2.3.4.4 Učení konvolučních neuronových sítí

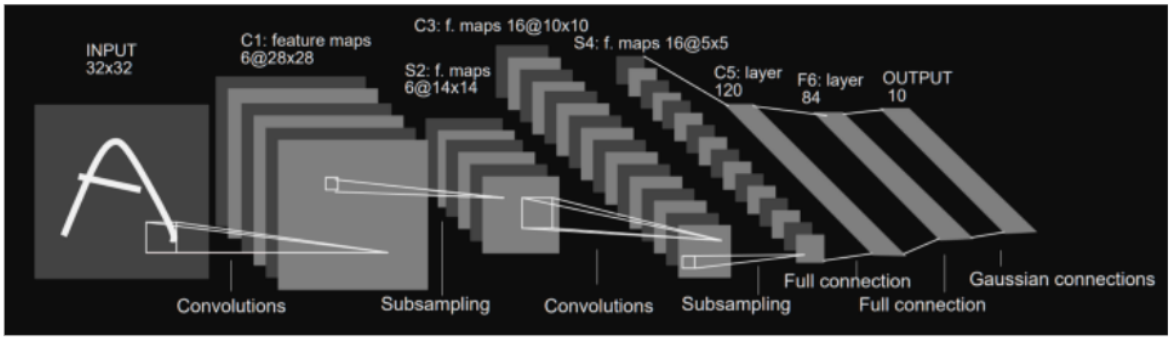
Zásadním rozdílem v procesu učení konvoluční neuronové sítě oproti klasickým neuronovým sítím je nutnost vhodného nastavení parametrů konvolučních a subsamplingových vrstev. Před zahájením učení jsou hodnoty konvoluční masky rozděleny náhodně, vhodné je například Gaussovo rozložení se směrodatnou odchylkou $\sqrt{\frac{2}{n}}$, kde n je počet výstupních neuronů. Síť při učení schopna hodnoty masky nastavit při procesu učení.

Vzhledem k navýšení složitosti oproti klasickým neuronovým sítím je i učení konvoluční neuronové sítě řádově časově náročnější. V praxi se tedy jako výchozí základ používá předučená síť, která je následně již doladěna dle požadovaných dat, příklady architektur takových předučených sítí jsou uvedeny dále [15].

LeNet

První úspěšná aplikace CNN, jenž vznikla již v roce 1989. Byla používána pro klasifikaci číselných znaků. Síť se skládá ze sedmi vrstev dle Obrázku 28. Po vstupní vrstvě následují konvoluční a subsamplingové vrstvy, transformující vstupní obraz 32x32x1 na rozměry1x1x84, poté je zařazena plně propojená skrytá vrstva a následně taktéž plně propojená výstupní vrstva, jejímž výstupem je 1x1x10, kde jednotlivé výstupy mají význam skóre pro příslušnost k jednomu z klasifikovaných znaků [18].

Původní architektura sítě LeNet o pěti vrstvách má 3246 parametrů a 139 402 propojení. Na základě této architektury byly dále vyvinuty další varianty sítě LeNet a jejich složitost narůstala. Poslední z nich, LeNet-5 má již 60000 parametrů a obsahuje 340 068 propojení.



Obrázek 28: Struktura sítě LeNet [18]

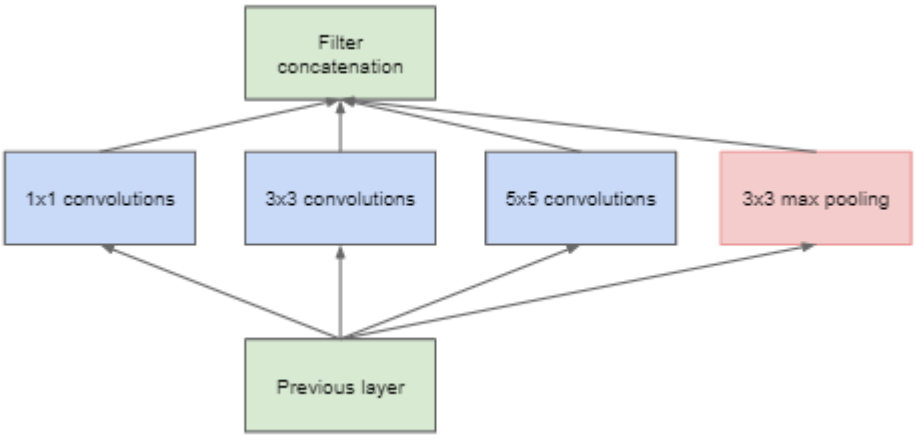
AlexNet

Architektura byla navržena pro soutěž ImageNet, jenž spočívala v tréninku sítě pro klasifikaci 1,3 milionu snímků do 1000 různých tříd. Soutěž v roce 2012 vyhrála se značným náskokem před ostatními modely neuronových sítí a značně zpopularizovala CNN v oblasti počítačového vidění.

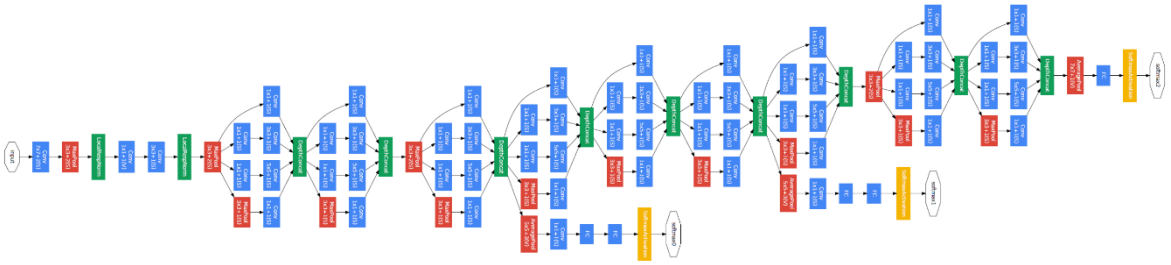
Síť je založena na LeNet, má však hlubší konvoluční vrstvu, což způsobuje i vyšší výpočetní náročnost a po vstupní vrstvě, která má oproti LeNet větší rozměr a také hloubku, následuje pět konvolučních a tři maxpoolingové vrstvy, jež následují tři plně propojené vrstvy, jejichž výstup již má rozměr 1000 a představuje příslušnost ke klasifikovaným třídám. Celkově má architektura síě AlexNet přes 62 milionů parametrů, ačkoli tedy vychází z architektury AlexNet, její složitost a výpočetní náročnost učení se značně zvýšily [19].

GoogLeNet

Síť byla vyvinuta v roce 2014 pro praktické použití na serverových farmách společnosti Google. Síť obsahuje 22 vrstev, kde jsou konvoluční vrstvy uspořádány do paralelní kombinace filtrů 1x1, 3x3 a 5x5, jak ukazuje Obrázek 29. Pro GoogLeNet je charakteristické právě využívání 1x1 konvolučních bloků, které redukují počet prvků před náročnějšími paralelními bloky, čímž je zásadně snížen počet parametrů sítě (4 miliony v porovnáním s 62 miliony u AlexNet), přitom není ztracena původní informace nesená v obrazu. Podoba architektury GoogLeNet je na Obrázku 30. V následujících letech byla síť dále vylepšována a vytvořeny varianty Inception V2, V3 a V4, které se odlišují především ve variantách po sobě jdoucích 3x3 konvolučních filtrů [16].



Obrázek 29: Schéma paralelního uspořádání bloků neuronové sítě GoogLeNet [16]

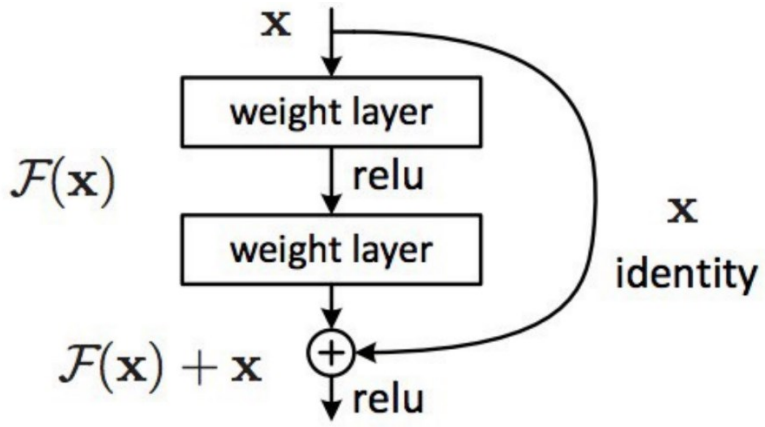


Obrázek 30: Neuronová sít GoogLeNet [16]

ResNet

Experimentálně bylo zjištěno, že i jednoduché architektury mohou být úspěšné pokud jsou dostatečně hluboké. Tato architektura vznikla v roce 2015 a je jednou z prvních, která umožnila vytvořit a natrénovat neuronovou síť se stovkami vrstev. Výsledek je po operacích ve dvou vrstvách sloučen s hodnotou před těmito vrstvami, jak ukazuje Obrázek 31. Koncept přeskočení jedné vrstvy již existoval, avšak na rozdíl od přeskočení dvou vrstev nedosahoval výrazného zlepšení. Sítě s větším počtem vrstev využívají konvoluční bloky pro redukci parametrů, podobně jako u GoogLeNet. Používá poměrně jednoduchou inicializační vrstvu následující konvolucí filtrem 7x7 a pooling vrstvou. Jakožto konečný klasifikátor využívá ResNet pooling vrstvu s vrstvou softmax.

Architektura sitě ResNet-50 má přes 23 milionů trénovatelných parametrů, dosahuje však nejmenší chyby na klasifikaci obrazů z datasetu ImageNet ze všech výše zmiňovaných architektur [20].



Obrázek 31: Koncept neuronové sítě ResNet [20]

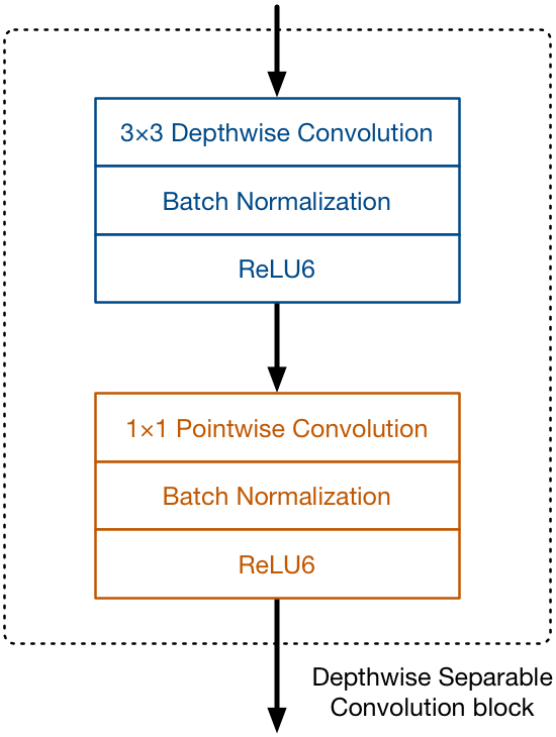
MobileNet **v1**

Klasifikační konvoluční neuronová síť zaměřená na mobilní zařízení, která disponují omezeným výpočetním výkonem (chytré telefony, tablety, raspberry pi, apod.). Pracuje s efektivnějším způsobem provádění konvoluce v konvolučních vrstvách. Tato konvoluční vrstva nese název Hloubkově oddělitelná a je rozdělena do dvou podvrstev. První podvrstvou je tzv. Hloubková konvoluce (hloubková konvoluce), následovaná druhou podvrstvou tzv. 1×1 Bodové konvoluce. Hloubková konvoluční vrstva provádí konvoluci jádra se všemi vstupy individuálně. Výstupem takové konvoluce je stejný počet matic jako byl počet vstupních kanálů. Tyto výstupní matice jsou následně vstupem pro 1×1 Bodovou konvoluční vrstvu, která slouží pro vytvoření lineární kombinace matic. Uspořádání vrstev je uvedeno na Obrázku 32. Takové rozdělení do podvrstev snižuje výpočetní náročnost [21].

Síť architektury MobileNet je již tak relativně malá, má pouze 4,2 milionu parametrů, nicméně výkon mobilních zařízení může vyžadovat další snížení výpočetní náročnosti. Toho lze dosáhnout pomocí parametru α nazývaného násobitel šířky (width multiplier) a jeho úkolem je redukovat šířku každé vrstvy v síti a tím i množství parametrů sítě. V základním modelu má hodnotu 1 a může nabývat hodnot (0,1>. Druhým parametrem je násobitel rozlišení (resolution multiplier) ρ, kterým je násobena velikost vstupního obrázku a tím dochází k redukci složitosti sítě. Důsledek změny těchto parametrů na množství parametrů sítě je znázorněn v tabulkách níže.

Násobitel šířky	Násobitel rozlišení	Přesnost	Počet mat. operací [mil.]	Počet parametrů [mil.]
1	224	70,60 %	569	4,2
0,75	224	68,40 %	325	2,6
0,5	224	63,70 %	149	1,3
0,25	224	50,60 %	41	0,5

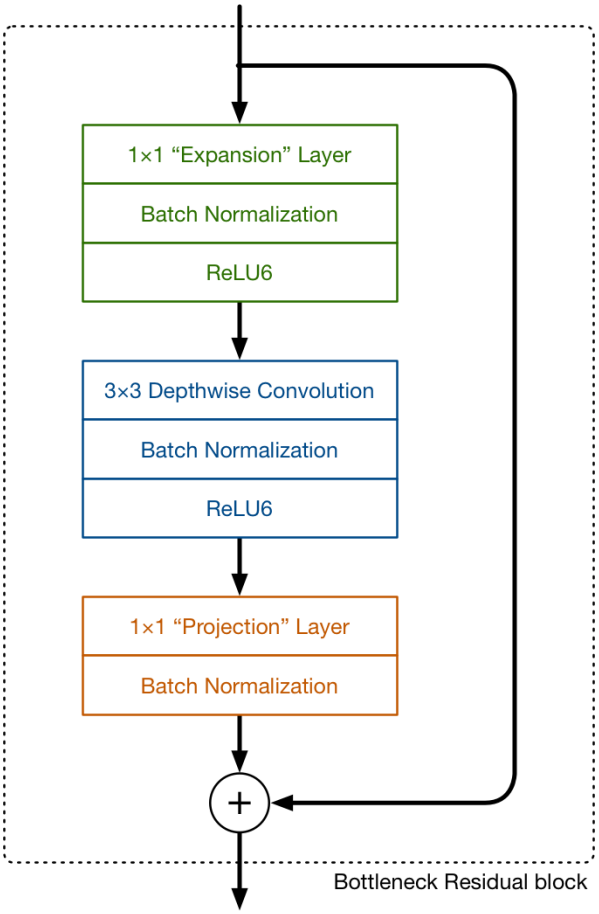
Násobitel šířky	Násobitel rozlišení	Přesnost	Počet mat. operací [mil.]	Počet parametrů [mil.]
1	224	70,60 %	569	4,2
1	196	69,10 %	418	4,2
1	160	67,20 %	290	4,2
1	128	64,40 %	186	4,2



Obrázek 32: Proces konvoluce v síti architektury MobileNet v1 [21]

MobileNet v2

Zdokonalená verze architektury využívá, podobně jako verze MobileNet v1, Hloubkově oddělitelnou konvoluční vrstvu, ovšem v upravené verzi, která u této verze nese název Bottleneck Residual block. Tato vrstva se skládá ze dvou podvrstev známých z předešlé verze architektury – Depthwise convolution a 1×1 Pointwise convolution. Navíc je zde Expansion layer (rozšiřující vrstva). V MobileNet v2 je vrstva Pointwise convolution pozměněna. Zatímco ve verzi MobileNet v1 zachovávala počet kanálů, ve verzi MobileNet v2 je tomu naopak a tato konvoluce snižuje počet kanálů. Z toho důvodu byl její název změněn na projekční – promítá data s vysokým počtem kanálů na tenzor s menším množstvím kanálů. Expansion layer, která do tohoto bloku byla přidána, má za cíl navýšit počet kanálů vstupujících do Depthwise convolution vrstvy. Scématické znázornění sítě je na Obrázku 33. V podstatě dělá rozšiřující vrstva opak toho, co dělá vrstva projekční. Míra, jakou jsou data rozšířena v rozšiřující vrstvě je dána hodnotou expansion factor (faktor rozšíření). Rozšiřující vrstva funguje jako dekomprese dat, nad nimiž je provedena hloubková konvoluce a následně jsou opět zkomprimována projekční vrstvou [22].



Obrázek 33: Proces konvoluce v síti architektury MobileNet v2 [21]

II. PRAKTICKÁ ČÁST

3 NVIDIA JETSON NANO

Následující kapitola bude detailně popisovat embedded platformu použitou v této práci, jak po stránce hardware, tak po stránce programového vybavení.

3.1 Nvidia Jetson Nano Developer Kit

Nvidia Jetson Nano je nejlevnější a nejméně výkonný zástupce ze série Nvidia Jetson. Celá série se stává ze 4 modelů: Jetson Nano, Jetson TX2 Series, Jetson Xavier NX a Jetson AGX Xavier.

Vývojový kit Nvidia Jetson Nano je jednodeskový počítač vybavený Jetson Nano modulem, který byl vyvinut pro aplikace s umělou inteligencí či jiných, jež vyžadují vysokou úroveň paralelizace. Hlavní předností tohoto počítače je grafický čip, poskytující vysoký výkon vzhledem k ceně a spotřebě elektrické energie. Podobu vývojového kitu můžeme vidět na Obrázku 34.

Výkon komponent závisí na způsobu napájení a nastaveném režimu. První možnost napájení je přes USB standardním napětím o velikosti 5V a proudem o hodnotě 2A. Druhá možnost napájení je pomocí bareljack konektoru stejným napětím a proudem, který může nabývat velikosti až 4A a je zejména vhodný pro aplikace, kdy periferie odebírají více proudu. Dále je možné vybrat ze dvou výkonových režimů, základním 10 W a sníženým 5W režimem, kdy je omezena frekvence a počet využívaných jader procesoru. V demonstrační úloze bylo zařízení v režimu 10W a napájeno 4 ampérovým konektorem.

Deska je osazena video výstupy HDMI a DP, gigabitovým ethernet konektorem, čtyřmi USB 3.0 typu A, slotem pro WiFi, PCIe, MIPI sériovým rozhraním pro kameru a 40 pinovým rozšiřujícím konektorem. Také se zde nachází slot pro mikroSD paměťovou kartu, která slouží jako trvalé paměťové úložiště.

Obrázek 34: Vývojový kit Nvidia Jetson Nano

3.1.1 Nvidia Jetson Nano modul

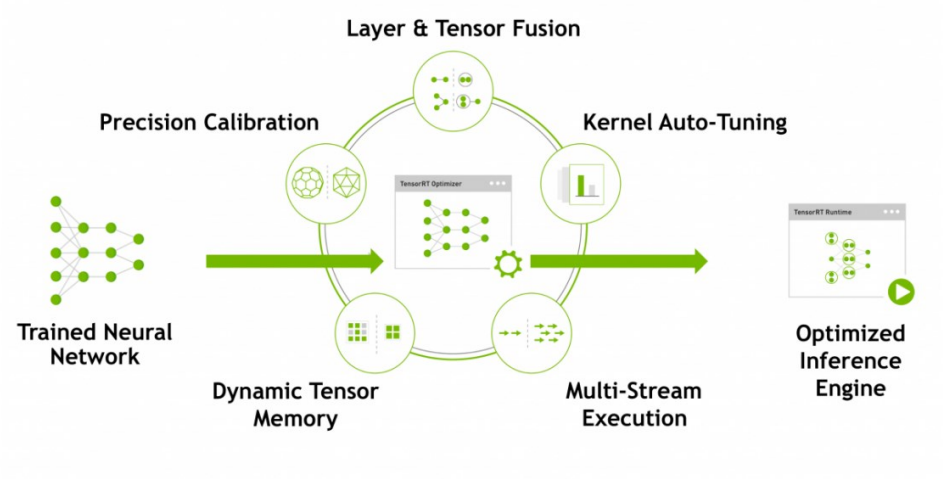
Základem modulu Jetson Nano je grafický čip Nvidia Tegra X1, jenž se skládá ze čtyřjádrového procesoru ARM Cortex A-57 MPCore s taktem 1,43 GHz, grafické karty architektury Maxwell se 128 jádry CUDA s maximální pracovní frekvencí 921 MHz a 4 GB 64bitové paměti LPDDR4 s frekvencí 1600 MHz a propustností 25,6 GB/s. Grafická karta má výkon 472 GFLOPS při FP16 operacích. Modul je vybaven digitálními vstupy a výstupy a rozhraní pro sériovou komunikaci I2C, SPA a UART.

3.2 JetPack SDK

Všechen nezbytný software pro práci s vývojovým kitem je dodán společností Nvidia v balíčku, který obsahuje kromě OS Linux a ovladačů Nvidia L4T i TensorRT, knihovny CUDA, cuDNN, knihovny pro počítačové vidění a další nástroje. Jednotlivé součásti budou detailněji popsány v následujícím textu.

3.2.1 Nvidia TensorRT

TensorRT je platforma pro optimalizaci neuronových sítí založená na CUDA. Umožňuje optimalizovat již naučenou neuronovou síť pro rychlejší běh na konkrétním hardware. Algoritmus TensorRT slučuje vrstvy případně optimalizuje maxpooling, aby neuronová síť měla nižší nároky na výkon a paměť. Umožňuje také další snížení nároků snížením přesnosti sítě [23].



Obrázek 35: Diagram TensorRT

3.2.2 cuDNN (Cuda Deep Neural Network)

Nízkoúrovňová knihovna implementující funkce používané při tvorbě umělých neuronových sítí. Umožňuje používání standardních funkcí, jako jsou konvoluce, pooling, normalizace a aktivační funkce akcelerované využitím grafických karet.

3.2.3 CUDA (Computed Unified Architecture)

Platforma pro pro paralelní programování, programovací model a rozhraní vytvořené firmou NVidia, kdy při správně koncipované tvorbě aplikací lze pomocí CUDA dramaticky navýšit výpočetní výkon využitím grafických karet pro výpočty, na které se běžně používá CPU. S použitím více čipů na grafické kartě, případně použitím více grafických karet lze výpočty dále paralelizovat.

Grafické karty v posledních letech vykazovaly dramatický nárůst výkonu při operacích s plovoucí desetinou čárkou, důvodem je specializace na náročné paralelní výpočty, běžně využívané při zobrazování. Grafické karty jsou koncipovány tak, aby lépe zvládaly pro-

blémy, které mohou být vyjádřitelné jako výpočty s datovým paralelismem a velkým poměrem aritmetických k paměťovým operacím.

4 NÁSTROJE PRO VÝVOJ UMĚLÉ INTELIGENCE

Pro implementaci umělých neuronových sítí vznikla řada frameworků a knihoven, které tento proces usnadňují. S jejich využitím je možné značně zredukovat úsilí nutné pro samotný vývoj aplikace a věnovat větší pozornost samotnému učení sítě. Při výběru je možné volit z několika frameworků vytvořených předními společnostmi a týmy zabývajícími se hlubokým učením neuronových sítí.

Caffe

Prostředí pro hluboké učení vyvinuté univerzitou v Berkley ve vývojovém centru BVLC (Berkley Vision and Learning Center) je vydáváno pod licencí BSD2, tedy volně dostupné. Je vyvíjeno v C++ s rozhraním pro Python a Matlab a jeho filozofií je rychlost, modularita a velká otevřenost komunitě. Architektura prostředí podporuje inovace ze strany komunitních vývojářů, díky čemuž se stále vyvíjí.

PyTorch

Klon frameworku Torch, původně vytvořené v jazyce Lua, vydal tým Facebook AI v roce 2007. Oproti TensorFlow disponuje dynamickými výpočtovými grafy, které mohou být výhodné při práci s velmi variabilními vstupními daty.

TensorFlow

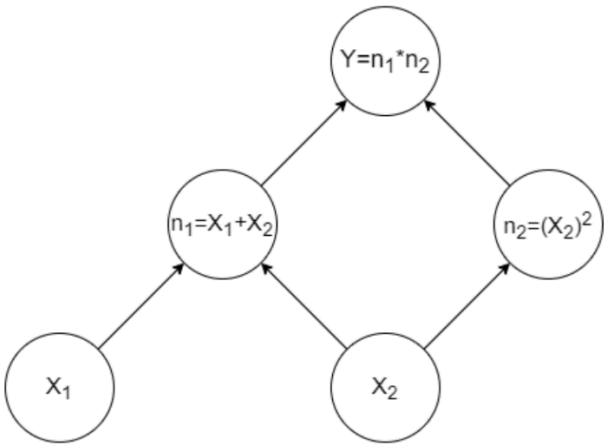
Knihovna pro tvorbu neuronových sítí, především v jazyce Python a C++. Původně vyvinuta pro interní potřeby společnosti Google v roce 2015 jako náhrada za zastaralou knihovnu Theano. Nyní je open-source a dále vyvíjena týmem Google Brain.

Vzhledem k relativně vysoké míry standardizace a dobré implementaci ve vysokoúrovňové knihovně Keras byl tento framework použit pro realizaci praktické části této práce. Bude podrobněji popsán v následujících kapitolách.

4.1 TensorFlow

TensorFlow je knihovna pro matematické úlohy, především zjednodušuje vývoj aplikací založených na neuronových sítích, jejichž příkladem může být zpracování dat, obrazu a porozumění jazyka. Podporuje programovací jazyky Python, JavaScript, C++ a Java. Pro Python má v současnosti nejkompletněji zpracované API jednoduché na použití, a proto byl programovací jazyk Python vybrán pro zpracování této práce.

Tensorflow pracuje s tzv. výpočtovými grafy, jenž se skládají z hran, kterými proudí data a uzlů, ve kterých jsou nad daty prováděny matematické operace. Podobu části výpočtového grafu ukazuje Obrázek 36. Tensorflow pak na distribuovaném systému provádí operace ve výpočtovém grafu paralelně, s využitím plánovacích algoritmů a řízeným sdílením paměti běžícími procesy.



Obrázek 36: Výpočtový graf

4.2 Keras

Vzhledem k tomu, že je knihovna Tensorflow nízkoúrovňová, je její použití pro koncové aplikace relativně komplikované a zdlouhavé. Pro použití je proto praktičtější knihovna Keras, což je vysokoúrovňové API v jazyce Python pro neuronové sítě, které umožňuje rychlejší vývoj a nasazení neuronových sítí. Původně implementovalo frameworky TensorFlow, Theano, Microsoft Cognitive Toolkit a PlaidML, nicméně od verze 2.4, vydané v červnu 2020 již podporuje pouze TensorFlow

Hlavním přínosem je zrychlení vývoje a prototypování díky uživatelsky přívětivému prostředí a modularitě.

Ukázka kódu v jazyce Python využívající knihoven Keras je na Obrázku 37, kde je vytvořen model neuronové sítě pro klasifikaci obrazů o velikosti 640x640 do dvou kategorií.

Na prvních dvou řádcích je proveden import knihoven potřebných pro vytvoření modelu. Dále je do proměnné model vytvořena instance třídy *Sequential*. V této třídě jsou vytvořeny jednotlivé vrstvy neuronové sítě, počínaje první vrstvou *Rescaling* pro normalizaci vstupních hodnot do rozsahu <0, 1>. Následuje série konvolučních a maxpoolingových

vrstev s různou velikostí jádra. Na řádku 12 je pak definována vrstva *Flatten*, která převede vstupní pole dat do tenzoru, jenž je dále vyhodnocován dvojicí vrstev *Dense*, plně propojených vrstev. Druhá z těchto vrstev je výstupní vrstvou modelu a má velikost dva neurony. Tento jednoduchý model by tedy mohl být použit například pro klasifikaci obrazu do dvou tříd.

```
1 from tensorflow.keras import layers
2 from tensorflow.keras.models import Sequential
3
4 model = Sequential([
5   layers.Rescaling(1./255, input_shape=(640, 640, 3)),
6   layers.Conv2D(16, 3, padding='same', activation='relu'),
7   layers.MaxPooling2D(),
8   layers.Conv2D(32, 3, padding='same', activation='relu'),
9   layers.MaxPooling2D(),
10  layers.Conv2D(64, 3, padding='same', activation='relu'),
11  layers.MaxPooling2D(),
12  layers.Flatten(),
13  layers.Dense(128, activation='relu'),
14  layers.Dense(2)
15 ])
```

Obrázek 37: Ukázka kódu v Keras

4.3 Detekční modely

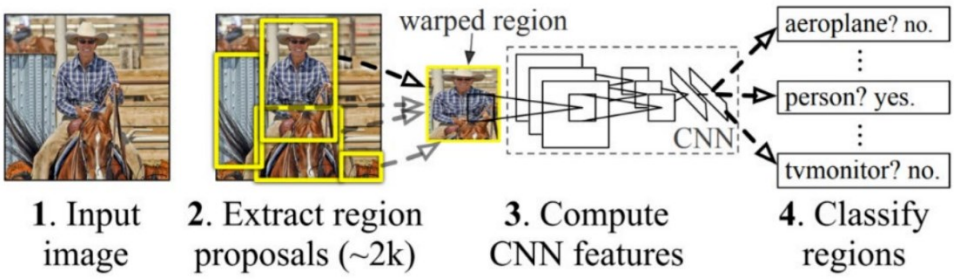
Výše popisované architektury neuronových sítí jsou ze své podstaty klasifikační – jejich úkolem je aktivace příslušného neuronu ve výstupní vrstvě, který představuje příslušnost k dané klasifikované třídě. Pro lokalizaci objektů v obraze jsou pak používány detekční algoritmy, které s použitím neuronové sítě jako klasifikátoru analyzují obraz a určí nejenom přítomnost ale i polohu objektu v obraze.

Nejpřirozenější přístup k detekci objektu by byl takový, že by byl obraz rozdělen na části a každá z nich by byla následně analyzována neuronovou sítí. Tento přístup je však velmi neefektivní a především zdlouhavý – při uvažování všech možných pozic, rotací a velikostí hledaného objektu by bylo prozkoumání obrazu náročné na čas a výpočetní výkon. Pro urychlení tohoto procesu vznikly detekční algoritmy, které budou dále popsány.

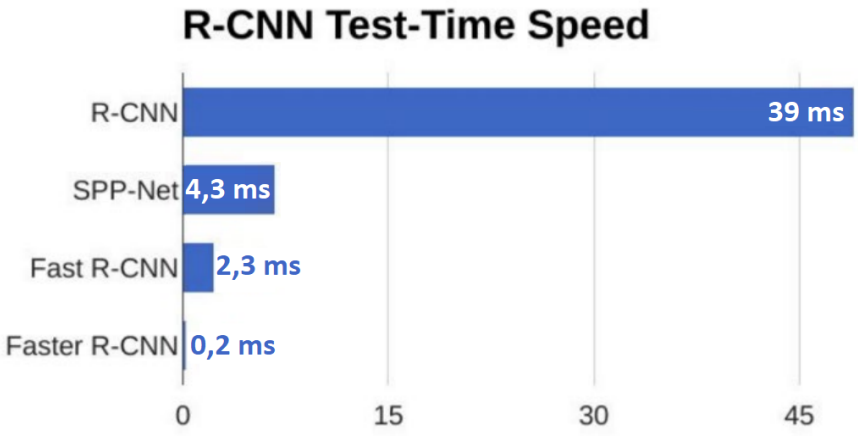
4.3.1 R-CNN

Metoda pro detekci objektů v obraze vytvořená v roce 2014, z níž vzniklo množství modifikací, jako například Fast R-CNN nebo Mask R-CNN. Díky těmto modifikacím je tato metoda dodnes jednou z nejpoužívanějších.

Na Obrázku 38 je znázorněn princip algoritmu R-CNN. Na vstup sítě je vložen obrázek, který je předzpracován algoritmem pro selektivní hledání. Ten obraz rozdělí do několika segmentů, například na základě barvy, kontrastu apod. Segmenty v obrazu jsou následně pospojovány na základě podobných vlastností a navrženy pro zpracování neuronovou sítí. Na výstupu původního algoritmu R-CNN je přibližně 2000 oblastí navržených pro vyhledávání, u následných verzí však byly algoritmy návrhu oblastí dále vylepšovány pro zmenšení množství oblastí, které je nutné dále vyhledávat. Na Obrázku 39 je porovnání časové náročnosti detekce objektů původním algoritmem R-CNN a jeho dalšími modifikacemi [24].



Obrázek 38: Princip funkce R-CNN [24]

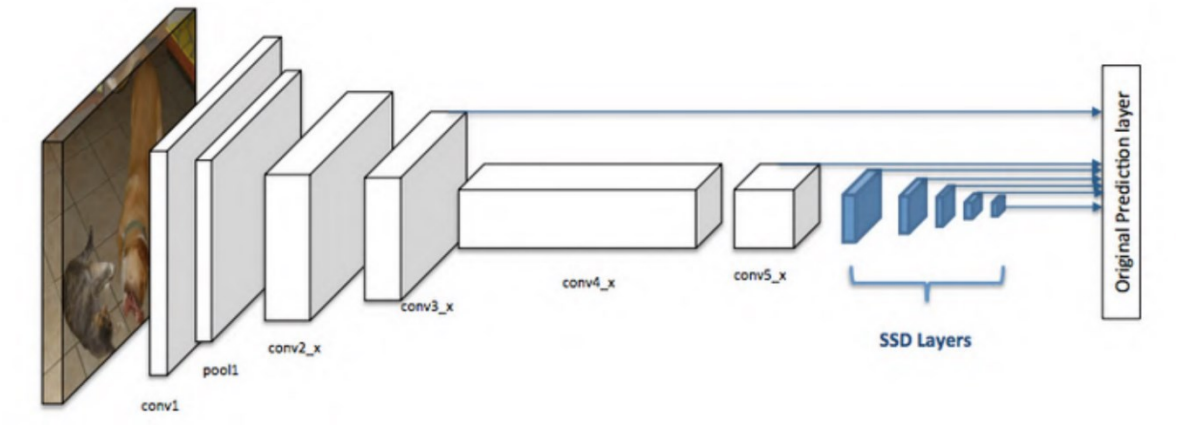


Obrázek 39: Porovnání rychlosti vyhledávacích algoritmů R-CNN [24]

4.3.2 SSD – Single Shot multibox Detector

Detekční model byl představen v roce 2016 a je navržen pro aplikace pracující v reálném čase. Na rozdíl od R-CNN kde byly navrhovány oblasti pro detekci a jejich postupné prohledávání je zde vstupní obraz rozdělen na několik segmentů, které jsou pro prohledávány najednou. To přináší značné zvýšení rychlosti detekce, ovšem za cenu nižší přesnosti. Pro zvýšení robustnosti, především na obrázcích s nižším rozlišením jsou voleny segmenty s různých velikostí.

Algoritmus SSD je implementován do struktury neuronové sítě, dle Obrázku 40. Síť pak lze rozdělit na tzv. *backbone*, což je předtrénovaná síť pro klasifikaci a SSD head, která představuje algoritmus pro výběr vyhledávacích oblastí. Má podobu několika konvolučních vrstev, jejichž výstupy jsou interpretovány jako ohraničující rámečky a třídy objektů v prostorovém umístění finálních vrstev.



Obrázek 40: Detekční model SSD

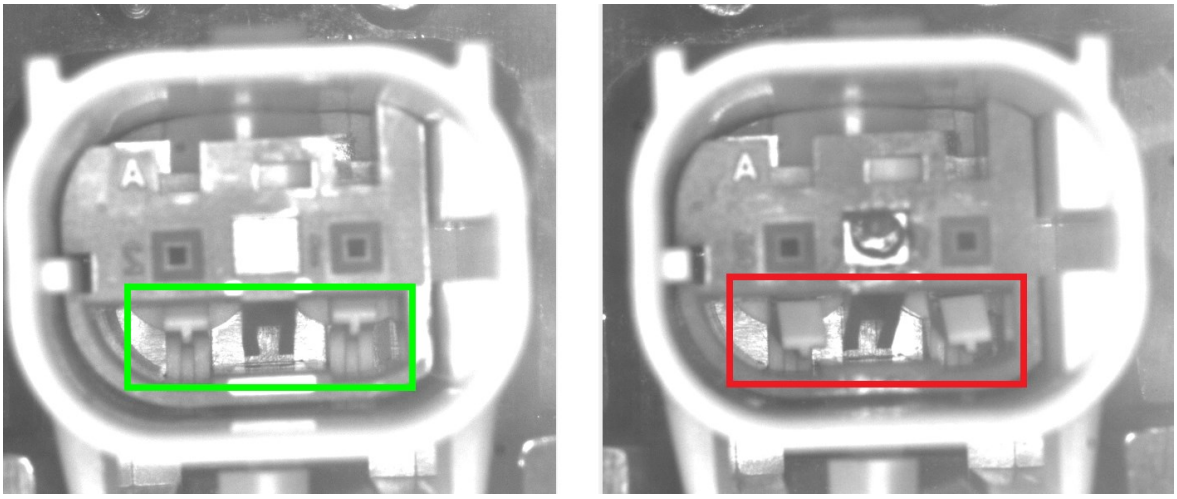
5 IMPLEMENTACE HLUBOKÉ NEURONOVÉ SÍTĚ

Pro otestování možností zařízení Jetson Nano pro detekci vad byla vybrána jedna úloha pro klasifikaci obrazu a jedna úloha detekce objektu v obrazu. Ačkoli se v obou případech jedná o kontrolu kvality výrobků, postup při učení neuronových sítí je značně rozdílný. V případě klasifikace provádí veškerou práci s obrazem neuronová síť, kdy výstupem sítě je klasifikace do jednotlivých tříd, tedy v případě dále popsaných inspekcí OK/NOK. V aplikaci detekce je neuronová síť doplněna o algoritmus SSD, jehož výstup je region v obrazu, kde byl detekován trénovaný vzor. Postup od výběru vhodných snímků po implementaci neuronové sítě na embedded platformě bude popsán v následujících kapitolách.

5.1 Popis vybraných úloh

5.1.1 Kontrola asambláže konektoru

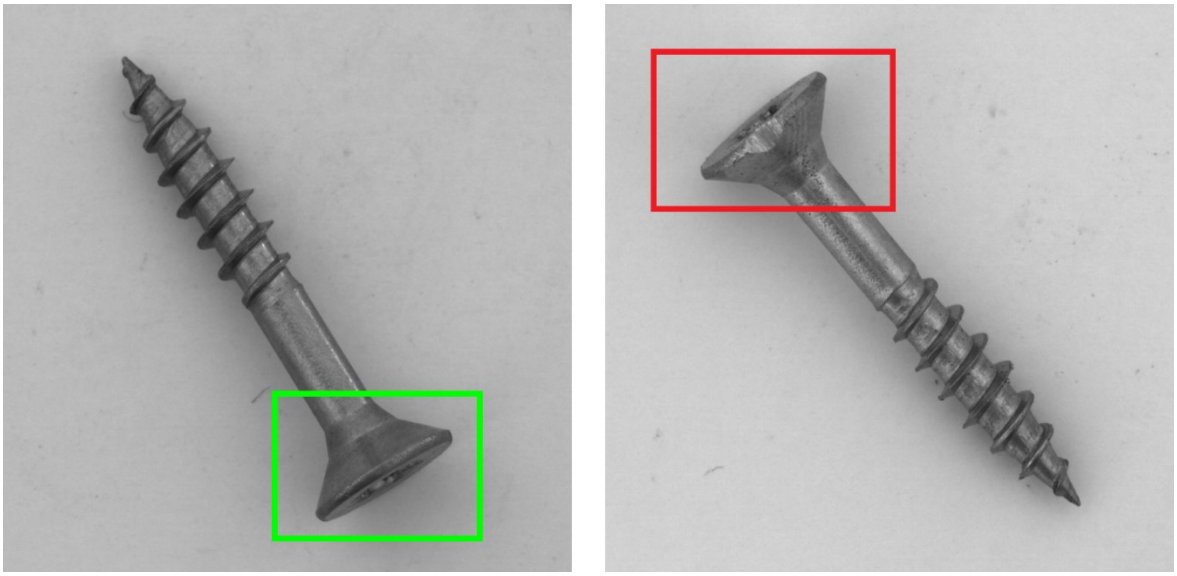
Pro úlohu klasifikace obrazu, kdy neuronová síť klasifikuje celý obraz do tříd dle naučeného modelu byly zvoleny snímky vnitřní části konektoru po asembláži. Její správnost je kontrolována pomocí pozice dvojice pacek, které jsou po úspěšné asembláži v určené pozici, viz. Obrázek 41.



Obrázek 41: Příklad snímku správně asemblovaného konektoru (vlevo) a vadného konektoru (vpravo)

5.1.2 Detekce vad na povrchu šroubku

Pro úlohu detekce objektu v obrazu byly vybrány snímky z databáze společnosti MVTec [https://www.mvtec.com/company/research/datasets/mvtec-ad], konkrétně vady přítomné na hlavičce šroubku, viz. Obrázek 42. Vzhledem k tomu, že podoba takovýchto vad je velmi variabilní, pro konvenční strojové vidění je obtížné spolehlivě vypočítat příznaky potřebné pro jejich detekci. Využití hlubokých neuronových sítí je proto pro detekci nespecifických povrchových vad v praxi typické.

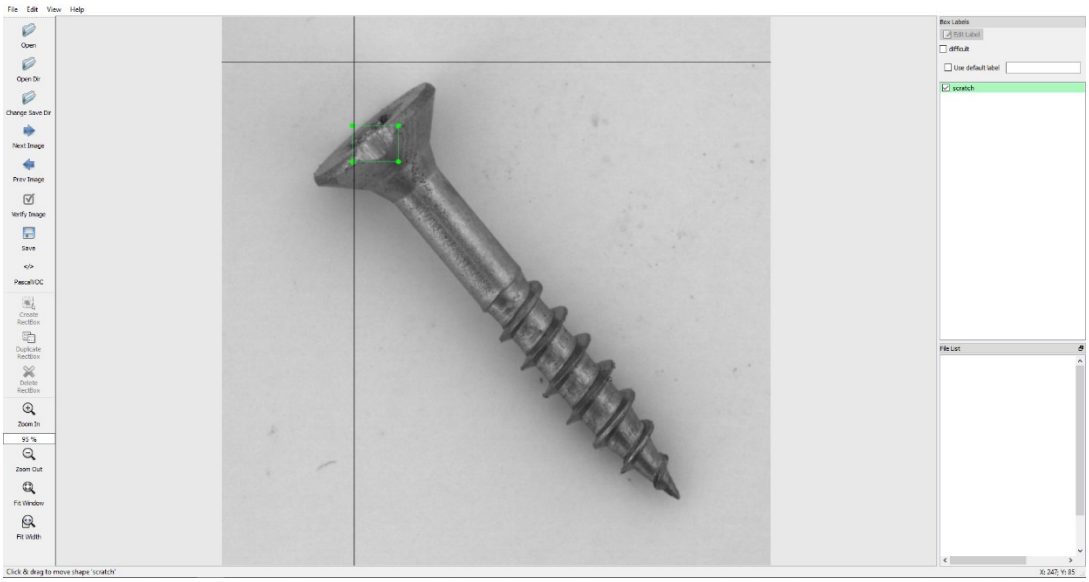


Obrázek 42: Příklad snímků z datasetu s vadami šroubků. Na je šroubek bez vady (vlevo) a s vadou na hlavičce šroubku (vpravo)

5.2 Tvorba datasetu

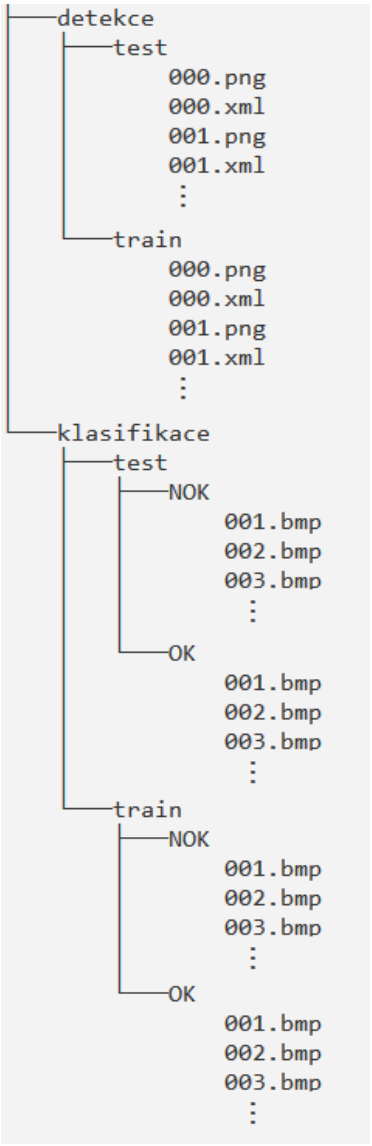
Pro obě úlohy byly shromážděny sady obrázků a rozděleny do trénovací a testovací množiny. Trénovací množina pak vždy byla použita při nastavování vah sítě ve fázi jejího učení a testovací množina obrázků pak sloužila k ověření správného naučení.

Vzhledem k tomu, že úlohy představené v této práci spočívají v hlubokém učení s učitelem, dalším krokem byla anotace obrázků. Pro úlohu klasifikace spočívá anotace v umístění obrázků do složek dle názvu klasifikovaných tříd, v našem případě OK a NOK. V případě úlohy detekce je třeba při anotaci označit umístění vyhledávaného objektu, v našem případě vady na šroubku. Pro účely této práce byl použit program LabelImg [https://github.com/tzutalin/labelImg], který poskytuje jednoduché rozhraní pro anotaci pro operační systémy Windows i Linux. Jeho výstupem je v závislosti na zvoleném anotačním formátu buď soubor *.xlm pro formát Pascal/VOC (Visual Object Classes), jenž je použit i v této práci nebo soubor *.txt při použití formátu YOLO. Podoba prostředí v programu LabelImg je na Obrázku 43.



Obrázek 43: Prostředí pro anotaci LabelImg

Struktura výsledného adresáře s anotovanými datasety pak vypadá dle Obrázku 44.



Obrázek 44: Adresář s datasety pro klasifikaci a detekci

5.3 Učení sítě

Pro zkrácení procesu implementace úlohy hlubokého učení bylo použito principu přeneseného učení, kdy byla na základě datasetů z předchozí kapitoly přeučena již vytrénovaná NN. Tento postup je značně časově kratší než trénování celé sítě znovu od začátku. Vybraný model sítě byl následně přeučen na platformě Google Colaboratory, na adrese *https://colab.research.google.com* a přenesen na Nvidia Jetson Nano, kde bylo otestováno chování NN přímo na tomto zařízení. Jednotlivé body tohoto postupu budou podrobněji popsány v následujících kapitolách.

5.3.1 Výběr modelu neuronové sítě

K dispozici je množství kvalitních předtrénovaných modelů, například ze stránek https://modeldepot.io, http://pretrained.ml nebo kolekce ModelZoo [https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md] od tvůrců frameworku TensorFlow, která byla použita i v této práci. Pro trénování těchto modelů bylo použito velkých databází obrázků, například IILSVRC (ImageNet Large Scale Visual Recognition Challenge). Trénování modelu obdobně velkým objemem dat by kladlo velké nároky na čas a velikost trénovacího datasetu.

K dispozici jsou známé architektury, například ResNet, GoogLeNet, či MobileNet v různých verzích. Při pokusu o implementaci větších NN, například ResNet na Nvidia Jetson Nano nebylo možné načíst model sítě z důvodu malé operační paměti zařízení. Pro aplikaci detekce objektů byla proto zvolena architektura SSD MobileNet V2 FPNLite 640x640, určená pro mobilní zařízení, která klade malé nároky na paměť a výpočetní výkon. Pro klasifikaci pak obdobně MobileNet V2 224x224.

5.3.2 Platforma použitá pro učení modelu neuronové sítě

Vzhledem k omezenému výkonu Nvidia Jetson Nano, nebylo trénování neuronové sítě prováděno přímo na zařízení, ale bylo využito cloudové služby Google Colaboratory. Ta umožňuje zdarma použít výpočetní techniku poskytnutou z cloudu ke spouštění skriptů v jazyce Python prostřednictvím vývojového prostředí v internetovém prohlížeči. Prostředí již po inicializaci obsahuje potřebné knihovny, je tedy třeba minimum zásahů, dále umožňuje propojení s uživatelským účtem Google Drive a GitHub.

Pro běh aplikace v prostředí Google Colaboratory je možné využít GPU poskytovanou taktéž z Google cloudu, což dále urychluje učení a testování neuronové sítě. Služba nicméně nezaručuje přidělení konkrétní typ grafické karty, ten se liší dle aktuálního vytížení od ostatních uživatelů. Mezi nejvíce používané typy grafických karet patří Nvidia Tesla K80, Nvidia Tesla T4 a Nvidia Tesla P100. Poslední jmenovaná, konkrétně ve verzi PCi s 16 GB RAM byla použita pro učení neuronových sítí v této práci. Výkon této karty použité pro trénování sítě byl tedy nepoměrně větší v porovnání s výkonem GPU osazené na Nvidia Jetson Nano.

5.3.3 Učení modelu pro klasifikaci

Síť MobileNet v2, vybraná pro klasifikaci je dostupná pomocí příkazu z knihovny Keras, viz. Obrázek 45. Tímto příkazem je stažena naučená síť bez poslední klasifikační vrstvy. Následně je zakázáno trénování této sítě a na její konec je připojena poolingová a následně plně propojená vrstva dle Obrázku 46. Tyto vrstvy zabezpečí interpretaci dat zpracovaných zbytkem neuronové sítě. Vzniklý model neuronové sítě byl poté zkompilován a naučen na vytvořeném trénovacím datasetu.

```
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet')
```

Obrázek 45: Import sítě MobileNet v2 pomocí příkazu Keras

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
```

Obrázek 46: Přidání poolingové a plně propojené vrstvy pro klasifikaci

Před začátkem procesu učení byla přesnost klasifikace původním modelem s inicializačními váhami na posledních vrstvách stanovena na 64 %. Z výpisu z procesu učení na Obrázku 47 je patrné, že již po deseti epochách učení posledních vrstev dosahoval model přesnosti klasifikace 92,5 % na trénovací a 100 % přesnosti na testovací množině snímků.

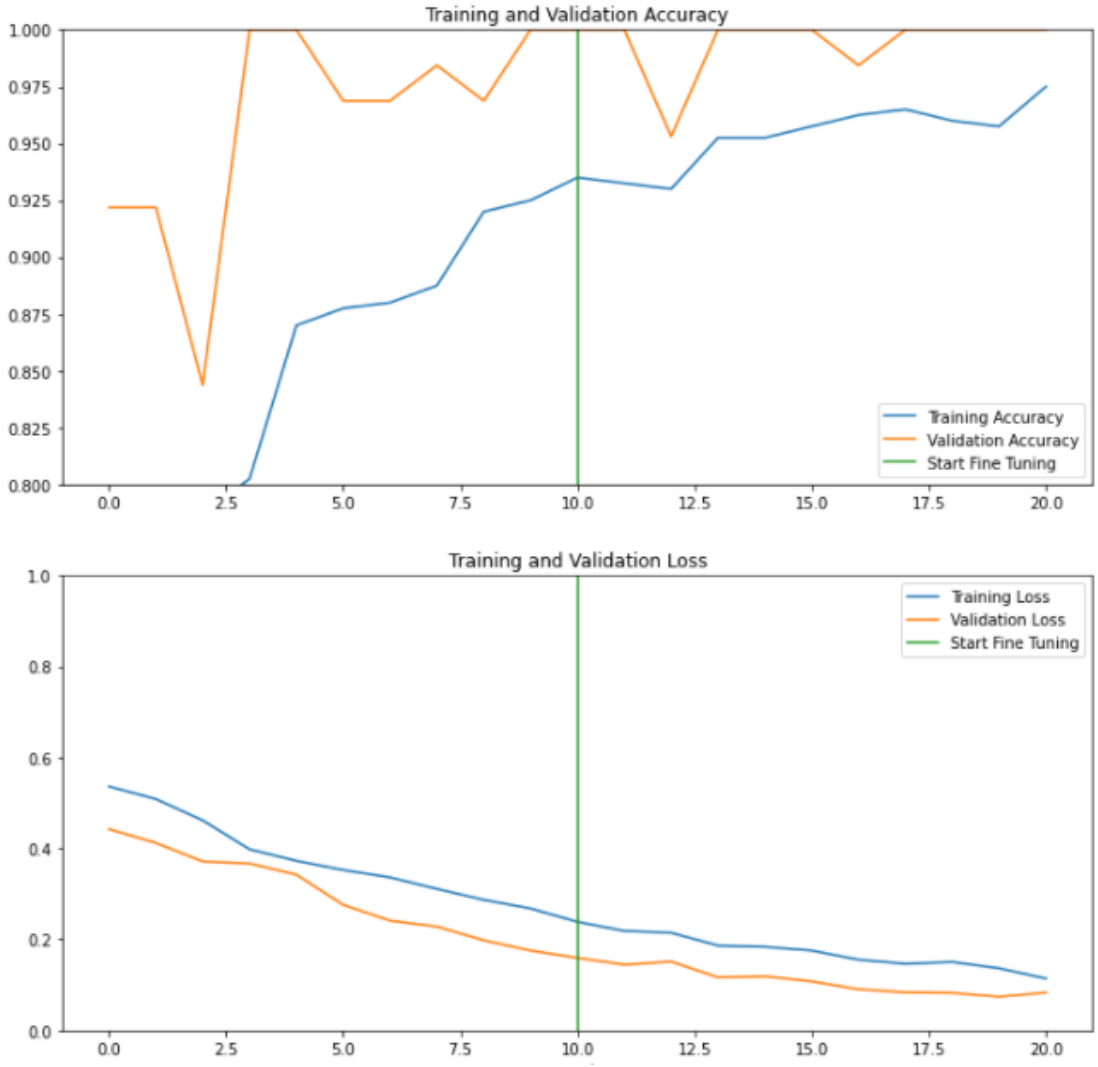
```
4/4 [==============================] - 25s 26ms/step - loss: 0.5119 - accuracy: 0.6406
initial loss: 0.51
initial accuracy: 0.64
Epoch 1/10
25/25 [==============================] - 76s 3s/step - loss: 0.5361 - accuracy: 0.7075 - val_loss: 0.4423 - val_accuracy: 0.9219
Epoch 2/10
25/25 [==============================] - 2s 66ms/step - loss: 0.5085 - accuracy: 0.7600 - val_loss: 0.4119 - val_accuracy: 0.9219
Epoch 3/10
25/25 [==============================] - 2s 66ms/step - loss: 0.4616 - accuracy: 0.7875 - val_loss: 0.3712 - val_accuracy: 0.8438
Epoch 4/10
25/25 [==============================] - 2s 66ms/step - loss: 0.3979 - accuracy: 0.8025 - val_loss: 0.3667 - val_accuracy: 1.0000
Epoch 5/10
25/25 [==============================] - 2s 65ms/step - loss: 0.3726 - accuracy: 0.8700 - val_loss: 0.3421 - val_accuracy: 1.0000
Epoch 6/10
25/25 [==============================] - 2s 67ms/step - loss: 0.3524 - accuracy: 0.8775 - val_loss: 0.2757 - val_accuracy: 0.9688
Epoch 7/10
25/25 [==============================] - 2s 65ms/step - loss: 0.3362 - accuracy: 0.8800 - val_loss: 0.2410 - val_accuracy: 0.9688
Epoch 8/10
25/25 [==============================] - 2s 67ms/step - loss: 0.3109 - accuracy: 0.8875 - val_loss: 0.2280 - val_accuracy: 0.9844
Epoch 9/10
25/25 [==============================] - 2s 69ms/step - loss: 0.2868 - accuracy: 0.9200 - val_loss: 0.1978 - val_accuracy: 0.9688
Epoch 10/10
25/25 [==============================] - 2s 65ms/step - loss: 0.2678 - accuracy: 0.9250 - val_loss: 0.1752 - val_accuracy: 1.0000
```

Obrázek 47: Učení neuronové sítě pro klasifikaci

Pro další zlepšení přesnosti klasifikace bylo počínaje stou vrstvou povoleno trénování sítě a dále pokračoval proces učení, tentokrát pouze v pěti epochách. Dle výpisu na Obrázku 48 dosáhla přesnost klasifikace na trénovací množině rostla na dosáhla 97,5 %, přičemž přesnost na testovacím datasetu v druhé fázi učení dosahovala 100 %. Hodnoty ztrátových funkcí při učení dále klesaly. Hodnoty přesnosti a ztrátové funkce byly vyneseny do grafu na Obrázku 49.

```
Epoch 10/20
25/25 [==============================] - 7s 96ms/step - loss: 0.2385 - accuracy: 0.9350 - val_loss: 0.1592 - val_accuracy: 1.0000
Epoch 11/20
25/25 [==============================] - 2s 67ms/step - loss: 0.2185 - accuracy: 0.9325 - val_loss: 0.1443 - val_accuracy: 1.0000
Epoch 12/20
25/25 [==============================] - 2s 68ms/step - loss: 0.2144 - accuracy: 0.9300 - val_loss: 0.1510 - val_accuracy: 0.9531
Epoch 13/20
25/25 [==============================] - 2s 66ms/step - loss: 0.1864 - accuracy: 0.9525 - val_loss: 0.1164 - val_accuracy: 1.0000
Epoch 14/20
25/25 [==============================] - 2s 65ms/step - loss: 0.1835 - accuracy: 0.9525 - val_loss: 0.1186 - val_accuracy: 1.0000
Epoch 15/20
25/25 [==============================] - 2s 65ms/step - loss: 0.1754 - accuracy: 0.9575 - val_loss: 0.1075 - val_accuracy: 1.0000
Epoch 16/20
25/25 [==============================] - 2s 66ms/step - loss: 0.1553 - accuracy: 0.9625 - val_loss: 0.0901 - val_accuracy: 0.9844
Epoch 17/20
25/25 [==============================] - 2s 66ms/step - loss: 0.1462 - accuracy: 0.9650 - val_loss: 0.0835 - val_accuracy: 1.0000
Epoch 18/20
25/25 [==============================] - 2s 66ms/step - loss: 0.1502 - accuracy: 0.9600 - val_loss: 0.0821 - val_accuracy: 1.0000
Epoch 19/20
25/25 [==============================] - 2s 66ms/step - loss: 0.1364 - accuracy: 0.9575 - val_loss: 0.0736 - val_accuracy: 1.0000
Epoch 20/20
25/25 [==============================] - 2s 67ms/step - loss: 0.1138 - accuracy: 0.9750 - val_loss: 0.0829 - val_accuracy: 1.0000
```

Obrázek 48: Doučení další části neuronové sítě



Obrázek 49: Průběh přesnosti klasifikace a ztrátových funkcí při učení klasifikační neuronové sítě

5.3.4 Učení modelu pro detekci objektů

Výchozí architektura pro přeučení SSD MobileNet v2 FPNLite 640x640 byla získána z repozitáře tensorflow Model Zoo.

Pro vlastní učení sítě byla využita knihovna funkcí Tensorflow Object detection API, které výrazně zkracují proces implementace učení a nasazení neuronových sítí pro detekci objektů.

Po stažení instalaci výše zmiňovaných knihoven je spuštěním skriptu *generate_tfrecord.py* zpracován trénovací a testovací dataset a následně obdobně spuštěním skriptu *mo-*

del_main_tf2.py spuštěno učení sítě. Během učení jsou po sto krocích zobrazovány aktuální hodnoty ztrátové funkce, na základě které je trénování zastaveno. Příklad jednoho výpisu učení je na Obrázku 50. Celý proces od učení sítě až po její uložení se tedy děje pomocí již připravených funkcí z knihovny Object detection, kterým jsou pouze předávány parametry, jako například cesty k anotovaným datasetům, výchozímu modelu sítě a podobně.

```
I0516 16:18:24.922659  5508 model_lib_v2.py:708] {
 'Loss/classification_loss': 0.023486575,
 'Loss/localization_loss': 0.0024119979,
 'Loss/regularization_loss': 0.05594641,
 'Loss/total_loss': 0.081844985,
 'learning_rate': 8.2254405e-07}
```

Obrázek 50: Příklad výpisu z učení neuronové sítě pro detekci objektů

6 IMPLEMENTACE HLUBOKÉHO UČENÍ NA ZAŘÍZENÍ NVIDIA JETSON NANO

V kapitole bude popsáno uvedení do provozu a instalace potřebného programového vybavení zařízení Nvidia Jetson Nano.

6.1 Instalace Nvidia Jetson Nano

Instalace operačního systému Linux, konkrétně distribuce Ubuntu byla provedena dle oficiálního návodu výrobce.

V rámci prvního spuštění a nastavení je potřeba mít k dispozici microSD kartu, klávesnici, myš a monitor. Další způsob prvního spuštení je tzv. headless mode, kdy je k zařízení přistupováno přes SSH (Secure Shell) z jiného počítače, nicméně pro jednoduchost použití byl zvolen standardní režim s grafickým výstupem přes HDMI. V našem případě byl také k dispozici 5V adaptér pro napájení prostřednictvím DC Barrel Jack konektoru.

Výrobce nabízí volně ke stažení JetPack SDK, popsaný v předchozích kapitolách, ze kterého byl pomocí programu BalenaEtcher (https://www.balena.io/etcher/) vytvořen obraz na microSD kartě. Po vložení karty a připojení napájení je spuštění zařízení signalizováno zelenou LED, která se nachází vedle microUSB konektoru. Během prvním spuštění je uživatel proveden úvodním nastavením operačního systému Ubuntu.

6.2 Implementace neuronových sítí v prostředí TensorFlow a Keras

Pro spouštění skriptů v jazyce python a využívání knihoven TensorFlow a Keras je nutné doinstalovat systémové balíčky a knihovny. Instalace je provedena pomocí příkazů v terminálu systému Linux Ubuntu.

Nejdříve je proveden update aplikací a instalace potřebných balíčků pomocí následujících příkazů.

```
$ sudo apt-get update
$ sudo apt-get install libhdf5-serial-dev hdf5-tools libhdf5-dev zlib1g-
dev zip libjpeg8-dev liblapack-dev libblas-dev gfortran
```

Dále je nainstalován instalační manažer pro Python pip3.

```
$ sudo apt-get install python3-pip
$ sudo pip3 install -U pip testresources setuptools==49.6.0
```

Pomocí pip3 je následně možné nainstalovat potřebné knihovny pro Python.

```
$ sudo pip3 install -U numpy==1.16.1 future==0.18.2 mock==3.0.5
h5py==2.10.0 keras_preprocessing==1.1.1 keras_applications==1.0.8
gast==0.2.2 futures protobuf pybind11
```

Nakonec je pomocí pip3 nainstalováno TensorFlow

```
$ sudo pip3 install --pre --extra-index-url
https://developer.download.nvidia.com/compute/redist/jp/v44 tensorflow
```

Pokud všechny tyto příkazy proběhnou úspěšně, je možné na zařízení Jetson Nano spouštět programy v jazyce Python.

6.3 Přenos natrénované neuronové sítě na Nvidia Jetson Nano

Model je uložen při trénování v paměti zařízení a pro přenos naučené sítě je model uložen ve vhodném formátu. Pro uložení naučeného modelu neuronové sítě byly v této práci použity rozdílné příkazy. V aplikaci klasifikace obrazu byl pro uložení modelu použit příkaz z knihovny Keras, zatímco model pro detekci objektů byl exportován s využitím knihovny Object Detection API. Oba tyto formáty budou dále popsány podrobněji.

6.3.1 Uložení pomocí příkazů Tensroflow a Keras

Knihovny Tensorflow a Keras nabízí několik možností pro uložení naučeného modelu sítě které budou dále popsány. Při známé architektuře sítě lze použít příkaz *tf.keras.layers.Layer.set_weights()*, kterým lze uložit naučené váhy mezi neurony a následně na cílovém zařízení je nastavit. Dále je možné model uložit ve formátu HDF5, kde je model uložen v rámci jednoho souboru, v němž je uložena architektura modelu, váhy a data z procesu kompilace. Nejsou zde však uloženy výpočtové grafy Tensorflow a nelze tedy obnovit vlastní objekty, například vlastní definované třídy apod.

Vzhledem ke svoji univerzálnosti byl pro uložení klasifikační sítě použit příkaz *model.save()*, kterým je uložen celý model, včetně naučených vah mezi neurony, Tensorflow výpočtových grafů a případných vlastních vytvořených objektů. Ačkoli použitý formát má největší nároky na kapacitu úložiště z výše jmenovaných, zabezpečuje kompletní přenos naučené sítě. Takto uložený model měl velikost 58,4MB.

6.3.2 Uložení modelu Object Detection API

Vysokoúrovňová knihovna Object detection poskytuje kromě funkcí pro učení sítě také funkce pro její export. Po naučení modelu je tedy pomocí skriptu *exporter_main_v2.py* uložen model do zvoleného adresáře. Kromě celého modelu s natrénovanými vahami je uložen i trénovací checkpoint a konfigurační soubor pipeline.config.

6.4 Inference na Jetson Nano

6.4.1 Klasifikace obrazu

Načtení naučené sítě pro klasifikaci, jejíž souhrn architektury je na Obrázku 51, do paměti Jetson Nano, trvalo 132,1 sekundy. Dále byla spuštěna inference na validační množině obrázků, která obsahovala snímky padesáti OK a padesáti NOK kusů. Z tohoto ověření na validační množině byly zaznamenány výsledky, viz. Obrázek 52, na němž je patrné, že validační snímky klasifikoval správně ve 100 % případů a provedení jedné inference trvalo přibližně 0,23 sekundy. Vzhledem k nižší přesnosti na trénovací množině v kapitole 5.3.3 je však pravděpodobné, že na velkém množství snímků v reálném provozu by přesnost byla o něco menší než 100 %. Tyto výsledky byly uspořádány do Tabulky 1.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 244, 244, 3)]     0
_________________________________________________________________
tf.math.truediv (TFOpLambda) (None, 244, 244, 3)       0
_________________________________________________________________
tf.math.subtract (TFOpLambda (None, 244, 244, 3)       0
_________________________________________________________________
mobilenetv2_1.00_224 (Functi (None, 8, 8, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dropout (Dropout)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 1281
=================================================================
Total params: 2,259,265
Trainable params: 1,862,721
Non-trainable params: 396,544
_________________________________________________________________
```

Obrázek 51: Architektura sítě na Jetson Nano

```
...
>>>
Number of OK parts predicted as OK parts: 50
Number of OK parts predicted as NOK parts: 0
>>>
Number of NOK parts predicted as NOK parts: 50
Number of NOK parts predicted as OK parts: 0
>>>
Accuracy in prediction: 100.0 %
>>>
Overal time for inference: 23.3329 seconds
Average time for single inference: 0.2333 seconds
```

Obrázek 52: Výpis z inference při klasifikaci testovací množiny snímků

Tabulka 1: Kontingenční tabulka s výsledky klasifikace obrazu na zařízení Jetson Nano

	OK (detekované	NOK (detekované)	Celkem
OK (anotované)	50	0	50
NOK (anotované	0	50	50

6.4.2 Detekce objektů v obrazu

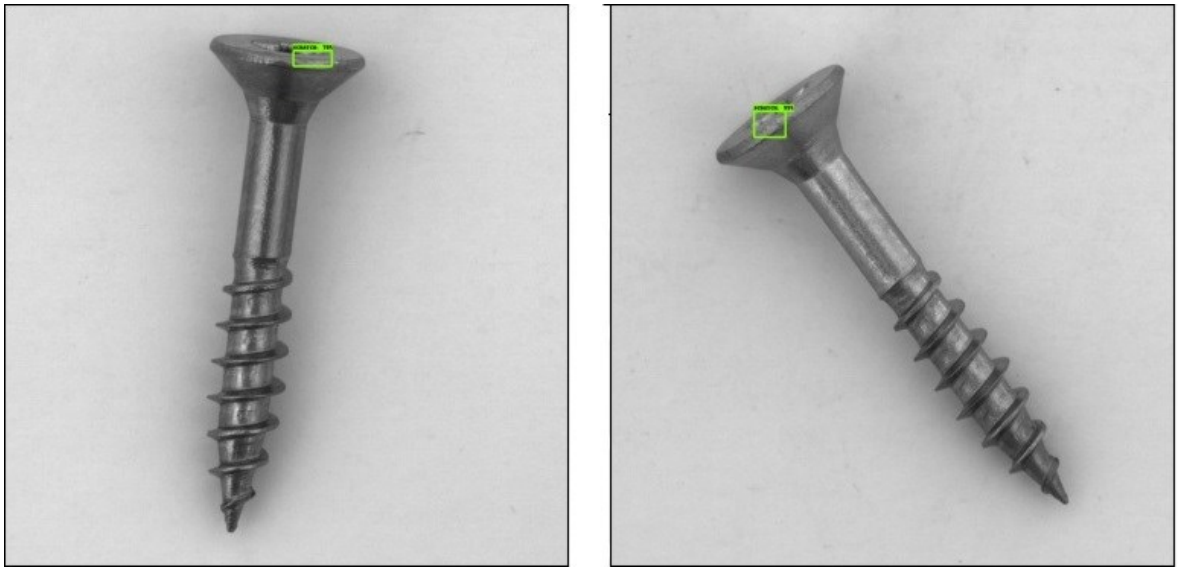
Načtení modelu sítě trvalo 184,15 sekund, což je podstatně více než u modelu pro klasifikaci v předchozím případě, což je pravděpodobně způsobeno větší složitostí modelu, z důvodu přítomnosti detekčního modelu SSD, který v tomto případě doplňuje síť MobileNet. Z inference na validační množině snímků, která se skládala z 24 OK a 9 NOK, byla vypočítána přesnost 93,9 % a provedení jedné inference trvalo průměrně 0,42 sekund. Do této doby nebyl započítáván čas první inference, která pravděpodobně kvůli inicializaci trvala násobně delší dobu. Chybné detekce se vyskytovali v množině OK snímků, kde ve dvou případech síť detekovala poškození kusu, viz. Obrázek 53. Z výsledků detekce byla vytvořena Tabulka 2. Snímky výrobků, na nichž byly nesprávně detekovány vady lze nalézt na Obrázku 54.

```
>>>
Done, total inference time: 13.935643196105957 seconds
Average time for one inference: 0.4222922180638169 seconds
Number of images: 33
>>>
Number of OK parts with no defects: 22
Number of OK parts with defects: 2
Number of NOK parts with defects: 9
Number of NOK parts with no defects: 0
>>>
```

Obrázek 53: Výpis z inference při detekci vad na validační množině snímků

Tabulka 2: Kontingenční tabulka s výsledky detekce vad na zařízení Jetson Nano

	OK (detekované	Nelze určit	NOK (detekované)	Celkem
OK (anotované)	22	2	0	24
NOK (anotované	0	0	9	9



Obrázek 54: Výrobky, na kterých byly nesprávně detekované vady.

7 SROVNÁNÍ S KOMERČNÍM SYSTÉMEM PRO STROJOVÉ VIDĚNÍ

Výsledky získané z aplikací klasifikace obrazu a detekce objektu v obraze budou dále porovnány s komerčně používaným systémem pro strojové vidění využívajícím neuronové sítě, Cognex ViDi Suite 4.1, který pomocí několika jednoduše implementovatelných nástrojů umožňuje učení neuronových sítí pro strojové vidění v průmyslu.

7.1 Popis systému Cognex ViDi

Firma Cognex je rozšířeným dodavatelem systémů pro strojové vidění, například čteček kódů, smart kamer a PC-based systémů strojového vidění. Nabízí také několik produktů využívajících hlubokého učení neuronových sítí, z nichž byla pro porovnání vybráno prostředí Cognex ViDi Suite, v němž lze využít celkem tři nástroje založené na hlubokém učení určené pro různé typy úloh a tyto nástroje budou dále popsány podrobněji.

Okno prostředí Cognex ViDi Suite lze rozdělit na několik logických částí dle rozmístění ovládacích a zobrazovacích prvků. Tyto části jsou zvýrazněny na Obrázku 55 a jsou dále popsány.

1 – Okno s právě označeným snímkem, se kterým právě uživatel pracuje.

2 – Dostupné datasety se snímky a výsledky z jejich zpracování.

3 – Prvky pro nastavení variací učeného vzorového obrazu. Zde je možné volit uvažovaný úhel otočení, symetrickou i asymetrickou změnu velikosti či změnu kontrastu a jasu učených vzorů. Tímto lze dosáhnout lepšímu se přizpůsobení reálným podmínkám v reálném procesu i při malém počtu trénovacích snímků.

4 – Nastavení neuronové sítě a jejího trénování, například volba učení s učitelem či bez učitele a počet trénovacích epoch.

Obrázek 55: Prostředí programu Cognex ViDi Suite

7.1.1 Blue Locate tool

Tzv. Blue tool slouží k vyhledání a lokalizaci prvků v obrazu dle naučených anotovaných vzorů. Také je tento nástroj možné použít pro učení vzorových znaků a jejich následné rozpoznání v aplikacích OCR.

7.1.2 Red Analyze tool

Analyze tool slouží pro detekci anomálií a vad objektu v obrazu. V režimu učení bez učitele jsou nástroji předkládány pouze snímky OK objektů, dle kterých je naučena neuronová síť. Následně nástroj detekuje anomálie při uvažování zadaných tolerancí.

V režimu učení s učitelem jsou naopak vyžadovány anotované snímky s možnými druhy vad. Tyto vzory vad je možné zatížit zvolenou variabilitu a následně dle nich naučit neuronovou síť, jež pak hledá naučené vady na objektech v předložených snímcích. Uvedený postup je využit při učení neuronové sítě pro detekci vad na Nvidia Jetson Nano.

7.1.3 Green Classify tool

Nástroj umožňující klasifikaci objektu nebo celé scény, kdy pro učení musejí být k dispozici snímky přiřazené rozlišovaným třídám. Typickým využitím jsou aplikace třídě-

ní nebo kontrola vad založená na podobě celého obrázku, bez potřeby dále lokalizovat detekovanou vadu. Popsaný nástroj byl použit pro porovnání s aplikací klasifikace na Nvidia Jetson Nano v této práci.

7.2 Implementace úloh v prostředí Cognex ViDi

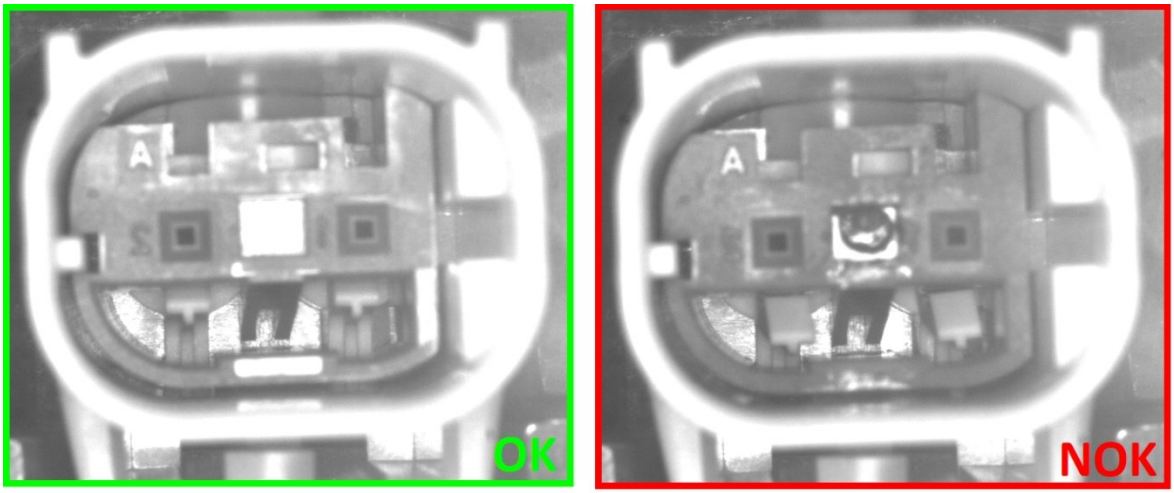
Pro práci s Cognex ViDi, včetně učení neuronových sítí v tomto prostředí byl použit notebook s procesorem Intel Core i7-10750H 2,6 GHz, 32 GB RAM s GPU NVIDIA Quadro T1000 Max-Q, která byla využívána pro výpočty při učení neuronových sítí. Doba učení pak bude záležet především na použité grafické kartě, například výrobce uvádí dobu učení sítě přibližně 6 minut s nejvýkonnější běžně dostupnou GPU. Doba učení na výše uvedené sestavě se pohybovala od 15 do 30 minut pro klasifikaci a 20 až 40 minut pro úlohu detekce objektů, dle nastavených parametrů učení.

Příprava datasetů i následné učení sítí byla prováděna s důrazem na rovné podmínky pro obě srovnávaná zařízení. Trénovací i validační množina byla složena z totožných snímků a pro učení sítí bylo nastaveno pouze nezbytné množství parametrů na doporučené hodnoty. Přesnost vyhodnocení oběma zařízeními by tedy mohla být při vynaložení dodatečného úsilí větší, nicméně pro korektní srovnání bylo trénování sítí prováděno bez zásahu nebo optimalizací, pouze učícími algoritmy.

7.2.1 Klasifikace obrazu

Klasifikace obrazu byla provedena nástrojem Classify Tool, kterému byl předložen připravený dataset snímků a následně označena příslušnost snímků k třídě OK a NOK. Dále byl nastaven parametr Feature Size, jenž má význam velikosti charakteristického rysu, kterým se jednotlivé třídy odlišují. Poté byl nastaven počet trénovacích epoch a spuštěno učení sítě, které trvalo přibližně 15 minut.

Po dokončení procesu učení byla naučenému nástroji pro klasifikaci předložena validační množina, která se obdobně jako v případě Jetson Nano sestávala ze snímků 50 OK a 50 NOK kusů. Dle výsledků klasifikace těchto snímků bylo dosaženo velmi dobré přesnosti, kdy bylo správně klasifikováno 100 % snímků. Průměrná doba zpracování jednoho snímku přitom byla 88,8 ms. Výsledky klasifikace jsou v Tabulce 3. Příklad klasifikace obrazu s grafickým výstupem OK/NOK pro operátora je na Obrázku 56.



Obrázek 56: Příklad klasifikace snímku vadného kusu s výstupem pro operátora

Tabulka 3: Kontingenční tabulka s výsledky klasifikace obrazu v systému Cognex ViDi

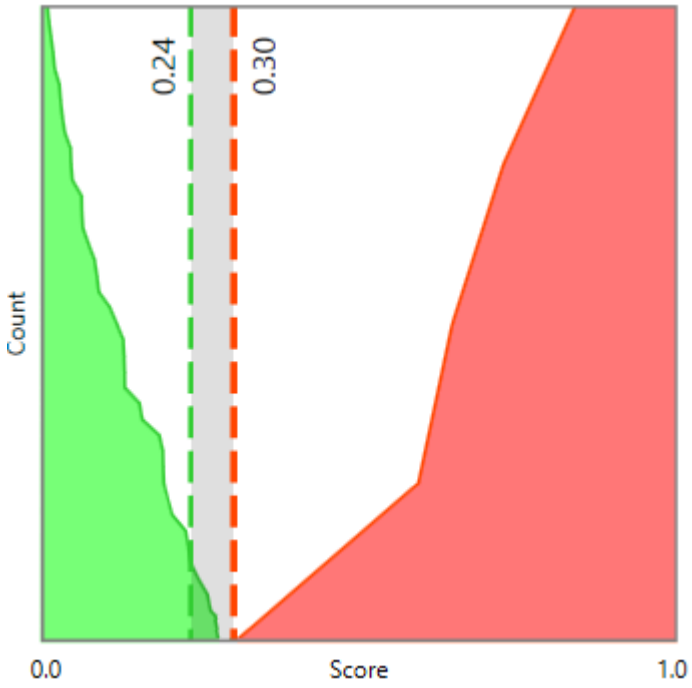
	OK (detekované	NOK (detekované)	Celkem
OK (anotované)	50	0	50
NOK (anotované	0	50	50

7.2.2 Detekce vad výrobku

Nástroj Analyze tool s nastavením učení s učitelem, který byl použit pro úlohu detekce objektů, vyžadoval anotaci obrázků. Pomocí k tomu určeného nástroje tedy byly na trénovacích obrázcích vyznačeny oblasti, ve kterých se nacházely vady určené k detekci. Následně po této přípravě datasetů byl nastaven počet trénovacích epoch, parametr Feature size a také rozsah variability anotovaných vad. Díky tomu jsou pak při učení sítě uvažovány i procedurálně generované vady, vypočítané z předložených datasetů a zadaném rozsahu variability. Takto lze docílit vyšší přesnosti detekce v reálném provozu i při méně obsáhlém trénovacím datasetu. Poté již bylo možné spustit učení sítě, které trvalo přibližně 20 minut, přičemž při nastavení vyšší uvažované variability se potřebná doba pro učení značně zvyšovala.

Ověření naučeného modelu bylo provedeno na validační množině snímků 24 OK kusů bez vad a 9 NOK vadných kusů. Z výsledků inference byl vytvořen graf pro vyhodnocení správnosti detekce, kde jsou vyneseny počty OK a NOK snímků vyšlé z detekce v závislosti na hodnotě skóre, kterých při inferenci dosáhly. V tomto grafu, který lze nalézt

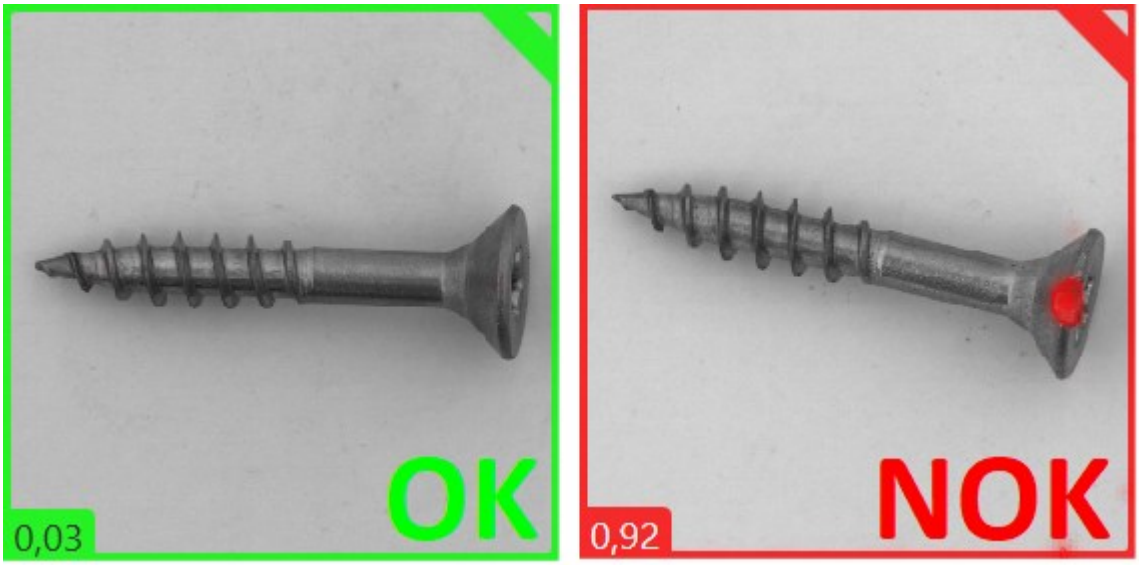
na Obrázku 57, jsou také vyneseny prahy pro přiřazení snímků do množin OK, NOK a do množiny jejíž příslušnost nelze určit.



Obrázek 57: Graf s výsledky inference detekce vad

Z tohoto grafu je patrné, že dle hodnotícího parametru skóre na sebe detekovaná OK a NOK množina snímků téměř navazuje. Je tedy pravděpodobné, že v reálném prostředí by tento nástroj vyhodnotil některé vadné kusy jako OK, a naopak dobré jako NOK. Z tohoto důvodu byly posunuty prahové hodnoty tak, aby byly odhaleny všechny špatné kusy, přičemž byla zachována alespoň minimální vzdálenost mezi oběma prahy, což se bohužel projevilo zhoršenou detekcí OK kusů.

Po výše uvedeném nastavení prahu bylo na validační množině snímků dosaženo přesnosti 96,9 %, která však vychází z umístění prahové hodnoty z výše uvedených důvodů. Průměrná doba zpracování jednoho snímku byla 189,9 ms. Výsledky z detekce vad jsou uspořádány v Tabulce 4. Příklad detekce vady na bezvadném výrobku a s detekovanou vadou s grafickým výstupem OK/NOK a označením oblasti vady pro operátora lze nalézt na Obrázku 58.



Obrázek 58: Příklad výsledku detekce v případě OK výsledku a v případě kdy byla detekována vada

Tabulka 4: Kontingenční tabulka s výsledky detekce vad výrobku v systému Cognex ViDi

	OK (detekované	Nelze určit	NOK (detekované)	Celkem
OK (anotované)	23	1	0	24
NOK (anotované	0	0	9	9

Informace o kvalitě detekce lze získat také z hodnot *Recall* a *Precision*, které jsou při inferenci vypočteny. *Recall* udává procento z celkové anotované oblasti defektu, která je pokryta nalezeným defektem na všech zpracovávaných snímcích. *Precision* je procento z celkové anotované oblasti defektu, která je pokryta nalezenou oblastí defektu, na všech zpracovávaných snímcích. Tyto parametry jsou uspořádány do Tabulky 5.

Tabulka 5: Informace o kvalitě detekce na validační množině v prostředí Cognex ViDi Suite

	Recall	Precision	F-Score
Defect	66,2	85,3	75,8

7.3 Porovnání dosažených výsledků

V této kapitole budou porovnány poznatky získané z implementace úloh klasifikace obrazu a detekce objektu v obrazu hlubokým učením na zařízeních Jetson Nano a komerčního systému pro strojové vidění Cognex ViDi.

Vzhledem k rozdílům ve výkonu GPU, kterým disponuje Jetson Nano a grafickou kartou Nvidia Quadro T1000 použitou při práci s Cognex ViDi, je nejdříve uvedeno srovnání výkonu obou zařízení. Hardwarové prostředky přitom přímo neovlivňují kvalitu detekce či klasifikace, mají však rozhodující vliv na potřebnou dobu učení sítě a délku jedné inference. Obě zařízení podporují technologii CUDA umožňující paralelní zpracování a mají stejnou velikost paměti, zásadně se však liší počtem jader a rychlostí operací s plovoucí řádovou čárkou. Srovnání výkonu obou zařízení je uvedeno v Tabulce 7. V reálných aplikacích se systémem Cognex ViDi jsou však v praxi využívány ještě řádově výkonnější grafické karty (dle manuálu Cognex je aktuálně doporučenou GPU série Nvidia GeForce RTX 30).

Tabulka 6: Porovnání výkonu GPU Nvidia Jetson Nano a Nvidia Quadro T1000 použité k práci s Cognex ViDi Suite

	Nvidia Jetson Nano	Nvidia Quadro T1000	Nvidia GeForce RTX 3070
Počet jader	128	896	5888
Rychlost operací	472 GFLOPS	2,5 TFLOPS	12,1 TFLOPS
Velikost paměti	4 GB LPDDR4	4 GB GDDR6	GDDR6 8 GB

K implementaci výše uvedených úloh pro kontrolu vad výrobku byl k dispozici totožný trénovací dataset snímků a při učení jim bylo věnováno obdobné úsilí. Srovnání přesnosti a doby vyhodnocení obou zařízení je uvedeno v Tabulce 7.

Tabulka 7: Porovnání přesnosti a doby zpracování pomocí zařízení Nvidia Jetson Nano a Cognex ViDi Suite

	Klasifikace		Detekce	
	Přesnost [%]	Doba zpracování [ms]	Přesnost [%]	Doba zpracování [ms]
Nvidia Jetson Nano	100	233,3	93,9	420
Cognex ViDi Suite	100	88,8	96,9	189,9

V klasifikaci snímku vnitřku konektoru dosahovaly obě zařízení shodné přesnosti 100 %. V úloze detekce vady na výrobku dosahovala neuronová síť na Jetson Nano přesnosti 93,9

% (2 OK kusy s falešně detekovanými vadami, 0 NOK kusů označených jako dobrých) a síť naučená v prostředí Cognex ViDi Suite 96,9 % (1 OK kus s falešně detekovanými vadami, 0 NOK kusů označených jako dobrých). Lze tedy usoudit, že naučená neuronová síť pro detekci objektů ozkoušená na Jetson Nano dosahuje mírně nižší přesnosti než model naučený v prostředí Cognex Vidi Suite. Vzhledem k uživatelskému nastavení prahových hodnot pro vyhodnocení je však tento rozdíl v přesnosti velmi malý.

Výraznější byl rozdíl v době zpracování, které pro Jetson Nano přibližně 2 – 2,5krát delší, což je způsobeno především použitím hardware o rozdílném výkonu, který je popsán v předchozích odstavcích.

ZÁVĚR

V prvních dvou kapitolách je uveden přehled technických prostředků používaných ve strojovém vidění a základní metody detekce objektů v obraze, především je kladen důraz na metody využívající hluboké učení konvolučních neuronových sítí. Také je zde uveden přehled a srovnání vlastností architektur neuronových sítí používaných pro detekci objektů, z nichž byla následně pro svoje nízké nároky na paměť vybrána pro realizaci na Nvidia Jetson Nano architektura MobileNet v2.

Ve třetí kapitole je popsán vývojový kit Jetson Nano, jeho periferie a hardware specifikace. Také jsou popsány programové prostředky balíčku JetPack dodávaného výrobcem, především knihovny pro implementaci umělých neuronových sítí cuDNN a CUDA.

Čtvrtá kapitola je věnována nejvíce používaným frameworkům pro práci s hlubokými neuronovými sítěmi, zvláště je věnována pozornost knihovnám Tensorflow a Keras, které byly vybrány pro realizaci praktické části této práce. Dále je popsán význam detekčních modelů a popsány algoritmy modelů R-CNN a také SSD, který byl použit pro aplikaci detekce vad na výrobku v následujících kapitolách.

Kapitola číslo pět se již zabývá praktickým učením neuronových sítí, kterým jsou zde řešeny dvě úlohy pro kontrolu kvality ve výrobě. V rámci první úlohy je prováděna kontrola správného sestavení konektoru klasifikací obrazu, druhá úloha spočívá v detekci povrchových vad na šroubu. Síť pro klasifikaci byla učena na vzorku 240 OK a 240 NOK snímků a 50 OK a 50 NOK snímků ve validační množině pro ověření funkce. V případě aplikace detekce objektu v obraze, kde byl využit dataset z databáze MVTec bylo k dispozici 15 NOK snímků s vadnými výrobky pro trénování, validační dataset se pak skládal z 24 snímků OK výrobků a 9 snímků s přítomným vadným kusem. Jelikož je zařízení Jetson Nano z hlediska výkonu a paměti nedostatečné pro efektivní učení neuronových sítí, bylo použito cloudové prostředí pro vývoj v jazyce Python Google Colaboratory, které pro výpočty zdarma poskytuje GPU s řádově vyšším výkonem. V tomto prostředí byl pomocí příkazů knihovny Keras a přeneseného učení natrénován model již předučené sítě MobileNet v2 dle připravených snímků konektoru. Jako výchozí model sítě pro detekci povrchových vad byl vybrán model sítě MobileNet v2 FPNLite 640x640, jejíž přeučení bylo provedeno s využitím knihovny Tensorflow Object Detection API.

Dále je popsán proces implementace naučeného modelu sítě na zařízení Nvidia Jetson Nano. V úvodu kapitoly je uveden postup prvního spuštění a instalace potřebných knihoven,

jejichž verze jsou pro spolehlivou reprodukovatelnost uvedeny ve výpisu instalovaného programového vybavení v příloze. Následně byla provedena inference na validační množině snímků, která se stávala z 50 OK a 50 NOK obrázků pro úlohu klasifikace a 24 OK a 9 NOK obrázků pro úlohu detekce vad.

Kapitola 7 se zabývá komerčním systémem strojového vidění založeným na hlubokém učení neuronových sítí, Cognex ViDi Suite, ve kterém byly implementovány vizuální kontroly sestavení správného sestavení konektoru a detekce povrchových vad pro porovnání s výsledky obdrženými z inference na Jetson Nano. Využitým hardwarovým akcelerátorem pro zpracování obrazu byl použit osobní počítač s GPU Nvidia Quadro T1000. Výkon této grafické karty je násobně vyšší než výkon, kterým disponuje Jetson Nano, porovnání výkonu obou zařízení a také grafické karty doporučené pro aplikace s Cognex ViDi je uveden v Tabulce 6. Porovnání přesnosti klasifikace a detekce, stejně jako dobu potřebou pro zpracování jednoho snímku na obou zařízeních je uvedeno v Tabulce 7. Z něj vyplývá shodná přesnost při úloze kontroly správného sestavení konektoru klasifikaci obrazu pro obě zařízení, která dosahovala 100 %, přičemž zpracování jednoho snímku na Jetson Nano trvalo 233 ms a na výše uvedeném zařízení se systémem Cognex ViDi 89ms. Při detekci povrchových vad na šroubu vykazoval Jetson Nano mírně nižší přesnost detekce 94 % s dobou inference 420 ms oproti Cognex Vidi, které dosahovalo přesnosti 97 % s průměrným časem 190 ms na snímek.

Z porovnání výsledků z vypracovaných úloh pro klasifikaci obrazu a detekci objektů vyplývá, že embedded zařízení a komerční systém Cognex Vidi dosahovaly srovnatelné přesnosti při klasifikaci obrazu, nicméně v aplikaci detekce vad byla úspěšnost inference námi naučené sítě na Jetson Nano mírně nižší než na systému Cognex. Doba potřebná pro vyhodnocení jednoho snímku byla však v případě Jetson Nano přibližně 2 až 2,5krát delší.

Lze tedy předpokládat, že lze zařízení Jetson Nano použít v méně náročných aplikacích, kde doposud nebylo ekonomicky výhodné nasazení komerčního systému strojového vidění s hlubokým učením. Cena zařízení nvidia Jetson Nano v době vzniku této práce je 99 $, naproti tomu licence pro použití jedné kamery v systému Cognex ViDi cca 18 500 $. Jako zásadní problém hodnotím nedostatek paměti, kvůli kterému nelze na zařízení implementovat architektury neuronových sítí s větší hloubkou, s jejichž využitím by optická kontrola dosahovala vyšší přesnosti. Další skutečností limitující použití Jetson Nano v reálných průmyslových aplikacích je doba zpracování snímku, jenž je řádově vyšší než u komerčních systémů s použitím doporučených GPU.

Nasazení Nvidia Jetson Nano tak představuje cenově atraktivní alternativu komerčním systémům, avšak aplikace, ve kterých bude použita jsou zásadně omezeny složitostí inference a relativně dlouhým časem výrobního cyklu. Pro komerční využití v reálném provozu lze spíše doporučit některá z výkonnějších zařízení z řady Nvidia Jetson, disponující větším výkonem a pamětí, která jsou přitom stále cenově dostupnější než komerční systémy strojového vidění.

Neméně důležitou vlastností pro nasazení v reálné aplikaci v průmyslu, kterou však Jetson Nano postrádá, je přítomnost grafického rozhraní nejen pro vizualizaci výsledků inspekce, ale také umožňující jednoduchou možnost přeučení neuronové sítě dle zadaných snímků se změnou parametrů učení. Tato možnost je zásadní pro údržbu správné funkce zařízení a rychlou reakci na požadavky řízení kvality, například při změně výrobního procesu. Knihovny Tensorflow a Keras jsou však obsluhovány skripty v jazyku Python a nedisponují výše popsaným grafickým rozhraním, jakékoliv zásahy do naučené neuronové sítě tedy vyžadují znalost těchto knihoven, programovacího jazyka a prostředí Linux. Vytvoření aplikace, která by umožňovala výše zmíněnou údržbu zařízení například aplikačním inženýrem bez programátorských znalostí značně zvýší jeho užitnou hodnotu.

SEZNAM POUŽITÉ LITERATURY

[1] HAVLE, Otto. Strojové vidění: Principy a charakteristiky. In: Automa [online]. 2008 [cit. 2012-03-11]. Dostupné z: *http://www.odbornecasopisy.cz/index.php?id_document=36550* .

[2] KNÁPKOVÁ, Eva. Aplikace systémů strojového vidění, Zlín 2012, 75 s. Bakalářská práce. Univerzita Tomáše Bati, Fakulta aplikované informatiky, Ústav bezpečnostního inženýrství. Vedoucí práce: Ing. Jan Valouch, Ph.D.

[3] [Keyence Corporation]. In: Keyence.com [online]. [cit. 2022-03-01]. Dostupné z: *https://www.keyence.eu/cscz/products/vision/vision-sensor/iv/models/iv-hg500ma/*

[4] [Cognex Corporation]. In: Cognex.com [online]. [cit. 2022-03-01]. Dostupné z: https://www.cognex.com/products

[5] *Opto Engineering basics* [online]. Mantova, Italy., 2021 [cit. 2021-03-07]. Dostupné z: *https://www.opto-e.com/basics*

[6] SOLEM, J. E. Programming Computer Vision with Python: Tools and algorithms for analyzing images, O'Reilly Media, 2012, ISBN 978-1449316549

[7] JANŠTOVÁ, Michaela. Segmentace měkkých tkání v obličejové části myších embrií v mikrotomografických datech. Brno, 2019, 82 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav biomedicínského inženýrství. Vedoucí práce: Ing Chmelík Jiří.

[8] DAVIES, E. R. Computer and machine vision: theory, algorithms, practicalities. 4th ed. Boston: Elsevier, 2012. ISBN 978-0123869081

[9] Martin Sedlář, Jaromír Štámek, Ondřej Ráček, Vojtěch Mornstein: Získávání a analýza obrazové informace. Učební text LF MU, 2011. online: *http://www.med.muni.cz/biofyz/Image/ucebnice.pdf*

[10] CAGAŠ, Roman. Identifikace objektů v prostředí strojového vidění VisionLab, In: [online]. Praha: Automa, 2015 [cit. 2022-02-06]. Dostupné z: *http://www.mii.cz/download/company/articles/Automa_08_2015.pdf*

[11] JANÁK, Zdeněk. Klasifikace světelných křivek s pomocí neuronových sítí, Brno 2012, 54 s, Diplomová práce. Masarykova Univerzita, Přírodovědecká fakulta, Ústav teoretické fyziky a astrofyziky. Vedoucí práce: Mgr. Viktor Votruba, Ph.D.

[12] GOODFELLOW, I. BENGIO, Y. COURVILLE, A. Deep learning. Cambridge, Massachussetts: MIT Press, 2016, ISBN 978-0262035613

[13] BLAHA, Milan. Umělá inteligence [online]. [cit. 2022-02-08]. Dostupné z: *https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence*

[14] Koncept umělé neuronové sítě [online]. [cit. 2022-02-08]. 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] MITRENGA, M. Konvoluční neuronová síť pro segmentaci obrazu. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. 55s. Vedoucí bakalářské práce doc. Ing. Václav Jirsík, CSc..

[16] FERGUS, R. Neural Networks: Facebook AI Research Machine Learning Summer School 2015 [cit. 2022-04-13]. Dostupné z: *http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf* .

[17] HORÁK, Karel. Introduction to Convolutional Neural Networks [pdf]. VUT, 2017 [cit. 2022-02-08]. Dostupné z: *http://midas.uamt.feec.vutbr.cz/ROZ/roz_cz.php*

[18] LECUN, Y., BOTTOU, L., BENGOI, Y., HAFFNER, P. Gradient-Based Learning Applied to Document Recognition [online]. Listopad 1998 [cit. 2022-04-14]. Dostupné z: *http://.vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf*

[19] KRIZHRVSKY, A., SUTSKEVER, I., HINTON, G., ImageNet Classification with Deep Convolutional Neural Networks [online]. 2012 [cit. 2022-04-14]

[20] KAIMING, H., XIANGIU, Z., SHAOQUING, R., JIAN, S. Deep residual learning for image recognition [online]. 2015 [cit. 15-04-2022]. Dostupné z: *https://arxiv.org/pdf/1512.03385.pdf*

[21] HOLLEMANS, M. MobileNet version 2 [online]. Duben 2018 [cit. 2022-04-13]. Dostupné z: *https://machinethink.net/blog/mobilenet-v2/* .

[22] SANDLER, M., HOWARD, A., ZHU, M., ZHMOGINOV, A. a CHEN, L.-C. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019.

[23] Nvidia Corporation. Nvidia TensorRT [online]. [cit. 2022-04-02]. Dostupné z: *https://developer.nvidia.com/tensorrt*

[24] GANDHI, R. R-CNN, Fast R-CNN, Faster R-CNN, YOLO-Object Detection Algorithms [online]. [cit. 2022-04-10]. Dostupné z: *https://www. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e*

[25] CHOLLET, F. Deep learning v jazyku Python, Grada, 2019, 978-8024731001

[26] ROSEBROCK, A. Deep learning for computer vision with Python, PyimageSearch, 2017, ISBN 978-1986538138

[27] FORSYTH, D. PONCE, J. 2011. Computer Vision: A Modern Approach (2nd Edition). Pearson. ISBN-978-9332550117

[28] MUSIL, Petr. Klasifikace dat metodou SVM, Praha 2020, 24 s. Bakalářská práce. České Vysoké Učení Technické, Fakulta strojní, Ústav přístrojové a řídící techniky. Vedoucí práce: Ing. Vladimír Hlaváč, Ph.D.

[29] Framework Torch: využití konvolučních sítí pro rozpoznávání a klasifikaci obrázků [online]. 14.12.2017 [cit. 2022-02-05]. Dostupné z: *https://www.root.cz/clanky/framework-torch-vyuziti-konvolucnich-siti-pro-rozpoznavani-a-klasifikaci-obrazku/*

[30] CS231n Convolutional Neural Networks for Visual Recognition [online]. 2017 [cit. 2017-12-30]. Dostupné z: *http://cs231n.github.io/convolutional-networks/*

[31] RANJAN, Chitta. Understanding the Kernel with fundamentals [online]. 2019 [cit. 2022-02-07]. Dostupné z: *https://towardsdatascience.com/truly-understanding-the-kernel-trick-1aeb11560769*

[32] DE LUCA, Gabriele. *SVM vs Neural Network* [online]. [cit. 2022-01-10]. Dostupné z: *https://www.baeldung.com/cs/svm-vs-neural-network*

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

OK Z anglického „Okay“. V průmyslové výrobě označuje kusy, které mají požadované vlastnosti.

NOK Z anglického „Not Okay“. V průmyslové výrobě označuje kusy, které nemají požadované vlastnosti.

PC Z anglického „Personal Computer“, osobní počítač

OCR Z anglického „Optical Character Recognition“. Soubor metod pro strojové rozpoznávání tištěného písma.

CMOS Z anglického „Complementary Metal-Oxide-Semiconductor“. Technologie pro výrobu logických integrovaných obvodů

CCD Z anglického „Charged-coupled device“. Technologie pro snímání obrazu

SVM Support Vector Machines – Metoda podpůrných vektorů užívaná pro klasifikaci vstupních dat

CNN Z anglického „Convolutional neural networks“. Konvoluční neuronové sítě

NN Neural networks – Neuronové sítě

UI Umělá inteligence

API Application Programming interface (rozhraní pro programování aplikací)

BSD Berkley Software Distribution (licence pro svobodný software vytvořená na University of California)

CPU Central processing unit (centrální procesorová jednotka)

GPU Graphics Processing Unit (grafický procesor)

ReLU Rectified Linear Unit (usměrněná lineární aktivační funkce)

CUDA Compute Unified Device Architecture (architektura jednotného výpočetního zařízení)

cuDNN NVIDIA CUDA Deep Neural Network (GPU akcelerovaná knihovna funkcí pro implementaci hlubokých neuronových sítí)

BVLC Berkley Vision and Learning Center – centrum vývoje umělé inteligence na uni-

verzitě v Berkley

BSD Berkley Software Distribution, také název distribuce OS Unix vydaný univerzitou v Berkley

SSD Z anglického „Single Shot Multibox Detector“ Algoritmus pro detekci objektů v obraze pomocí neuronových sítí.

SSH Z anglického „Secure Shell“. Zabezpečený komunikační protokol.

HDMI Z anglického „High-Definition Multimedia Interface“. Standard pro přenos nekomprimovaného obrazového signálu.

API Z anglického „Application Programming Interface“. Rozhraní pro programování aplikací.

FLOPS Z anglického „Floating-Point Operations Per Second“. Počet operací v plovoucí řádové čárce za sekundu, obvyklé měřítko výkonosti výpočetní techniky.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH PŘILOŽENÉHO DATOVÉHO NOSIČE

Přiložené DVD obsahuje obrázky použité pro tvorbu datasetů k úlohám popsaným v této práci. Vzhledem k jejich velkému počtu jsou uvedeny pouze jejich kořnové adresáře.

/……kořenový adresář přiloženého DVD
Nvidia Jetson Nano…….pracovní adresář použitý při práci s zařízením
detekce……...……pracovní adresář pro úlohu detekce objektů
annotations……..……adresář s anotačním souborem label_config.pb
exported-models……..…..adresář s modelem použitým pro inferenci
images……adresář se snímky použitými pro úlohu detekci objektů
test……...……testovací množina snímků
train……...…….trénovací množina snímků
validation……validační množina snímků
detekce.py…….skript pro inferenci na validační množině
klasifikace……...……..pracovní adresář pro úlohu klasifikace obrazu
exported-models……..…..adresář s modelem použitým pro inferenci
images……...……adresář se snímky použitými pro úlohu klasifikace obrazu
test……..……testovací množina snímků
OK…….……..OK snímky z testovací množiny
NOK…….NOK snímky z testovací množiny
train…….…….trénovací množina snímků
OK…….……..OK snímky z trénovací množiny
NOK…….NOK snímky z trénovací množiny
validation……..validační množina snímků
OK…….……..OK snímky z validační množiny
NOK…….NOK snímky z validační množiny
klasifikace.py……..…….skript pro inferenci na validační množině