Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Systém pro detekci objektů v obrazových datech

Diplomová práce

2022 Bc. Vítek Rais

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Vítek Rais**
Osobní číslo:	**I20214**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Systém pro detekci objektů v obrazových datech**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Postup: Cílem práce je návrh a implementace systému pro detekci objektů v obrazových datech pomocí hlubokých neuronových sítí. Student v rámci práce vytvoří zařízení pro snímání a vyhodnocování obrazových dat a toto zařízení vybaví softwarem zajištění požadovaných funkcionalit. Zařízení bude minimálně umožňovat detekci a lokalizaci jednoho typu objektů ve statických obrazových datech.
Teoretická část: Stručná rešerše existujících nástrojů pro detekci a lokalizaci objektů v obrazových datech založených na metodách umělé inteligence. Popis senzorové techniky pro sběr dat. Metodika sběru dat. Popis softwarových nástrojů použitých pro řešení praktické části.
Praktická část: Návrh a implementace zařízení pro sběr a vyhodnocování dat. Sběr a analýza dat. Návrh a implementace softwarového nástroje pro zařízení založeného na vybraném paradigmatu umělých neuronových sítí. Komplexní testování aplikace. Dokumentace k aplikaci včetně testovacího scénáře demonstrujícího použití.

Rozsah pracovní zprávy: **cca 60 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

CHOLLET, François. *Deep learning v jazyku Python: knihovny Keras, Tensorflow*. Přeložil Rudolf PECINOVSKÝ. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.
GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning*. Cambridge, Massachusetts: The MIT Press, [2016]. ISBN 978-02-620-3561-3.

Vedoucí diplomové práce: **doc. Ing. Petr Doležel, Ph.D.**
Katedra řízení procesů

Datum zadání diplomové práce: **8. listopadu 2021**
Termín odevzdání diplomové práce: **20. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2021

Prohlašuji:

Práci s názvem Systém pro detekci objektů v obrazových datech jsem vypracoval(a) samostatně. Veškeré literární prameny a informace, které jsem v práci využil(a), jsou uvedeny v seznamu použité literatury.

Byl(a) jsem seznámen(a) s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 25. 8. 2022

Bc. Vítek Rais v. r.

PODĚKOVÁNÍ

Rád bych poděkoval doc. Ing. Petru Doleželovi, Ph.D. za odborné vedení, za pomoc a podnětné rady při zpracování této práce.

ANOTACE

Tato práce se věnuje vytvoření detektoru přesné polohy objektů v reálném čase, jehož výstupy jsou dostatečně přesné pro následnou robotickou manipulaci s těmito objekty. Nejprve jsou představeny základní úlohy počítačového vidění a architektura konvolučních neuronových sítí. Následuje rešerše moderních metod pro detekci a klasifikaci objektů založených na hlubokých neuronových sítích. Praktická část se věnuje návrhu a implementaci samotného systému, který propojuje detekční model YOLOv5 a klasifikační EfficientNet.

KLÍČOVÁ SLOVA

počítačové vidění, konvoluční neuronové sítě, detekce objektů, YOLOv5, EfficientNet

TITLE

Object Detection in Visual Data

ANNOTATION

The present thesis focuses on the creation of a detector of precise location of objects in real time. The outputs of this detector are precise enough for the subsequent robotic manipulation with these objects. First, the text presents the basic tasks of the computer vision and the architecture of the convolutional neuron networks. Second, it researched modern methods for detection and classification of objects that are based on deep neuron networks. The practical part is devoted to the design and implementation of the system itself, which links the YOLOv5 detection model and EfficientNet classification model.

KEYWORDS

computer vision, convolutional neural networks, object detection, YOLOv5, EfficientNet

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

AP – average precision

BN – batch normalization

CNN – convolutional neural network

FAIR – Facebook AI Research

FLOPs – floating point operations

FN – false negative

FPN – Feature Pyramid Network

FPS – frames per second

FP – false positive

IDE – Integrated development environment

IoU – Intersection over Union

mAP – mean average precision

MBConv – mobile inverted bottleneck

PyPI – Python Package Indexu

R-CNN – Region-based Convolutional Neural Network

ReLU – Rectified Linear Unit

ResNet – residual network

RoI – region of interest

RPN – region proposal network

SSD – Single Shot Multibox Detector

Tanh – Hyperbolický tangens

TPU – Tensor Processing Unit

TP – true positive

VGG – Visual Geometry Group

Xception – Extreme Inception

XLA – Accelerated Linear Algebra

YOLO – You Only Look Once

ÚVOD

Jedním z výrazných trendů posledních let je zavádění robotizace do průmyslové výroby. Přesto stále existuje mnoho procesů, kde je velice problematické tuto automatizovanou výrobu zavést. To je velmi často způsobeno odlišnou podstatou člověka a stroje, kdy se člověk dokáže přizpůsobit jistým odchylkám, kdežto stroj pro své správné fungování typicky potřebuje znát naprosto přesnou polohu výrobků, se kterými manipuluje.

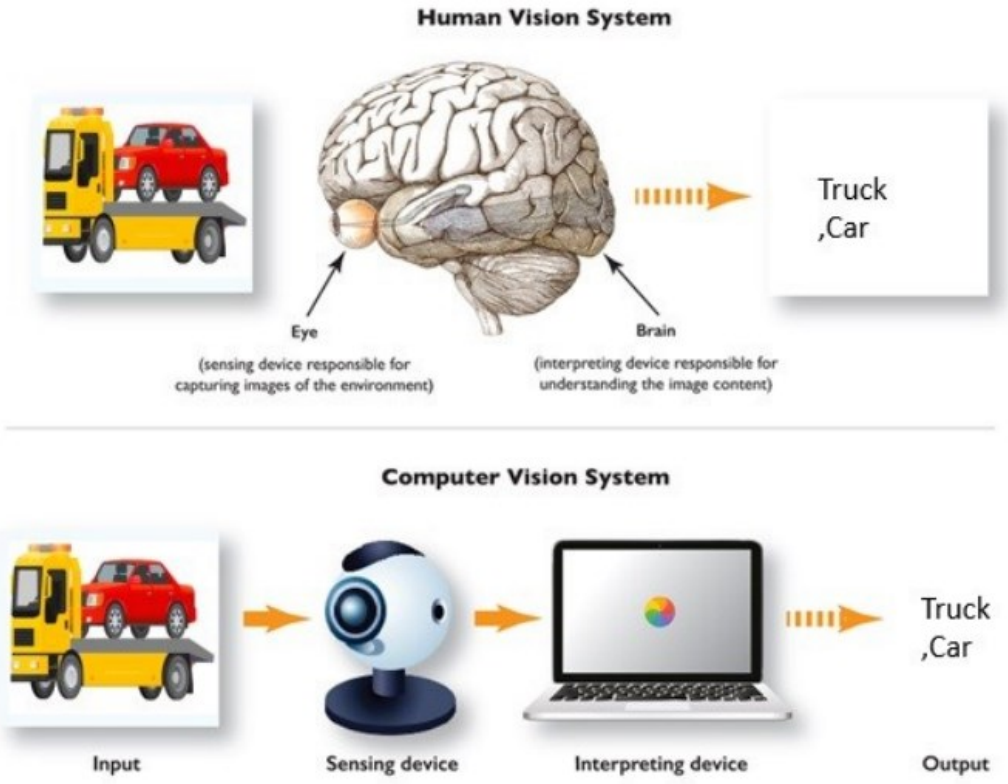
K odstranění problémů s nepřesnostmi umístění jednotlivých výrobků lze využít algoritmy strojového vidění, které zažívají v posledních letech velmi bouřlivý vývoj. Jejich rychlost a přesnost již dosahují takových hodnot, že je s nimi experimentováno i ve velmi náročných úlohách jako je např. autonomní řízení aut. Tento velmi rychlý rozvoj je způsoben značným pokrokem v oblasti hlubokých neuronových sítí, mezi kterými hrají hlavní roli především konvoluční neuronové sítě.

Cílem praktické části této práce je návrh a vytvoření systému, který bude přesně detekovat polohu objektů z obrazových dat. Nebude určována jen poloha, kde se daný objekt nachází, ale i plocha, na které objekt leží, a dále také úhel otočení objektu od horizontální osy při pohledu shora. Tyto údaje by měly být natolik přesné, aby mohly sloužit jako vstup do systému řídicímu výrobní stroj či manipulátor, který bude s těmito objekty dále pracovat. Další důležitou vlastností systému je schopnost poskytovat tyto údaje v reálném čase, tedy dostatečně efektivně, aby systém nezpožďoval celý výrobní proces V souladu s aktuálním stavem vývoje technologií v této oblasti budou využity algoritmy založené na hlubokých neuronových sítích.

V teoretické části budou představeny základní úlohy počítačového vidění. Dále bude zmapován aktuální stav vědy a techniky v oblastech věnujícím se detekci a klasifikaci objektů v obrazových datech, včetně základních metrik používaných k jejich hodnocení. Z této rešerše budou následně vybrány nejvhodnější algoritmy pro tvorbu praktické části práce. Dále budou představeny principy a základní stavební kameny konvolučních neuronových sítí, na kterých jsou založeny téměř všechny moderní algoritmy v těchto oblastech.

1 POČÍTAČOVÉ VIDĚNÍ

Pojmem „počítačové vidění“ se obvykle vyjadřuje oblast umělé inteligence či strojového učení, která se zaměřuje na zpracování obrazových dat (videa, obrázky), jež se snaží interpretovat či z nich získat co nejvíce relevantních informací. Obvyklým cílem je získat podobné informace o viděných objektech jako poskytuje lidský mozek, případně i informace podrobnější. Celý proces počítačového vidění ostatně poměrně přesně kopíruje proces vidění lidského, jak je vidět na obrázku 1. [1]

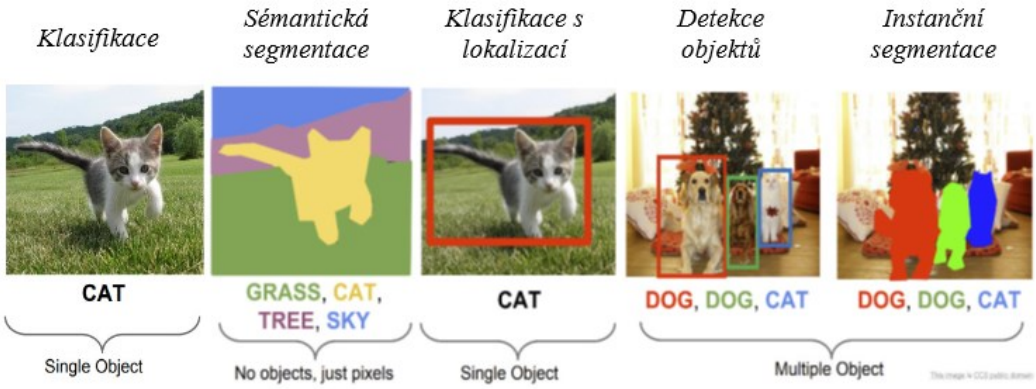


Obrázek 1 – Porovnání principu lidského a počítačového vidění, zdroj: [2]

Úloh počítačového vidění je celá řada: těmi základními jsou *klasifikace*, *klasifikace s lokalizací*, *detekce objektů*, *sémantická* a *instanční segmentace*. Klasifikace obrazových dat je nejjednodušší úlohou počítačového vidění, ve které jde o zařazení obrázku do určité třídy podle zobrazeného objektu. Při klasifikaci s lokalizací je výstupem nejen třída, ale i vektor umístění daného objektu v obrázku. Tento vektor je obvykle ve tvaru (x, y, w, h), kde x a y jsou souřadnice středu, w je šířka a h je výška rámečku ohraničující daný objekt. [3], [4]

Podobnou úlohou je i detekce objektů, ve které jde taktéž o určení třídy a polohy objektů. Oproti klasifikaci s lokalizací se při detekci může v jednom obrázku vyskytovat více objektů stejných či různých tříd. [3]

Sémantická segmentace je poněkud odlišnou metodou zpracování obrazu, jejím principem je zařazení každého pixelu na obrázku do příslušné třídy. Nejsou zde rozlišovány různé instance stejné třídy. Instanční segmentace je té sémantické velmi podobná, ovšem zde jsou již od sebe odlišovány jednotlivé instance objektů patřících do stejné třídy. [3]



Obrázek 2 – Základní úlohy počítačového vidění, zdroj: [4]

V této práci je využito hned dvou metod, a to detekce objektů pro zjištění polohy jednotlivých součástek a klasifikace pro zjištění jejich úhlu otočení. Problematika těchto metod je podrobněji probírána v kapitolách 3, resp. 4. Původně bylo uvažováno o využití metod instanční segmentace, ovšem vzhledem k daleko složitější přípravě datové sady a zároveň jejich výrazně nižší rychlosti zpracování obrazu od nich bylo upuštěno.

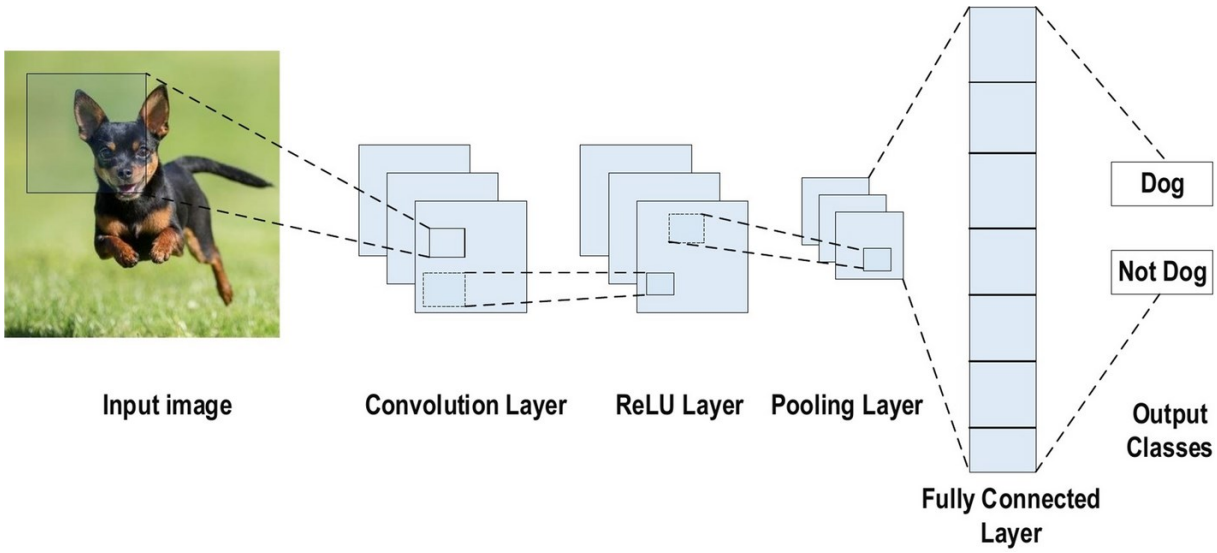
Počítačové vidění není otázkou pouze posledních let. První systémy pro počítačové vidění byly navrženy již v 60. letech 20. století. Tyto projekty ovšem obvykle neskončily úspěchem. Posun nastal až na začátku 80. let, kdy Kunihiko Fukushima navrhl *Neocognitron* [5], první konvoluční neuronovou síť. Za první moderní konvoluční neuronovou síť lze považovat *LeNet-5* [6] představený v roce 1989. Dalším milníkem byl rok 2001, kdy Paul Viola a Michael Jones představili první framework pro detekci tváří pracující v reálném čase. [7] Tento framework ovšem není založený na architektuře konvolučních neuronových sítí. [8]

S postupným vývojem nových algoritmů bylo zapotřebí vytvořit veřejně dostupná datová sada sloužící jako benchmark pro tyto algoritmy. Mezi nejznámější patří například *Pascal VOC* [9] či *ImageNet* [10], na jehož základě se od roku 2010 koná každoroční soutěž. V prvních dvou letech se pohybovala chybovost (*top-5*, více viz kapitola 3) nejlepších algoritmů okolo 26 %. Rok 2012 ovšem znamenal revoluci. Vědci z univerzity v Torontu přihlásili do soutěže konvoluční neuronovou síť *AlexNet* (viz kapitola 3.1), která vyhrála s na tu dobu ohromující přesností detekce (chybovost 16,4 %). Od té doby jsou všechny úspěšné modely postaveny na základě konvolučních neuronových sítí, přičemž dnešní modely vykazují chybovost v řádu nízkých jednotek procent. [8]

2 KONVOLUČNÍ NEURONOVÉ SÍTĚ

Konvoluční neuronové sítě, zkráceně *CNN* (z anglického *convolutional neural network*), jsou zvláštním případem neuronových sítí využívaným pro zpracování dat s mřížkovou strukturou známé velikosti. Například se může jednat o zpracování signálů nebo obrázků. Tento typ sítí se v posledních letech velice dobře uplatňuje v praktických aplikacích v oblastech, jako je například strojové vidění nebo rozpoznávání řeči. Jeho největší výhodou je automatická identifikace důležitých vlastností bez asistence člověka. [9], [12]

Architektura těchto sítí se liší v závislosti na typu řešené úlohy. Typicky se ovšem začíná vstupní vrstvou, která vstupní data nijak netransformuje. Následuje série konvolučních vrstev, mezi kterými jsou aktivační vrstvy. Mezi konvolučními vrstvami (typicky jen některými, ne všemi) se ještě vyskytují *pooling* vrstvy. Na konci sítě se nachází plně propojené vrstvy, které jsou stejné jako v případě plně propojených dopředných vícevrstvých neuronových sítí. [12]

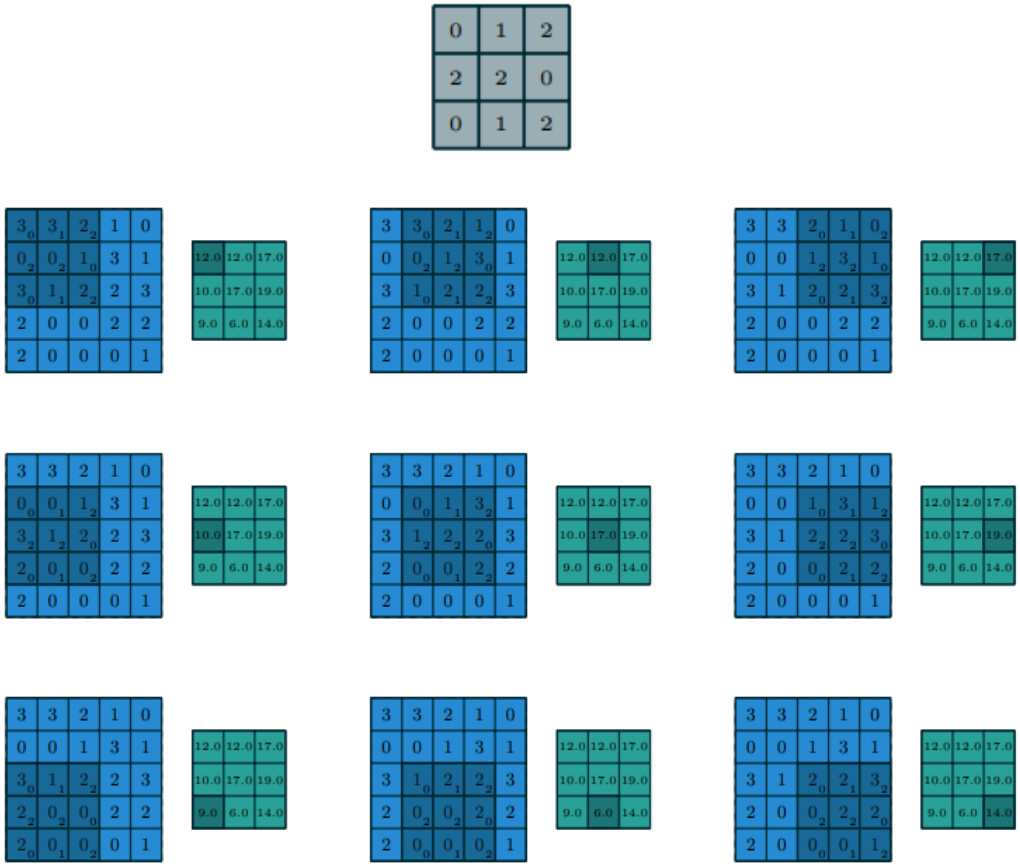


Obrázek 3 – Ukázka architektury jednoduché CNN pro klasifikaci obrázků, zdroj: [12]

2.1 Konvoluční vrstva

Konvoluční vrstvy jsou základem těchto sítí. Nesou název podle matematické operace *konvoluce*, kterou využívají. Zjednodušeně se jedná o aplikaci *filtru* (někdy také označován jako *jádro* či *kernel*) na vstupní data. Filtr je obvykle výrazně menší než vstupní data. Hodnoty filtru se v oblasti neuronových sítí obvykle nazývají *váhy*. Aplikací je myšleno vynásobení jednotlivých překrývajících se hodnot jádra a vstupních dat, které jsou následně sečteny, čímž vznikne jedna hodnota. Filtr je postupně posouván přes vstupní data. Takto vzniklá matice hodnot je výstupem operace. Výstupní matice dat se obvykle označuje jako *příznaková mapa* nebo anglickým *feature map*. [13]

Podle definice konvoluce by měl být ještě filtr před každou aplikací přetočen. V oblasti neuronových sítí se obvykle toto přetáčení vynechává. V takovém případě se používaná operace správně nazývá *cross-correlation*, ale typicky je stále označována jako konvoluce. Stejně tomu je i v případě této práce. [14]



Obrázek 4 – Příklad konvoluce, šedá barva – jádro, modrá – vstupní data, zelená – příznaková mapa, zdroj: [13]

Na obrázku 4 lze vidět příklad 2-D konvoluce, ale tato operace může být zobecněna na *n* dimenzí. Cílem operace je extrakce nějaké významné vlastnosti ze vstupních dat. Konvoluční vrstva obvykle obsahuje více než jeden filtr, typicky 32 až 512. Tyto filtry jsou paralelně aplikovány na vstupní data a tím vzniká adekvátní množství příznakových map, a tedy je extrahováno větší množství vlastností. [13], [14]

Velmi důležitým parametrem konvoluce je velikost kroku při posouvání filtru, obvykle označovaný anglickým výrazem *stride*. Čím větší je stride, tím bude výstupní příznaková mapa menší a operace rychlejší, ovšem za cenu ztráty detailů. S výpočetním výkonem současných nejmodernějších strojů se proto obvykle využívá stride o velikosti 1. [15]

Dalším parametrem ovlivňujícím velikost výstupu je *padding*, který vyjadřuje počet nul přidaných na okraje vstupních dat. Typicky je tento parametr nastaven na hodnotu 0, což je

označováno jako *valid padding*, nebo na takovou hodnotu, aby se rovnala velikost vstupu a výstupu, což je označováno jako *same padding*. Ten se vypočítá podle vzorce $P = \frac{f-1}{2}$, kde f je rozměr filtru. Same padding má výhodu v zachování informací vyskytujících se na okraji vstupních dat, které jsou v případě valid paddingu zanedbány. [15]

Konvoluční neuronová síť obvykle obsahuje více než jednu konvoluční vrstvu. To znamená, že vstupem konvolučních vrstev nemusí být pouze surová data, ale i již předzpracovaná data jinými (konvolučními) vrstvami. To umožňuje hierarchickou dekompozici vstupu. První vrstvy tedy extrahují vlastnosti na nízké úrovní abstrakce, jako jsou jednotlivé hrany a přechody, zatímco poslední vrstvy pracují na vysoké úrovni abstrakce, a tedy zpracovávají komplexní útvary jako jsou například oči či ústa. [14]

2.2 Aktivační vrstva

Tato vrstva plní roli aktivační funkce, známé z klasických neuronových sítí. V konvolučních neuronových sítích je tato funkce často extrahována do zvláštní vrstvy. V takovém případě typicky následuje po každé vrstvě obsahující váhy (konvoluční, plně propojené vrstvy). Obvykle využívané aktivační funkce v konvolučních neuronových sítích jsou:

- *Sigmoid*, která je využívána převážně po plně propojených vrstvách. Výstupem funkce jsou reálná čísla od 0 do 1 a lze ji matematicky zapsat jako

$$f(x) = \frac{1}{1 + e^{x}} \tag{1}$$

- *Hyperbolický tangens* (*Tanh*) se také používá převážně po plně propojených sítích, a je tedy alternativou k předchozí funkci. Hlavní rozdíl je v tom, že výstupy z této funkce jsou v rozmezí -1 až 1 a matematický zápis je

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2}$$

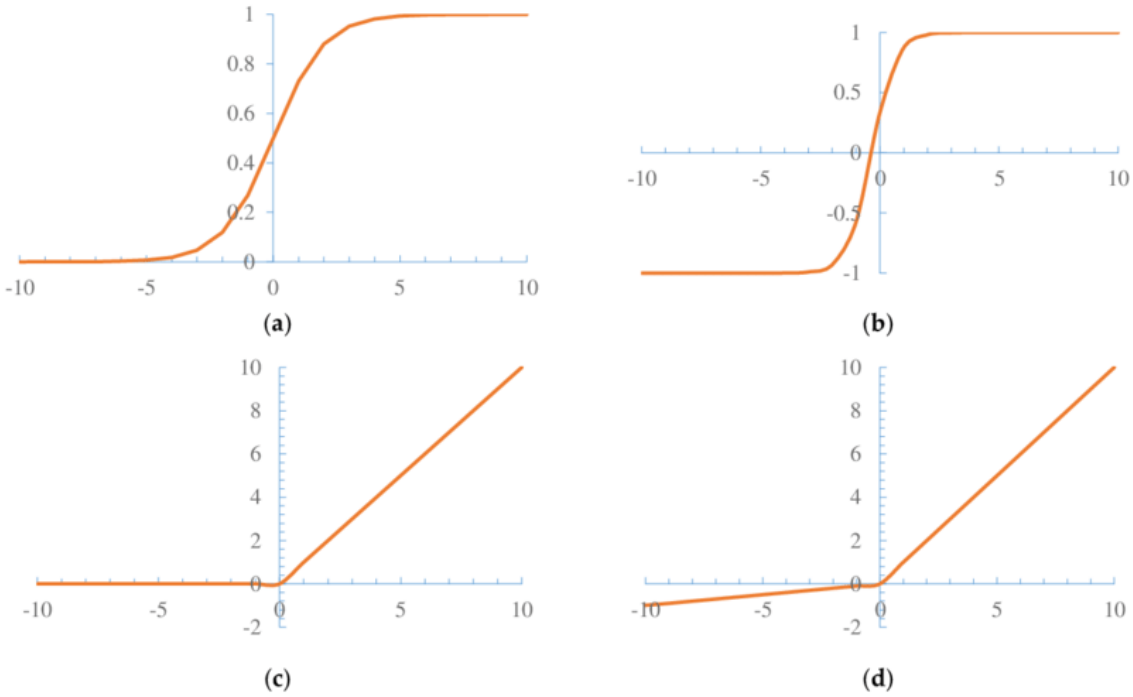
- *ReLU* (*Rectified Linear Unit*), která je velmi jednoduchou funkcí typicky využívanou po konvolučních vrstvách. Výhodou této funkce je krátký výpočetní čas. Lze ji zapsat jako

$$f(x) = \max(0, x) \tag{3}$$

- *Leaky ReLU,* která je modifikací předchozí funkce. Oproti ní částečně zachovává i negativní hodnoty, takže žádné vstupy nejsou ignorovány. Zápis této funkce je

$$f(x) = \begin{cases} x & pokud\ x > 0 \\ mx & pokud\ x \leq 0 \end{cases}, \tag{4}$$

kde *m* je *leak faktor*. Typicky se jedná o velmi malé číslo jako je 0,001. [12]

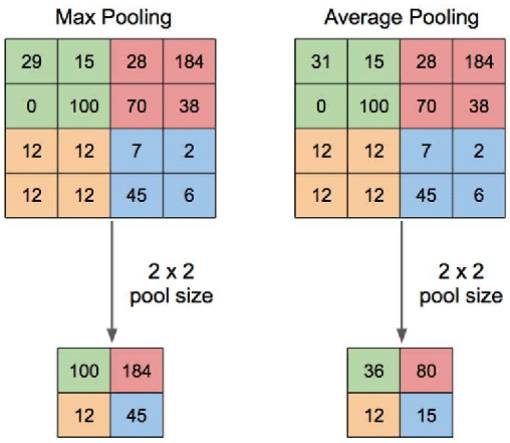


Obrázek 5 – Průběh aktivačních funkcí. (a) Sigmoid, (b) Tanh, (c) ReLU, (d) Leaky ReLU, zdroj: [16]

2.3 Pooling vrstva

Tato vrstva obvykle následuje po konvoluční vrstvě a má za cíl zmenšit velikost vstupních dat, čímž zmenšuje časovou i paměťovou náročnost dané sítě. Dalším benefitem této vrstvy je částečné odstranění závislosti jednotlivých vlastností ve vstupu na jejich pozici. [9]

Základními parametry této operace jsou velikost *okna* (někdy též označováno jako *filtr* či *pool size*) a *stride,* který určuje velikost kroku, se kterým je okno posouváno přes vstupní data. Při každém kroku hodnoty v daném okně agregovány do jedné hodnoty. Jako agregační funkce se typicky používá *max pooling*, při kterém je vybrána maximální hodnota v daném okně, nebo *average pooling*, při které je z daných hodnot vypočítán aritmetický průměr. [17]



Obrázek 6 – Ukázka max a average poolingu, zdroj [18]

3 ALGORITMY PRO KLASIFIKACI OBJEKTŮ

Tato kapitola představuje výběr z nejznámějších a nejúspěšnějších algoritmů pro klasifikaci objektů posledních let. Všechny tyto algoritmy jsou založeny na principu konvolučních neuronových sítí. Hodnoceny jsou na základě několika důležitých metrik.

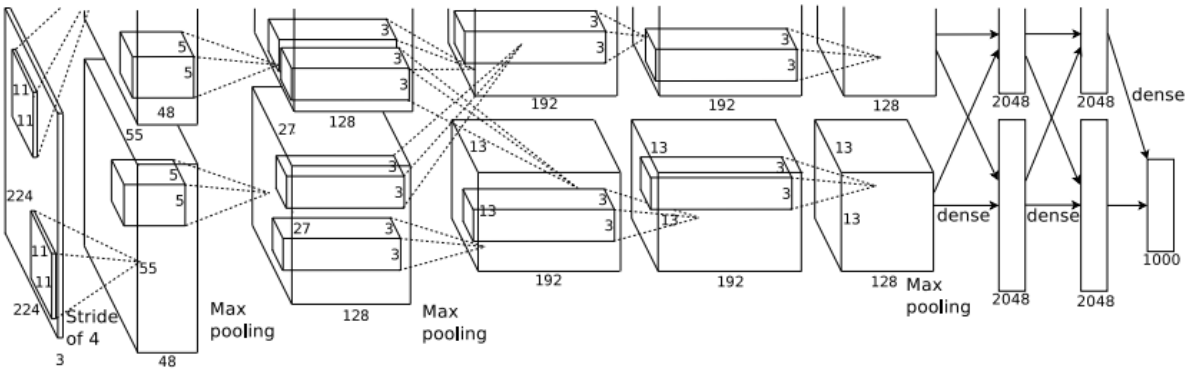
Tou nejdůležitější je přesnost klasifikace. Výstupem klasifikačního algoritmu je obvykle výčet tříd a pravděpodobnost, s jakou by mohl vstupní obrázek do každé z nich spadat. Typicky se udává tzv. *top-1* nebo *top-5* přesnost vyhodnocena na známé veřejně dostupné datové sadě jako je ImageNet [10], nebo Pascal VOC [9]. Za úspěšnou klasifikaci je v případě top-1 přesnosti považováno, pokud se shoduje skutečná a nejpravděpodobnější předpovězená třída. V případě top-5 přesnosti stačí, pokud se skutečná třída vyskytuje mezi pěti nejpravděpodobnějšími variantami. [19]

Dalším důležitým parametrem je velikost dané konvoluční neuronové sítě. Nejčastěji se udává ve formě počtu parametrů, které lze v rámci sítě trénovat, nebo jako počet *operací s plovoucí řádovou čárkou* (*floating point operations*), značené jako *FLOPs*, při klasifikaci jednoho obrázku. Velmi často se také udává informace o hloubce neuronové sítě, z této informace ovšem nelze odvodit velikost dané sítě. S velikostí sítě také úzce souvisí i její rychlost, která se typicky udává jako doba (v ms), za kterou je provedena klasifikace jednoho obrázku.

3.1 AlexNet

AlexNet je pravděpodobně nejznámější model konvoluční neuronové sítě, který se zasloužil o široké přijetí tohoto typu neuronové sítě a zároveň velmi rozšířil zájem o oblast počítačového vidění a umělé inteligence jako takové. Byl představen týmem vědců z univerzity v Torontu v roce 2012. [8]

Tato síť se skládá celkem z pěti konvolučních a tří plně propojených vrstev. Dále síť obsahuje tři max pooling vrstvy, které se nacházejí za první, druhou a pátou konvoluční vrstvou. Jako aktivační funkce všech vrstev (kromě poslední plně propojené, kde je to funkce softmax) byla zvolena funkce ReLU, při které jsou dosaženy výrazně lepší výsledky než s dříve používanou funkcí tanh. [20]

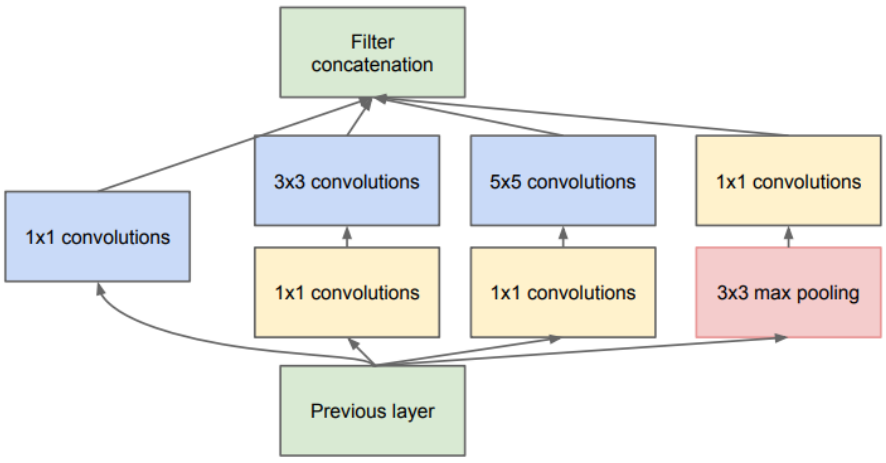


Obrázek 7 – Architektura sítě AlexNet, zdroj: [20]

3.2 Inception V1 (GoogLeNet)

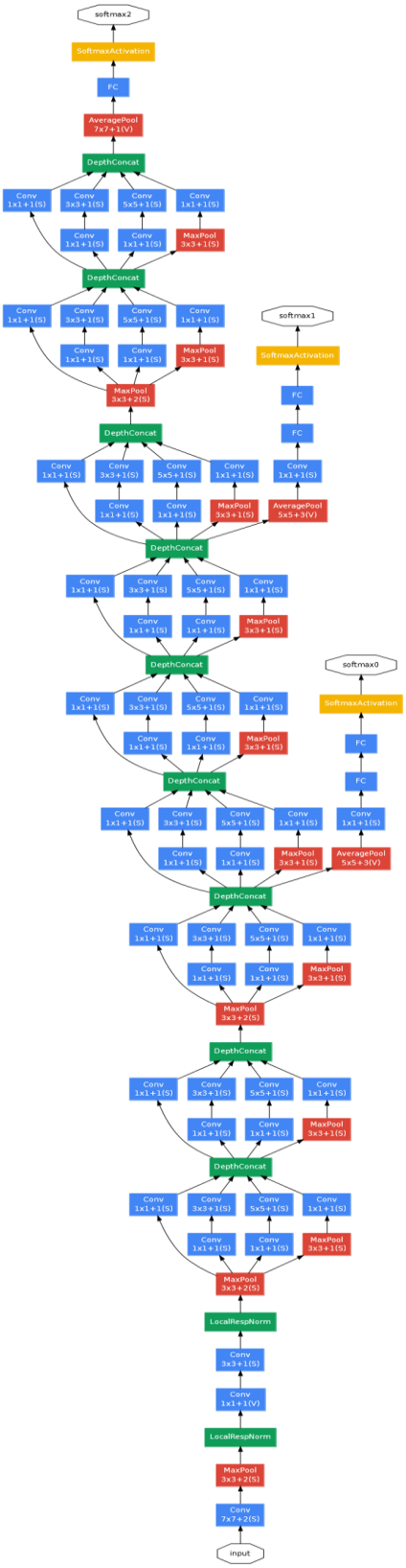
Architekturu Inception a od ní odvozený model GoogLeNet vytvořili výzkumníci ze společnosti Google v roce 2014. Jedná se o velmi hlubokou síť, model GoogLeNet obsahuje hned 22 vrstev. Z toho důvodu byla architektura pojmenována na základě internetového memu „We need to go deeper“ [21]. Název modelu GoogLeNet odkazuje na model LeNet-5 [6], první moderní CNN. Tento model dosáhl při klasifikaci datové sady ImageNet na svou dobu rekordně nízké chybovosti [22]

I přes velký počet jednotlivých vrstev je zachována poměrně malá velikost modelu. Počet parametrů sítě je 12× menší než v případě modelu AlexNet. Architektura sítě je založena na tzv. „Inception modules“. Ty zapouzdřují několik konvolučních vrstev s různou velikostí filtru a také max pooling vrstvu. Pomocí filtrační vrstvy jsou poté výstupy z těchto vrstev spojeny, aby mohly být vstupem pro další modul. [22]



Obrázek 8 – Architektura Inception modulu, zdroj: [22]

Model GoogLeNet obsahuje hned 9 těchto modulů. Aby byla zachována přiměřená velikost parametrů, obsahuje model před těmito moduly nejprve několik standardních konvolučních a poolingových vrstev, které významně redukují celkový počet parametrů. [22]



Obrázek 9 – Architektura GoogLeNet, zdroj: [22]

3.3 VGG

VGG je architektura konvoluční neuronové sítě vytvořená v roce 2014 na univerzitě v Oxfordu. Název architektury je zkratkou z Visual Geometry Group, což je název týmu, který tuto architekturu navrhl. Bylo představeno hned pět variant této sítě, široce používané jsou ovšem převážně dvě nejsložitější, v originále označované jako varianty *D* a *E*. V praxi se pro ně ovšem spíše vžilo označení VGG-16, resp. VGG-19, podle počtu vrstev s vahami. [23]

Architektura sítě je založena na použití konvolučních vrstev s malými filtry, typicky (3 × 3). Dalším znakem je poměrně malý počet filtrů, konkrétně 64, v prvních vrstvách modelu. Tento počet je po každé max poolingové vrstvě zdvojnásoben, dokud není dosaženo počtu 512. Model celkem obsahuje pět max poolingových vrstev a tři plně propojené. [23]

Tabulka 1 – Přehled architektury jednotlivých modelů VGG, zdroj: [23]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

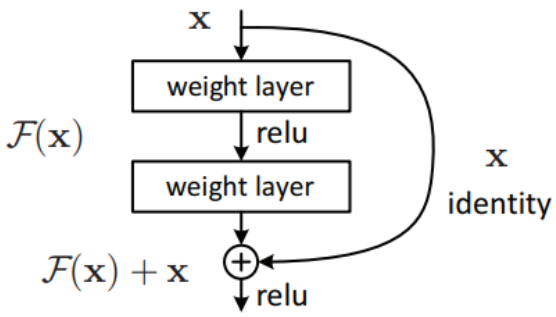
3.4 ResNet

Residual network, zkráceně ResNet je architektura konvoluční neuronové sítě využívající oproti doposud úspěšným architekturám extrémní množství konvolučních vrstev, jichž jsou desítky, v některých variantách i více než 100. Tato architektura byla vyvinuta v roce 2015 výzkumníky ze společnosti Microsoft. Bylo vytvořeno hned několik modelů, typicky jsou pojmenovány podle počtu vrstev, např. ResNet-34, který obsahuje 34 konvolučních vrstev. V době svého vzniku vykazoval tento model na datové sadě ImageNet nejnižší chybovost ze všech. [24]

Tabulka 2 – Porovnání chybovosti ResNet s vybranými modely na datové sadě ImageNet, zdroj: [24]

method	top-5 err. (test) %
VGG (ILSVRC'14)	7.32
GoogLeNet (ILSVRC'14)	6.66
VGG (v5)	6.8
PReLU-net	4.94
BN-inception	4.82
ResNet (ILSVRC'15)	3.57

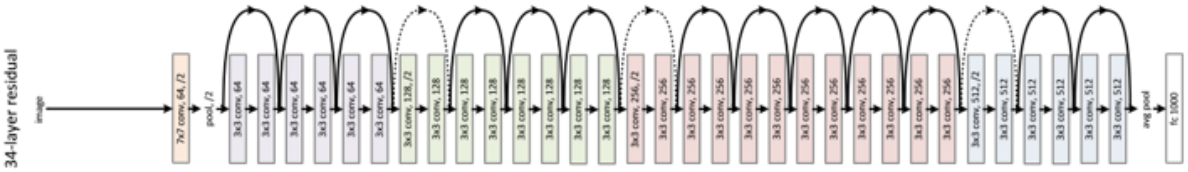
Základem této architektury je tzv *reziduální učení* (*residual learning*), které si lze představit jako propojení jedné konvoluční vrstvy s vrstvou, která se nachází v architektuře hlouběji, např. 3. vrstva propojená s 6. Tomuto principu se také někdy říká *zkratkové spojení* (*shortcut connection*). Díky zavedení tohoto principu nedochází k degradaci neuronové sítě při využití velmi velkého počtu konvolučních vrstev. [24]



Obrázek 10 – reziduální učení – základní blok, zdroj: [24]

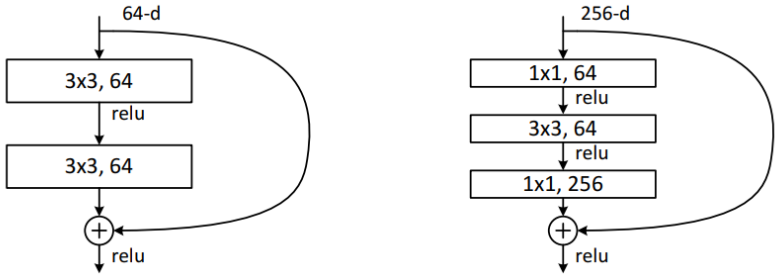
Architektura těchto sítí začíná konvoluční vrstvou s poměrně velikým filtrem (7 × 7). Po něm následují sekvence bloků reziduálního učení. Je aplikováno postupné navyšování počtu filtrů v konvolučních vrstvách, tak jako je tomu v případě architektury VGG, ovšem ke zmenšení dimenze příznakové mapy nejsou použity poolingové vrstvy, ale je aplikován větší krok posunu

na konvoluční vrstvě. Dále byl využit princip *dávkové normalizace* (*batch normalization*) [25] pro všechny konvoluční vrstvy. [24]



Obrázek 11 – Architektura ResNet-34, zdroj:

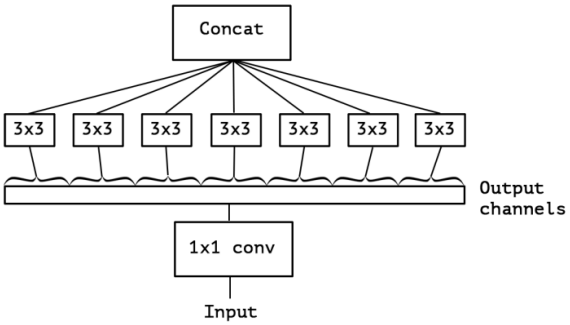
Pro model ResNet-34 je využit blok se dvěma konvolučními vrstvami s filtrem o velikost (3 × 3). Pro hlubší sítě byla podoba bloku změněna. Obsahuje 3 konvoluční vrstvy, z čehož dvě mají filtry o velikosti (1 × 1) a pouze jedna má filtr (3 × 3). Na základě tohoto bloku byly vytvořeny modely ResNet-50, kde byly pouze nahrazeny jednodušší bloky složitějšími, dále ResNet-101 a ResNet-152. I přes tento obrovský počet vrstev má model ResNet-152 menší počet parametrů než model VGG-16. [24]



Obrázek 12 – Základní stavební bloky architektury ResNet, zdroj: [24]

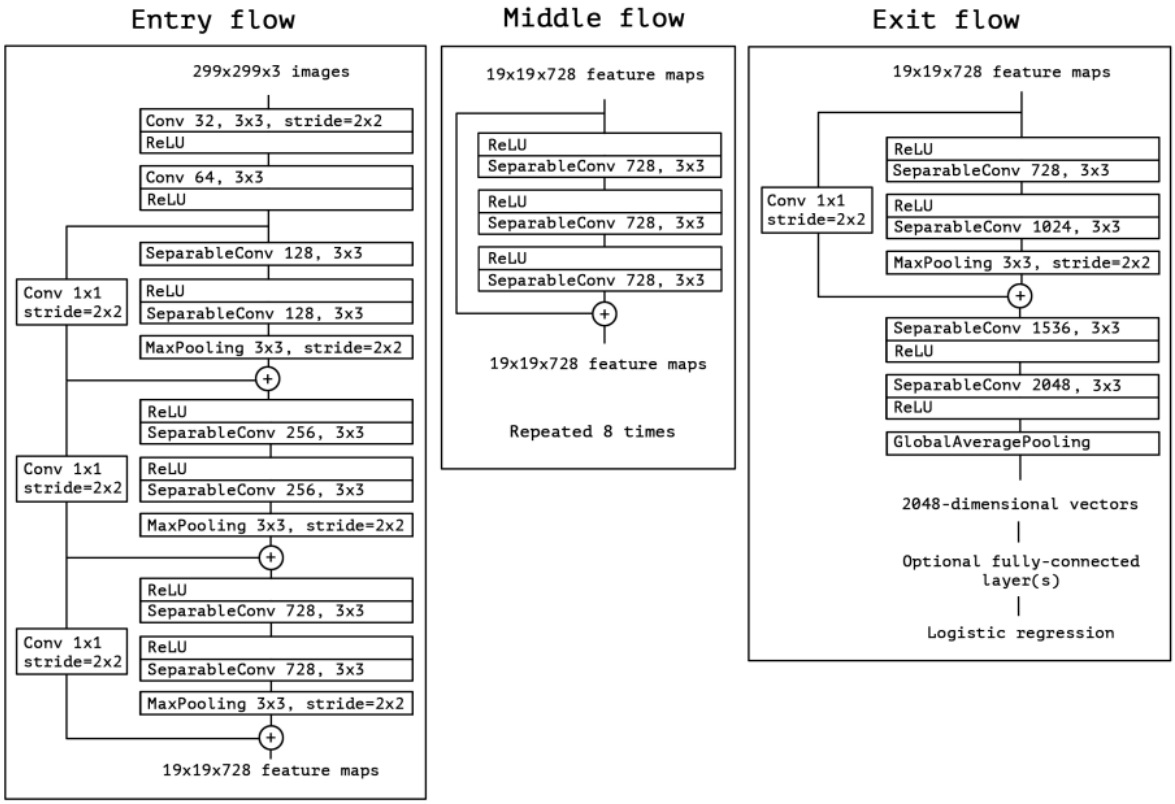
3.5 Xception

Tato architektura byla představena Francoisem Cholletem ze společnosti Google v roce 2017. Vychází z architektury Inception, ze které byla využita upravená varianta Inception modulu. Tento tzv. „*extrémní*“ modul je založen na konvoluci s filtrem (1 × 1), který mapuje vzájemné korelace, které následně rozděluje na výstupní kanály. Z tohoto principu vychází i název architektury, Xception je zkratka z „Extreme Inception“. [26]



Obrázek 13 – Extrémní verze Inception modulu, zdroj: [26]

Celkem se tato síť skládá z 36 konvolučních vrstev, které jsou strukturovány do 14 modulů. Mezi jednotlivými moduly kromě prvního a posledního jsou také reziduální spojení, obdobné jako v případě architektury ResNet. Po těchto modulech mohou volitelně následovat ještě plně propojené vrstvy. Model poté ukončuje vrstva *logistické regrese*. [26]



Obrázek 14 – Architektura modelu Xception, zdroj: [26]

Velkou výhodou této architektury je velmi jednoduchá definice v knihovnách jako je Keras, kde lze celou definici modelu vytvořit na 30 až 40 řádcích. Mezi další přednosti patří vysoká přesnost klasifikace a malá velikost modelu. [26]

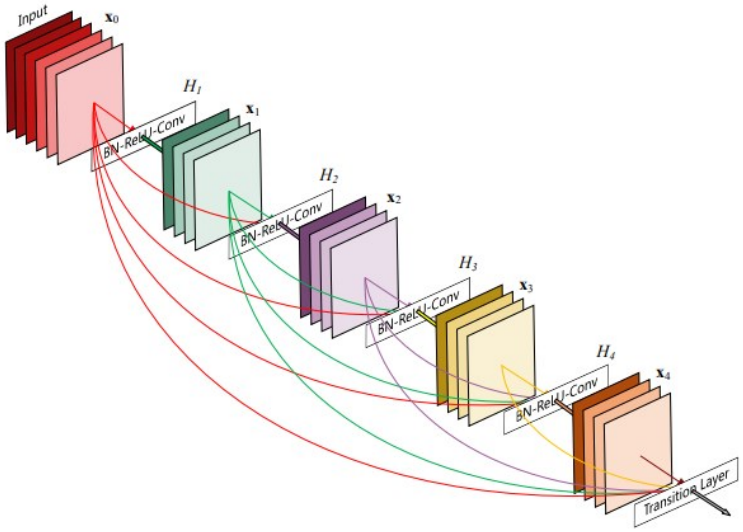
Tabulka 3 – Porovnání přesnosti Xception s vybranými modely na datové sadě ImageNet, zdroj: [26]

	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	0.790	0.945

3.6 DenseNet

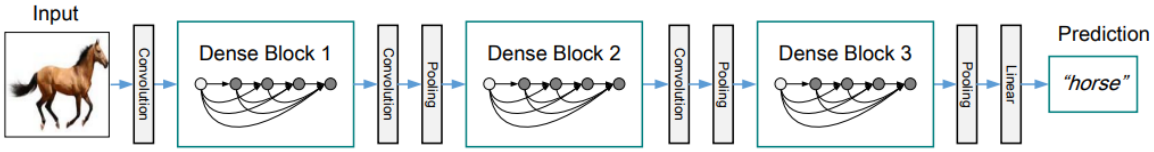
Tato architektura vznikla v roce 2017 na základě architektury ResNet. Obdobně jako ona pracuje s extrémně velkým množstvím konvolučních vrstev (řádově stovky), ale posouvá myšlenku reziduálních spojení vrstev do extrému. V rámci tzv. *„dense“ bloku* je každá vrstva

propojena se všemi dalšími, které po ní následují. Tím se dostáváme z L propojení v tradičních konvolučních sítích až na $\frac{L(L+1)}{2}$ propojení, kde L je počet vrstev v dané síti. Z této hustoty propojení mezi vrstvami vychází i jméno této architektury. [27]



Obrázek 15 – Architektura dense bloku, zdroj: [27]

Jako vstupní vrstva modelu je typicky použita konvoluční vrstva s velikostí filtru (7 × 7), kterých je 16 nebo 32. Následují 3 až 4 dense bloky, každý obsahující 12 až 128 konvolučních vrstev. Tyto vrstvy jsou uspořádány do dvojic, kde první vrstva, obsahující filtry o velikosti (1 × 1), slouží k redukci počtu příznakových map. Druhá je standardní konvoluční vrstva s velikostí filtru (3 × 3). Mezi jednotlivými dense bloky se vyskytuje konvoluční vrstva s filtry (1 × 1) následovaná average poolingovou vrstvou. [27]



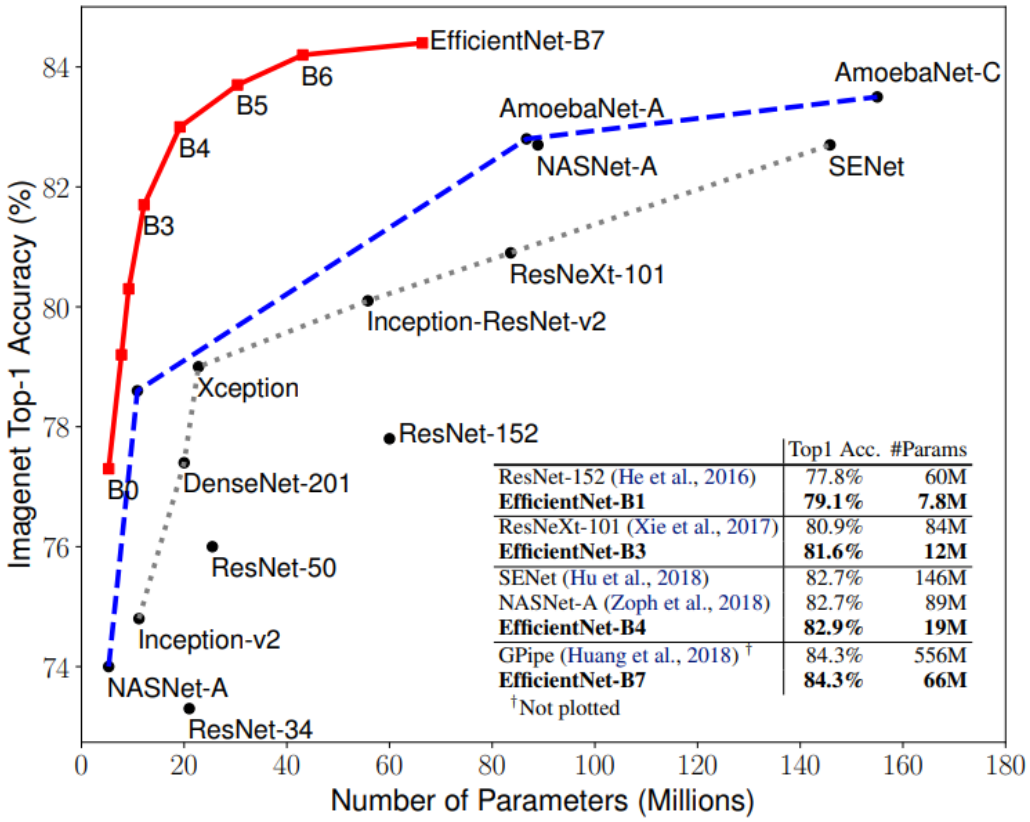
Obrázek 16 – Zjednodušená architektura sítě DenseNet, zdroj: [27]

Mezi výhody této architektury patří vysoká přesnost a nízký počet parametrů. V porovnání s architekturou ResNet je zapotřebí pouze 1/3 parametrů k dosažení stejné přesnosti detekce. Další výhodou je velká odolnost proti degradaci sítě a *přetrénování* (*overfitting*) [28] i při použití velmi velkého počtu konvolučních vrstev. [27]

3.7 EfficientNet

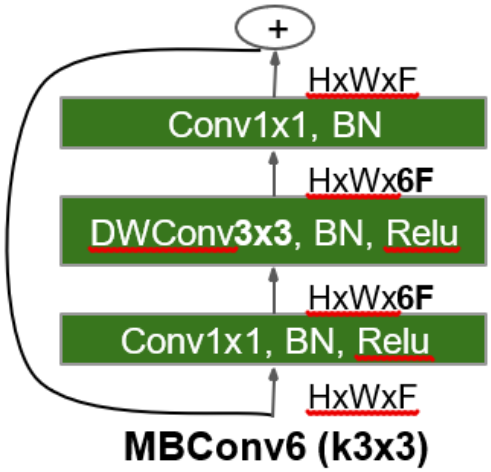
Jedná se o rodinu modelů, která vznikla v roce 2019 s cílem vytvořit různé varianty detekčních sítí, které jsou při zachování obdobné přesnosti jako konkurenční modely výrazně menší a rychlejší. Z tohoto požadavku také vychází její název. Tento cíl byt splněn, největší model z této

rodiny, EfficientNet-B7, dosahuje větší přesnosti než do té doby nejlepší model, přičemž je 8,4× menší a 6,1× rychlejší. [29]



Obrázek 17 – Porovnání EfficientNet s konkurenčními modely, zdroj: [29]

Architektura této sítě byla inspirována sítí MnasNet [30]. Základním stavebním kamenem obou těchto sítí je *mobilní invertovaný páteřní blok* (*mobile inverted bottleneck*), zkráceně MBConv [31], jehož architektura je k vidění na obrázku 18 kde písmena *H* a *W* označují výšku, resp. šířku vstupu a *F* počet filtrů. Zkratka *BN* vyjadřuje využití dávkové normalizace. [29]



Obrázek 18 – Architektura MBConv bloku, zdroj: [30]

Na vstupu sítě je umístěna standardní konvoluční vrstva s velikostí filtru (3 × 3). Po ní následuje série MBConv bloků. Síť uzavírá konvoluční vrstva s velikostí filtru (1 × 1), po které následuje poolingová a plně propojená vrstva. Postupně je u jednotlivých bloků snižována velikost příznakových map a zároveň zvětšován počet filtrů. [29]

Tabulka 4 – Architektura sítě EfficientNet-B0, zdroj: [29]

Stage i	Operator $\hat{F}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

4 ALGORITMY PRO DETEKCI OBJEKTŮ

V této kapitole jsou představeny některé z nejvýznamnějších algoritmů pro detekci objektů vzniklých v posledních letech. Obvykle jsou založeny na architekturách algoritmů pro klasifikaci objektů. Nejdůležitějšími metrikami těchto algoritmů jsou přesnost a rychlost detekce, které se typicky měří na známých volně dostupných datových sadách jako je Pascal VOC [9] či MS COCO [32]. Rychlost se typicky měří jako počet zpracovaných snímků za vteřinu (*frames per second*, zkráceně *FPS*). [33]

Měření přesnosti detekce již není tak triviální. Obvykle se udává ve formě *average precision* (*AP*) nebo *mean average precision* (*mAP*). Pro jejich definici je zapotřebí vysvětlit několik dalších pojmů:

- *Confidence score* – Pravděpodobnost, že daný bounding box obsahuje objekt, typicky vypočteno klasifikátorem.
- *Intersection over Union* (*IoU*) – jedná se o číselné vyjádření přesnosti predikce polohy objektu B_p oproti skutečnému umístění objektu B_{gt}. Vypočte se podle rovnice

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{5}$$

- *True positive* (*TP*) – Jedná se o korektní detekci. Za tu je detekce považována, pokud je confidence score větší než prahová hodnota (např. 0,5), predikovaná třída odpovídá té skutečné a zároveň je hodnota IoU větší než prahová hodnota.
- *False positive* (*FP*) – Jde o chybnou detekci. Za tu je detekce považována, pokud není splněna druhá nebo třetí podmínka TP.
- *False negative* (*FN*) – Jde o případ, kdy je Confidence score menší než prahová hodnota, takže není provedena detekce, i když se v dané oblasti objekt ve skutečnosti vyskytuje.
- *Precision* – jedná se o poměr správně detekovaných objektů ku všem detekovaným objektům neboli

$$precision = \frac{TP}{TP + FP} \tag{6}$$

- *Recall* – Jde o poměr správně detekovaných objektů ku počtu skutečných objektů neboli

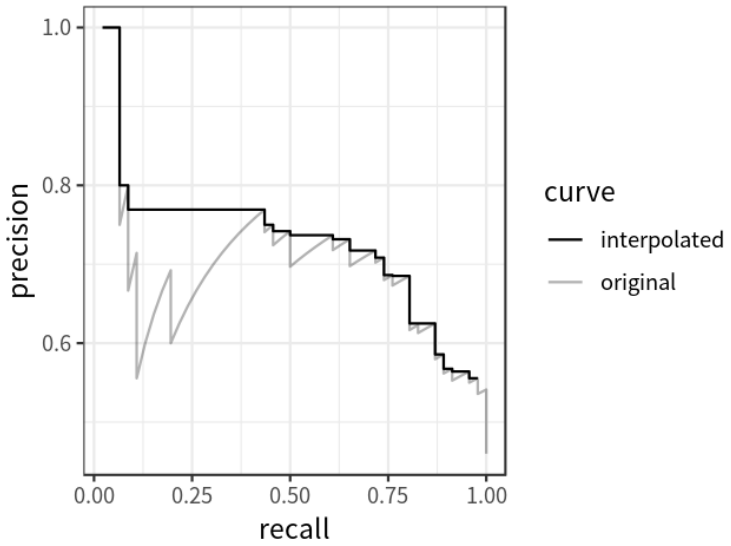
$$recall = \frac{TP}{TP + FN} \tag{7}$$

Postupnou změnou prahové hodnoty lze poté vytvořit *precision-recall křivku*, která je poté využita k určení celkové přesnosti daného algoritmu. Vzhledem k obtížnému porovnání na

základě křivek v grafu je z této křivky následně vypočtena jedna hodnota, která se nazývá average precision. [33]

Nejprve je ovšem pro jednodušší výpočet tato křivka interpolována pomocí vzorce

$$P_{interp}(r) = max\ p(r'), \text{kde } r' \geq r. \tag{8}$$



Obrázek 19 – křivka precision-recall, zdroj: [33]

Z takto interpolované křivky lze poté vypočítat average precision pomocí vzorce

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) p_{interp}(r_{i+1}), \tag{9}$$

kde $r_1, r_2, \ldots, r_n$ jsou úrovně, ve kterých byla původní křivka interpolována. Tato hodnota je ovšem vyhodnocena na základě pouze jedné třídy, obvykle ovšem model detekuje *K* tříd. Mean average precision je poté definován jako průměr AP napříč všemi třídami. [33]

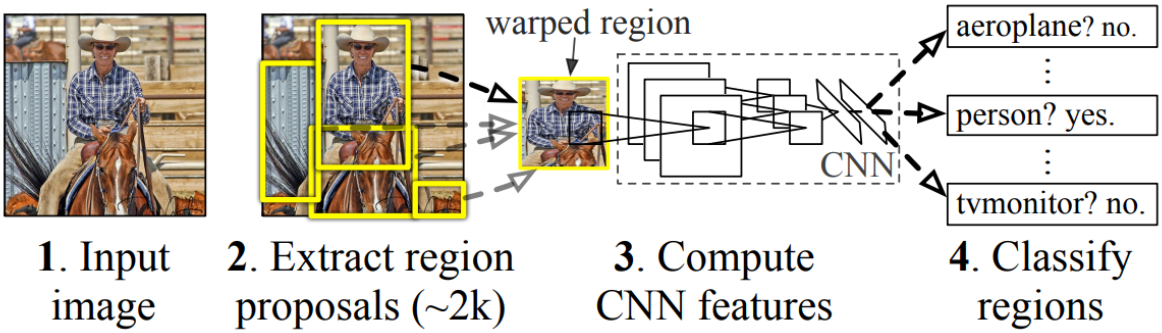
$$mAP = \frac{\sum_{i=1}^{K} AP_i}{K} \tag{10}$$

4.1 R-CNN

Region-based Convolutional Neural Network, zkráceně R-CNN, je architektura neuronové sítě, která kombinuje architekturu konvoluční neuronové sítě s generováním návrhů regionů, ve kterých se vyskytují požadované objekty. Tato architektura byla představena vědci z Kalifornské univerzity v Berkeley v roce 2014. [34]

Oproti klasickým konvolučním neuronovým sítím je po načtení vstupního obrazu nejprve aplikován algoritmus *selektivního vyhledávání* [35], který vyhledá 2000 návrhů regionů. Tyto regiony jsou následně transformovány do čtvercového tvaru o velikosti 227 × 227 pixelů a

propagovány do konvolučních vrstev. Těchto vrstev je pět, výstupem z nich je příznakový vektor o délce 4096. O finální klasifikaci se poté starají dvě plně propojené vrstvy. Po ohodnocení všech regionů je ještě aplikován pro každou třídu algoritmus *greedy non-maximum suppression* [36] pro zabránění vícenásobné detekce stejného objektu. [34]



Obrázek 20 – Ukázka funkce R-CNN, zdroj: [34]

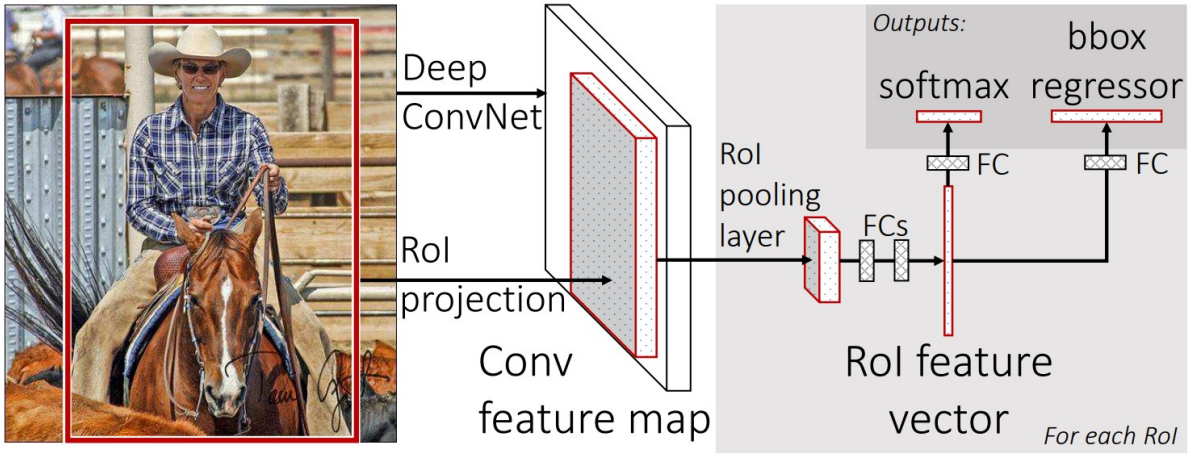
Stala se velmi známou pro svou přesnost, ovšem trpí i některými nedostatky. Trénování i následná detekce objektů jsou velmi časově náročné operace. Pro zpracování jednoho obrázku je totiž nutné zpracovat všech 2000 regionů pomocí konvolučních vrstev separátně. Z tohoto důvodu není využitelná v systémech pracujících s obrazovými daty v reálném čase. Pro ilustraci, detekce objektů v jednom obrázku trvá průměrně 47 sekund (testováno na grafické kartě Nvidia K40 přetaktované na 875 MHz). [34], [37]

4.2 Fast R-CNN

Architektura neuronové sítě *Fast Region Based Convolutional Neural Network*, zkráceně Fast R-CNN, staví na předchozí architektuře R-CNN a snaží se odstranit její nedostatky. Trénování této sítě je až 9× rychlejší a detekce objektů až 213× rychlejší než v případě R-CNN. Kromě tohoto výrazného zrychlení byla i zvýšena přesnost detekce. [37]

Výše zmíněná rychlost trénování i detekce je způsobena změnou architektury. Vstupem do konvoluční neuronové sítě je obrázek a návrhy regionů získané z tohoto obrázku pomocí algoritmu selektivního vyhledávání. Vstupní obrázek je jako celek zpracován konvolučními vrstvami, čímž vznikne příznaková mapa. Až tato mapa je poté rozdělena na jednotlivé regiony, které jsou následně transformovány na čtvercové rozměry a dále zpracovávány. Tím je zajištěno, že konvoluce proběhne pro zpracování každého snímku pouze jednou a nikoliv 2000× jako v případě R-CNN. [37], [38]

Jednotlivé regiony následně vstupují do *region of interest* (*RoI*) pooling vrstvy [37], ve které jsou transformovány na příznakové vektory fixní délky. Klasifikaci poté na základě těchto vektorů dokončí standardní plně propojené vrstvy. [37]

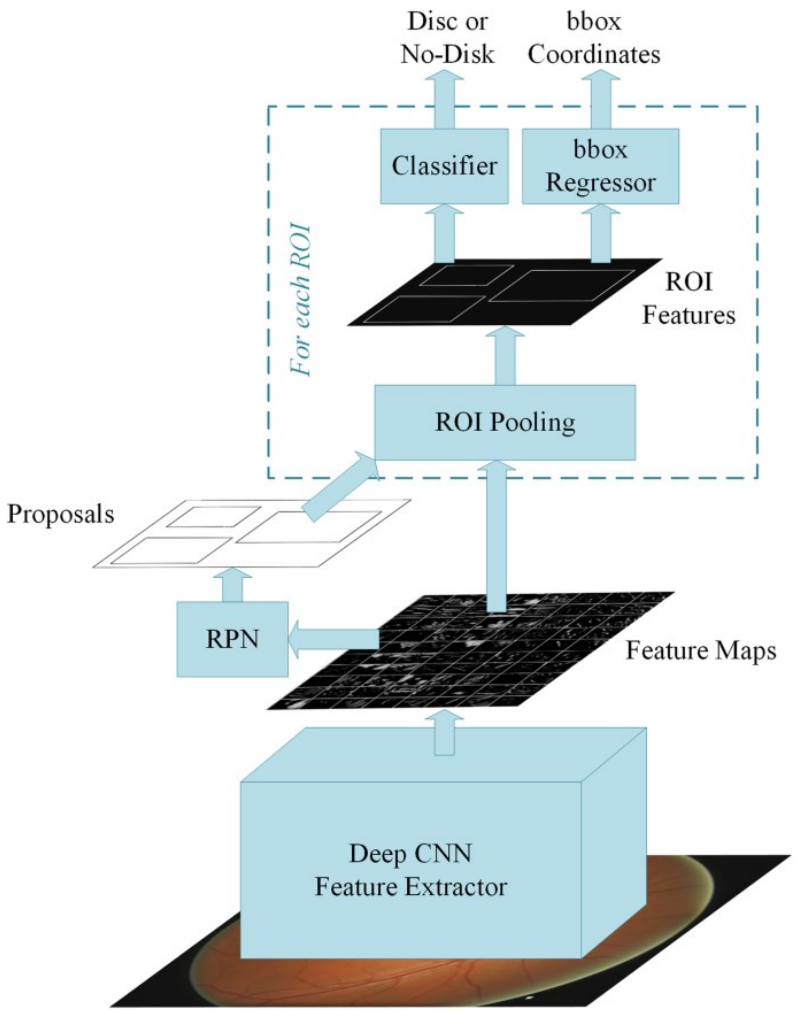


Obrázek 21 – Zjednodušený model Fast R-CNN, zdroj: [37]

4.3 Faster R-CNN

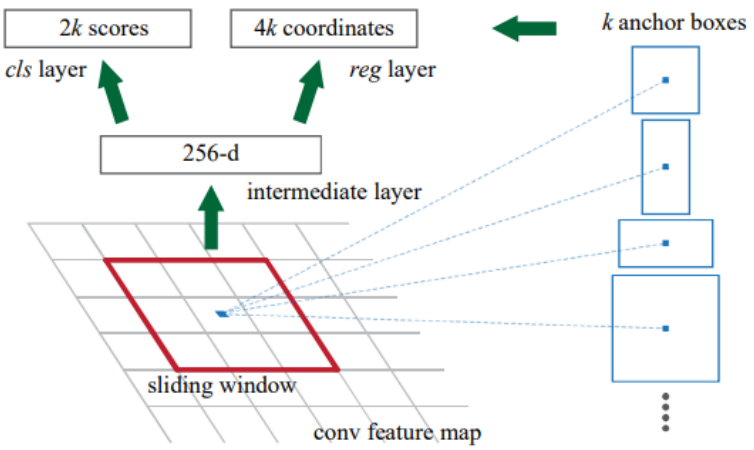
I přes výrazné zrychlení architektury Fast R-CNN oproti jeho předchůdci, tato architektura stále nedosahuje rychlosti potřebné k využití v real-time systémech. Úzkým hrdlem systému jsou algoritmy pro generováním návrhů regionů jako je algoritmus selektivního vyhledávání. Architektura Faster R-CNN řeší tento problém vynecháním tohoto algoritmu a představením *konvoluční sítě pro návrh regionů* (anglicky *region proposal network*, zkráceně *RPN*). [38]

Tato síť přijímá na vstupu obrázek libovolných rozměrů. Tento obrázek je nejprve zpracován standardními konvolučními vrstvami poskytujícími příznakovou mapu. Z této mapy jsou poté pomocí RPN získány jednotlivé regiony. Ty jsou následně převedeny na příznakový vektor. Tento vektor je poté zpracován plně propojenými vrstvami, které poskytují informace o tom, jaký se v daném regionu nachází objekt a jakou má polohu. [37]



Obrázek 22 – Architektura Faster R-CNN, zdroj: [39]

Vstupem do RPN je příznaková mapa získaná z poslední konvoluční vrstvy. Po této příznakové mapě je posouváno malé okno, které slouží jako vstup do sítě. Ta následně vytvoří k návrhů regionů. Jednotlivé návrhy jsou parametrizované relativně k referenčním boxům nazývaným *kotvy* (*anchors*). Každá kotva je centrována na střed okna s rozdílnými měřítky a poměry stran. Obvykle jsou využívána 3 měřítka a 3 poměry stran, což znamená, že $k = 9$ kotev. Návrhy regionů jsou následně zpracovány do formy vektoru a putují do plně propojených vrstev, které poskytují informace o pozici a třídě objektu. Důležitým aspektem je, že kotvy i funkce určující regiony na jejich základě jsou nezávislé na poloze okna. [39]

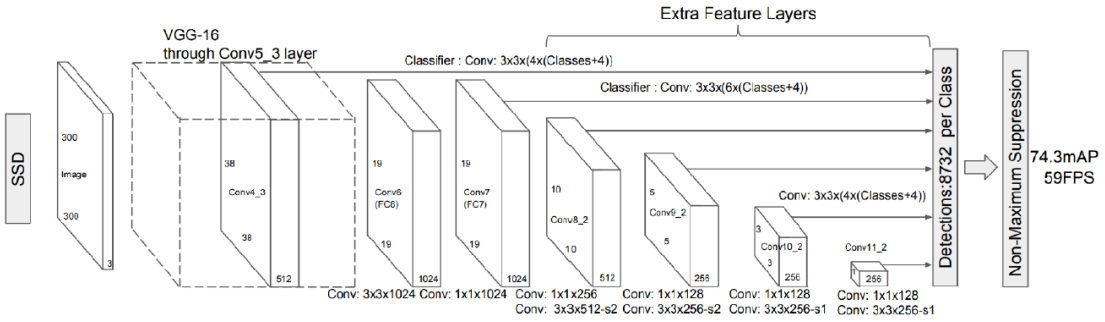


Obrázek 23 – Architektura RPN, zdroj: [38]

4.4 SSD

Detektor objektů *Single Shot Multibox Detector*, zkráceně SSD, vznikl v roce 2016 s cílem představit vysoce přesný detektor, který je možné využít v systémech pracujících v reálném čase. V testu na datové sadě Pascal VOC dosáhl do té doby rekordních hodnot přesnosti 74,3 % mAP při 59 FPS. [40]

Detektor je založen na principu dopředné konvoluční sítě poskytující pevný počet bounding boxů a ohodnocení, zda se v příslušných boxech nachází objekty. Na závěr je umístěna ještě vrstva implementující algoritmus non-maximum suppression produkující finální řešení. [40]



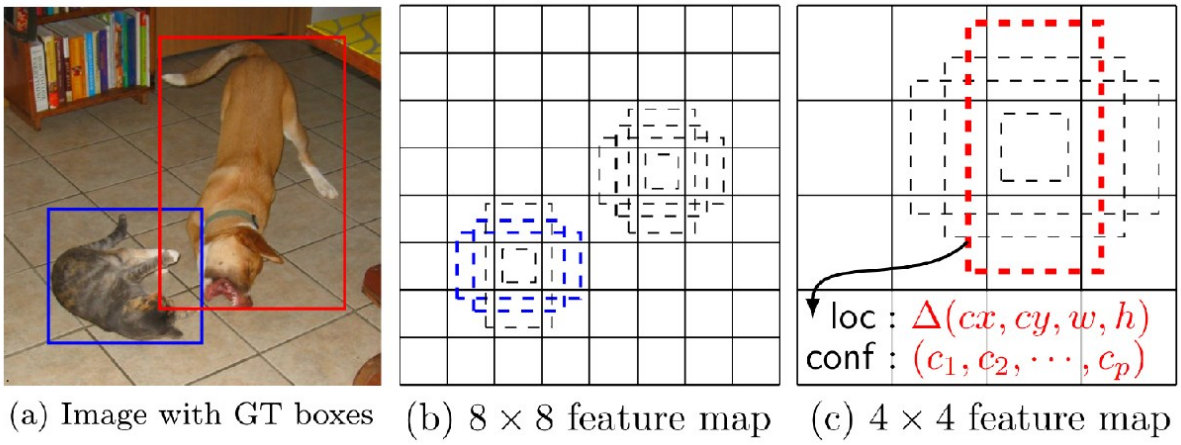
Obrázek 24 – Architektura SSD, zdroj: [40]

Vrstvy na začátku modelu jsou založeny na standardních architekturách pro klasifikaci objektů, obvykle VGG-16 (více viz kapitola 3.3), díky své vysoké kvalitě při klasifikaci objektů. Tyto vrstvy jsou nazývány *základní síť*. Po nich následují další konvoluční vrstvy, které se postupně zmenšují a tím pomáhají v detekci pro různá měřítka objektů. [40]

Predikce není prováděna pouze z příznakové mapy poslední konvoluční vrstvy, ale z příznakových map vrstev napříč celou sítí, včetně vrstev základní sítě. Jednotlivé detekce jsou

založeny na principu konvolučních filtrů, které se pro jednotlivé vrstvy liší. Jsou uvedeny ve vrchní části obrázku 24. [40]

Samotná detekce potom využívá bounding boxy s různými poměry stran. Při vyhodnocení jsou pro každou pozici vypočteny polohy jednotlivých boxů a zároveň je pro každou rozpoznávanou třídu vypočteno skóre určující přítomnost objektu v daném boxu. Tento přístup je zobrazen na obrázku 25. [40]



Obrázek 25 – Princip detekce objektů v SSD, zdroj: [40]

4.5 RetinaNet

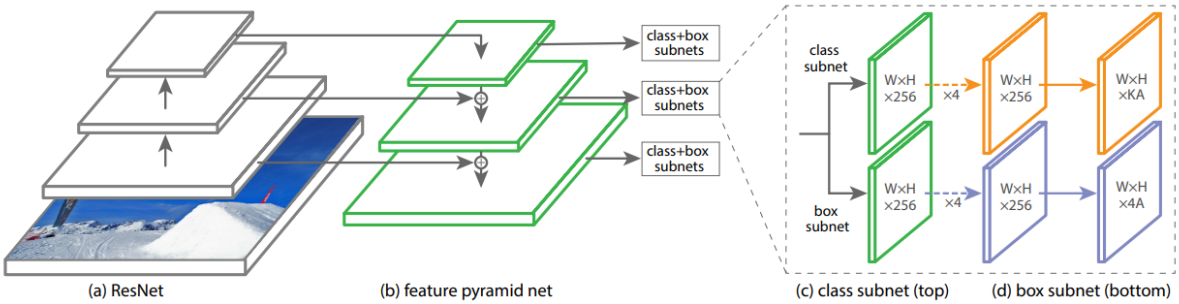
RetinaNet je konvoluční neuronová síť pro detekci objektů, která se vyznačuje vysokou přesností a zároveň rychlostí detekce objektů, která je dostatečná k použití v real-time systémech. Vznikla v roce 2018 ve výzkumném centru *FAIR* (*Facebook AI Research*). V rámci tohoto modelu byla představena nová účelová funkce *focal loss*, která výrazně zlepšuje přesnost detekce objektů. [41]

Architektura tohoto modelu se skládá z *páteře* (*backbone*) a dvou podsítí. Páteř je zodpovědná za zpracování vstupního obrázku a vytvoření příznakových map na jeho základě. První podsíť je zodpovědná za klasifikaci objektů a druhá podsíť za výpočet jejich umístění. [41]

Páteř je založena na architektuře *Feature Pyramid Network* (*FPN*) [42]. Tato architektura modifikuje standardní konvoluční sítě do formy pyramid a přidává spojení mezi jednotlivé vrstvy ve stejné výšce, což umožňuje lepší detekci objektů vyskytujících se v různých měřítkách. Jako základ FPN je v tomto případě využita architektura ResNet (více viz kapitola 3.4). Pro získání jednotlivých regionů je využit princip kotev, který je velmi podobný systému v rámci RPN (více viz kapitola 4.3). [41]

Klasifikační podsíť má architekturu plně konvoluční sítě a je připojena ke každé úrovni pyramidy páteře, přičemž parametry této sítě jsou pro všechny úrovně sdíleny. Vstupem do této

podsítě jsou jednotlivé kotvy, na jejichž základě je pro každou třídu objektu vypočtena pravděpodobnost přítomnosti tohoto objektu. Podsíť určující umístění objektů je téměř identická s klasifikační, přičemž je taktéž připojena ke každé úrovni pyramidy páteře. [41]

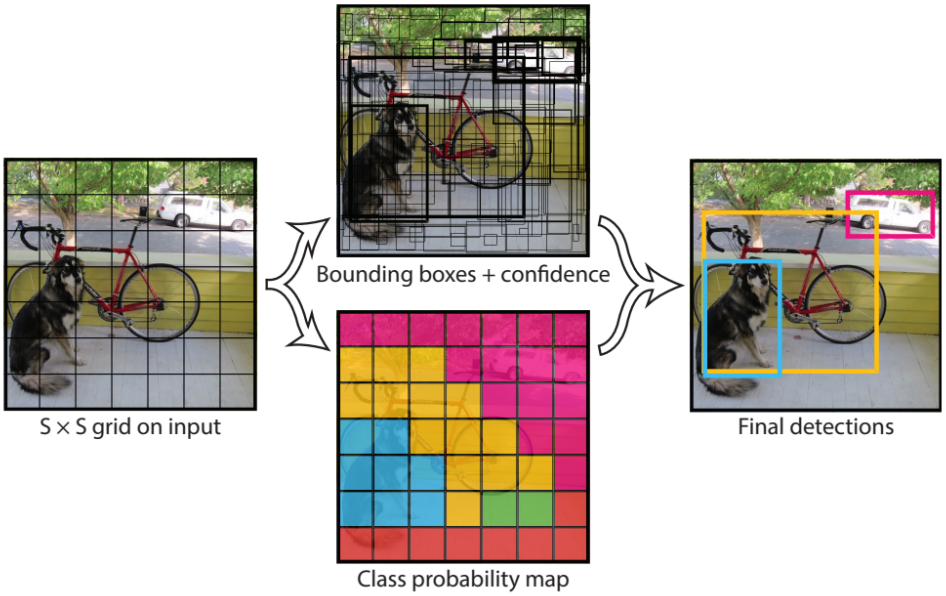


Obrázek 26 – Architektura sítě RetinaNet, zdroj: [41]

4.6 YOLO

Architektura *You Only Look Once* (YOLO) je založena na principu získání všech potřebných informací (třída objektu a jeho umístění) v rámci jednoho vyhodnocení, z čehož také vychází její název. Architektura vznikla v roce 2016 za účelem přesné detekce objektů v reálném čase. Velká výhoda této architektury je, že se učí velmi obecné reprezentace objektů, takže je schopná poměrně přesné detekce, i když se testovací obrázky výrazněji liší od trénovacích (například obrázky z přírody a umělecká díla). [43]

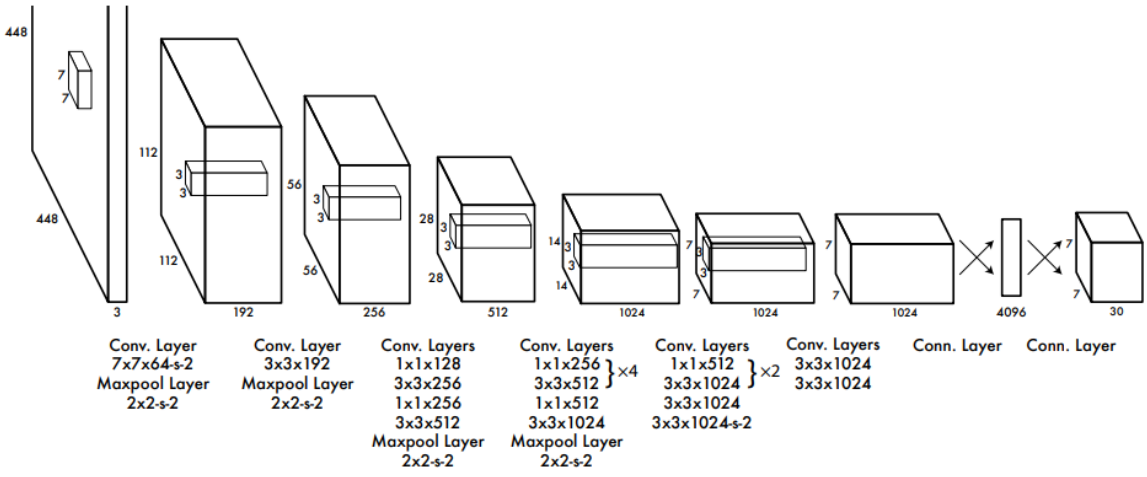
Tato architektura využívá pro predikci každého bounding boxu příznaky z celého obrázku a zároveň je predikuje pro všechny třídy objektů současně. Princip této detekce začíná rozdělením vstupního obrázku podle mřížky s velikostí $S \times S$. Každá buňka mřížky se poté stará o detekci objektů, jejíž střed leží v této buňce. Pro každou buňku je poté předpovězeno B bounding boxů obsahujících polohu středu objektu, jeho výšku a šířku vzhledem k celému obrázku, a pravděpodobnost, že se zde nachází nějaký objekt. Zároveň každá buňka vypočte vektor C pravděpodobností, že daná buňka obsahuje daný objekt. Hodnota C je rovna počtu tříd, pro které je daný model trénován. Tento vektor je vždy jeden nezávisle na hodnotě B. Veškeré predikce jsou poté uloženy jako *tenzor* o velikosti $S \times S \times (B * 5 + C)$. [43]



Obrázek 27 – Zjednodušený princip detekce objektů architekturou YOLO, zdroj: [43]

Nevýhodou tohoto přístupu je omezení počtu objektů, které je schopna tato architektura detekovat, obzvláště pokud se vyskytují blízko sebe. Dále model velmi špatně detekuje malé objekty. Z těchto důvodů není model vhodný pro detekci některých typů objektů jako je např. hejno ptáků. Dále má tato architektura problémy s detekcí objektů, pokud se vyskytují v jiných poměrech stran. [43]

Architektura samotné sítě je založena na jedné konvoluční síti, není tedy rozdělena na více podsítí. Je inspirovaná architekturou GoogLeNet pro klasifikaci objektů (viz kapitola 3.2). Síť obsahuje 24 konvolučních vrstev, které jsou následovány dvěma plně propojenými vrstvami. Varianta Fast YOLO potom obsahuje pouze 9 vrstev s méně filtry v každé z nich. [43]



Obrázek 28 – Architektura YOLO, zdroj: [43]

4.6.1 YOLOv2 (YOLO9000)

YOLOv2 je druhá verze detektoru objektů YOLO představená v roce 2017. Snaží se odstranit některé nedostatky předchozí verze a výrazně zvýšit přesnost detekce objektů při zachování stejné rychlosti. Na základě této architektury byl vytvořen a natrénován detektor YOLO9000, který dokáže detekovat přes 9000 tříd objektů v reálném čase. [44]

Mezi nejvýraznější změny patří:

- Použití *dávkové normalizace* (*batch normalization*), což vede ke zlepšení konvergence a eliminuje potřebu ostatních forem regularizace. Zavedením dávkové normalizace na všechny konvoluční vrstvy zvýší přesnost predikce (mAP) až o 2 %.
- Využití klasifikátoru s vyšším rozlišením. Všechny architektury pro detekci objektů využívají klasifikátory předtrénované na datové sadě ImageNet. Ty typicky pracují s obrázky v rozlišení 256 × 256 pixelů, nebo menšími. Původní klasifikátor v architektuře YOLO je předtrénován při rozlišení 224 × 224, které je pro detekci zvýšeno na 448 × 448 pixelů, což způsobuje problémy. Nyní je při předtrénování využito pro posledních 10 epoch plné rozlišení. To zlepšuje přesnost detekce až o 4 %.
- Pro získání bounding boxů jednotlivých objektů jsou místo plně propojených vrstev využity vrstvy konvoluční, které využívají princip kotev obdobný jako v architektuře Faster R-CNN. Tato změna nepatrně snižuje přesnost detekce, ovšem zvyšuje senzitivitu modelu, což poskytuje více prostoru pro další vylepšování architektury.
- Architektura celé sítě byla změněna z původní, která vycházela ze sítě GoogLeNet, na novou, která nese název *Darknet-19*. Tento model obsahuje 19 konvolučních vrstev oproti 24 v původním modelu, což znamená menší počet parametrů k trénování. Tento nový model i přes toto zmenšení velikosti poskytuje lepší výsledky než původní architektura. [44]

4.6.2 YOLOv3

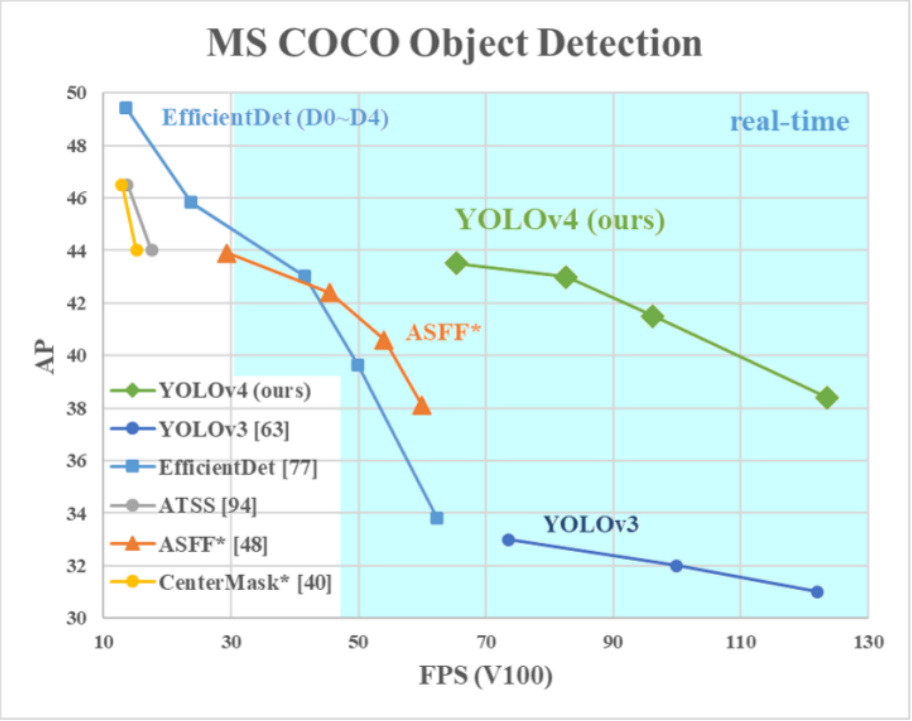
V roce 2018 byla představena již 3 verze tohoto modelu. Ta se opět zaměřuje na zvýšení přesnosti detekce. Tím byla o něco snížena její rychlost, ta je ovšem stále dostatečná pro detekci objektů v reálném čase. [45]

Byl vylepšen systém pro predikci třídy objektu. Byla vypuštěna vrstva obsahující aktivační funkci softmax a místo ní jsou využity nezávislé *logistické klasifikátory*. To umožňuje přiřadit jednomu objektu více než jednu třídu. Tento princip pomáhá zlepšit detekci u složitějších datových sad, kde může docházet k překryvu jednotlivých tříd objektů (např. žena a osoba). [45]

Další změnou je zavedení predikce objektů v různých měřítkách. Konkrétně jsou prováděny predikce ve 3 měřítkách. K tomu je využit princip, který je velmi podobný architektuře FPN. Také byla použita nová konvoluční síť pro extrakci příznaků. Základ této sítě je stejný jako v předchozí verzi, byla ovšem výrazně rozšířena. Obsahuje hned 53 konvolučních vrstev, od čehož je také odvozen její název *Darknet-53*. Tato síť je výrazně efektivnější než její předchůdce Darknet-19, ovšem kvůli své velikosti je také výrazně pomalejší. [45]

4.6.3 YOLOv4

Již čtvrtá verze tohoto detektoru vznikla v roce 2020 pod taktovkou Alexeye Bochkovskiyho. Autor původních tří verzí Joseph Redmon totiž v témže roce zanechal výzkumu z důvodu obav o možné zneužití této technologie pro vojenské účely. Tato verze se snaží zlepšit paralelizaci výpočtů a zároveň vytvořit detektor objektů, který bude dosahovat dobrých výkonů i na komerčních grafických kartách (např. Nvidia 1080Ti nebo 2080Ti). Oproti předchozí verzi je tento model přesnější i rychlejší o 10, respektive 12 %. [46], [47]



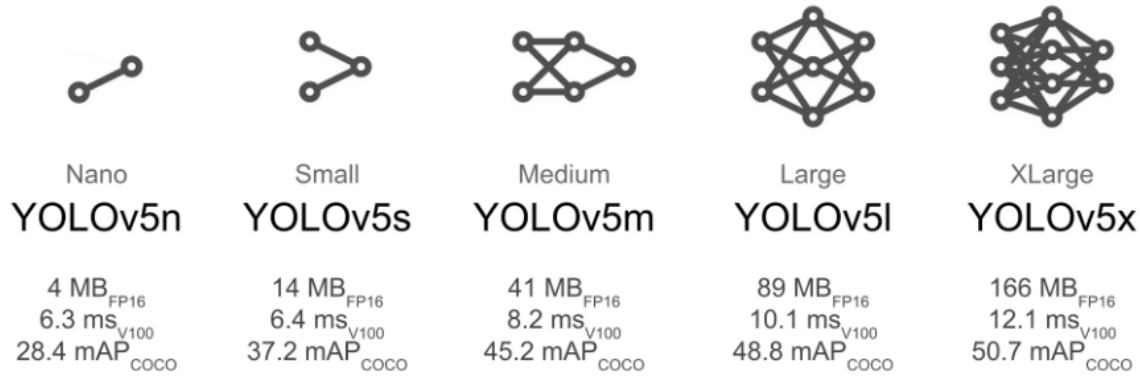
Obrázek 29 – Porovnání parametrů YOLOv4 s dalšími vybranými detektory, zdroj: [47]

Pravděpodobně největší změnou v této verzi je zavedení tzv. „*bag of freebies*“ a „*bag of specials*“. Pojmem bag of freebies jsou označovány metody, které modifikují pouze strategii trénování sítě, nebo pouze zvyšují náročnost trénování. Tyto metody mají za cíl zlepšit přesnost výsledného modelu, aniž by byla ovlivněna rychlost detekce. Typickým příkladem těchto metod je augmentace dat. Oproti tomu plugin modely a metody post-processingu, které mírně

zvyšují čas detekce, ale výrazně zvyšují přesnost modelu, jsou označovány jako bag of specials. Tyto pluginy typicky modifikují určité vlastnosti modelu, jako je například zvětšení *smyslového prostoru* (*receptive field*). [47]

4.6.4 YOLOv5

Pouhé 2 měsíce po vydání předchozí verze byla vydána další, již pátá verze tohoto detektoru, kterou představil Glenn Jocher se společností Ultralytics. Jedná se o open-source projekt, který je stále v aktivním vývoji. Aktuálně je vytvořeno hned pět variant, které jsou rozlišeny podle velikosti a s ní související rychlostí a přesností. [48]



Obrázek 30 – Modely YOLOv5, zdroj: [48]

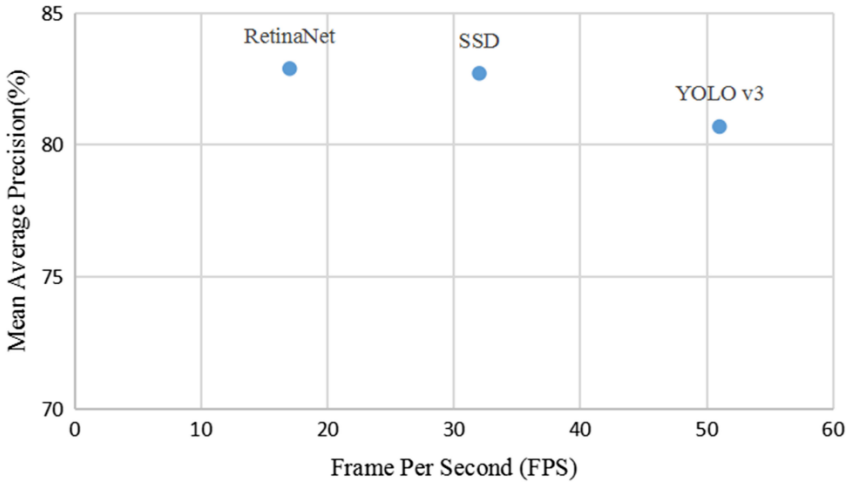
Vydání této verze se ovšem neobešlo bez několika kontroverzí. První problémem byla forma vydání, protože nebyl zveřejněn žádný oficiální dokument popisující tuto verzi, ale pouze GitHub repositář. Vedly se také diskuse ohledně pojmenování tohoto modelu jako YOLOv5, a to především pro nahrazení původní architektury Darknet za implementaci sítě nativně ve frameworku PyTorch. Dalším problémem bylo vydání porovnání rychlosti a přesnosti detekce této verze s předchozí, které nebylo zcela objektivní. [49]

I přes tyto kontroverze se tato je tato verze hojně využívána (přes 29 000 hvězd na GitHubu), především pro vysokou přesnost a rychlost detekce a velmi jednoduchý trénink modelu na vlastní datové sadě. Pět poskytovaných variant také nabízí velkou variabilitu použití. [48]

5 POUŽITÉ ALGORITMY A TECHNOLOGIE

5.1 Výběr algoritmů

Nejprve byl proveden výběr detekčního algoritmu, přičemž bylo uvažováno hned několik kritérií. Hlavním požadavkem byla dostatečná rychlost detekce umožňující využití algoritmu v reálném čase. Tím byly z výběru vyřazeny modely vycházející z architektury R-CNN. Při porovnání modelů SSD, RetinaNet a YOLOv3 byla všemi modely dosažena téměř srovnatelná přesnost, ovšem jednoznačně nejvyšší rychlosti dosáhl model YOLOv3. Detaily výsledků a celková metodika testu je dostupná na [50].



Obrázek 31 – Porovnání výkonu SSD, RetinaNet a YOLOv3, zdroj: [50]

Vzhledem k vyšší rychlosti i přesnosti detekce modelu YOLOv4 oproti YOLOv3 [47] bylo ve finále vybíráno mezi architekturami YOLOv4 a YOLOv5. Na datové sadě MS COCO je YOLOv4 nepatrně přesnější, ovšem kvůli vyšší rychlosti trénování a jednoduchosti použití byl nakonec zvolen YOLOv5, konkrétně druhý nejmenší model YOLOv5s. Kompletní porovnání těchto algoritmů viz [49].

Tento detekční model byl použit pouze pro detekci umístění objektu a plochy, na které leží, nikoliv pro zjištění úhlu natočení od horizontální osy. Vzhledem k výrazné generalizaci objektů v rámci tohoto algoritmu by bylo pro tento účel velmi složité připravit dostatečně velkou datovou sadu a následně dostatečně kvalitně natrénovat model. Využití jiného algoritmu pro určení úhlu otočení přináší i výhodu vyšší flexibility systému, kdy tento algoritmus může být jednoduše vyměněn za jiný, ať už kvůli vyšší přesnosti nebo rychlosti.

Úhel otočení nebude určován přesně, objekt bude pouze zařazen do intervalu, ve kterém se skutečný úhel otočení nachází, např. 0-30°. K tomuto účelu se za předpokladu získání výřezu daného objektu velmi dobře hodí algoritmy klasifikace obrázků.

Při výběru konkrétního algoritmu pro klasifikaci úhlu otočení byla posuzována stejná kritéria jako v případě algoritmu pro detekci objektů, tedy snadné použití, přesnost a rychlost klasifikace. Pro co nejjednodušší použití byl výběr zúžen na předpřipravené modely v rámci *Keras Applications* [51]. Pro svou vysokou rychlost a přesnost byl vybrán model architektury EfficientNet, konkrétně nejmenší a nejrychlejší z nich, tedy model EfficientNet-B0.

5.2 Kamera Basler Ace

Německá společnost Basler AG je přední výrobce kvalitních kamer a jejich doplňků zaměřující se na strojové vidění. Pro tuto práci byla vybrána kamera od této společnosti ze série *ace Classic,* konkrétně model *acA2500-14uc*. Důvodem výběru tohoto modelu byla vysoká kvalita, nízká pořizovací cena, malá velikost (pouze 29 × 29 mm), velké rozlišení (2590 × 1942 pixelů) a v neposlední řadě možnost připojení k počítači pomocí USB konektoru. [52], [53]

Velmi důležitým příslušenstvím jakékoliv kamery je objektiv. Jeho výběr není ovšem vůbec jednoduchý, musí být totiž kompatibilní s vybranou kamerou, ale ani další parametry, jako je ohnisková vzdálenost nebo zorný úhel, nejsou příliš intuitivní. Naštěstí existují nástroje pro usnadnění výběru vhodného objektivu. Jedním z nich je i *Lens Selector* od společnosti Basler. Do něj se zadá typ kamery a část parametrů jako například rozměry a vzdálenost sledovaného objektu a zbylé parametry se automaticky dopočítají. Software je dostupný na [54].

Na základě takto získaných parametrů byl vybrán vhodný objektiv, konkrétně *H0514-MP2* od společnosti Computar. Tato společnost je již přes 40 let světovým lídrem v oblasti optik pro průmyslová zařízení. [55] Podrobná specifikace vybraného objektivu je k dispozici na [56].

5.3 Pylon Viewer

Jde o oficiální software od společnosti Basler pro nastavení kamer od této společnosti a vyhodnocení kvality získaného obrazu. K tomuto účelu obsahuje řadu nástrojů jako například indikátor ostrosti obrazu nebo možnost zobrazení jeho histogramu. Je poskytován zdarma, nicméně se nejedná o otevřený software. [57], [58]

5.4 Python

Python je vyšší, interpretovaný, objektově orientovaný programovací jazyk. Tento jazyk umožňuje velice rychlý vývoj aplikací a skriptů, a to především díky svým vlastnostem jako například dynamické typování a vazby nebo vestavěné vysokoúrovňové datové struktury.[59], [60]

Mezi jeho další vyzdvihované vlastnosti patří vysoká úroveň abstrakce a důraz na čitelný zdrojový kód. Z tohoto důvodu má jazyk velmi striktní syntaxi, ve které mají význam i bílé

znaky. To znamená, že i při změně odsazení či odřádkování je změněn význam kódu, který se tím i může stát nespustitelným. [59], [60]

Součástí instalace je i debugger, který je sám napsán v jazyce Python, a umožňuje jednoduché a rychlé ladění kódu. Volitelnou součástí instalace je i balíčkovací systém *pip*, který slouží k velmi jednoduché instalaci balíčků třetích stran, které se nacházejí v *Python Package Indexu* (*PyPI*).[59], [61]

5.5 Pypylon

Pypylon je API pro práci s kamerami společnosti Basler s využitím jazyka Python. Umožňuje velmi jednoduché a rychlé začlenění kamer do projektu. Zároveň poskytuje široké možnosti nastavení těchto kamer v kódu, takže není nutné spoléhat na manuální nastavení kamery jiným softwarem. Jedná se o otevřený software. [62]

5.6 Tensorflow

Tensorflow je platforma vyvinutá společností Google pro usnadnění vývoje aplikací využívajících strojové učení. Platforma byla zveřejněna jako otevřený software v roce 2015, první stabilní verze až v roce 2017. Název je odvozen od multidimenzionálního pole nazývaného *tenzor*, které je základním stavebním kamenem této platformy. [63]

Architektura platformy funguje na principu grafu, kde jednotlivé vrcholy reprezentují matematické operace a hrany reprezentují tenzory, které jsou jejich vstupy a výstupy. Díky tomu je zápis jednotlivých algoritmů přehledný. [64]

Platforma dokáže využívat i výpočetní výkon GPU pro výrazně rychlejší běh algoritmů, než při použití CPU. [64] V současné době jsou podporovány grafické karty od společnosti NVIDIA s architekturou CUDA a výpočetní kapacitou alespoň 3,5. Pro její využití je dále potřeba nainstalovat příslušný software. Kompletní seznam požadavků je dostupný na [65].

Dále je pro zrychlení výpočtů možné využít specializovaný procesor zaměřený na práci s tenzory, podle čehož také nese označení TPU (*Tensor Processing Unit*). Byl vyvinut společností Google přímo pro platformu TensorFlow. [64], [66]

5.7 Keras

Keras je API pro hluboké učení vytvořené pro co nejjednodušší práci s neuronovými sítěmi. Je napsáno v jazyce Python a staví na platformě TensorFlow. Oproti TensorFlow poskytuje vyšší úroveň abstrakce, ovšem se zaměřením pouze na hluboké učení. [67]

Základními strukturami jsou zde vrstvy a modely. Důležité informace, jako je například počet neuronů či aktivační funkce jsou do nich předávány pomocí parametrů. To umožňuje tvořit neuronové sítě pomocí několika málo řádcích kódu. [67]

5.8 Pytorch

Pytorch je framework vytvořený společností Facebook pro tvorbu aplikací a modelů v oblasti hlubokého učení. Byl napsán v jazyce Python a zveřejněn jako otevřený software v roce 2017. [68]

Framework Pytorch je vlastnostmi i základní stavbou velmi podobný platformě TensorFlow. Základní datovou strukturou je tenzor a je využit princip grafu. Také podporuje výpočty na GPU se stejnými omezeními jako TensorFlow. [68]

Pytorch ovšem nativně nepodporuje výpočty na TPU. Podpora pro výpočty na TPU je zajištěna pomocí integrace s kompilátorem pro lineární algebru *XLA* (*Accelerated Linear Algebra*), který podporuje CPU, GPU i TPU, projektem *PyTorch/XLA*. [68], [69]

Využití platformy TensorFlow a frameworku Pytorch v rámci jednoho projektu není úplně obvyklé. V rámci této práce k tomu došlo kvůli využití projektu YOLOv5 pro detekci objektů, který je postavený na frameworku Pytorch, a následné nutnosti vytvoření modelu pro klasifikaci otočení objektů co možná nejjednodušeji a nejrychleji. K tomu se nejlépe hodí API Keras, které je nadstavbou platformy TensorFlow. [70], [71]

5.9 Matplotlib

Matplotlib je v současné době nejpopulárnější knihovna v jazyce Python pro vizualizaci dat. Je oblíbená pro své široké možnosti a zároveň jednoduchost použití. Vznikla jako otevřený software již v roce 2002 pouze pro tvorbu dvourozměrných grafů, ale postupně byly možnosti knihovny rozšířeny i o tvorbu trojrozměrných vizualizací. [72]

5.10 Google Colaboratory

Colaboratory, zkráceně Colab je produkt od Google Research, který umožňuje psát a spouštět kód napsaný v jazyce Python ve webovém prohlížeči bez nutnosti jakékoliv instalace. Tyto dokumenty kombinují bloky spustitelného kódu a bloky textu, které mohou obsahovat i další prvky jako jsou obrázky. Jsou ukládány na Google Drive, takže je lze jednoduše sdílet. [73], [74]

Colaboratory bylo vytvořeno především pro využití v oblastech datové analýzy a umělé inteligence. To znamená, že virtuální stroje, na kterých dokumenty běží, jsou pro tyto účely přednastavené, tedy mají nainstalovány moduly využívané v těchto oblastech, jako je například

Matplotlib nebo Tensorflow. Dále je možné připojit GPU nebo TPU akcelerátory, což významně urychluje operace jako je trénování neuronových sítí. [73], [74]

Tato služba je v základní verzi zdarma. Je ovšem limitována délka běhu na maximálně 12 hodin a je vynucena interaktivita ze strany uživatele. Taktéž není garantována dostupnost hardwarových akcelerátorů. Pro náročnější uživatele ovšem existuje i placená verze Colab Pro a Colab Pro+, které mají mnohem mírnější omezení. V rámci této práce byla využita verze zdarma. [73]

5.11 PyCharm

PyCharm je komplexní IDE (integrované vývojové prostředí) zaměřené hlavně na vývoj v jazyce Python. Bylo vyvinuto firmou JetBrains zaměřující se na vývoj produktů pro vývojáře, která stojí například za velmi populárním vývojovým prostředím IntelliJ IDEA zaměřeným na vývoj v jazyce Java. [75]

Výhodou využití tohoto prostředí je velké množství integrovaných nástrojů, například pro inteligentní napovídání kódu, práci s databází či s verzovacím systémem. Nejde ovšem jen o integrované nástroje, je možné doinstalovat ještě větší množství pluginů, díky kterým je možné si toto prostředí libovolně přizpůsobit. Další oceňovanou vlastností tohoto produktu je podpora pro práci v dalších programovacích jazycích, jako je například *JavaScript*. [75]

Jedná se o multiplatformní software fungující na všech rozšířených platformách (Windows, Mac i Linux) a je vydáván v bezplatné, ale omezené edici Community a placené edici Professional. Dále ještě existuje edice Edu, která je rovnocenná s edicí Professional, ale je pro studenty a učitele zdarma. [75]

5.12 MATLAB Image Labeler

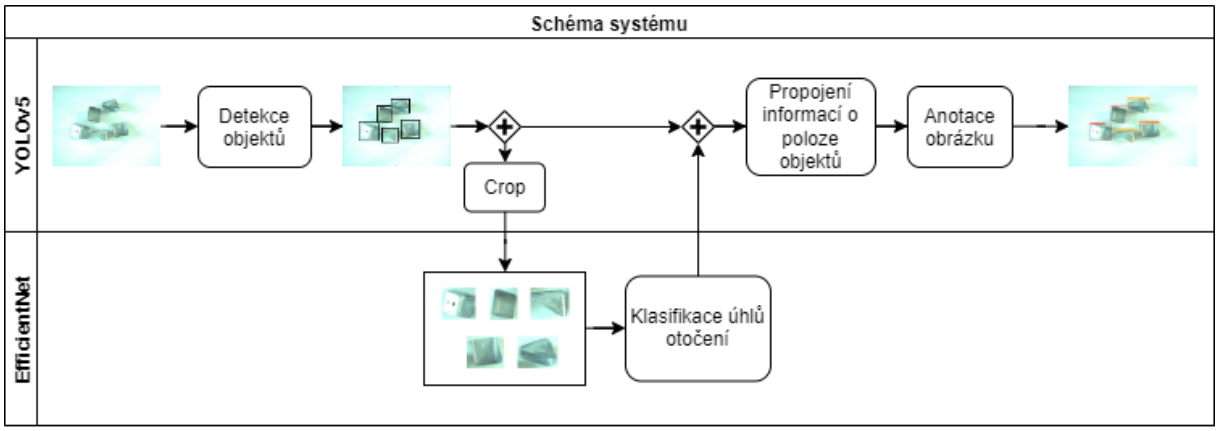
MATLAB je programovací jazyk a platforma, která zapouzdřuje výpočetní, vizualizační a programovací prostředí. Původně vznikl v roce 1984 pro zjednodušení přístupu k maticovým software vyvinutými projekty LINPACK a EISPACK. Z toho byl odvozen i jeho název, který je zkratkovým slovem z „*matrix laboratory*“. Obsahuje velké množství aplikací pro zjednodušení velkého množství úkonů. [76]

Jednou z nich je i Image Labeler, který slouží k jednoduchému interaktivnímu anotování obrázků. Jde o velmi intuitivní nástroj umožňující vytvářet vlastní třídy objektů a vytvářet anotace pro různé úlohy počítačového vidění. Mezi základní typy patří čtvercový bounding box pro detekci objektů a pixelové rozdělení pro sémantickou segmentaci. Dále umožňuje seskupovat třídy do skupin a vytvářet hierarchii tříd. [77]

Zajímavou možností této aplikace je vytváření anotací automatizovaně pomocí algoritmů. K dispozici je několik vestavěných algoritmů nebo je možné vytvořit vlastní. Pro jednoduchý přehled je také k dispozici shrnutí informací o jednotlivých třídách napříč celou datovou sadou. [77]

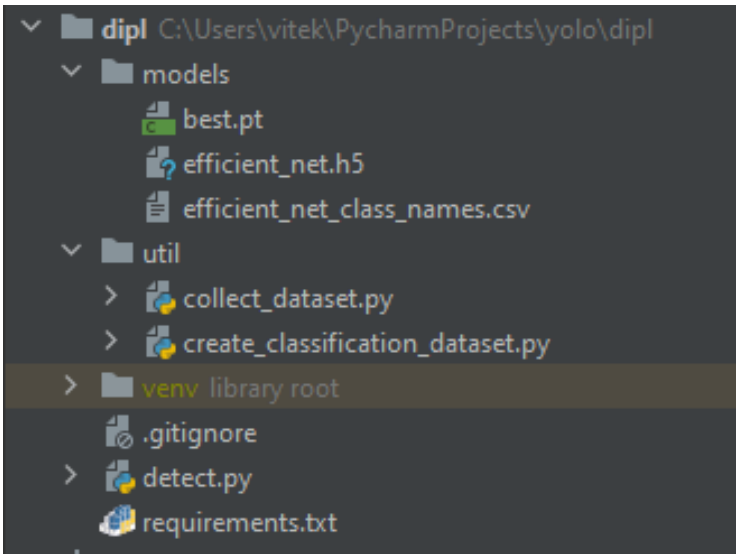
6 NÁVRH A IMPLEMENTACE SYSTÉMU

Cílem systému je detekovat objekty na pracovní ploše a přesně určit jejich polohu, čímž je myšleno nejen umístění, ale i plocha, na které daný objekt leží, a jeho úhel natočení od horizontální osy při pohledu shora. Takto podrobné informace o poloze objektů jsou potřeba, aby mohli být následně využity dalšími systémy, například systémem pro ovládání robotického manipulátoru. Jednotlivé objekty se pro správnou detekci nesmí překrývat. Kamera je umístěna kolmo nad pracovní plochou, přičemž její poloha je neměnná a známá. Pro její nastavení byl využit nástroj Pylon Viewer. Výstup systému je ve formě obrazových dat a volitelně i formátu JSON, který umožní relativně jednoduchou integraci s dalšími systémy. Ta však již není součástí této práce.



Obrázek 32 – Schéma činnosti systému, zdroj vlastní

Ústřední komponentou systému je skript *detect.py*, který obsahuje většinu logiky, jako je propojení neuronových sítí. Také je využíván pro spuštění systému. Složka *models* obsahuje veškeré komponenty natrénovaných modelů neuronových sítí. Ve složce *util* se nachází pomocné skripty, které slouží především pro sběr dat a tvorbu datových sad. Součástí systému je i soubor *requirements.txt*, který obsahuje informace o knihovnách potřebných k jeho korektnímu spuštění.

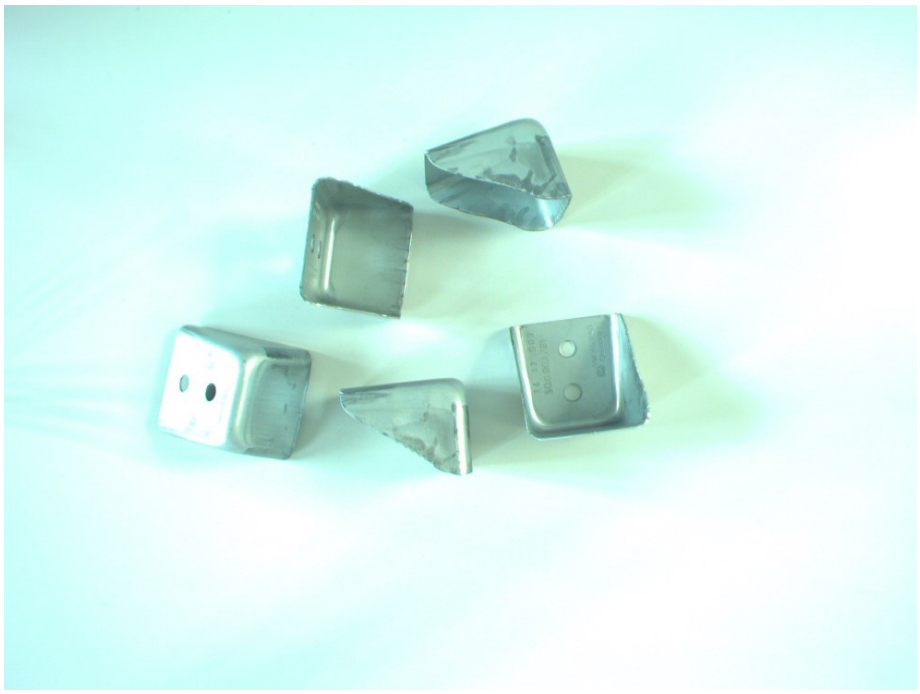


Obrázek 33 – Adresářová struktura systému, zdroj vlastní

6.1 Sběr dat a tvorba datových sad

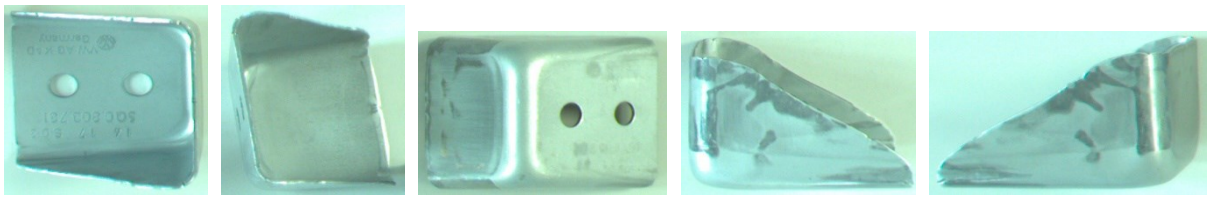
Pro sběr dat byl využit skript *collect_dataset.py*, který přijímá volitelný parametr *Path* (přepínač *-p* nebo *--path)*. Ten udává složku, do které budou získané snímky uloženy, přičemž výchozí hodnota je *../images.* V případě, že taková složka neexistuje nebo není připojena kamera je vyvolána výjimka a skript je ukončen.

Pro ovládání skriptu slouží tři klávesy, konkrétně enter, mezerník a escape. Pomocí klávesy escape je program korektně ukončen, po stisku kláves enter či mezerník je získán snímek typu *.png* a uložen do požadované složky. Název snímku je tvořen datem a časem jeho vytvoření ve formátu *YYYY.mm.dd hh-mm-ss* (např. *2022.03.01 09-28-14.879459*). Takto bylo získáno 500 snímků s rozlišením 2590 × 1942 pixelů, přičemž na každém se vyskytuje dva až pět náhodně rozmístěných objektů.



Obrázek 34 – Ukázka snímku z datové sady, zdroj vlastní

Všechny snímky byly následně anotovány v programu MATLAB Image Labeler. Ačkoliv je rozpoznáván pouze jeden typ objektu, bylo vytvořeno pět tříd, pro každou polohu objektu jedna. Ty byly pracovně nazvány „*roof*“, „*smallBottom*“, „*bigBottom*“, „*rightSideDown*“ a „*leftSideDown*“.



bigBottom smallBottom roof leftSideDown rightSideDown

Obrázek 35 – Příklad objektu pro každou třídu datové sady, zdroj vlastní

Po vytvoření všech anotací bylo zapotřebí provést export anotací do formátu *darknet*. V rámci tohoto formátu je pro každý snímek vytvořen textový soubor obsahující anotace ve tvaru

$$\begin{matrix} class & x_r & y_r & w_r & h_r \\ class & x_r & y_r & w_r & h_r \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{matrix},$$

kde *class* je číselné vyjádření třídy, x_r a y_r jsou relativní polohy středu daného objektu v rozsahu 0 až 1, w_r a h_r jsou relativní výška a šířka daného objektu, opět v rozsahu od 0 do 1. Tento formát ovšem MATLAB nepodporuje, takže bylo nutné exportovat data v jiném formátu. Konkrétně byl vybrán formát *COCO JSON*. K tomuto exportu byla využita funkce *exportGroundTruthToJSON*. Poté byl pro převod do správného formátu využit projekt

JSON2YOLO od společnosti Ultralytics [78]. Takto vytvořená datová sada byla rozdělena na trénovací, validační a testovací množinu v poměru $70:20:10$.

Dále je zapotřebí vytvořit datovou sadu pro natrénování sítě pro klasifikaci úhlu otočení objektů. K tomu byla využita již dříve natrénovaná detekční síť, konkrétně byla využita její funkce *crop*, která vyřízne každý detekovaný objekt z předložených obrázků a uloží ho do složky dle názvu třídy. Pro tento účel byl vytvořen skript *create_classification_dataset.py*.

Objekty v rámci každé třídy byly následně rozděleny do 12 složek podle úhlu otočení od horizontální osy. Jednotlivé skupiny jsou tedy rozděleny po 30°, například všechny objekty na obrázku 35 jsou ze skupin v úhlu otočení 0-30°. Celkem tedy bude klasifikační síť rozlišovat 60 tříd. Rozdělení na jednotlivé množiny zůstalo zachováno včetně jejich poměrů. Obě datové sady jsou k dispozici na [79].

6.2 Trénování modelu YOLOv5

Pro trénování tohoto modelu byl využit jen mírně modifikovaný oficiální notebook pro trénování této architektury na vlastní datové sadě, který je k dispozici na [48]. Tento notebook byl pro vyšší rychlost spouštěn na platformě Google Colaboratory s využitím GPU akcelerátoru výpočtů.

Všechna potřebná data, tedy připravená datová sada a konfigurační soubor *metal-comps.yaml*, jsou na běhové prostředí nahrána prostřednictvím Google Disku. Tento konfigurační soubor obsahuje informace o umístění trénovací, validační a testovací množiny. Dále obsahuje informaci o počtu tříd a jejich seznam ve formě pole, přičemž název třídy je s jejím číselným označením v souborech s anotacemi párován podle indexu v tomto poli.

Vzhledem k malé velikosti datové sady je vhodné pro lepší výsledky využít augmentace dat. Ta je v rámci trénování pomocí skriptu *train.py* nastavena automaticky. Ve výchozím nastavení, které bylo použito i v tomto případě, je prováděna například změna jasu, vertikální převrácení obrázku či jeho posun. Všechny metody včetně konkrétních hodnot jsou nastaveny v konfiguračním souboru *hyp.scratch.yaml.*

Výše zmíněný skript pro trénování má celou řadu parametrů, jako je např. --*img* pro velikost vstupního obrázku či --*batch* pro velikost dávky. V rámci této práce byla většina parametrů ponechána na výchozích hodnotách. Výjimkou jsou pouze parametry --*data* pro zadání konfiguračního souboru datové sady, --*epochs* pro nastavení počtu epoch a --*weights* pro nastavení výchozích vah. Bohužel v době psaní této práce není v dokumentaci dostupný přehled všech parametrů s jejich výchozími hodnotami. Pro zobrazení parametrů je možné spustit

trénovací skript s přepínačem *-h*, ovšem veškeré výchozí hodnoty jsou dostupné pouze v jeho zdrojovém kódu v metodě *parse_opt.*

Pro trénování vzhledem ke specifičnosti datové sady nebylo využito předtrénovaných vah a počet epoch trénování byl nastaven na 100. Tento počet epoch není dostačující, metrika přesnosti detekce *mAP_0.5* se na validační množině pohybuje na úrovni 0,85, ovšem bylo dosaženo limitů základní verze platformy Colaboratory. Tento nedostatek byl obejit opakovaným spuštěním trénování, přičemž jako váhy (parametr *--weights*) jsou nastaveny výsledky předchozího trénování (soubor *best.pt*). Kompletní výsledky trénování viz kapitola 8.1.

6.3 Trénování modelu EfficientNet-B0

Trénování tohoto modelu bylo obdobně jako v předchozím případě provedeno na platformě Google Colaboratory s využitím GPU. Pro tento účel byl využit mírně modifikován notebook pro trénování modelů této architektury, který je dostupný na oficiálních stránkách knihovny Keras [80].

Pro nahrání datové sady je také využit Disk Google. Jednotlivé množiny dat (trénovací, testovací a validační) jsou poté připraveny za pomoci funkce *image_dataset_from_directory*. Ta má jeden povinný parametr, kterým je cesta k připravené datové sadě. Dále následuje celá řada volitelných parametrů, přičemž v této práci byly využity následující:

- *image_size*, který udává hodnotu, na kterou je automaticky změněna velikost obrázků po jejich načtení z disku. Zadává se v pixelech jako *tuple* ve formátu *(výška, šířka)*, výchozí hodnota je *(256, 256)*. Model EfficientNet-B0 přijímá obrázky o velikosti *(224, 224)*
- *seed*, který udává, seed pro generátor pseudonáhodných čísel použitý pro automatické zamíchání datové sady.

I v tomto případě byla využita augmentace dat. V tomto případě, kdy jsou klasifikovány stejné objekty, pouze je rozlišováno jejich otočení, je zapotřebí zvýšená opatrnost. Není totiž možné použít všechny techniky, protože by tím došlo k porušení jednotlivých tříd a model by se tedy nemohl správně natrénovat. Mezi tyto techniky patří například otočení či překlápění obrázku. Z tohoto důvodu bylo využito pouze posunutí, zvětšení/zmenšení obrázku a změna kontrastu.

Počet epoch trénování nebyl určen pevně, ale pro jeho určení bylo využito principu ukončení trénování v případě, že se přesnost modelu nezlepší po dobu 50 epoch od posledního zlepšení. Pro toto nastavení bylo využito *Callbacks API* [81], které slouží k nastavení provádění různých akcí mezi jednotlivými epochami trénování. Toto API bylo také využito pro průběžné ukládání

modelu při jeho zlepšení. Keras totiž jinak ukládá pouze model po provedení poslední epochy, přičemž nastavení jeho vah nemusí být (a obvykle také není) to nejlepší možné.

Vzhledem k poměrně vysoké rychlosti trénování (průměrně 18 s na epochu) bylo trénování provedeno hned dvakrát, jednou bez využití předtrénovaných vah a jednou s nimi. Při trénování s využitím předtrénovaných vah proběhlo trénování na dvě fáze. V první fázi byly zmraženy všechny vrstvy kromě vrchních, které se starají o extrakci důležitých vlastností. Váhy ve zmražených vrstvách nejsou při tréninku modifikovány. Kompletní výsledky trénování viz kapitola 8.2.

6.4 Propojení detekční a klasifikační sítě

O propojení obou sítí do jednoho systému se stará třída *CustomDetections.* Ta je potomkem třídy *Detections* z modelu YOLOv5, která se stará o zpracování výsledků detekce do různých výstupních formátů, jako je např. anotovaný obrázek či *pandas dataframe*. Tento přístup byl zvolen pro zachování co nejvíce funkcionalit detekčního modelu.

CustomDetections přijímá v konstruktoru 4 parametry, konkrétně svého předka, klasifikační neuronovou síť, seznam názvů jednotlivých tříd rozlišovaných klasifikační sítí a velikost vstupu do klasifikační sítě ve formě *(šířka, výška)*. Většina metod této třídy zůstala zachována nezměněná, přepsány byly pouze metody *display* a *pandas*.

Metoda display je stěžejní metodou. Slouží k vytvoření výřezů objektů ze vstupních obrázků, které následně předává na vstup klasifikační síti. Výstupy této sítě jsou uloženy a následně propojeny s informacemi z detekční sítě a se vstupními obrázky, čímž vznikají anotované snímky.

Metoda pandas slouží k propojení informací z klasifikační a detekční sítě, které jsou následně transformovány do podoby *pandas dataframe*. Tyto objekty jsou následně v aplikaci využívány k jednoduchému poskytnutí informací ve formátu JSON.

7 UŽIVATELSKÁ DOKUMENTACE

Jedná se především o experimentální systém, a proto nebylo vytvořeno grafické rozhraní. Jedná se tedy o konzolovou aplikaci, která je spouštěna skriptem *detect.py*. Pro jeho ukončení se potom využívá standardní ukončovací klávesová zkratka *ctrl+c*. Skript přijímá několik parametrů:

- *--help* nebo *-h*, který zobrazí nápovědu k použití
- *--directory* nebo *-d*, který udává složku, ze které budou načteny obrázky k detekci. V případě, že parametr není uveden, je pro detekci využito video z připojené kamery.
- *--image-type nebo -i*, který udává typ detekovaných obrázků. Má smysl pouze v kombinaci s parametrem directory. Výchozí hodnota je *png*.
- *--json nebo -j*, po jehož zadání jsou výstupem i data ve formátu JSON, které jsou posílána na standardní výstupní kanál (*stdout*). Ostatní hlášky pro uživatele jsou posílány na standardní chybový kanál (*stderr*), což umožňuje jejich jednoduché oddělení a případně přesměrování do jiného systému.
- *--fps*, po jehož zadání jsou na chybový výstupní kanál poskytovány průběžné informace o rychlosti detekce ve formě počtu zpracovaných snímků za sekundu. Tyto informace jsou také zobrazovány v levém horním rohu grafického výstupu.
- *--no-display*, kterým se vypíná grafický výstup detekce.

```
(venv) PS C:\Users\vitek\PycharmProjects\yolo\dipl> python detect.py -h
usage: detect.py [-h] [-d DIRECTORY] [-i IMAGE_TYPE] [-j] [--fps] [--no-display]

Script pro detekci objektů z kamery/obázků

optional arguments:
  -h, --help            show this help message and exit
  -d DIRECTORY, --directory DIRECTORY
                        složka,ze které budou načteny obrázky pro detekci, když žádná, tak je bráno video
  -i IMAGE_TYPE, --image-type IMAGE_TYPE
                        formát obrázků, výchozí je png, má smysl pouze s přepínačem -d
  -j, --json            zobrazení výstupu ve formátu Json
  --fps                 zobrazení hodnoty fps
  --no-display          detekce objektů bez zobrazení grafického výstupu
```

Obrázek 36 – Nápověda výsledné aplikace, zdroj vlastní

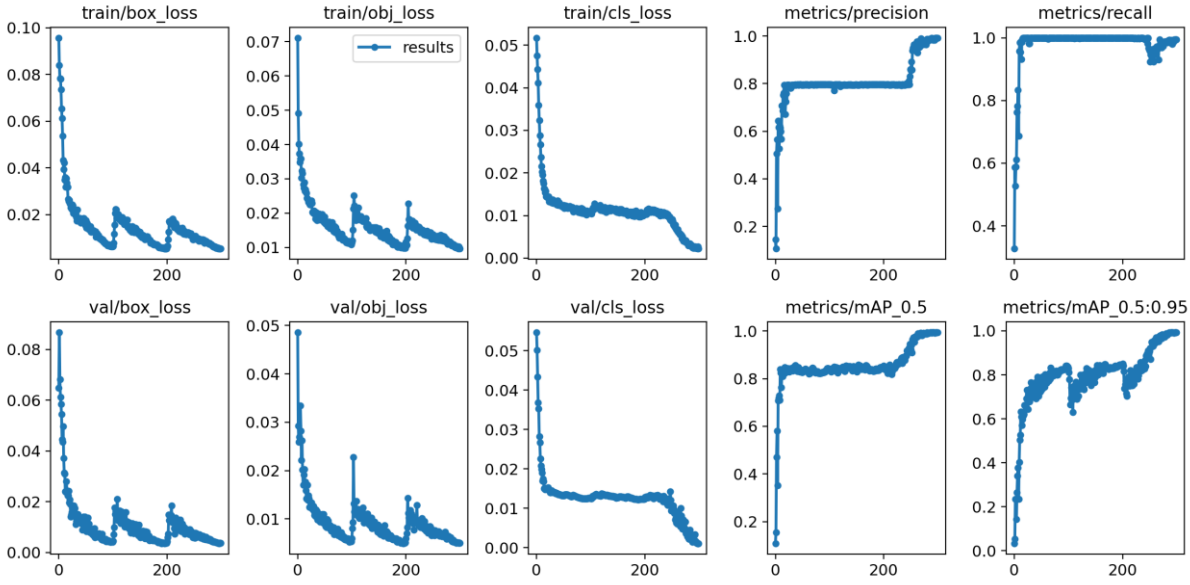
8 TESTOVÁNÍ A VÝSLEDKY

Tato kapitola shrnuje výsledky trénování jednotlivých modelů a ověřuje je testováním na předem připravených testovacích množinách, které nebyly využity při trénování. Dále se věnuje ověření funkčnosti a rychlosti systému jako celku.

Testování jednotlivých modelů proběhlo obdobně jako jejich trénování v prostředí Google Colaboratory. Testování kompletního systému bylo provedeno na osobním počítači s procesorem *AMD Ryzen 4600H* s frekvencí 3 GHz, 16 GB paměti RAM a grafickou kartou *NVIDIA GeForce 1650 Ti*.

8.1 Výsledky trénování a testování modelu YOLOv5s

Pro natrénování modelu YOLOv5s byly využity tři cykly trénování po 100 epochách, celkem tedy proběhlo 300 epoch trénování. Po nich byla dosažena velice dobrá přesnost detekce, konkrétně 0,99 *mAP_0.5* na validační množině. Opakované spouštění trénování má jeden drobný nedostatek, kterým je resetování rychlosti učení pro každý cyklus. To se projevuje dočasným zhoršením některých metrik v prvních epochách trénování, což je názorně vidět na obrázku 37.



Obrázek 37 – Průběh důležitých metrik při trénování modelu YOLOv5, zdroj vlastní

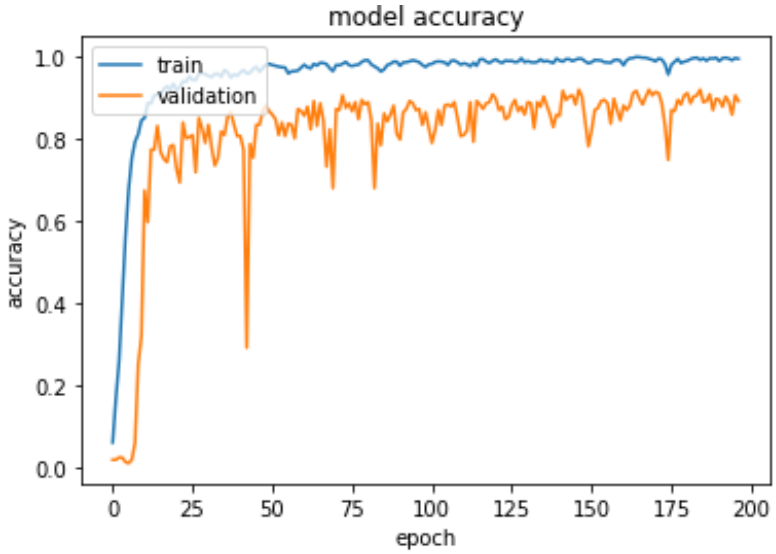
Pro otestování přesnosti detekce bylo využito scriptu *val.py*, kterému byly nastaveny parametry *--data* a *--weights* obdobně jako při trénování. Dále byl využit parametr *--task* s hodnotou test, díky kterému jsou vyhodnoceny hlavní metriky na testovací datové sadě definované v konfiguračním souboru. Z výsledků uvedených v tabulce 5 je zřejmé, že přesnost tohoto modelu je více než dostatečná.

Tabulka 5 – Hodnoty hlavních metrik při testování modelu YOLOv5s, zdroj vlastní

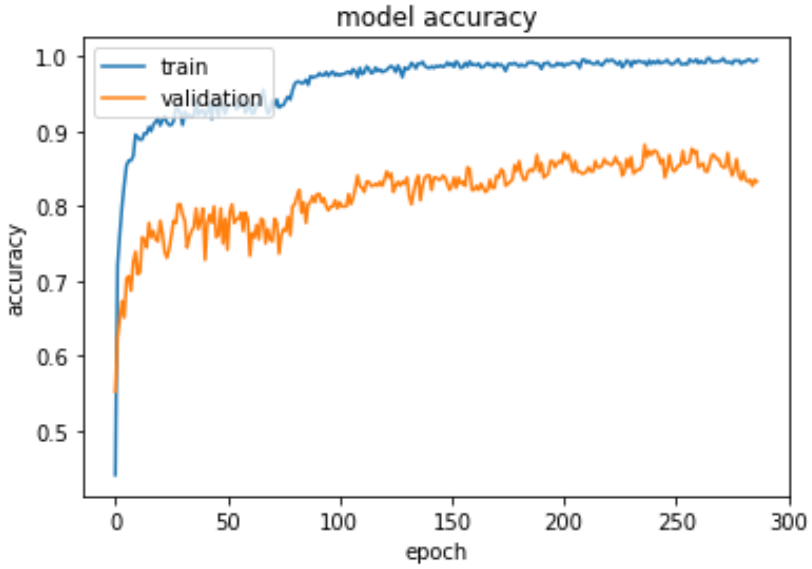
Třída	Precision	Recall	mAP_.5	mAP_.5:.95
Celkově	0,997	0,999	0,995	0,992
roof	0,998	1	0,995	0,995
smallBottom	0,997	1	0,995	0,995
bigBottom	0,998	1	0,995	0,995
rightSideDown	0,992	1	0,995	0,987
leftSideDown	1	0.996	0,995	0,991

8.2 Výsledky trénování a testování modelu EfficientNet-B0

Při trénování modelu bez využití předtrénovaných vah proběhlo celkem 197 epoch, přičemž přesnost klasifikace na validační množině byla v nejlepším případě 91,75 % a hodnota ztrátové funkce 0,421. Při trénování s využitím připravených vah na datové sadě ImageNet proběhlo v první fázi se zmraženými vrstvami 79 epoch s přesností 80,22 % a hodnotou ztrátové funkce 1,49 na validační množině. V druhé fázi proběhlo 209 epoch, přičemž se přesnost modelu na validační množině zlepšila na 88,18 % a hodnota ztrátové funkce na 1,415.



Obrázek 38 – Vývoj přesnosti klasifikace při trénování modelu EfficientNet-B0 bez využití předtrénovaných vah, zdroj vlastní



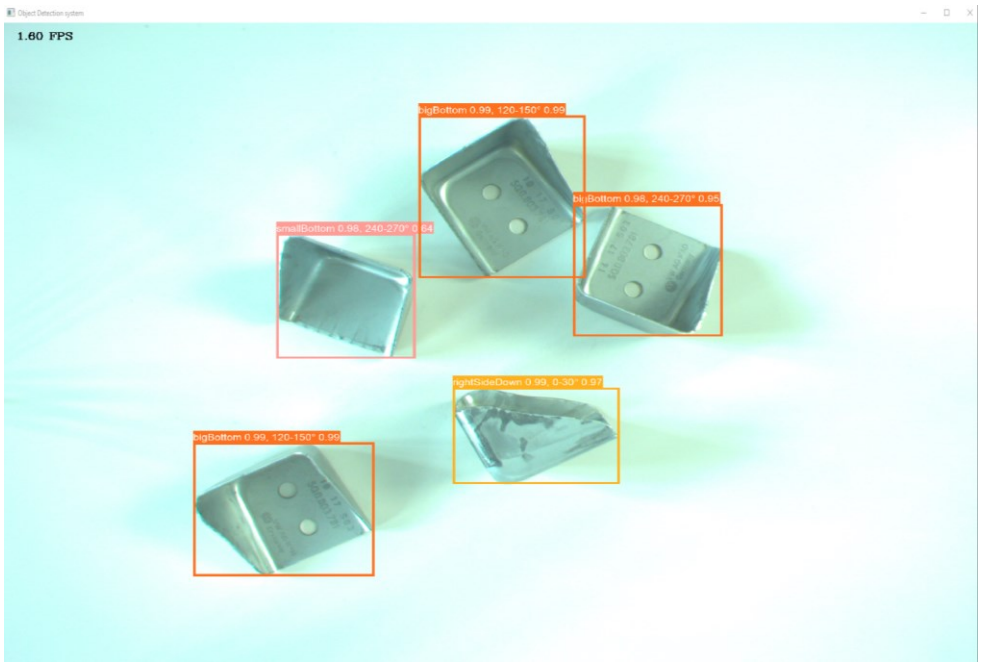
Obrázek 39 – Vývoj přesnosti klasifikace při trénování modelu EfficientNet-B0 s využitím předtrénovaných vah, zdroj vlastní

Při testování byly rozdíly mezi modely ještě markantnější, byť oba modely dosáhly lepších výsledků než na validační množině. U modelu bez přetrénovaných vah byla dosažena přesnost klasifikace 95,1 % a hodnota ztrátové funkce dosáhla hodnoty 0,244. Model s předtrénovanými vahami potom dosáhl přesnosti klasifikace 88,24 % a hodnoty ztrátové funkce 1,13. Vzhledem k tomuto rozdílu byl v systému využit model trénovaný bez využití předtrénovaných vah.

Důvodem jeho vyšší přesnosti je pravděpodobně velká specifičnost datové sady, kdy se jednotlivé třídy objektů liší pouze úhlem otočení, zatímco v klasických datových sadách, jako je ImageNet se vyskytují objekty jedné třídy v různých úhlech otočení. Tento opačný přístup k úhlu otočení pravděpodobně zvýrazňuje problémy modelu s předtrénovanými vahami s přetrénováním (overfitting), které se ovšem v menší míře nevyhýbají ani modelu bez předtrénovaných vah, jak je vidět na obrázcích 38 a 39.

8.3 Testování systému

Účelem testování systému jako celku nebylo zjištění přesnosti, ta byla testována pro každou neuronovou síť zvlášť, ale pouze ověření funkčnosti a také zjištění rychlosti detekce systému. Testování bylo provedeno jak na testovací datové sadě pro neuronovou síť YOLOv5, tak i na živých záběrech z připojené kamery.



Obrázek 40 - Ukázka detekce objektů systémem na testovací sadě, zdroj vlastní

Z obrázku výše je patrné, že detekce objektů včetně určení jejich úhlu natočení probíhá korektně. Rychlost detekce ovšem osciluje přibližně v rozmezí od 1,5 do 3 FPS v závislosti na počtu detekovaných objektů, a to jak na testovací datové sadě, tak na živých záběrech z kamery. Tato rychlost je nižší než se očekávalo, nicméně stále může být dostatečná pro určité typy aplikací, např. při odebírání věcí z přepravky.

ZÁVĚR

V této práci byly nejprve představeny základní úlohy počítačového vidění. Následně byly popsány základy konvolučních neuronových sítí, které stojí za masivním rozvojem této oblasti umělé inteligence a strojového učení. Dále byl zmapován vývoj nejznámějších a nejúspěšnějších architektur pro detekci a klasifikaci objektů v obrazových datech, včetně představení základních metrik určujících jejich přesnost.

V praktické části byl vytvořen detektor kovových součástek určující jejich přesnou polohu, plochu, na které leží, i jeho úhel otočení od horizontální osy při pohledu shora. Pro tento účel byl propojen algoritmus pro detekci objektů YOLO, který určuje polohu těchto součástek a plochu, na které leží, s klasifikačním algoritmem EfficientNet-B0. Ten přijímá výřezy jednotlivých objektů, na jejichž základě provádí klasifikaci jejich úhlu otočení. Vstupem do systému mohou být jednotlivé obrázky uložené ve složce, což je využitelné především pro testování systému, nebo data z připojené kamery, která jsou předávána v reálném čase. Výstupem jsou poté jak obrazová data zobrazující detekované objekty, tak i informace o těchto objektech ve formátu JSON, které potenciálně mohou složit jako vstup do dalších systémů, například pro ovládání robotického manipulátoru.

Pro natrénování jednotlivých modelů byly vytvořeny vlastní datové sady. Samotné trénování i testování přesnosti těchto modelů bylo provedeno, vzhledem k její rychlosti a jednoduchosti použití, na platformě Google Colaboratory. Následně byl systém nasazen na konvenční notebook s grafickou kartou obsahující CUDA jádra, která byla využita pro akceleraci výpočtů neuronových sítí systému. Pro snímání obrazových dat byla využita výkonná, ale levná kamera od společnosti Basler. Při implementaci systému byl kladen důraz na jednoduchost a rychlost vývoje, takže byl použit jazyk Python, který umožňuje velice jednoduché využití knihoven třetích stran. Při výběru těchto knihoven byl mimo jiné kladen důraz na otevřený software, takže byly použity pouze knihovny umožňující jejich volné šíření.

Výsledkem práce rozhodně není produkt připravený k reálnému nasazení do provozu, ale pouze experiment demonstrující možnost využití těchto technologií při řešení podobného typu úloh. Z tohoto důvodu nebylo vytvářeno žádné grafické rozhraní umožňující jednoduché ovládání systému, ale pouze konzolové rozhraní pro testování jednotlivých funkcionalit. Tento prototyp systému ovšem dokazuje způsobilost zvoleného principu k tvorbě systémů vhodných k nasazení do praxe.

POUŽITÁ LITERATURA

[1] SUMA V. COMPUTER VISION FOR HUMAN-MACHINE INTERACTION-REVIEW. In: *Journal of Trends in Computer Science and Smart Technology* [online]. 2019 1(2), 131-139 [cit. 29. 7. 2022]. ISSN: 2582-4104. Dostupné z: https://doi.org/10.36548/jtcsst.2019.2.006

[2] CIPOLLA Roberto a Alex PENTLAND. *Computer Vision for Human-Machine Interaction.* Cambridge: Cambridge university press, 1998. ISBN 978-0521622530.

[3] CACCIOTTI Niccolò. Computer vision: the ultimate guide on the 4 main tasks. In: *Smart Interaction* [online]. 14. 7. 2022 [cit. 30. 7. 2022]. Dostupné z: http://www.smart-interaction.com/2022/07/14/computer-vision-the-ultimate-guide-on-the-4-main-tasks/

[4] LI, Fei-Fei, Justin JOHNSON a Serena YEUNG. *Lecture 11: Detection and Segmentation* [online prezentace]. 10. 5. 2017 [cit. 30. 7. 2022]. Dostupné z: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

[5] FUKUSHIMA, Kunihiko. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. In: *Biological Cybernetics* [online]. 1980, 36, 193-202 [cit. 1. 8. 2022]. ISSN 1432-0770. Dostupné z: https://doi.org/10.1007/BF00344251

[6] LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. In: *Neural Computation* [online]. 1989, 1(4), 541–551. [cit. 9. 8. 2022]. ISSN 0899-7667. Dostupné z: https://doi.org/10.1162/neco.1989.1.4.541

[7] VIOLA, Paul a Michael JONES. Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* [online]. 2001, 1, 511-518. ISSN 1063-6919. Dostupné z: https://doi.org/10.1109/CVPR.2001.990517

[8] DEMUSH, Rostyslav. A Brief History of Computer Vision (and Convolutional Neural Networks). In: *Hackernoon* [online]. 26. 2. 2019 [cit. 1. 8. 2022]. Dostupné z: https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3

[9] EVERINGHAM, Mark et al. The PASCAL Visual Object Classes (VOC) Challenge. In: *International Journal of Computer Vision* [online]. 2010, 88, 303-338. ISSN 1573-1405. Dostupné z: https://doi.org/10.1007/s11263-009-0275-4

[10] DENG, Jia et al. ImageNet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* [online]. 2009. ISSN 1063-6919 Dostupné z: https://doi.org/10.1109/CVPR.2009.5206848

[11] GOODFELLOW Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning.* Cambridge, Massachusetts: The MIT Press, [2016]. ISBN 978-02-620-3561-3.

[12] ALZUBAIDI, Laith, et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. In: *J Big Data* [online] 2021, 8, 53 [cit. 2. 8. 2022]. ISSN 2196-1115. Dostupné z: https://doi.org/10.1186/s40537-021-00444-8

[13] DUMOULIN Vincent a Francesco VISIN. A guide to convolution arithmetic for deep learning. In: *arXiv* [online]. 11. 1. 2018 [cit. 2. 8. 2022]. Dostupné z: https://doi.org/10.48550/arXiv.1603.07285

[14] BROWNLEE Jason. How Do Convolutional Layers Work in Deep Learning Neural Networks? In: *Machine learninig mastery* [online]. 17. 4. 2020 [cit. 3. 8. 2022]. Dostupné z: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks

[15] SEB. Understanding Padding and Stride in Convolutional Neural Networks. In: *Programmathically* [online]. 3. 12. 2021 [cit. 3. 8. 2022]. Dostupné z: https://programmathically.com/understanding-padding-and-stride-in-convolutional-neural-networks

[16] YANG, Jing a Guanci YANG. Modified Convolutional Neural Network Based on Dropout and the Stochastic Gradient Descent Optimizer. In: *Algorithms* [online] 2018, 11 (3), 28 [cit. 5. 8. 2022]. ISSN 1999-4893. Dostupné z: https://doi.org/10.3390/a11030028

[17] BROWNLEE Jason. A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. *Machine learninig mastery* [online]. 5. 7. 2019 [cit. 4. 8. 2022]. Dostupné z: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks

[18] YANI Muhamad, Budhi IRAWAN a Casi SETIANINGSIH. Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. In: *Journal of Physics: Conference Series*. [online] 2019 volume 1201 [cit. 4. 8. 2022]. ISSN: 1742-6596. Dostupné z: https://doi.org/10.1088/1742-6596/1201/1/012052

[19] NAGDA, Rushabh. Evaluating models using the Top N accuracy metrics. In: *medium* [online]. 8. 11. 2019 [cit. 4. 8. 2022]. Dostupné z: https://medium.com/nanonets/evaluating-models-using-the-top-n-accuracy-metrics-c0355b36f91b

[20] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey E. HINTON. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems* [online]. 2012, 25 [cit. 5. 8. 2022]. ISBN 9781627480031. Dostupné z: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[21] We Need To Go Deeper. In: *Know Your Meme* [online]. 18. 4. 2013, last modified on 18. 3. 2018 [cit. 9. 8. 2022]. Dostupné z: https://knowyourmeme.com/memes/we-need-to-go-deeper

[22] SZEGEDY, Christian, et al. Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. 2015 [cit. 9. 8. 2022]. ISBN: 978-1-4673-6964-0. Dostupné z: https://doi.org/10.1109/CVPR.2015.7298594

[23] SIMONYAN, Karen a Andrew ZISSERMAN. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *3rd International Conference on Learning Representations, ICLR 2015* [online]. 2015 [cit. 9. 8. 2022]. Dostupné z: https://doi.org/10.48550/arXiv.1409.1556

[24] HE, Kaming et al. Deep Residual Learning for Image Recognition. *In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. 2016 [cit. 9. 8. 2022]. ISBN 978-1-4673-8851-1. Dostupné z: https://doi.org/10.1109/CVPR.2016.90

[25] IOFFE, Sergey a Christian SZEGEDY. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: *Proceedings of the 32nd International Conference on Machine Learning* [online]. 2015, 37. 448-456 [cit. 9. 8. 2022]. Dostupné z: https://doi.org/10.48550/arXiv.1502.03167

[26] CHOLLET, Francois. Xception: Deep Learning with Depthwise Separable Convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. 2017, 1251-1258 [cit. 10. 8. 2022]. ISBN 978-1-5386-0457-1 Dostupné z: https://doi.org/10.1109/CVPR.2017.195

[27] HUANG, Gao et al. Densely Connected Convolutional Networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online] 2017, 4700-4708 [cit. 10. 8. 2022]. ISBN 978-1-5386-0457-1. Dostupné z: https://doi.org/10.1109/CVPR.2017.243

[28] YING, Xue. An Overview of Overfitting and its Solutions. In: *Journal of Physics: Conference Series* [online]. 2019, 1168 (2). ISSN: 1742-6596. Dostupné z: https://doi.org/10.1088/1742-6596/1168/2/022022

[29] TAN, Mingxing a Quoc LE. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In: *Proceedings of the 36th International Conference on Machine Learning, PMLR* [online]. 2019, 97, 6105-6114 [cit. 10. 8. 2022]. ISSN 2640-3498. Dostupné z: https://doi.org/10.48550/arXiv.1905.11946

[30] TAN, Mingxing et al. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. 2019, 2820-2828 [cit. 11. 8. 2022]. ISBN 978-1-7281-3293-8. Dostupné z: https://doi.org/10.1109/CVPR.2019.00293

[31] SANDLER, Mark et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. 2018, 4510-4520 [cit. 11. 8. 2022]. ISBN 978-1-5386-6420-9. Dostupné z: https://doi.org/10.1109/CVPR.2018.00474

[32] LIN, Tsung-Yi et al. Microsoft COCO: Common Objects in Context. In: *Lecture Notes in Computer Science* [online]. 2014, 8693. ISBN 978-3-319-10602-1. Dostupné z: https://doi.org/10.1007/978-3-319-10602-1_48

[33] ZENG, Guangyu. An Introduction to Evaluation Metrics for Object Detection. In: *NickZeng*| *曾广宇* [online]. 16. 12. 2018 [cit. 10. 8. 2022]. Dostupné z: https://blog.zenggyu.com/en/post/2018-12-16/an-introduction-to-evaluation-metrics-for-object-detection/

[34] GIRSHICK, Ross, et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [online] 2014. 580–587 [cit. 5. 8. 2022]. ISSN: 1063-6919. Dostupné z: https://doi.org/10.1109/CVPR.2014.81

[35] UIJLINGS, Jasper RR, et al. Selective search for object recognition. In: *International journal of computer vision*, [online] 2013, 104, 154-171 [cit. 5. 8. 2022]. ISSN: 1573-1405. Dostupné z: https://doi.org/10.1007/s11263-013-0620-5

[36] HOSANG, Jan, Rodrigo BENESON a Bernt SCHIELE. Learning Non-maximum Suppression. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [online] 2017, 6469-6477 Dostupné z: https://doi.org/10.1109/CVPR.2017.685

[37] GIRSHICK, Ross. Fast R-CNN. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. [online] 2015, 1440-1448 [cit. 5. 8. 2022]. ISSN 2380-7504. Dostupné z: https://doi.org/10.1109/ICCV.2015.169

[38] REN, Shaoqing, et al. Faster R-CNN: Towards real-time object detection with region proposal networks. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. [online] 2017, 39 (6), 1137-1149 [cit. 6. 8. 2022]. ISSN 1939-3539. Dostupné z: https://doi.org/10.1109/TPAMI.2016.2577031

[39] BAJWA, Muhammad Naseer, et al. Two-stage framework for optic disc localization and glaucoma classification in retinal fundus images using deep learning. In: *BMC Medical Informatics and Decision Making.* [online] 2019 19 (136). ISSN 1472-6947. Dostupné z: https://doi.org/10.1186/s12911-019-0842-8

[40] LIU, Wei, et al. SSD: Single shot multibox detector. *ECCV: European Conference on Computer Vision.* [online] 2016, 21-37 [cit. 6. 8. 2022] ISSN 1611-3349. Dostupné z: https://doi.org/10.1007/978-3-319-46448-0_2

[41] LIN, Tsung-Yi, et al. Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* [online] 2020, 42 (2), 318-327 [cit. 6. 8. 2022]. ISSN 1939-3539. Dostupné z: https://doi.org/10.1109/TPAMI.2018.2858826

[42] LIN, Tsung-Yi, et al. Feature Pyramid Networks for Object Detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition CVPR).* [online] 2017, 2117-2125 [cit. 7. 8. 2022]. ISBN 978-1-5386-0457-1. Dostupné z: https://doi.org/10.1109/CVPR.2017.106

[43] REDMON, Joseph, et al. You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* [online] 2016 [cit. 7. 8. 2022]. ISBN 978-1-4673-8851-1. Dostupné z: https://doi.org/10.1109/CVPR.2016.91

[44] REDMON, Joseph a Ali FARHADI. YOLO9000: Better, Faster, Stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* [online] 2017 [cit. 7. 8. 2022]. ISBN 978-1-5386-0457-1. Dostupné z: https://doi.org/10.1109/CVPR.2017.106

[45] REDMON, Joseph a Ali FARHADI. YOLOv3: An Incremental Improvement. In: *arXiv* [online]. 8. 4. 2018 [cit. 7. 8. 2022]. Dostupné z: https://doi.org/10.48550/arXiv.1804.02767

[46] REDMON, Joseph. [I stopped doing…]. In: *Twitter* [online]. 20. 2. 2020 [cit. 8. 8. 2022]. Dostupné z: https://twitter.com/pjreddie/status/1230524770350817280

[47] BOCHKOVSKIY, Alexey, WANG, Chien-Yao a Hong-Yuan Mark LIAO. YOLOv4: Optimal Speed and Accuracy of Object Detection. In: *arXiv* [online]. 23. 4. 2020 [cit. 8. 8. 2022]. Dostupné z: https://doi.org/10.48550/arXiv.2004.10934

[48] JOCHER, Glenn et al. YOLOv5. In: *GitHub* [online]. 29. 5. 2020, last modified on 8. 8. 2022 [cit. 8. 8. 2022]. Dostupné z: https://github.com/ultralytics/yolov5

[49] NELSON, Joseph a Jacob SOLAWETZ. Responding to the Controversy about YOLOv5. In: *Roboflow Blog* [online]. 12.6.2020 [cit. 8. 8. 2022]. Dostupné z: https://blog.roboflow.com/yolov4-versus-yolov5/

[50] TAN, Lu et al. Comparison of RetinaNet, SSD, and YOLO v3 for real-time pill identification. In: *BMC Medical Informatics and Decision Making* [online]. 2021, 21, 324 [cit. 13. 8. 2022]. ISSN 1472-6947. Dostupné z: https://doi.org/10.1186/s12911-021-01691-8

[51] CHOLLET, François, et al. Keras Applications. *Keras* [online]. c2022 [cit. 13. 8. 2022]. Dostupné z: https://keras.io/api/applications/

[52] Basler AG. Company Profile. *Basler* [online]. c2022 [cit. 28. 7. 2022]. Dostupné z: https://www.baslerweb.com/en/company/about-us/

[53] Basler AG. Basler ace – Area Scan Cameras. *Basler* [online]. c2022 [cit. 28. 7. 2022]. Dostupné z: https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/#productline=aceclassic

[54] Basler AG. Lens Selector. *Basler* [online]. c2022 [cit. 29. 7. 2022]. Dostupné z: https://www.baslerweb.com/en/sales-support/tools/lens-selector/

[55] CBC AMERICA LLC. About Computar. *Computar* [online]. c2022 [cit. 29. 7. 2022]. Dostupné z: https://computar.com/page/612/about-computar

[56] CBC AMERICA LLC. MPZ Machine Vision Series H0514-MP2: Manual Iris: Megapixel Monofocal Lenses. *Computar* [online]. c2022 [cit. 29. 7. 2022]. Dostupné z: https://computar.com/product/551/H0514-MP2

[57] Basler AG. pylon Open Source. *Basler* [online]. c2022 [cit. 28. 7. 2022]. Dostupné z: https://www.baslerweb.com/en/products/basler-pylon-camera-software-suite/pylon-open-source-projects/

[58] Basler AG. pylon Viewer – pylon Camera Software Suite. *Basler* [online]. c2022 [cit. 28. 7. 2022]. Dostupné z: https://www.baslerweb.com/en/products/basler-pylon-camera-software-suite/pylon-viewer/

[59] Python Software Foundation. What is Python? Executive Summary. *Python* [online]. c2022 [cit. 23. 7. 2022]. Dostupné z: https://www.python.org/doc/essays/blurb

[60] RedHat. What is Python? *Opensource* c2022 [cit. 23. 7. 2022]. Dostupné z: https://opensource.com/resources/python

[61] ASCANY Phillip. Using Python's pip to Manage Your Projects' Dependencies. *Realpython* [online]. 2. 2. 2022 [cit. 24. 7. 2022]. Dostupné z: https://realpython.com/what-is-pip/

[62] Basler AG. pylon Open Source. *Basler* [online]. c2022 [cit. 29. 7. 2022]. Dostupné z: https://www.baslerweb.com/en/products/basler-pylon-camera-software-suite/pylon-open-source-projects/

[63] JOHNSON Daniel. What is TensorFlow? How it Works? Introduction & Architecture. *Guru99* [online]. 16. 7. 2022 [cit. 26. 7. 2022]. Dostupné z: https://www.guru99.com/what-is-tensorflow.html

[64] NVIDIA Corporation. What is TensorFlow? *Nvidia* [online]. 2. 7. 2021 [cit. 26. 7. 2022]. Dostupné z: https://www.nvidia.com/en-us/glossary/data-science/tensorflow/

[65] Google. Install TensorFlow with pip. *TensorFlow* [online]. 3. 6. 2022 [cit. 26. 7. 2022]. Dostupné z: https://www.tensorflow.org/install/pip

[66] DUTH Parth. Understanding Tensor Processing Units. GeeksforGeeks [online]. 4. 8. 2018 [cit. 26. 7. 2022]. Dostupné z: https://www.geeksforgeeks.org/understanding-tensor-processing-units

[67] CHOLLET, François, et al. About Keras. *Keras* [online]. c2022 [cit. 26. 7. 2022]. Dostupné z: https://keras.io/about/

[68] NVIDIA Corporation. What is PyTorch? *Nvidia* [online]. 2. 7. 2021 [cit. 26. 7. 2022]. Dostupné z: https://www.nvidia.com/en-us/glossary/data-science/pytorch/

[69] SPISAK Joe. Get started with PyTorch, Cloud TPUs, and Colab. *Medium* [online]. 10. 3. 2021 [cit. 26. 7. 2022]. Dostupné z: https://medium.com/pytorch/get-started-with-pytorch-cloud-tpus-and-colab-a24757b8f7fc

[70] JOCHER Glenn. *YOLOv5 Documentation* [online]. 18. 5. 2022 [cit. 28. 7. 2022]. Dostupné z: https://docs.ultralytics.com/

[71] O'CONNOR Ryan. PyTorch vs TensorFlow in 2022. In: *AssemblyAi* [online]. 14. 12. 2021 [cit. 28. 7. 2022]. Dostupné z: https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/

[72] MADAN Krisma. What is Matplotlib in Python? In: *Scaler* [online]. 25. 6. 2022 [cit. 26. 7. 2022]. Dostupné z: https://www.scaler.com/topics/matplotlib-in-python

[73] Google. Colaboratory, Frequently Asked Questions. *Research Google* [online]. c2022 [cit. 24. 7. 2022]. Dostupné z: https://research.google.com/colaboratory/

[74] Google. Welcome To Colaboratory. *Colab Research Google* [online]. c2022 [cit. 24. 7. 2022]. Dostupné z: https://colab.research.google.com/

[75] ARORA Simran Kaur. What is PyCharm? Features, Advantages & Disadvantages. In: *hackr.io* [online]. 2. 3. 2022 [cit. 28. 7. 2022]. Dostupné z: https://hackr.io/blog/what-is-pycharm

[76] HOUCQUE, David. Introduction to matlab for engineering students. Northwestern University. 2005 [cit. 12. 8. 2022]. Dostupné z: https://www.mccormick.northwestern.edu/documents/students/undergraduate/introduction-to-matlab.pdf

[77] The MathWorks Inc. *Get Started with the Image Labeler. MathWorks* [online]. c2022 [cit. 11. 8. 2022]. Dostupné z: https://www.mathworks.com/help/vision/ug/get-started-with-the-image-labeler.html

[78] JOCHER, Glenn et al. JSON2YOLO. In: *GitHub* [online]. 11. 5. 2019, last modified on 14. 5. 2022 [cit. 16. 8. 2022]. Dostupné z: https://github.com/ultralytics/JSON2YOLO

[79] RAIS, Vítek. Object detection in visual data. In: *Kaggle* [online]. 24. 8. 2022, [cit. 24. 8. 2022]. Dostupné z: https://www.kaggle.com/datasets/vitekrais/diploma-theses-object-detection-in-visual-data

[80] FU, Yixing. Image classification via fine-tuning with EfficientNet. In: *keras* [online]. 30. 6. 2020, last modified on 16. 7. 2020 [cit. 16. 8. 2022]. Dostupné z: https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/

[81] CHOLLET, François, et al. Callbacks API. *Keras* [online]. c2022 [cit. 17. 8. 2022]. Dostupné z: https://keras.io/api/callbacks/