VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

AUTOMATICKÁ DETEKCE OVLÁDACÍCH PRVKŮ VÝTAHU ZPRACOVÁNÍM DIGITÁLNÍHO OBRAZU
AUTOMATIC DETECTION OF ELEVATOR CONTROLS USING IMAGE PROCESSING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Martin Černil**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Jiří Krejsa, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Bc. Martin Černil**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**doc. Ing. Jiří Krejsa, Ph.D.**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Automatická detekce ovládacích prvků výtahu zpracováním digitálního obrazu

Stručná charakteristika problematiky úkolu:

Jednou z netriviálních úloh v mobilní robotice je schopnost mobilního robotu pohybovat se mezi patry budov pomocí běžného osobního výtahu. Ke splnění této úlohy musí být robot schopen detekovat ovládací prvky výtahu. Cílem diplomové práce je implementovat a ověřit metodu, která bude schopna automatické detekce ovládacích prvků z obrazu palubní kamery robotu.

Cíle diplomové práce:

1. Seznamte se s metodami zpracování obrazu.
2. Vytvořte dostatečně mohutnou množinu obrazů ovládacích prvků výtahu.
3. Vyberte vhodnou metodu detekce.
4. Metodu implementujte, ověřte na vytvořené množině obrazů a vyhodnoťte.

Seznam doporučené literatury:

Antonio GULLI, Amita KAPOOR, Sujit PAL: Deep Learning with TensorFlow 2 and Keras: Regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API, Packt Publishing, 2019.

Tensorflow online dokumentace (www.tensorflow.org).

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2020/21

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt

Tato diplomová práce se zabývá automatickou detekcí ovládacích prvků osobních výtahů v digitálním obraze pomocí metod počítačového vidění. V teoretické části je provedena rešerše metod zpracování obrazu s návazností na rozpoznávání objektů v obraze a průzkum předchozích řešení, které vede k probádání oblasti konvolučních neuronových sítí. V praktické části je probráno vytvoření datové sady obrazů ovládacích prvků výtahu, výběr, učení a vyhodnocení modelů za použití TensorFlow Object Detection API 2 a implementace robustního algoritmu využívajícího detektoru ovládacích prvků výtahu. Závěr práce pojednává o vhodnosti modelu pro zadanou úlohu.

Abstract

This thesis deals with the automatic detection of elevator controls in personal elevators through digital imaging using computer vision. The theoretical part of the thesis goes through methods of image processing with regards to object detection in image and research of previous solutions. This leads to investigation into the field of convolutional neural networks. The practical part covers the creation of elevator controls image dataset, selection, training and evaluation of the used models and the implementation of a robust algorithm utilizing the detection of elevator controls. The conclussion of the work discusses the suitability of the detection on given task.

Klíčová slova

Detekce objektů, Počítačové vidění, Konvoluční neuronové sítě, Tensorflow Object Detection API 2, Detekce ovládacích prvků výtahu

Keywords

Object Detection, Computer Vision, Convolutional Neural Networks, Tensorflow Object Detection API 2, Elevator Controls Detection

Bibliografická Citace

ČERNIL, Martin. *Automatická detekce ovládacích prvků výtahu zpracováním digitálního obrazu.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. 2021. 88 s., Vedoucí diplomové práce: doc. Ing. Jiří Krejsa, PhD.

Prohlášení

Prohlašuji, že jsem diplomovou práci na téma „Automatická detekce ovládacích prvků výtahu zpracováním digitálního obrazu“ vypracoval samostatně s použitím odborné literatury a pramenů uvedených v seznamu použitých zdrojů.

Martin Černil

Brno 18.5.2021 . . . . . .

Poděkování

Děkuji tímto doc. Ing. Jiřímu Krejsovi, PhD. za cenné připomínky, výtky a rady při vypracování diplomové práce.

Martin Černil

Obsah

1 Úvod

Již od šedesátých let se začalo experimentovat s počítačovým viděním na univerzitách zabývajících se umělou inteligencí. Snaha o napodobení lidského smyslu vidění pro vytvoření vhledu o zpracovávaném obraze byla jednou z prvních aplikací počítačového vidění. Dále se vývoj tohoto odvětví ubíral směrem vytváření algoritmů vyhledávajících hrany, odhadující pohyb a extrahujících prominentní rysy objektů v obraze. S nástupem grafických výpočetních jednotek se tyto metody staly dostupnějšími.

Recentní průlom v posledním desetiletí u metod hlubokého strojového učení dopomohl k prominentnímu postavení počítačové vize v rámci technologií umělé inteligence. Počítačová vize se stává součástí každodenního života, například u chytrých telefonů se uplatňuje v jejich odemykání obličejem vlastníka, v interpretaci a vylepšení pořízených fotografií či u překladu cizojazyčných názvů z obrazu. Výčtem dalších uplatnění počítačového vidění je kontrola úrody a zvířat v zemědělství, monitorování dopravy a autonomní řízení, sledování pohybů osob v obchodních střediscích, diagnóza pacienta na základě rentgenového snímku, kontrola produktů ve výrobě či interpretace okolí mobilního robotu.

Přestože navigace robotů ve vnitřních prostorech je v recentní době úspěšně probádaným tématem, robotické platformy nejsou běžně schopny pohybovat se autonomně skrze všechny podlaží budovy. Nejčastěji realizována možnost přesunu mezi podlažími je využitím osobních výtahů budovy. Pro automatizaci tohoto kroku je klíčovým krokem schopnost autonomní obsluhy výtahového zařízení, přičemž při generalizaci této schopnosti na neznámá výtahová zařízení nelze využít předem naprogramovaných pokynů. Pro realizaci změny podlaží robotu přivoláním výtahu ze stanice následované stiskem žádaného podlaží v kabině je tedy stěžejní tyto ovládací prvky rozeznat.

Tato práce se zabývá automatickou detekcí ovládacích prvků (například ovládačů volby podlaží, ovládačů otevírání kabinových dveří apod.) výtahu v digitálním obrazu. Práce je součástí většího systému, jenž umožní robotické platformě autonomně obsluhovat výtahové zařízení a změnit podlaží v rámci budovy, a to bez nutnosti úpravy výtahového zařízení. Realizace této dílčí detekční úlohy využívá metod počítačového vidění a umělé inteligence.

Cílem práce je implementovat a ověřit vhodnou metodu, která bude schopna tyto ovládací prvky výtahového prostředí rozeznat z dodaných obrazových dat. Jedná se tedy o detekci instancí objektů, jejichž rešerší se (společně s průzkumem předchozích specifických řešení) zabývá úvodní část. Dále je nutné vytvořit dostatečně mohutnou množinu obrazů ovládacích prvků výtahů a zvolit vhodnou metodu jejich detekce. Tuto zvolenou metodu je poté potřeba implementovat a ověřit její funkčnost.

Vybraná metoda bude v budoucnu použita jako část systému autonomní obsluhy výtahového zařízení robotickou platformou, kde umožní detekci ovládačů výtahových zařízení z obrazu palubní kamery robotu.

2 Rešeršní část

2.1 Zpracování digitálního obrazu

Tato kapitola rešeršní části se zabývá nejčastějšími typy zpracování obrazu s ohledem na detekci objektů. Konkrétně se jedná o práci s digitálním obrazem, který je interpretován jako funkce a matice, filtrování obrazu, principy segmentace obrazu a detekci charakteristických rysů v obraze.

2.1.1 Digitální obraz jako funkce a matice

Obraz lze považovat za funkci $f(x, y)$, kde každý pixel má přiřazeny souřadnice x, y a intenzitu 0-255 (pro 24 bitovou hloubku obrazu, nejčastější). Obraz může mít jeden či více barevných kanálů, konkrétně v práci byl zvažován barevný prostor RGB, sestávající ze tří kanálů (červená, zelená, modrá), kde jsou každému pixelu přiřazeny právě tři intenzity pro každou z uvedených barev. S obrazem tedy lze pracovat jako s maticí, která obsahuje stejný počet seřazených prvků, jako je počet pixelů v obrazu, a každý prvek obsahuje tolik členů, kolik má obraz barevných kanálů.

Obrázek 2.1: Vzorový digitální obrázek použitý dále v kapitole

2.1.2 Filtrování obrazu

Filtrování obrazu je důležitý proces během jeho zpracování, který se používá především pro odstranění šumu, odstranění rozmazání, zviditelnění hran a jim opačných operací. Rozlišují se dva typy algoritmů filtrování, a to lineární a nelineární.[1] V následující tabulce 2.1 jsou znázorněny nejčastější algoritmy společně s jejich využitím ve zpracování obrazu.

Typ	**Použití**	**Myšlenka**	**Ukázka**
Gaussian	vyhlazování obrazu	použití Gaussova filtru, dis-krétně například $\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}$	
Median	vyhlazování obrazu, zachování hran	pixel výstupního obrazu je mediánem svého okolí ve vstupním obraze	
Bilateral	vyhlazování obrazu, zachování hran	filtrace se zvážením prostorové i hodnotové blízkosti pixelů	
Laplacian	zvýraznění oblastí s velkou změnou hodnoty pixelů v okolí	výpočet druhé derivace obrazu, používán kernel $\begin{bmatrix}0 & -1 & 0\\-1 & 4 & -1\\0 & -1 & 0\end{bmatrix}$	
Unsharp	zvýraznění hran	odečtení vyhlazeného obrazu od originálního	
Sobel	extrakce hran	kombinovaný gradient oblastí, kombinace kernelů $\begin{bmatrix}1 & 0 & -1\\2 & 0 & -2\\1 & 0 & -1\end{bmatrix}$ a $\begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix}$	

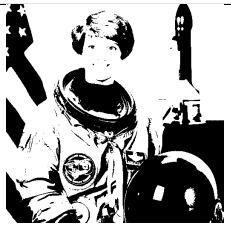
Tabulka 2.1: Příklady filtrace digitálního obrazu

2.1.3 Segmentace obrazu

Segmentace obrazu spočívá v rozdělení oblastí obrazu na jednotlivé části, nejčastěji podle nespojitostí, či naopak podobností v obraze, při předem zvolených kritériích.[2] V případě rozdělení dle nespojitostí je předpokládaná hranice mezi částmi obrazu dostatečně rozdílná

od svého okolí a pozadí. V těchto případech je hlavní používanou metodou segmentace na základě hran (hledání bodů, přímek, hran).

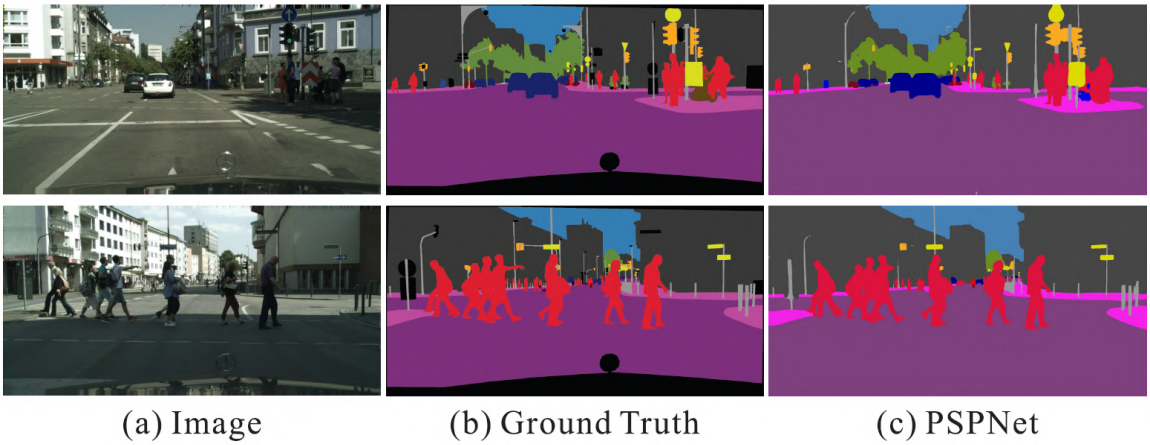
V případě rozdělení dle podobností v obraze lze využít prahování[1] (například dle intenzity zvolené barvy), rozrůstání regionů, shlukování a superpixelů. Znázornění jmenovaných procesů je nastíněno v tabulce 2.2.

Typ segmentace	**Myšlenka**	**Ukázka**
Detekce hran	Extrakce hran a vyplnění těchto oblastí	
Binární prahování	Výběr vhodného prahu pro binarizaci obrazu, zde metoda Otsu	
Rozrůstání regionů Watershed	Vyplnění oblastí s nižší intenzitou, zde použit gradient obrazu	
Clustering SLIC	Spojování oblastí s podobnými vlastnostmi, využití algoritmu K-means	
Na základě reprezentace obrazu grafem Felzenszwalb	Reprezentace regionů obrazu jako uzly a spojování pomocí podobných hran mezi nimi	

Tabulka 2.2: Segmentace digitálního obrazu

V recentní době se také rozmáhá segmentace na základě hlubokého učení, a to od nejjednodušších enkodérů-dekodéru po plně propojené konvoluční neuronové sítě, rekurentní neuronové sítě, generativní sítě a mnoho dalších.[4] Ukázka segmentace metodami hlubokého učení s názvem použité metody je nastíněna v obrázku 2.2.

[1] přiřazení pravdivostní hodnoty pixelu na základě splněné podmínky



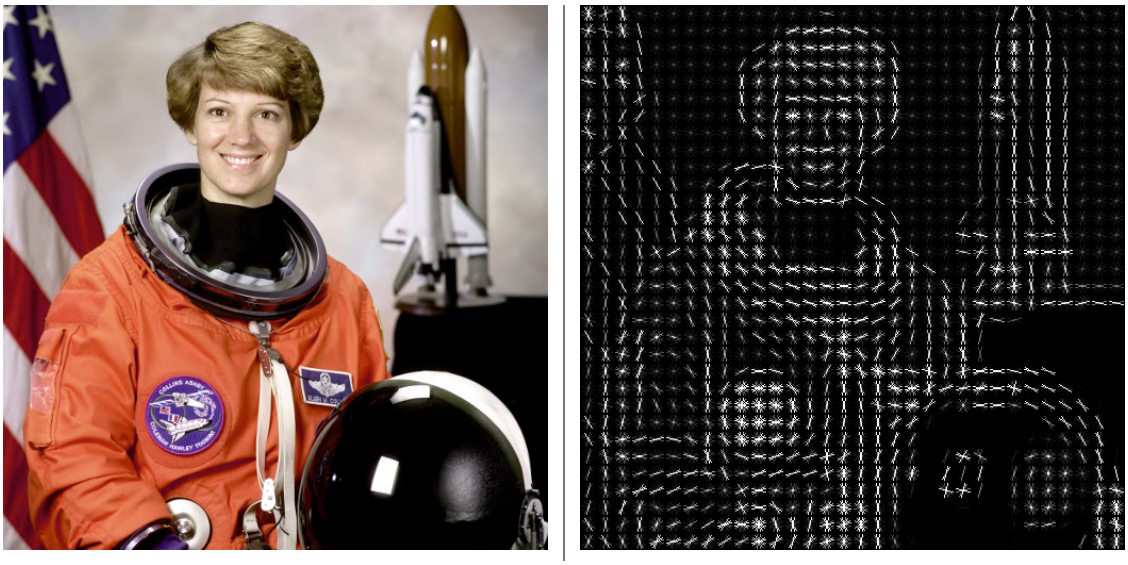
Obrázek 2.2: Segmentace digitálního obrazu neuronovou sítí PSPNet[3]

2.1.4 Extrakce charakteristických rysů

Typickým dalším krokem po segmentaci obrazu je extrakce charakteristických rysů, která sestává z jejich detekce a popisu. Detekce rysů se zabývá vyhledáváním rysů v obraze, regionu, či hranici. Popis rysu poté přiřazuje těmto rysům kvantitativní znaky, jako je například orientace a poloha. Je snaha učinit tyto popisy invariantní vůči rozměrům, translaci, rotaci, osvětlení a perspektivě.

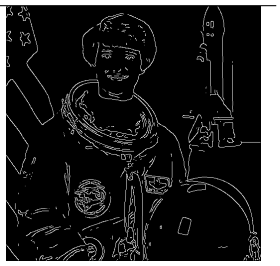
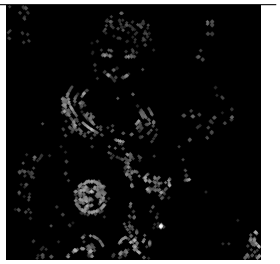
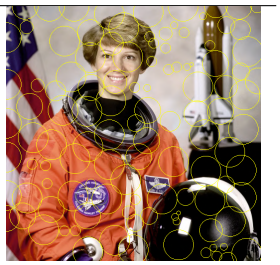
Nejčastějšími typy vyhledávaných rysů jsou hrany, rohy (body), blobs[2], ridges[3] a tvary. Některé detektory těchto vyhledávaných rysů jsou znázorněny v tabulce 2.3.

Pro popis těchto rysů je poté používáno algoritmů SIFT, SURF, ORB, kdy tyto algoritmy jsou podrobněji přiblíženy v článku Akalanky S. Perery uvedeném ve zdroji [5]. Metoda popisů rysů HOG (histogram orientovaných gradientů) je vizualizována na obrázku 2.3.



Obrázek 2.3: Ukázka popisu rysů metodou HOG

[2]regiony obrazu odlišující se svými vlastnostmi
[3]skupiny sousledných bodů s obdobně odlišnými vlastnostmi

Typ extrahovaných rysů	**Použitá metoda**	**Ukázka**
Hrany	Canny detektor hran	
Rohy	FAST - extrakce rysů testováním akcelerovaného segmentu	
Blobs	s použitím DoG - diference obrazů zpracovaných gaussovým filtrem	
Blobs	extrakcí lokálních extrémů DoH - determinant hessiánu obrazu	
Tvary	selektivní extrakcí lokálních extrémů v Houghově prostoru - zde extrakce kružnic	
Tvary	CCL - značení spojených komponent	

Tabulka 2.3: Extrakce charakteristických rysů obrazu

2.2 Rozpoznávání objektů v obraze

Snaha o rozpoznání instancí sémantických objektů z digitálního obrazového vstupu je hlavním cílem klasifikace objektů. Nejprobádanější doménou této technologie je detekce dopravy, chodců a obličejů.[6]

V této kapitole jsou přiblíženy možné postupy k řešení problému klasifikace objektů v digitálním obraze. Tyto metody lze dělit následovně:

- Klasifikace na základě srovnávání se vzorem
- Klasifikace na základě optima statistické formulace
- Klasifikace na základě neuronových sítí

První dvě jsou užívány pro aplikace, u kterých je známa podstata objektu. Tato podstata objektu se poté promítá ve zřejmých a jednoznačných charakteristických rysech a prvcích objektu, pro které člověk dokáže navrhnout klasifikátor. Postup s neuronovými sítěmi je méně závislý na znalosti podstaty objektu, jelikož algoritmus se sám učí využívat význačné charakteristické rysy a prvky objektů.[18]

Společnou charakteristikou těchto metod je jejich základ na parametrech, které musí být specifikovány nebo naučeny z předložených vzorů[4]. Tyto parametry následně reflektují specifický problém rozpoznávání objektů, který je aktuálně řešen. Vzory použité v těchto metodách mohou být:

- Označené – známe třídu každého vzoru (například při rozpoznávání jednotlivých znaků A-Z, 0-9)
- Neoznačené - neznáme třídu každého vzoru (například při hledání čárových kódů v obraze)

2.2.1 Klasifikace na základě srovnání se vzorem

Hlavním cílem tohoto druhu klasifikace je učinit rysy objektů tak unikátními a jednoduše rozlišitelnými, aby klasifikace samotná se stala jednoduchou úlohou (například čtení strojového textu známého druhu písma). Při této klasifikaci vzorů se rozlišuje mezi kvantitativními vzory (vektorové vzory) a strukturálními vzory (symboly uspořádané ve formě řetězců, stromů nebo grafů). Vektorové vzory jsou reprezentovány jako vektor:

$$\boldsymbol{x}^{\mathbf{T}} = [x_1, x_2, \ldots x_n] \tag{2.1}$$

kde každý element x_i reprezentuje i-tý kvantitativní znak rysu a n je počet popisů rysu. Tyto vektorové vzory lze vytvořit vektorizací celého obrazu, pomocí vektorizací hranic a regionů[18] či určením vlastností binárního obrazu. Jako příklad použitelných kvantitativních znaků rysů lze uvést polohu v obraze, momenty, zaoblení hran, homogenitu a excentricitu objektu.

Strukturální vzory jsou méně používanou alternativou vektorových vzorů. Jejich použití je velmi vhodné u aplikací, kde je důležitý tvar objektu. Samotné strukturální vzory jsou reprezentovány strukturálními rysy, například řetězcem symbolů reprezentujících hranici objektu, stromem popisujícím celý obraz sestaveným z dílčích popisů částí obrazů či grafem popisujícím vlastnosti významných objektů v obraze a vztahy mezi nimi.[18]

[4]instance objektů, které jsou předmětem klasifikace

Principem klasifikace je následně rozdělení stavového prostoru těchto vzorů na samostatné kategorie objektů.

Nejčastější a nejjednodušší metodou srovnávání se vzorem je klasifikátor dle nejmenší vzdálenosti.[18] Tento klasifikátor vypočítává vzdálenost mezi neznámým vzorkem a reprezentantem každé ze tříd vzorů. Neznámý vzorek je klasifikován třídou s nejmenší vzdáleností od neznámého vzorku.

Výhodou těchto metod je možnost učinit problém rozpoznání objektu v digitálním obraze triviálním a výpočtově nenáročným, pokud lze zaručit jednoznačnost vzorků a neměnné prostředí, ve kterém jsou vzorky generovány.

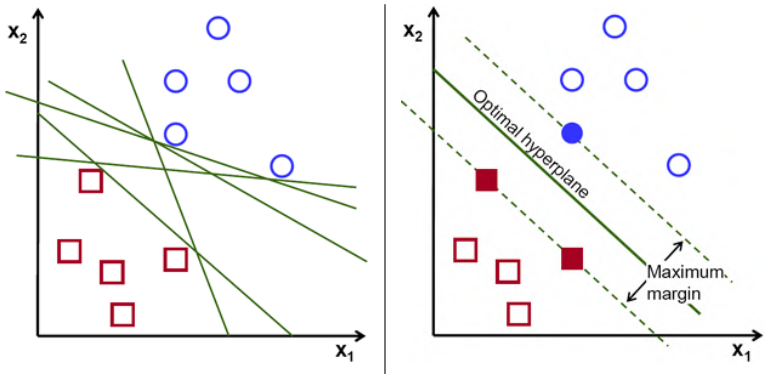
2.2.2 Klasifikace na základě optima statistické formulace

Princip tohoto druhu klasifikace spočívá ve vyřešení optimalizačního problému. Pro klasifikaci objektů v digitálním obraze v souvislosti s prostudovanou literaturou se nejčastěji používá následujících metod spadajících do oblasti strojového učení:

- Support vector machine (SVM) – Metoda podpůrných vektorů
- K-nearest neighbors (KNN) – Algoritmus k-nejbližších sousedů
- Hidden Markov model (HMM) – Skrytý Markovův model
- Decision Trees (DT) – Rozhodovací stromy

Metoda podpůrných vektorů

Cílem SVM algoritmu je nalezení optimální hyperroviny[5], která rozděluje n-dimenzionální stavový prostor, kde n je počet analyzovaných rysů. Tento algoritmus se snaží maximalizovat vzdálenost hyperroviny od prvků reprezentujících různé třídy za pomoci podpůrných vektorů, což jsou právě prvky stavového prostoru nejblíže situované k navrhované hyperrovině. Tyto blízké prvky, reprezentující hranici žádaného rozdělení tříd, ovlivňují pozici a orientaci hyperroviny.[7]

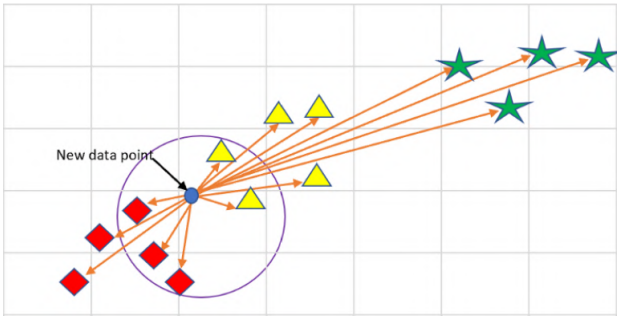


Obrázek 2.4: Vizualizace metody SVM, možné hyperroviny (vlevo) a nalezená optimální hyperrovina (vpravo)[7]

Pro data, která nelze lineárně separovat, je využito převodu analyzovaných rysů do prostoru vyšších dimenzí tak, aby se vstupní data stala lineárně separabilními.

[5] v p dimenzionálním prostoru se jedná o $p-1$ dimenzionální útvar, příkladem 2D prostor – 1D přímka

Algoritmus *k*-nejbližších sousedů
Jedná se o algoritmus strojového učení, který při klasifikaci obrazových dat pracuje s předpokladem, že data s podobnými rysy se budou shlukovat na obdobných pozicích. Obrazová data jsou reprezentovaná n-dimenzionálním vektorem rysů v n-dimenzionálním prostoru. Klasifikátor poté zvolí detekovanou třídu na základě k-nejbližších (vzdálenost určena pomocí zvolené metriky) tříd vzorků. Mezi často zvolené metriky vzdáleností jsou Euklidovská nebo Hammingova vzdálenost.



Obrázek 2.5: Vizualizace metody k-nejbližších sousedů, algoritmus by při této zvolené vzdálenosti predikoval zadaný bod jako červený kosočtverec[8]

Z principu použité metody klasifikace objektu je tento algoritmus velmi náchylný k zaujatosti pro více zastoupenou třídu. Je zapotřebí využívat datovou sadu vzorů, ve které jsou třídy stejnoměrně zastoupeny, popřípadě přidělovat jednotlivým zástupcům penalizační váhy vypočítané z velikosti rozdílu zastoupení vůči nejméně zastoupené třídě.

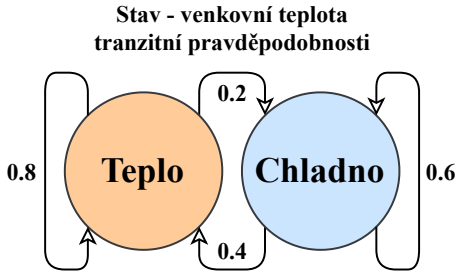
Skrytý Markovův model
Metoda využívá Markovův řetězec, což je stochastický model popisující sekvenci možných stavů, ve které pravděpodobnost každého stavu závisí pouze na předchozím stavu. Jednoduchý dvoustavový Markovův řetězec je uveden na obrázku 2.6, kde jsou zobrazeny tranzitní pravděpodobnosti. Tyto sekvence stavů lze sumarizovat stochastickou maticí:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,j} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ p_{i,1} & p_{i,2} & \cdots & p_{i,j} \end{bmatrix}$$

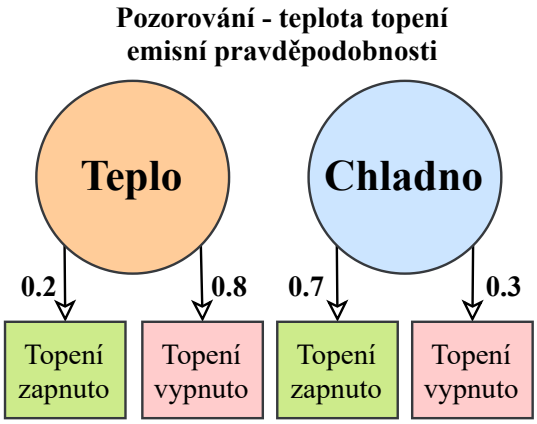
což je čtvercová nezáporná matice, jejíž řádkové součty jsou rovny jedné a jednotlivé členy $p_{i,j}$ definují pravděpodobnost přechodu ze stavu i do stavu j. U skrytého Markovova modelu nelze pozorovat stavy systému přímo, pouze jejich projevy na základě vnějšího pozorování.

Příkladem takového skrytého stavu může být venkovní teplota, přičemž se pozorovatel nachází v místnosti uvnitř budovy. Příkladem pozorování je teplota ústředního topení v téže místnosti. Nelze napřímo změřit teplotu venku (skrytý stav systému), ovšem předpokládá se (s danou pravděpodobností), že pokud je zapnuto ústřední topení (pozorování), venku je chladno. Tato emisní pravděpodobnost je znázorněna v obrázku 2.7.

Na základě po sobě jdoucích měření získáváme sekvenci pozorování, ze kterých lze určit nejpravděpodobnější sekvenci stavů systému. Tato sekvence stavů je vypočítána pomocí



Obrázek 2.6: Dvoustavový Markovův řetězec - příklad tranzitních pravděpodobností



Obrázek 2.7: Pozorování Markovova řetězce - příklad emisních pravděpodobností

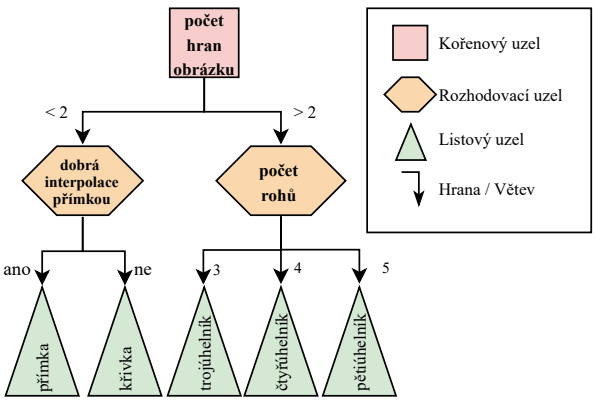
Viterbiho algoritmu.[9]

Pro klasifikaci obrazu pomocí HMM je možno využít následujícího postupu. Obraz je zpracován po předem definovaných částech extraktorem rysů, čímž je sestavena posloupnost rysů obrazu (sekvence pozorování). Z této sekvence pozorování lze odvodit nejpravděpodobnější sekvenci tříd (stavy systému). Tímto způsobem lze například detekovat obličeje při prohledávání obrazu shora dolů, a to vyhledáváním sekvence tříd, které reprezentují čelo, oči, nos a ústa.

Rozhodovací stromy

Rozhodovací stromy jsou typ modelu používaný pro úlohy klasifikace a regrese. Tyto stromy odpovídají na sekvenci dotazů na vstupní data, čímž postupují ke konečné predikci výsledku. Při použití rozhodovacích stromů pro účel klasifikace je klasifikátor uložen ve stromové formě, která je reprezentovaná následujícími objekty:

- Kořenový uzel - první dělící kritérium vstupu
- Rozhodovací uzel – definuje dělící kritérium
- Hrana / Větev – rozdělení na základě výsledku rozhodovacího uzlu
- Listový uzel – definuje výsledek klasifikace



Obrázek 2.8: Rozhodovací strom - příklad

Algoritmus tréninku klasifikátoru pomocí rozhodovacích stromů funguje na základě rekurzivní selekce mezí atributů (ASM – attribute selection measure). ASM je heuristikou selekce kritérií, která se pokouší dělit data nejlepším možným způsobem na jednotlivé podoblasti. Tyto kritéria rozhodují o výběru následující hrany, kterou se objekt v rozhodovacím stromu „vydá“. Nejpopulárnějšími selektory mezí atributů jsou Information Gain, Gain Ration a Gini.[10]

Nevýhodou rozhodovacích stromů je náchylnost na přetrénování u hlubších stromů. Při využití datové sady s nerovnoměrným zastoupením jednotlivých tříd rozhodovací stromy vykazují zaujatost vůči více zastoupené třídě. Tato metoda trpí velkým rozptylem klasifikace, kdy i malá změna atributů dat může vést k diametrálně odlišnému výsledku.

Pro detekci objektů v obraze je vhodnější zvolit pokročilejší metodu využívající rozhodovacích stromů, nazývanou náhodné lesy. Tato metoda redukuje rozptyl klasifikace tréninkem vícero rozhodovacích stromů na jednotlivých dílčích částech datové sady nebo použitím pouze části množiny atributů (kdy tato podmnožina je generována náhodně). Při nalezení významných atributů se poté sady rozhodovacích stromů stávají velmi korelujícími. Konečný výsledek klasifikace je modus (nejčastější zástupce) predikcí jednotlivých stromů.

2.2.3 Klasifikace na základě neuronových sítí

Neuronové sítě využívají velké množství nelineárních výpočetních jednotek (umělých neuronů), které jsou vzájemně propojeny.[16] Se zavedením algoritmu zpětného šíření chyby bylo možno začít efektivně trénovat tyto sítě.

Neuronové sítě se při tréninku automaticky učí reprezentaci dodaných vstupních objektů, přičemž každá vrstva sítě převádí tyto reprezentace do více abstraktní vrstvy. Tento proces, nazývaný hluboké učení, napodobuje chování lidského mozku při zpracování informací. Praktické použití neuronových sítí je často podmíněno existencí velké datové sady pro realizaci učení s učitelem.

Rozpoznávání objektů v digitálním obraze pomocí neuronových sítí je založeno na architekturách neuronových sítí typu konvoluční neuronové sítě. V dnešní době se tyto typy sítí používají například pro detekci úsměvů, rozpoznání obličejů, sledování dopravy, počítání objektů, autonomní řízení automobilů, čtení textu a v robotice.[6] Konvolučními neuronovými sítěmi se podrobněji zabývá samostatná kapitola 2.4.

2.3 Rešerše předchozích řešení

2.3.1 Autonomní operace nových výtahů pro navigaci robotů

V tomto článku z konference 2010 IEEE International Conference on Robotics and Automation (ICRA 2010) se studenti Stanfordské univerzity pod vedením profesora Andrew Y. Nga [12], průkopníka výuky hlubokého učení a umělé inteligence, zabývali metodou autonomní obsluhy nových výtahů pro navigaci robota ve výtahovém prostředí.

Článek se konkrétně zaměřoval na vývoj algoritmů pro identifikaci a lokalizaci ovládačů a rozpoznání jejich štítků z palubní kamery robotu, které bylo následováno vykonáním akce stisknutí ovládače robotickou paží. Tohoto chování bylo docíleno následujícím pseudoalgoritmem popsaným v tabulce 2.4.

Algoritmus byl natrénován rozmanitou datovou sadou 150 obrazů 61 odlišných výtahových panelů s rozdělením na 100 trénovacích a 50 evaluačních snímků. Každý krok algoritmu byl vyhodnocen samostatně. Datová sada využitá pro trénink neuronové sítě

1	**Extrakce ROI**
	Detekce ovládačů klouzavým oknem detektorem objektů byla založena na *metodě sdílených rysů podobných objektů*, která předložila prvotní odhady instancí objektů. Odhady byly poté filtrovány na základě dynamické hranice, určené z normalizace ovládačové kombinace.
2	**Uzpůsobení na mřížku**
	Bylo implementováno pomocí algoritmu *Expectation Maximalization*, který se snažil rozdělit panel na mřížky s uniformním rozdělením vzdálenosti tlačítek od sebe s penalizací počtu těchto mřížek.
3	**Využití neuronové sítě**
	Klasifikace štítků přilehlých u ovládačů na levé straně dle získané mřížky proběhla za použití neuronové sítě architektury typu **LeNet-5**.
4	**Prosazení soudržnosti**
	Pomocí *skrytého Markovova modelu* byla prosazena soudržnost mezi klasifikovanými štítky na základě pravděpodobnosti změn stavů. Toto způsobilo zvýšení robustnosti systému pochopením problému špatné detekce štítků.
5	**Interpretace sekvence štítků**
	Na základě bodů 3 a 4 byla pomocí *Viterbiho algoritmu* vybrána nejpravděpodobnější sekvence štítků ovládačů.
6	**Prosazení soudržnosti**
	Z předchozích kroků byly odvozeny pozice a štítky ovládačů volby podlaží, načež robotická ruka stiskla požadovaný ovládač volby podlaží.

Tabulka 2.4: Algoritmus identifikace a lokalizace ovládačů volby podlaží výtahové kabiny

z bodu 3 byla složena z binárních obrazů vytvořených pomocí superpixelového segmentačního algoritmu dle Felzenszwalba a Huttenlochera, kdy odkaz na tento algoritmus je uveden v této práci.

Otestování algoritmu proběhlo na robotické platformě univerzity s využitím kamery Canon SX100-IS ve třech dostupných výtazích. Robot byl postaven před výtahový panel a ze 14 pokusů splnil úlohu stisknutí požadovaného tlačítka podlaží ve 13 případech.

2.3.2 Autonomní rozpoznávání výtahových ovládačů založené na konvolučních neuronových sítích

Článek z konference IEEE International Conference on Robotics and Biomimetics (ROBIO 2017) doktorandů Čínské univerzity v Hong Kongu pod vedením profesora Max Q.-H. Menga [13] měl za účel představit metodu rozpoznání výtahových ovládačů, která by byla robustní, v reálném čase, a založena na použití konvolučních neuronových sítí.

Pro trénink neuronové sítě bylo pořízeno 800 snímků ovládacích panelů dvou výtahů, z kterých byla vytvořena sada 10 000 snímků ovládacích tlačítek výtahu za pomoci extrakčního algoritmu, jenž je přiblížen v následující tabulce 2.5.

Metoda byla evaluována po částech, kdy extrakce ROI[6] dosahovala přesnosti 98,3 % a klasifikace štítků přesnosti 93,8 %. Po úpravě zpracování výstupu neuronové sítě kroky 4 a 5 bylo dosaženo přesnosti 98,0 % na známých výtazích a 87,6 % na neznámých výtazích. Další část práce byla věnována robustnosti metody, a to vůči změně vzdálenosti a úhlu kamery vzhledem ke tlačítku, nejasnosti vstupu a změně jasu obrazu.

[6]Region of Interest - oblast zájmu

1	**Extrakce ROI**
	Byl zvolen předpoklad, že ovládače výtahu byly vždy *čtvercového nebo kruhového tvaru* s jasnou konturou. Vstupní obraz byl převeden do černobílé, zpracován mediánovým filtrem a předložen *canny detektoru hran s dynamickým prahem*. S předpokladem stejných tvarů a velikostí ovládačů byl použit *filtr pro odstranění prvků s abnormální velikostí či poměrem stran*. Pro odstranění falešných pozitiv a překrývajících se detekcí bylo využito jednoduchého *binárního klasifikátoru* a algoritmu *non-max suppression*.
2	**Klasifikátor**
	Klasifikátor byl založen na architektuře neuronové sítě *AlexNet,* která byla modifikovaná pro zvýšení kvality rozpoznávání. Inicializace neuronové sítě proběhla pomocí *předtrénovaného modelu* na datové sadě MNIST (obsahuje ručně psané číslice). Bylo využito moderních poznatků z hlubokého učení, konkrétně byla zavedena metoda regularizace *Dropout* pro prevenci přetrénování, změněna aktivační funkce sigmoid na ReLU a modifikována výstupní vrstva na max-pooling. Neuronová síť dosáhla *přesnosti 95 %* na předložené datové sadě.
3	**Zpracování dat klasifikátoru**
	Za pomoci neuronové sítě a vstupního obrazu zpracovaného v kroku 1 byly získány *míry jistoty klasifikace označení tlačítka*. Poté byla klasifikace s maximální mírou jistoty prohlášena za označení daného tlačítka. Všechny klasifikace byly uloženy pro možnost opravy označení, viz krok 4.
4	**Úprava dat neznámých štítků, detekce nesprávného označení**
	Byla vyvinuta úprava zpracování výstupu kroku 3 inspirovaná lidskou schopností odhadnout neznámý štítek mezi dvěma známými, pokud známe uspořádání ostatních štítků. Detekce byly zaprvé rozděleny do *sloupců a řádků* na základě jejich polohy, zadruhé byl posouzen *druh posloupnosti* tlačítek za pomoci porovnávací metriky a zatřetí bylo *detekováno přeskočení podlaží*. Na základě této znalosti pak byla definována funkce detekce špatně označeného tlačítka. Pro opravu špatně označeného ovládače pak byly použity data z bodu 3 a vhodné prvky odvozené z druhu posloupnosti ovládačů.

Tabulka 2.5: Algoritmus rozpoznávání výtahových ovládačů pomocí konvolučních neuronových sítí

2.3.3 Automatická lokalizace ovládacích prvků výtahu a kombinovaná detekce a sledování pro vícepatrovou navigaci

Nejrozsáhlejší článek rešerše předchozích řešení, publikován v časopise IEEE Access 2020, se věnoval kombinované lokalizaci a mapování ve vícepodlažních budovách a obsluze výtahu za pomoci počítačového vidění.[14]

Autoři poukázali na nevhodnost použití klasických metod (například detekce vzorů, shody vzorů), které jsou citlivé na změnu pozadí, a na vysokou výpočetní cenu konvolučních neuronových sítí, které nejsou vhodné pro systémy reálného času, kde způsobují selhání celého systému. Pro implementaci na mobilní robotické platformě bylo klíčovou myšlenkou použití sítě GAN[7], kdy toto konkrétní použití autoři přibližují ve třech krocích.

První krok detekce tlačítek byl realizován pomocí neuronové sítě typu SSD[8], které byly předloženy oblasti vstupního snímku zvolené detektorem hran. Nalezené oblasti jsou poté vizuálně sledovány za pomoci sledovacích algoritmů a neuronové sítě hlubokého učení.

[7]Generative Adversary Network - generativní soupeřící síť
[8]Single Shot Detector - jednofázový detektor (lokalizace + klasifikace objektu)

Druhým krokem se stala dvouúrovňová validace s binárním klasifikátorem a detektorem oblasti tlačítka typu SSD se zmenšeným vstupem a hloubkou. Poslední krok spojoval výstup sledování, binárního klasifikátoru a SSD sítě ve finální ROI. Tento krok byl realizován neuronovou sítí typu GAN, kdy výstup sledování byl považován za generátor, který byl společně s výsledkem binární klasifikace předán diskriminátoru s architekturou SSD.

Tímto rozdělením bylo docíleno vyšší výkonnosti metody díky rozdělení použitých dílčích metod na metody trénované offline (zajištění přesnosti) a na metody trénované online (zajištění robustnosti, používané opakovaně v reálném čase). Použité metody jednotlivých kroků jsou přiblíženy v následující tabulce 2.6.

1	**Extrakce ROI**
	Extrakce zajímavých oblastí byla realizována pomocí *canny detekce hran*. Tyto oblasti byly generovány za malých pohybů robota a vyfiltrovány implementovaným filtrem abnormálních velikostí. Bylo použito *binárního klasifikátoru*, respektive metody *non-maximum suppression* pro redukci nesprávně nalezených oblastí, respektive sjednocení překrývajících se kandidátů ovládače.
2	**Rozpoznání ovládače**
	Extrahované oblasti z kroku 1 byly expandované o 50 %. Ovládače byly detekovány použitím neuronové sítě architektury typu *SSD* s vnitřní sítí typu *MobileNet* (obrazový vstup 128 x 128 px).
3	**Vizuální sledování – inicializace**
	Poté byl obraz rozdělen na *popředí* (oblast detekce ovládače z bodu 2) a *pozadí* (oblast mimo detekci ovládače z bodu 2), kdy pro každou část byly uloženy jejich *detekované rysy* (především rohy) metodou *FAST* a *klíčové body* (invariantní vůči velikosti) metodou *BRISK*.
4	**Vizuální sledování – algoritmus**
	Sledování prvků vyžívalo metodu *Lucas-Kanade (LK)* optical flow. Byly sledovány trajektorie bodů, a to s kompenzací vůči změně natočení a změně velikosti sledovaných oblastí. Byl zaveden *detektor*, který porovnával posledních 20 snímků pomocí metody strojového učení *náhodného rozhodovacího lesa.* Z detektoru byla vybrána oblast s nejlepším skórem.
5	**Vizuální sledování – validace, aktualizace**
	Byl porovnán výstup LK a detektoru, a pokud překrytí jejich detekovaných oblasti bylo větší než 80 %, byl použit výsledek LK. V opačném případě byl spojen výsledek LK a detektoru. V této zvolené oblasti byla použita další, menší neuronová sítě typu SSD (se vstupem 32 x 32 px) pro detekci nově sledované instance ovldače, načež proběhla fúze výstupu sledování a aktualizace bodu 3 pro nově zvolenou sledovací oblast.
6	**Restrikce sítí GAN**
	Kroky 3-5 (vizuální sledování) sloužily jako generátor polohy snímků ovládačů, diskriminátor (posuzovač) těchto snímků byla neuronová síť typu SSD. Touto fází se metoda stávala invariantní vůči výjimečným situacím při sledování ovládačů a umožňuje zaměření algoritmu na specifický cíl nalezení ovládače.

Tabulka 2.6: Algoritmus lokalizace a detekce ovládačů výtahu se sledováním

Autoři práce vytvořili vlastní datovou sadu z videí 2 výtahů z různých vzdáleností a z vícero úhlů. Z těchto videí bylo anotováno 400 snímků ovládacích panelů kabiny a 200 snímků přivolávacích panelů. Data byla rozšířena změnami saturace a rozostřením snímků. Pro evaluaci bylo pořízeno dalších 200 snímků ze stejných dvou výtahů. Metoda byla

koncipována pro použití ve specifických výtazích, trénování sítí se nezabývalo konceptem přetrénování[9].

Dále autoři provedli evaluaci implementované metody s použitím vizuálního sledování a bez něj v porovnání s hlubokým učením (architektura AlexNet[16]), metodou strojového učení SVM[10] a rozpoznáváním textu.

Ve verzi metody se sledováním bylo dosaženo přesnosti 95 % s průměrnou rychlostí zpracování 40 FPS[11]. Ve verzi metody bez sledování bylo dosaženo přesnosti 90 % s průměrnou rychlostí zpracování 10 FPS. Hluboké učení s architekturou AlexNet dosahovalo přesnosti 85 % s průměrnou rychlostí zpracování 3 FPS. Metody SVM, respektive OCR vykazovaly nejhorších přesností, a to 15 %, respektive 14 %.

V poslední sekci uvedené práce byl proveden experiment pro výtahová tlačítka kruhového tvaru. Zde bylo použito modifikace metody za použití Houghovy transformace.

2.3.4 Rozpoznávání ovládacích prvků a jejich označení pro použití výtahu osobami se zrakovým postižením

Tento článek z publikace International Journal of Computer Applications (IJCA 2020) se zabýval možností interpretace ovládání výtahů pro zrakově postižené osoby.[15] Článek byl vypracován doktorandkou Jingya Liu pod vedením profesorky Yingli Tian na City College of New York.

Práce byla odlišná použitím dvou přístupů k metodě pro detekci tlačítek s rozpoznáním štítků podlaží. V prvním přístupu byla rozdělena detekce a rozpoznávání štítků na samostatné úlohy, které byly následně spojeny pomocí párování. Druhý přístup implementoval rozpoznání polohy tlačítek se štítky, které byly v dalším kroku společně klasifikovány. Algoritmy jsou přiblíženy v následující tabulce 2.7

<table>
<tr><th colspan="2">Algoritmus A
rozdělená detekce ovládačů a rozpoznání štítků</th><th>Algoritmus B
sloučená detekce a rozpoznání štítků</th></tr>
<tr><td colspan="3">Rozpoznání štítků</td></tr>
<tr><td colspan="3">Vstupní obraz byl zpracován pomocí ekvalizace histogramu a pro rozlišení oblasti s/bez ovládačů byl použit binární klasifikátor, kdy se v tomto případě jednalo o neuronovou síť architektury VGG16.</td></tr>
<tr><td>Detekce ovládačů</td><td>Rozpoznání štítků</td><td>Detekce ovládačů se štítky</td></tr>
<tr><td>Pro detekci tlačítka byla použita plně propojená neuronová síť (FCN).

Byla implementována korekce podmínkovými náhodnými poli, která byla použita jako rekurentní neuronová síť (CRFasRNN).</td><td>Detekce textové informace v obraze byla realizována pomocí neuronové sítě typu SSD.

Rozpoznání textu umožnila konvoluční rekurentní neuronová síť (CRNN).</td><td>Pro detekci oblastí s textem a ovládačem byla využita síť typu SSD.

Rozpoznání sémantické informace (čísla podlaží, symboly) zprostředkovány konvoluční rekurentní neuronovou sítí (CRNN).</td></tr>
<tr><td colspan="2">Párování detekce a rozpoznání</td><td rowspan="2"></td></tr>
<tr><td colspan="2">Ke každému tlačítku byl přiřazen štítek s nejmenší euklidovskou vzáleností v obraze.</td></tr>
</table>

Tabulka 2.7: Fáze rozpoznávání ovládacích prvků výtahu uvedené v článku, dva přístupy

[9] proces nadměrné specializace sítě na dodaná data
[10] Support Vector Machine - metoda podpůrných vektorů
[11] Frames per second - počet snímků za sekundu

Pro vyhodnocení metod byla vytvořena datová sada čítající 1 500 snímků, z čehož v 1 000 snímcích byly zastoupeny přivolávací a kabinové panely a v 500 snímcích byly zastoupeny snímky výtahového prostředí (negativní vzorky bez tlačítek). Datová sada byla náhodně rozdělena na 80 % pro trénink a evaluaci, zbylých 20 % bylo použito pro testování. Vyhodnocení metod je znázorněno v tabulce 2.8

Algoritmus	Krok	Přesnost
-	předselekce oblastí	92,32 %
Rozdělená detekce ovládačů a rozpoznání štítků	detekce tlačítek	74,20 %
	rozpoznání štítků	71,90 %
	párování detekce a rozpoznání	**73,00 %**
Sloučená detekce a rozpoznání štítků	-	**79,20 %**

Tabulka 2.8: Vyhodnocení metod detekce ovládačů pro $IoU_{0.5}$[12]

2.4 Konvoluční neuronové sítě

Tato podmnožina neuronových sítí je specializovaná pro práci s vícedimenzionálními signály (například obraz či zvuk), kde umožňuje stroji interpretovat tyto signály podobně jako lidský mozek. Ve své architektuře obsahují alespoň jednu konvoluční vrstvu.[18]

Konkrétně pro obrazová data je požadována extrakce dat porozumění obrazu, kterou lze využít například pro klasifikaci obrazů a videí. Použití konvolučních sítí v poslední době přibývá, mimo jiné díky dostupnosti výkonných počítačů, jenž umožnují hloubkové učení těchto sítí v dohledném čase.

Pro detekci objektů v digitálním obraze se poté využívá specializovaných architektur těchto sítí, jejichž recentní přehled je uveden ve zdroji [19].

2.4.1 Používané vrstvy sítí

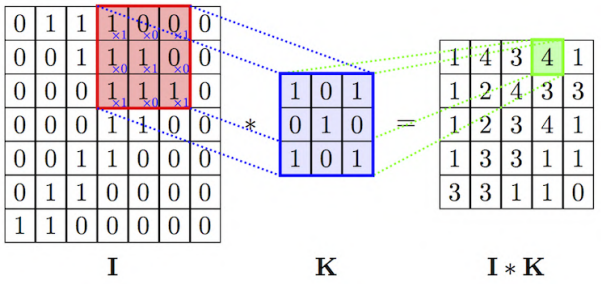
Konvoluční neuronová síť je složena z vrstev, kdy nejčastěji používané vrstvy konvolučních neuronových sítí pro typ úlohy rozpoznání objektů v obraze[20] jsou následující:

Konvoluční vrstva
Tato vrstva pracuje s konvolučními jádry (nazývané kernely/filtry). V případě práce s digitálním obrazem je použita diskrétní 2D konvoluce („okno“), která se „posouvá“ skrz obraz (2D vstupní data), jak je ukázáno na obrázku 2.9. Tato operace je popsána rovnicí 2.2. Tato vrstva je používána pro extrakci význačných rysů v dílčích částech obrazu.

$$g(x,y) = I * K(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} I(s,t)K(x-s,y-t) \tag{2.2}$$

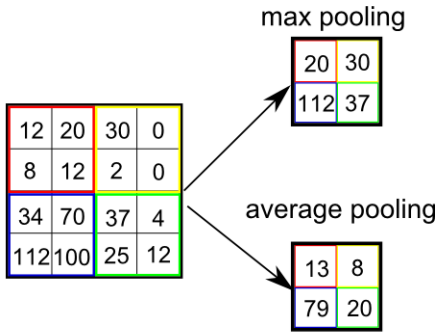
Pooling vrstva
Hlavní úlohou pooling vrstvy je zmenšení velikosti dat vstupní vrstvy, což přímo vede

[12]sjednocení průniku, podrobněji vysvětleno v kapitole 2.4.4



Obrázek 2.9: Vizualizace 2D diskrétní konvoluce[17]

ke snížení výpočtové náročnosti zpracování dat. Vrstva aplikuje podvzorkování pro každou hloubku vstupních dat samostatně a podobně jako u konvoluční vrstvy se „posouvá“ okno skrze obraz, načež je použita jedna z agregačních funkcí (maximum, resp. průměr, anglicky pojmenováno jako max-pooling, resp. average-pooling), jak je znázorněno v obrázku 2.10.



Obrázek 2.10: Vizualizace pooling vrstvy[17]

Použití vrstvy je vhodné pro extrakci dominantních rysů, které jsou invariantní vůči rotaci a změně pozice v obraze.

Normalizační vrstva

Normalizaci výstupu předchozí vrstvy pomocí odečtu průměru šarže a následným dělením směrodatnou odchylkou šarže zprostředkovává tato vrstva. Pro její implementaci přidává u každé použité vrstvy s normalizací dva trénovatelné parametry (parametr „průměr“ β ze šarže a parametr směrodatné výchylky γ). Výstup této funkce y_i lze tedy z normovaneho vstupu $\hat{x}_i$ vypočítat jako:

$$y_i = \gamma \hat{x}_i + \beta \tag{2.3}$$

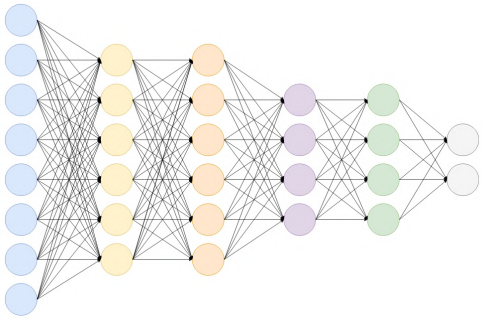
Plně propojená vrstva

U této vrstvy je každý neuron z vrstvy N spojen se všemi neurony vrstvy předchozí (vrstva M). Tuto vrstvu lze jednoduše definovat jako transformační funkci

$$y_j = \sum_{i}^{M} w_{ij} x_i \tag{2.4}$$

y_j značí hodnotu j-tého výstupního neuronu, x_i vstupní hodnotu neuronu z předchozí

vrstvy a w_{ij} váhu mezi těmito neurony. Vizualizace této vrstvy je zobrazena na následujícím obrázku 2.11.



Obrázek 2.11: Vizualizace plně propojených vrstev[17]

Softmax vrstva
Pro transformaci vstupu vrstvy do výsledného rozdělení pravděpodobnosti je využito vrstvy softmax. Využití nachází nejčastěji na výstupu neuronové sítě, kde převádí libovolný vícerozměrný vektor na vektor stejného rozměru s hodnotami v intervalu $(0, 1)$, jejichž součet nabývá hodnoty 1. Tento výstup lze při klasifikaci obrazů interpretovat jako rozdělení pravděpodobnosti výskytu jednotlivých unikátních rozlišovaných tříd.

2.4.2 Učení konvolučních neuronových sítí

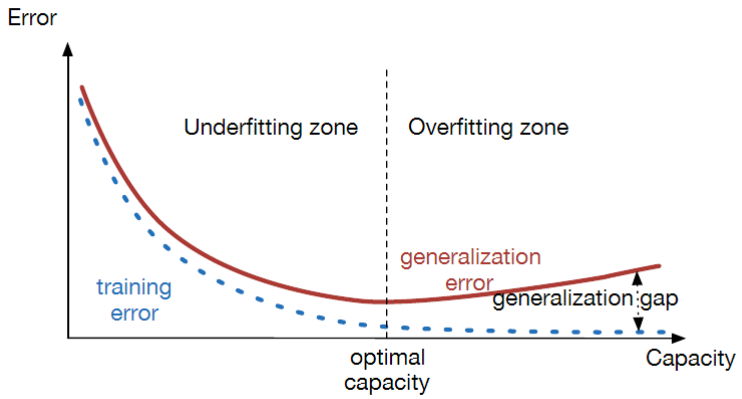
Trénink konvolučních neuronových sítí je metoda strojového učení typu učení s učitelem, jejíž snahou je nalezení takové funkce, která mapuje vstup na výstup na základě poskytnutých párů dat vstup-výstup.

Konkrétně u klasifikátorů digitálních obrazů se jedná o odvození této funkce z poskytnuté sady trénovacích vzorů. Každý trénovací vzor se skládá ze vstupního objektu (obrazová data) a žádaného výstupu (třída objektu). Tuto odvozenou funkci lze poté využít pro klasifikaci vzorů, u kterých není definován žádaný výstup.[11]

Cílem učení je tedy optimalizace vnitřních parametrů sítě takovým způsobem, aby byla minimalizována chyba predikce výstupu. Chyba výstupu je reprezentovaná účelovou funkcí (anglicky loss function), jejíž hodnota je v průběhu učení předmětem minimalizace. Učení probíhá v několika po sobě jdoucích fázích, typicky dopřednou propagací následovanou zpětnou propagací chyby a optimalizací použitých parametrů.[18]

Typicky jsou data rozdělena na trénovací a evaluační část, případně ještě doplněna částí testovací. Trénovací část je přímo použita na trénink neuronové sítě, načež je použita evaluační část dat pro vypočítání metrik umožňujících objektivně porovnat vytrénované modely. Doplňková testovací část obsahuje taková data, která nebyla spojena s použitou datovou sadou pro učení a slouží k objektivnímu zhodnocení způsobilosti modelu.

Díky dostupnosti výpočetního výkonu je běžné využívat neuronové sítě s desítkami milionů parametrů. S tímto počtem parametrů a dostatečně dlouhým časem učení se ovšem projevuje problém hlubokých sítí v přetrénování na dodanou datovou sadu. Místo, aby neuronová síť odvodila generalizovaný model, „naučí se zpaměti“ datovou sadu. Tento rozdíl generalizace je patrný při porovnání trénovací a testovací části dat, viz obrázek 2.12. Křížová validace, předběžné zastavení a metody regularizace jsou možnosti předcházení problému přetrénování sítě.[23]



Obrázek 2.12: Generalizační rozdíl, chyba trénovací datové sady (modrá křivka) vůči chybě testovací sady (červená křivka), rozdělení na nedotrénování (underfitting zone) a přetrénování (overfitting zone) sítě[25]

2.4.3 Hyperparametry sítí

Hyperparametry sítě jsou proměnné, které stanovují strukturu a vymezují postup učení neuronové sítě. Jsou nastaveny před trénováním neuronové sítě. [21]

Strukturu neuronové sítě lze přizpůsobit například následovně:

- Změnou počtu plně propojených vrstev a počtu jednotek v nich obsažených
- Zavedením regularizace typu Dropout[13]
- Volbou typu inicializace vah sítě (nejčastěji voleno uniformní rozdělení)
- Volbou aktivačních funkcí neuronů (nejběžněji použito ReLU a jeho alternativy)

a postup učení lze mimo jiné ovlivnit:

- Změnou typu a velikosti míry učení
- Volbou typu optimalizéru gradientního sestupu
- Změnou velikosti momentu učení
- Změnou počtu epoch sítě
- Volbou velikosti šarže
- Výběrem použitých regularizačních metod

K metodám úprav konvolučních neuronových sítí poskytuje přehled článek. [22]

Jednotlivým parametrům struktury a tréninku, které byly měněny při ladění neuronových sítí použitých v práci se podrobněji věnuje kapitola 5.3.

2.4.4 Evaluační metriky

Pro objektivní posouzení způsobilosti neuronové sítě řešit zadaný problém detekce tříd z digitálního obrazu byly pro soutěže zavedeny metriky, které byly následně přejaty jako standard pro porovnávání sítí.[24] Pro pochopení těchto metrik je nutno představit spjaté koncepty.

[13]předchází přetrénování sítě pomocí zavedení pravděpodobnosti vypnutí jednotlivých neuronů

Bounding box (ohraničující rámeček)
Ohraničující rámeček je vykreslený útvar ze dvou bodů obrazu se souřadnicemi (x, y), které ohraničují nejzazší body pozice detekovaného objektu. Tuto informaci poskytuje detektor objektů.

Confidence score (skóre jistoty)
Skóre jistoty je pravděpodobnost, že ohraničující rámeček obsahuje daný objekt. Tato informace je obvykle poskytována klasifikátorem a nabývá hodnot $(0, 1)$.

Intersection over Union – IoU (průnik ze sjednocení)
Tato metrika je definována jako plocha průniku predikovaného ohraničujícího rámečku B_p a skutečného ohraničujícího rámečku B_{gt}[14] dělena plochou sjednocení jejich ploch

$$IoU = \frac{A_p \cap A_{gt}}{A_p \cup A_{gt}} \tag{2.5}$$

kde A_p, respektive A_{gt} je plocha vymezená ohraničujícím rámečkem B_p, respektive B_{gt}. Skóre jistoty a IoU jsou poté použity jako kritéria pro určení, zda-li je detekce pravdivým pozitivem nebo falešným pozitivem.

Detekce je prohlášena jako *pravdivě pozitivní* (**TP** - z anglického true positive) právě tehdy splní-li následující podmínky:

- skóre jistoty > zvolený práh skóre jistoty
- predikovaná třída odpovídá skutečné třídě objektu
- predikovaný ohraničující rámeček má větší IoU se skutečným ohraničujícím rámečkem než zvolený práh (například hodnota 0,5)

Pokud nejsou splněny poslední dvě podmínky, jedná se o *falešné pozitivum* (**FP** - z anglického false positive). Pokud není splněna první podmínka pro detekci skutečného objektu, tato detekce je považována za *falešné negativum* (**FN** - z anglického false negative). Po nesplnění první podmínky při absenci objektu lze detekci prohlásit za *pravdivé negativum* (**TN** - z anglického true negative).

Precision (přesnost)
Přesnost je definována jako počet pravdivých pozitiv dělený sumou pravdivých a falešných pozitiv.

$$P = \frac{TP}{TP + FP} \tag{2.6}$$

Recall
Recall je definován jako počet pravdivých pozitiv dělený sumou pravdivých pozitiv a falešných negativ.

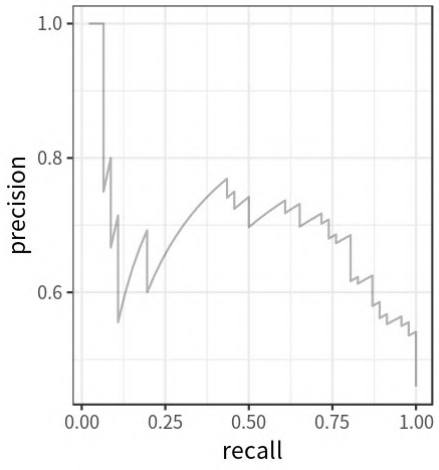
$$R = \frac{TP}{TP + FN} \tag{2.7}$$

Precision a recall se překládají do češtiny stejně jako přesnost, i když je jejich výpočet

[14] dolní index z anglického ground-truth

odlišný. V textu zmiňovaná přesnost odpovídá anglické precision.

Precision-recall
Vhodným měřítkem vhodnosti modelu na nestejnoměrně rozdělené datové sady je křivka *precision-recall*. Na ose x jsou vyneseny hodnoty precision a na ose y hodnoty recall pro *dané skóre jistoty*, viz 2.13. Lze pozorovat tendenci snižování precision se zvyšujícím se recall.



Obrázek 2.13: Křivka precision-recall[24]

Práce využívá metriky COCO[26] evalující sítě pomocí:

- average precision (AP)
- mean average precision (mAP)
- average recall (AR)
- mean average recall (mAR)

Average precision (AP)
Average precision využívá ke svému výpočtu interpolované křivky precision-recall

$$p_{interpolated}(r) = \max_{\acute{r}>=r} p(\acute{r}) \tag{2.8}$$

a samotný výpočet average precision je plocha pod touto interpolovanou křivkou

$$AP = \sum_{i=1}^{n-1} (r_{i+1} - r_i) p_{interpolated}(r_{i+1}) \tag{2.9}$$

Mean average precision (mAP)
Jakožto průměr AP skrze všechny detekované třídy K je mAP vypočítán:

$$mAP = \frac{\sum_{i=1}^{K} AP_i}{K} \tag{2.10}$$

Average recall (AR)
Average recall je vypočítán pomocí integrace jednotlivých distribucí *recall* napříč IoU práhy 0,5 až 1

$$AR = 2 \int_{0.5}^{1} recall(IoU)\ dIoU \tag{2.11}$$

Mean average recall (mAR)
Průměr AR skrze všechny detekované třídy K, mean average recall, je vypočítán:

$$mAR = \frac{\sum_{i=1}^{K} AR_i}{K} \tag{2.12}$$

Confusion matrix (matice záměn)
Vizualizována pomocí tabulky 2.14, tato metrika sumarizuje způsobilost klasifikačního algoritmu vůči jednotlivým klasifikovaným třídám. Poskytuje informace, které třídy jsou klasifikovány správně a při chybě klasifikace upozorňuje s jakými třídami došlo k záměně. Řádky matice reprezentují predikovanou třídu a sloupce skutečné třídy. Hodnoty na diagonále tedy reprezentují počet správných klasifikací daných tříd.

		True/Actual		
		Cat	Fish	Hen
Predicted	Cat	4	6	3
	Fish	1	2	0
	Hen	1	2	6

Obrázek 2.14: Příklad matice záměn[24]

Například hodnota 2 ve třetím řádku a druhém sloupci říká, že dvě instance třídy Fish byly predikovány jako instance třídy Hen. Využívá se i alternativa této matice používající relativních (namísto absolutních) hodnot.

2.4.5 Aplikační rámce strojového učení neuronových sítí

Caffe
Tento starší aplikační rámec byl vyvinutý vědci University of California Berkeley v roce 2013 a je používán především pro svou výtečnou výpočetní rychlost a zaměření na počítačové vidění. V dnešní době je však dokumentace zastaralá a práce s aplikačním rámcem není vhodná pro osoby nezkušené s hlubokým učením. Díky své výpočetní rychlosti je velmi vhodný pro nasazení do průmyslu. Podporuje programovací jazyky Python a `C++`.[27]

PyTorch
Tento aplikační rámec pro hluboké učení byl vytvořen skupinou Facebook AI Research

v roce 2016 a podporuje programovací jazyky Python, Java a `C++`, je založen na výpočetním rozhraní Torch. Mezi jeho přednosti vůči ostatním rámcům je flexibilita tvoření vlastních neuronových sítí, lepší podpora aplikačního rozhraní pro jazyky Java a `C++`, a velký ekosystém dalších aplikačních rámců pro nativní podporu dalších funkcí.[27]

Tensorflow 2, Keras
Vyvinut týmem Google Brain Team v roce 2015 je TensorFlow nejpopulárnějším aplikačním rámcem dnešní doby.[27] Z důvodu nesnadné práce s balíčkem TensorFlow jeden z výzkumníků vytvořil aplikační rámec Keras, který poskytuje více intuitivní a vysokoúrovňové rozhraní pro práci s TensorFlow. Mezi hlavní přednosti celého rámce patří snadnost použití Keras, flexibilita nasazování modelů, práce s modely neuronových sítí, rozsáhlá komunita uživatelů a velký výběr předtrénovaných modelů. S příchodem TensorFlow 2 v roce 2020 je Keras integrován do aplikačního rámce a je jeho plnohodnotnou součástí.

3 Upřesnění cílů

Třetí kapitola se zabývá upřesněním cílů na základě provedené rešerše v předchozí kapitole, kde byly přiblíženy základní metody zpracování obrazů, prozkoumány druhy metod rozpoznávání objektů v obrazu, práce s konvolučními neuronovými sítěmi a rešerše přechozích řešení v podobných aplikacích.

3.1 Vymezení úlohy

Úloha detekce ovládacích prvků výtahu je vymezena na základě získaných vstupních obrazových dat výtahového prostředí, doplněná restrikcemi uvedenými v normě ČSN ISO 4190-5:1992. Na základě těchto informací byly definovány třídy rozeznávaných prvků osobních výtahů.

3.1.1 Vstupní data

V průběhu sběru dat byla zjištěna velká variace v osvětlení, velikosti, tvaru i uspořádání ovládacích prvků výtahů. Umístění jednotlivých kombinací ovládačů nebylo konstantní, a to vzáleností od podlahy i polohou v rámci výtahového prostředí. Osvětlení v samotné kabině výtahu bylo proměnné, s různými typy použitých osvětlení. Ovládače byly nejčastěji kruhovitého či obdélníkového tvaru s mechanickou (akční) částí, byly zaznamenány i kapacitní dotykové či digitální displeje, které však byly v řešení ignorovány.

3.1.2 Vymezení normy ČSN ISO 41905 vůči ovládacím prvkům výtahů

V rámci dalšího upřesnění úlohy bylo přistoupeno k prostudování normy ČSN ISO 4190-5 Zřizování výtahů - Část 5: Ovládací prvky, signalizace a další příslušenství.[28] Tato norma existuje ve dvou dostupných verzích, a to verze vydaná v květnu 1992, která nahrazuje první vydání ČSN ISO 4190-5:1982, přičemž obsahuje její znění, a náhradu vydanou v lednu 2013, která je českou verzí mezinárodní normy ISO 4190-5:2006.

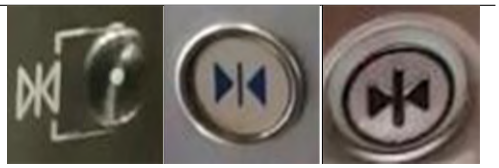
Vlastnost	**Požadavky**
Aktivní část ovládače	identifikovatelná vizuálně i dotykem plocha minimálně 280 mm^2, doporučeno minimálně 490 mm^2 rozměry umožnují vepsání kružnice o průměru alespoň 19 mm
Síla záznamu požadavku	ovládací síla na aktivní část ovládače 1-5N
Rozestupy mezi ovládači	vzdálenost mezi aktivními částmi dvou ovládačů alespoň 10 mm
Poloha ovládače	od podlahy minimálně 890 mm a maximálne 1 800 mm
Značka	provedena jako reliéf, kontrastní vůči svému pozadí velikost značky minimálně 15 mm a maximálně 40 mm poloha značky na aktivní části ovladače, popřípadě vlevo od aktivní části ve vzdálenosti mezi 10 mm a 15 mm

Tabulka 3.1: Požadavky na ovládače osobních výtahů

Norma stanovuje ovládací zařízení, ovládače a ukazatele, které musí být použity při konstrukci výtahu. Z důvodu velkého zastoupení získaných dat výtahů zřízených před rokem 2013 byl kladen důraz na verzi vydanou v květnu 1992.

Stěžejní informací byly požadavky na ovládače, jejichž relevantní část pro možnost použití mobilní robotickou platformou byla shrnuta v tabulce 3.1.

3.1.3 Definice tříd ovládacích prvků

Název prvku	**Definice**	**Ukázky prvku**
Ovládač volby podlaží	Typ: Ovládač splňující požadavky pro ovládače, viz tab. 3.1	
Ovládač ALARM	Reprezentace: značka obrysu zvonku / jednoduchý zvonek Poloha: od 900 mm do 1200 mm nad úrovní podlahy Typ: Ovládač splňující požadavky pro ovládač, viz tab. 3.1	
Ovládač POMOC	Reprezentace: značka stylizovaného sluchátka Poloha: od 900 *mm* do 1200 *mm* nad úrovní podlahy Typ: Ovládač splňující požadavky pro ovládač, viz tab. 3.1	
Ukazatel polohy kabiny	Reprezentace: číslo minimálně 13 *mm* veliké, doporučeno minimálně 30 *mm*, kontrastní barvy vůči okolnímu pozadí Poloha: v kabině - nad ovládačovou kombinací umístěn 1600 *mm* až 1800 *mm* nad podlahou	
Ovládačová kombinace stanice	Reprezentace směru: stylizované šipky vertikální Obsah: Záznamový/é ovládač/e dle typu řízení výtahu, splňující požadavky pro ovládače dle tab 3.1, s možným doplněním o ukazatel aktuálního podlaží kabiny	
Ovládač otevírání dveří	Reprezentace: stylizované šipky horizontální od sebe Typ: Ovládač splňující požadavky pro ovládač, viz tab. 3.1	
Ovládač zavírání dveří	Reprezentace: stylizované šipky horizontální k sobě Typ: Ovládač splňující požadavky pro ovládač, viz tab. 3.1	

Tabulka 3.2: Definice prvků osobního výtahu s ukázkami

Na základě analýzy vstupních obrazových dat a vymezení normy ČSN ISO 4190-5 byly

definovány třídy prvků, které byly běžně používány u osobních výtahů, představené v tabulce 3.2. Z důvodu zajištění robustnosti algoritmu bylo přihlíženo ke staršímu vydání normy z roku 1992. Definice tříd byly doplněny o obrázky jejich zástupců, které byly získané při sběru dat.

3.2 Definice úlohy

Jestliže je známo prostředí, ze kterého byl obrazový záznam pořízen, je implementovaná metoda schopna úspěšně detekovat ovládací prvky výtahu v obraze, přičemž rozlišuje jednotlivé třídy ovládacích prvků a nalézá akční části u ovládačů volby podlaží. Pořízený obraz je obdobné kvality jako u běžně používaných palubních kamer mobilních robotů schopných autonomní navigace.

Jako úspěšná detekce ovládacích prvků se považovala taková detekce, která lokalizuje správně oblast prvků alespoň 85 % všech instancí třídy ovládačů volby podlaží a alespoň 75 % všech instancí ostatních tříd při IoU = 0,5.

Pro specifické účely možnosti použití na mobilním robotu (realize schopnosti autonomního pohybu mezi patry) byl tedy kladen zvláštní důraz na vysoké procento lokalizovaných instancí třídy ovládačů volby stanice vůči ostatním třídám.

4 Datová sada

Tato kapitola se zabývá použitou datovou sadou v diplomové práci. Jedním z cílů práce bylo vytvoření dostatečně mohutné množiny obrazů ovládacích prvků výtahu. Pro zajištění robustnosti a obecnosti výsledného řešení pomocí strojového učení bylo zapotřebí získat obrazová data z kabin vícero vizuálně odlišných výtahových zařízení.

4.1 Specifikace dat

Z důvodu nedostupnosti dostatečně mohutné veřejné datové sady kabin výtahových zařízení bylo nutno vytvořit tuto sadu sběrem dat. Obrazová data výtahových zařízení pocházejí z veřejných i soukromých budov města Brna a Ostravy. Největší část dat (11 výtahových zařízení) pochází z oblasti VUT kampusu v Brně, Králově Poli. Datová sada byla složena z řady videozáznamů obsluhy výtahových zařízení, konkrétně pro každé výtahové zařízení byl vždy pořízen alespoň:

- jeden záznam při přivolání výtahu vnější ovládačovou kombinací stanice s detailem na panel ovládačové kombinace
- jeden záznam ovládačové kombinace kabiny výběru podlaží z vícero úhlů s proměnnými světelnými podmínkami
- jeden záznam na displej zobrazující aktuální podlaží výtahu, a to v průběhu jízdy z nejnižšího do nejvyššího patra a zpět (pokud výtah disponoval displejem aktuálního podlaží)

Dále datová sada disponuje záznamy vnějších informačních displejů aktuálního patra výtahů, dalšími záznamy ovládačových kombinací stanic, které se mohou lišit v závislosti na podlaží v budově, a záznamy vnitřních kabin při vystupování a nastupování do výtahové kabiny.

4.1.1 Sběr dat

Samotný sběr dat byl proveden pomocí mobilních telefonů s moderními senzory pořizujícími záznam v rozlišení alespoň 1080×1920 pixelů.

Vlastnost	**Hodnota**
Délka záběrů	56 minut 30 sekund
Počet souborů	98
Počet výtahových zařízení	34
Velikost obrazových dat	5682 MB

Tabulka 4.1: Statistiky sběru obrazových dat výtahových zařízení

U implementace automatické detekce byl poté zaveden předpoklad stejné nebo lehce

horší kvality vstupních snímků (alespoň 720×1280 pixelů). Statistiky pořízených záznamů jsou uvedeny v tabulce 4.1.

4.1.2 Vzorový případ

Tato podkapitola se zabývá nastíněním metodiky a klíčových okamžiků sběru obrazových dat ovládacích prvků na vzorovém případě výtahového zařízení, konkrétně výtahu v budově nákupního centra Vaňkovka v Brně. Tento záznam byl pořízen 1.10.2020 ve dvou videích.

V následující koláži digitálních obrazů 4.1 byly ze záznamu zachyceny klíčové záběry přivolávání výtahu ovládačovou kombinací stanice. V první části byl zahrnut celkový pohled na vnější část výtahových dveří společně s polohou ovládačové kombinace. Druhá část záběru byla zaměřena na zachycení detailů ovládačové kombinace z vícero perspektiv a vzdáleností. Třetí část záběru se zabývala odstoupením, otevřením výtahových dveří a nastoupením do výtahové kabiny.

Obrázek 4.1: Koláž digitálních obrazů záznamu přivolání výtahu ovládačovou kombinací stanice

Obrázek 4.2: Koláž digitálních obrazů záznamu ovládačové kombinace volby podlaží kabiny výtahu

Druhý pořízený záznam tohoto výtahového zařízení byl zaměřen na kombinaci ovládačů volby podlaží. V koláži snímků 4.2 bylo postupně ukázáno přiblížení k ovládačové

kombinaci volby podlaží, záběr na kombinaci z vícero úhlů a změna osvětlení při zavírání a otevírání dveří. Pro následnou anotaci dat bylo velmi vhodné pořizovat takové video, ve kterém se počet ovládacích prvků v záběru neměnil.

Poslední požadovaný záznam výtahového zařízení byl věnován displeji aktuálního podlaží, který byl zaznamenán při jízdě výtahem z nejnižšího podlaží do nejvyššího a zpět. Klíčové záběry tohoto výtahového prvku sestávaly z vícero úhlů záběru kamery, změny osvětlení při provozu a možnosti odrazů výtahového prostředí při pohledu na displej. V tomto případě byl pořízen záznam sedmisegmentového displeje podlaží s ukazatelem směru jízdy. V následující koláži snímků 4.3 byl zobrazen průběh záznamu.

Obrázek 4.3: Koláž digitálních obrazů záznamu displeje aktuálního podlaží při provozu výtahu

4.2 Typy záběrů prvků výtahů

V průběhu sběru dat bylo pořízeno mnoho záběrů s irelevantními daty pro použitou metodu detekce ovládacích prvků výtahu. Mezi tyto záběry patřily dlouhé záběry na prázdné stěny kabiny, pohledy do zrcadel, prudké otáčení v kabině či nástupy a výstupy z kabiny, kdy ani jeden z těchto typů záběrů neobsahuje obrazová data ovládacích prvků. Pro vytvoření dostatečně velké množiny obrazů ovládacích prvků výtahů bylo možno zvažovat tři relevantní typy pořízených záběrů prvků výtahy, a to:

- záběry ovládačové kombinace stanice
- záběry ovládačové kombinace volby podlaží
- záběry dipleje aktuálního podlaží výtahu

4.2.1 Ovládačová kombinace stanice

Ovládačovou kombinace stanice bylo možno definovat jako obdélníkový plochý panel s tlačítky pro přivolání výtahu, s případnou možností zvolení požadovaného směru jízdy. Některé přivolávací panely zahrnovaly i displej aktuálního podlaží výtahu, případně zdířku přivolání servisním klíčem.

Obrázek 4.4: Ukázky různých typů ovládačových kombinací stanice výtahových zařízení

Tyto panely se nacházely v blízkosti obsluhovaného výtahového zařízení, a to konkrétně buď na rámech výtahových dveří nebo zabudované ve zdech blízko výtahového zařízení (nejčastěji na pravé straně při pohledu na výtahové dveře zvenčí). Při výskytu vícero výtahů v jedné stěně byla možnost ovládat každý jednotlivý výtah vlastním panelem, nebo byly výtahy spjaty s jedním panelem umístěným v prostoru mezi nimi. V obrázku 4.4 byly zobrazeny různé typy přivolávacích panelů výtahových zařízení.

Na základě použitých elementárních prvků byly v datové sadě obsaženy ovládačové kombinace stanic:

- s ovládačem přivolání výtahu bez možnosti zvolení směru
- s ovládačem přivolání výtahu s volbou směru nahoru
- s ovládačem přivolání výtahu s volbou směru dolů
- se zdířkou na servisní klíč
- s displejem aktuálního podlaží výtahu

a jejich kombinacemi.

4.2.2 Dispej aktuálního podlaží

Prvek typu displej aktuálního podlaží bylo možno definovat jako sedmisegmentový, devítisegmentový, popřípadě digitální displej obdélníkového tvaru, vsazený buď do panelu výtahové kabiny, do rámu dveří výtahové kabiny, anebo do ovládačové kombinace stanice.

Tento prvek zobrazoval aktuální podlaží kabiny výtahu v budově pomocí číslice, případně byl vybaven i možností zobrazení doplňujících informací, mezi něž patřilo například zobrazení aktuální směru pohybu kabiny, infografika budovy, či symboly informačního a varovného charakteru.

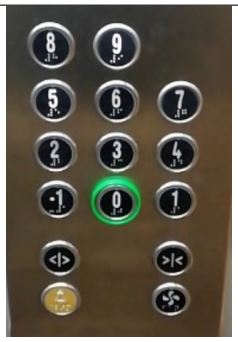
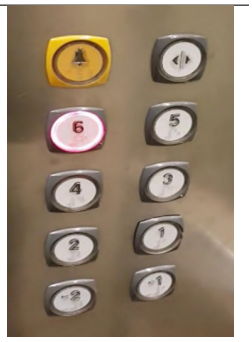
V obrázku 4.5 jsou přiblíženy některé typy displejů zahrnuté v datové sadě.

Obrázek 4.5: Ukázka typů displejů aktuálního podlaží výtahových zařízení

4.2.3 Ovládačová kombinace volby podlaží

Nejdůležitějším ovládacím prvkem pro možnost autonomního ovládání výtahu byla ovládačová kombinace volby podlaží kabiny výtahu. Tu bylo možno popsat jako uniformní panel kabiny obsahující ovládače výběru cílového podlaží a ovládače nouze. Některé panely byly vybaveny rovněž tlačítky otevírání a zavírání výtahových dveří, tlačítkem volání správce výtahu, otvory pro reproduktory, zdířkou pro servisní klíč, případě i čtečkou karet či čipů.

V získaných datech se objevily odlišné typy ovládačové kombinace volby podlaží, znázorněné v tabulce 4.2.

Typ odlišnosti	**Možnosti**	**Ukázky**
Rozložení ovládačů	alternativní, s vynechanou číslicí	 
	sloupcové, blokové	 
Způsob řazení ovládačů	po sloupcích, po řádcích	 
Směr inkrementace	zleva doprava, zprava doleva	 

Tabulka 4.2: Typy ovládačových kombinací volby podlaží výtahových zařízení

4.3 Vytvoření tréninkové sady

Možnost použití získaných obrazových dat pro detekci pomocí metod strojového učení byla podmíněna vytvořením anotačních dat. Proces anotování spočíval v přidělování označení jednotlivým částem obrazu, díky čemuž byl model schopen extrahovat informace o poloze a druhu objektů, které se nalézají se v obraze.

4.3.1 Anotace dat

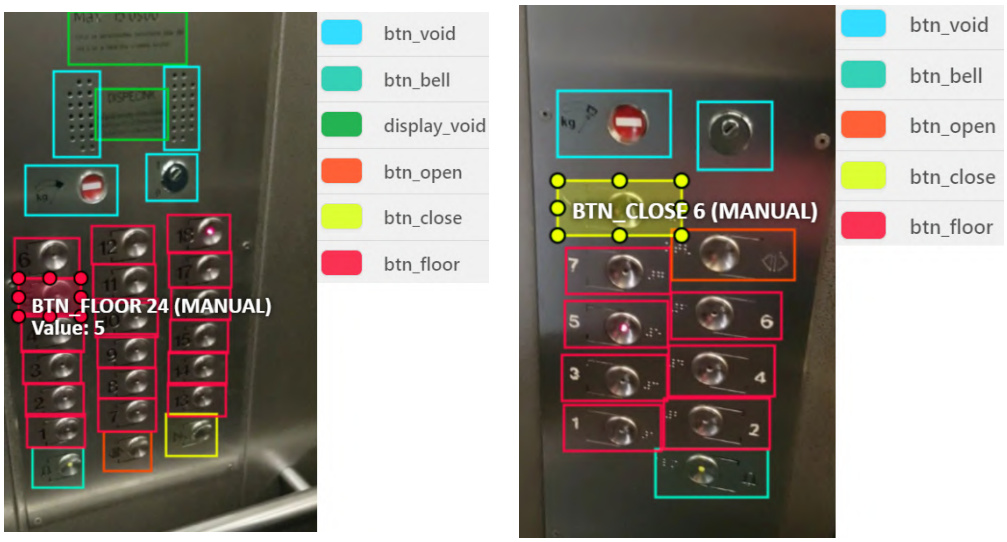
Samotná anotace dat byla provedena v otevřeném softwaru CVAT[30] (Computer Vision Annotation Tool – anotační nástroj počítačového vidění), který byl spouštěn pomocí lokální virtualizace skrze software Docker.

Každému ovládacímu prvku v záznamu byla přidělena třída a ohraničující rámeček, kde třída označovala typ ohraničeného objektu a ohraničující rámeček jednoznačně definoval jeho polohu ve snímku.

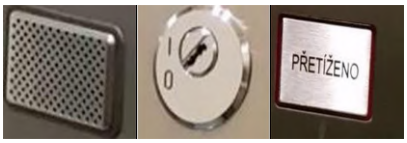
Pro zjednodušení anotačního procesu videí byl webový klient CVAT vybaven možností interpolace mezi manuálně zadanými klíčovými označeními, což byla preferovaná metoda značení pro videozáznam. U videozáznamů byl anotován vždy každý druhý snímek záznamu. Ohraničující rámeček byl tedy přesouván v průběhu videa tak, aby stejný objekt v obraze neustále ležel uvnitř tohoto rámečku.

Tyto označené objekty se poté staly oblastmi zájmu pro strojové učení. Pro jednoznačnou identifikaci typů ovládacích prvků byla použita označení tříd ovládacích prvků, jenž jsou shrnuta v tabulce 4.3. Byly zavedeny negativní třídy označené příponou *void* pro rozlišení mezi validními a invalidními typy prvků, jež byly vizuálně a poměrově velmi podobné. Pomocí webového klienta CVAT byly získané záběry rozděleny na části, které byly postupně označovány. Příklad takového označování je nastíněn v obrázku 4.6.

Obrazy prvků *display_valid* byly použity i v práci Bc. Martina Havelky[31] zabývající se detekcí aktuálního podlaží výtahu.



Obrázek 4.6: Vzorové příklady anotace obrazů v prostředí webového klienta CVAT

Název třídy	**DT**	**Popis**	**Ukázka třídy z anotace**
btn_floor	P	Objekt ovládače výběru cílového podlaží jízdy výtahem	
btn_bell	P	Objekt ovládače ALARM označený značkou zvonku	
btn_open	P	Objekt ovládače otevírání dveří výtahové kabiny	
btn_close	P	Objekt ovládače zavírání dveří výtahové kabiny	
btn_void	N	Objekt velmi se podobající ovládači, bez funkce (reproduktory, zdířky, tlačítka bez funkce, záslepky)	
display_valid	P	Objekt displeje aktuálního podlaží kabiny	
display_void	N	Objekt velmi se podobající displeji aktuálního podlaží, bez této funkce	
panel_call_valid	P	Objekt ovládačové kombinace stanice (přivolávací panel výtahu)	
panel_call_void	N	Objekt podobající se ovládačové kombinaci ve stanici, bez funkce přivolání výtahu	

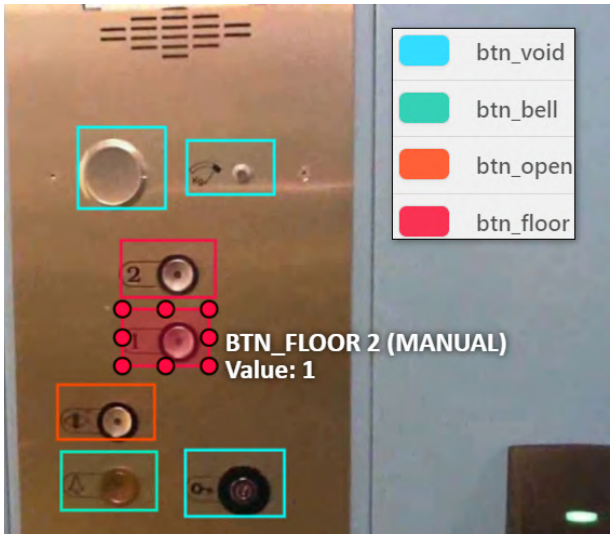
Tabulka 4.3: Označení tříd pro anotaci, DT - detekční typ: P (pozitivní třída), N (negativní třída)

4.3.2 Modifikace tréninkové sady na základě výsledků detekce

Na základě výsledků detekce bylo zaprvé rozhodnuto o eliminaci třídy *ovládač POMOC* (pro popis viz tabulka 3.2) z důvodu velmi malého zastoupení v datové sadě.

Zadruhé bylo nutno opravit značení snímků tam, kde nebyly označeny všechny prvky, které se ve snímku vyskytovaly. Tyto nesrovnalosti mátly učení neuronové sítě a nedovolovaly dosáhnout dostatečné přesnosti detekce, jelikož části obrazu, ve kterých nejsou označeny prvky, algoritmus učení považuje za pozadí, vůči kterému chce označené prvky rozlišit.

Zatřetí, po neúspěšných pokusech rozpoznat číslice u ovládačů pomocí dostupných metod rozpoznávání znaků byla přidána všem objektům třídy *btn_floor* (pro popis viz tabulka 4.3) proměnná typu celočíselné hodnoty zastupující právě číslici podlaží u tohoto ovládače, která byla využita pro trénink klasifikátoru podlaží z objektů třídy *btn_floor*. Ukázka přiřazení hodnoty číslici ovládače volby podlaží je uvedena v obrázku 4.7.



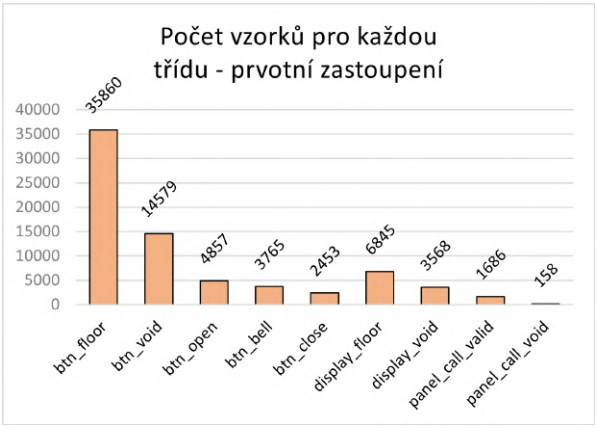
Obrázek 4.7: Přiřazení hodnoty 1 číslici ovládače volby podlaží při anotaci dat

4.3.3 Zastoupení objektů v datové sadě

Datová sada byla v průběhu diplomové práce rozšiřována dalšími daty, první iterace zastoupení objektů v datové sadě byla znázorněna v tabulce 4.8. Na této sadě byly provedeny prvotní experimenty trénování konvolučních neuronových sítí.

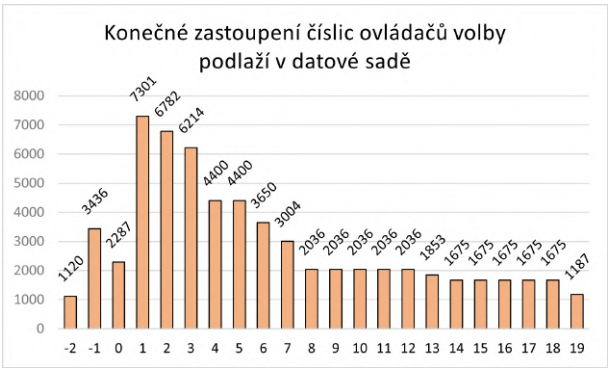
Datová sada disponovala 32 916 anotovaných obrázků s celkem 131 803 označenými instancemi tříd uvedených v tabulce 4.3. Konečné zastoupení jednotlivých tříd, které bylo použito pro trénink detektoru ovládacích prvků, respektive klasifikátoru číslic podlaží, bylo následně zobrazeno v obrázcích 4.9, respektive 4.10.

Jak je uvedeno v obrázku 4.9, zastoupení třídy *btn_floor* bylo několikanásobně větší než u ostatních tříd. Pro menší možnost zaujatosti neuronových sítí vůči jednotlivým třídám je vhodné předkládat pro trénink datovou sadu s uniformní distribucí tříd, případně kompenzovat tuto vlastnost vhodným zvolením architektury detektoru.



Obrázek 4.8: Prvotní zastoupení objektů v datové sadě

Obrázek 4.9: Konečné zastoupení objektů v datové sadě



Obrázek 4.10: Konečné zastoupení číslic ovládačů volby podlaží v datové sadě

4.3.4 Standardizovaný export datové sady

Pro využití výsledné datové sady byl proveden export do Pascal VOC[29] formátu, což byl standardizovaný soubor složek, obsahující anotace k jednotlivým obrázkům ve formátu XML, obrázky ve formátu JPEG a textový seznam odkazující na jednotlivé soubory anotací a obrázků.

5 Detekce ovládacích prvků výtahů

Pátá kapitola se zabývá principem a popisem použité technologie pro realizaci detekce ovládacích prvků pomocí konvolučních neuronových sítí. První část této kapitoly se zabývá volbou a popisem použitých architektur neuronových sítí za účelem detekce prvků výtahu a klasifikace číslic ovládačů volby podlaží. Dále kapitola popisuje práci s aplikační nástavbou Object Detection API balíčku Tensorflow 2 a poslední část kapitoly je věnována realizaci tréninku a evaluaci použitých neuronových sítí za pomoci balíčku Object Detection API (detektor) a Keras (klasifikátor).

5.1 Použité architektury

Detektor byl zvolen z TensorFlow Object Detection API 2 (dále TFOD2)[32], nástavby TensorFlow 2, která zprostředkovává možnosti tréninku detektorů objektů v obraze s již předtrénovanými modely na veřejně dostupných datových sadách.

TFOD2 disponovala řadou předtrénovaných modelů na veřejně dostupné datové sadě COCO.[26] Volba použité architektury sítě byla provedena za následujících požadavků:

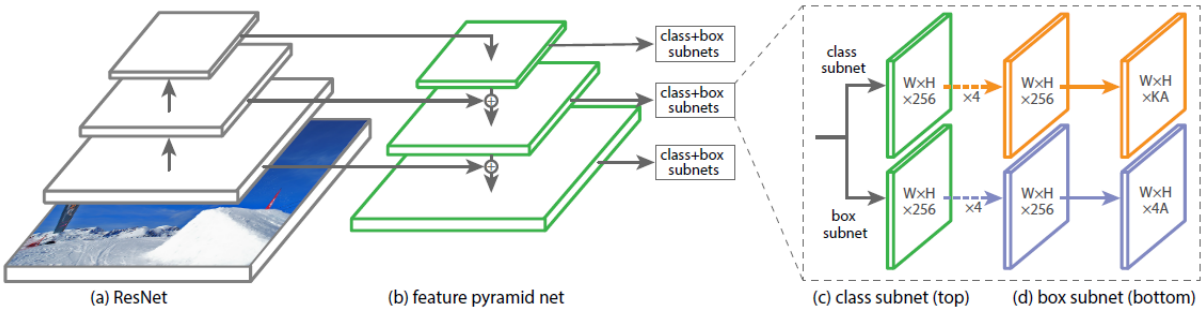
- Detektor byl použit na **neproměnné prostředí** a není nutný opakovaný dopředný průchod sítí v reálném čase
- Z důvodu neproměnného prostředí bylo vhodnější **zaměření na přesnost** detektoru (na úkor jeho rychlosti)
- Možnost chodu na platformě mobilního robotu omezuje velikost použité a aktuálně dostupné paměti pro realizaci detektoru

Článek Speed/accuracy trade-offs for modern convolutional object detectors[33] se zabýval právě možností volby moderní architektury detektoru, při srovnání jejich rychlosti, přesnosti a požadavků na paměť. Na základě prostudování tohoto článku, experimentací s architekturami a možnostmi výběru v předtrénovaných modelech TFOD2 a jejich skóre na datasetu COCO byla vybrána architektura **SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)**.

5.1.1 Detektor prvků výtahu

Účelem tohoto detektoru architektury SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50) bylo nalézt instance tříd definovaných v kapitole 4.3.2 v dodaném obrazu. Detektor využíval extraktor rysů typu SSD. Vizualizace architektury sítě je uvedena v obrázku 5.1.

Tato jednoúrovňová architektura přijímá obrazový vstup 1024x1024 pixelů a využívá síť pyramidy rysů (FPN)[35], která vylepšuje standardní konvoluční neuronové sítě o vícestupňovou pyramidu rysů extrahovanou z jednoho obrazu, kdy každý stupeň pyramidy může být použit pro detekci objektů jiné velikosti. Pyramida rysů byla v tomto případě napojena na *ResNet*[36] a společně tedy vytvářejí bohatou konvoluční pyramidu rysů vstupujících dále do sítě.



Obrázek 5.1: Vizualizace architektury RetinaNet použité jako detektor prvků výtahů[34]

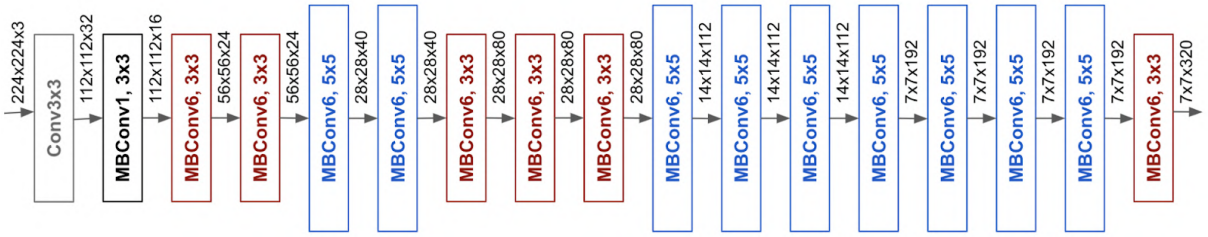
Toto jádro sítě bylo napojeno na dvě podsítě plně propojených vrstev, kdy jedna řešila klasifikaci navržených oblastí jádrem a druhá byla specializovaná na regresi navržených oblastí vůči skutečným ohraničujícím rámečkům objektů.

5.1.2 Klasifikátor číslic ovládačů volby podlaží

Po neúspěšných experimentech s metodami rozpoznávání znaků pomocí aplikací pro strojové čtění znaků a aplikace Tesseract (konkrétně knihovnou pro jazyk Python PyTesseract[37]) bylo rozhodnuto o implementaci klasifikátoru číslic ovládačů volby stanice pomocí neuronové sítě.

Účelem tohoto klasifikátoru bylo přiřadit dodané instanci ovládače volby podlaží štítek označující hodnotu číslice, která se u tohoto ovládače vyskytovala. Architekura byla zvolena na základě předešlého použití při rozeznávání textu a číslic, viz článek [40].

Síť byla založena na EfficientNet-B0[38] se vstupem 224×224 pixelů, jejíž architekura je přiblížena na obrázku 5.2. Síť využívá konvenčních konvolučních vrstev a bloků MBConv (mobile inverted bottleneck), které jsou stěžejními jednotkami architektury *MobileNetv2*.



Obrázek 5.2: Vizualizace vrstev architektury EfficientNet-B0 použité jako základ klasifikátoru ovládačů volby podlaží[38]

EfficientNet-B0 tedy slouží jako extraktor s předtrénovánými vahami z datové sady ImageNet[39], načež byly připojeny plně propojené vrstvy zakončené vrstvou softmax s 22 výstupy, které reprezentují hodnotu hledané číslice -2 až 19.

5.2 Použití TFOD2

V době vytváření diplomové práce knihovna TensorFlow Object Detection API přestupovala na TensorFlow verze 2.x.x, což zkomplikovalo některé operace a práci se sítěmi. Bylo tedy nutno upravovat zdrojový kód, případně implementovat vlastní skripty pro práci s knihovnou. Odkazy na tyto změny jsou uvedeny vždy v textu a vlastní skripty jsou

uvedeny v kapitole 5.2.2.

5.2.1 Příprava datové sady

Po vytvoření datové sady standardizovaným exportem do formátu *Pascal VOC* bylo nutné konvertovat tyto data do formátu podporovaného balíčkem TFOD2 *TFRecord*. *TFRecord* umožňoval uložení dat jako jednoduchou sekvenci binárních záznamů.

Následně byla datová sada ve formátu *TFRecord* rozdělena na trénovací a evaluační část v poměru 80 : 20 %, kdy z každé této části byly vytvořeny díly (anglicky sharding). Byl vytvořen skript `create_pascal_tf_record_sharding.py`, který zmíněné operace konvertování na *TFRecord* a rozdělení datové sady implementoval.

Pro iterativní přidávání obrazů do datové sady bylo nutno implementovat skript umožňující kombinovat jednotlivé datové sady typů Pascal VOC, což bylo provedeno skriptem `merge_multiple_datasets_pascal.py`. Podrobnější popis použitých skriptů je k nalezení v kapitole 5.2.2

Dále byl vytvořen seznam štítků mapující štítek (jméno objektu) k identifikačnímu kódu objektu. Tato mapa byla formátu PBTXT (typ Protobuf[42]). Dle použitých tříd z kapitoly 4.3.1 byla mapa štítků reprezentována:

```
item { id: 1
  name: 'btn_void'}
item { id: 2
  name: 'btn_floor'}
item { id: 3
  name: 'btn_bell'}
...
item { id: 9
  name: 'btn_open'}
item { id: 10
  name: 'btn_close'}
```

a celý soubor je přiložen v příloze A.

5.2.2 Implementované skripty

Byly vytvořeny skripty umožňující specifickou práci s daty, jejichž vzorem byly algoritmy zavedené v balíčku TFOD2.

`create_pascal_tf_record_sharding.py` sloužil ke konverzi datové sady typu *Pascal VOC* na *TFRecord*. Nejdůležitější myšlenkou této konverze bylo vytvoření sekvence sériových dat typu `tf.Example`. Zadáním složky s anotacemi (soubory typu XML, jejichž zástupce byl uveden v příloze B) společně se složkou obrázků a cestou k seznamu štítků vytvořil soubory `pascal.train` a `pascal.eval` rozdělené na zvolený počet tréninkových a evaluačních dílů (sharding). Toto dělení dovolovalo operaci s menšími soubory a možnost paralelních výpočtů z vícero dílů.

`merge_multiple_datasets_pascal.py` vytvořil z vícero zadaných složek datových sad *Pascal VOC* jednu, přičemž zachovával informaci o původu dat a eliminoval duplicitní pojmenování přidáním prefixu dle zadaného pořadí cest ke složkám datových sad.

`extract_class_images_from_pascal_and_sort_by_attribute.py` vyextrahoval obrázky zadané třídy z datové sady *Pascal VOC* a vytvořil dílčí složky dle hodnoty hledaného atributu. Tímto byla seřazena třída dle atributu, což bylo využito pro extrakci třídy *btn_floor*, kde byl hledán atribut *Value* označující číslici podlaží v obrázku pro trénink klasifikátoru.

`xml_rewrite_to_jpeg.py` sjednotil dva druhy přípon obrázků JPEG a JPG uvedené v anotačních souborech typu XML na JPEG, přičemž ignoroval přípony PNG.

`dataset_prune_negatives_pascal.py` odstraňoval takové snímky a anotační soubory datové sady *Pascal VOC*, které neobsahovaly žádný anotovaný prvek v anotačním souboru typu XML. Tímto skriptem byly odstraněny veškeré snímky, na kterých nebyly v rámci anotačního procesu definovány hledané objekty zájmu.

`detection_inference_tf2.py` modifikoval stejně nazvaný soubor balíčku TFOD2. Z důvodu použití nové verze *V2* ukládání modelů neuronových sítí bylo nutno pozměnit načítání modelu v metodě, kdy pro verze *V1* musel být uložený model skriptem sestaven, zatímco nová verze redukuje tuto funkcionalitu na jeden řádek načtení modelu, konkrétně `detect_fn = tf.saved_model.load(<cesta_k_modelu>)`.

`infer_detections_tf2.py` kompletně reimplementovala skript pro inferenci dat neuronovou sítí. TensorFlow2 nedisponuje `tf.Session` a zabudovanými globálními parametry `FLAGS`. Globální parametry a logování byly tedy zaimplementovány samostatným balíčkem `abseil`. Principem tohoto skriptu bylo vytvoření iterátoru na předložených datech typu *TFRecord* a načtení modelu sítě skrze skript `detection_inference_tf2.py`. Dále byla načtená data deserializována a získán digitální obraz, který byl inferován modelem, přičemž výstupem byla opět serializovaná data, která již byla doplněna informacemi o detekovaných instancích objektů.

5.3 Trénink a evaluace sítí

Bylo vytvořeno nové virtuální prostředí pomocí manažeru balíčků Conda s využitím programovacího jazyka Python 3.8.2 a balíčku pro strojového učení TensorFlow 2.3.1. Pro trénink na grafické kartě bylo použito podporovaných verzí cuDNN 7.6.5 a CUDA 10.0 ovladačů.

Jelikož TFOD2 používá *Protobuf*[1][42] souborů pro konfiguraci modelu a trénovacích parametrů, bylo nutno zkompilovat interpretaci těchto souborů pro programovací jazyk Python na platformě operačního systému Windows.

Trénink byl realizován na hardwaru uvedeném v následující tabulce 5.1.

Komponenta	Název
CPU	AMD Ryzen 5600
GPU	NVIDIA RX 2070
Paměť	Kingston DDR4 3200MHz 16GB x2

Tabulka 5.1: Použitý hardware pro učení neuronových sítí

[1]Protocol Buffer - knihovna umožňující vysokorychlostní multiplatformní zápis a četbu binárně serializovaných dat

5.3.1 Příprava na učení detektoru

Byl stažen předtrénovaný model vybrané architektury detektoru *RetinaNet50* z TFOD2 Zoo.[43] Složka s modelem poskytovala přístup k souboru *pipeline.config*, kde bylo možno nakonfigurovat celý proces učení. Tento soubor umožňoval nastavení desítek parametrů, například velikost šarže, míry trénování, parametrů regularizace, počet kroků učení, augmentizaci vstupních dat a mnoho dalších.

Parametry, které bylo nutno změnit v tomto souboru, byly počet detekovaných tříd, cesta k předtrénovanému modelu, cesta k seznamu štítků, typ uloženého předtrénovaného modelu a jeho verze a cest k trénovacímu a evaluačnímu datesetu:

```
model {
    ssd {
        num_classes: 10
...
    fine_tune_checkpoint:  "<...>/ckpt-0"
...
    fine_tune_checkpoint_type: "detection"
    fine_tune_checkpoint_version: V2
...
train_input_reader {
    label_map_path: "<...>/labelmap_detection.pbtxt"
    tf_record_input_reader {
        input_path: "<...>/train.record-?????-of-00020"
    }
}
...
eval_input_reader {
    label_map_path: "<...>/labelmap.pbtxt"
    tf_record_input_reader {
        input_path: "<...>/test.record-?????-of-00004"
    }
}
```

Neexistuje heuristická metoda volby hyperparametrů sítě. Pro vyladění a robustnost učení na základě provedených experimentů s tréninkem sítí byly iterativně upravovány v souboru *pipeline.config* vlastnosti učení do následující podoby:

```
train_config {
#zmenšení velikosti šarže kvůli nedostatečné paměti pro trénink
    batch_size:2
...
#odstranění augmentace horizontálním převrácením
    data_augmentation_options {
#přidání augmentace změnou velikosti obrazu
        random_image_scale {
            min_scale_ratio: 0.8
            max_scale_ratio: 1.5
```

```
    }
...
    optimizer {
        momentum_optimizer {
            learning_rate {
                cosine_decay_learning_rate {
# zvýšena počáteční míra tréninku pro rychlejší konvergenci k řešení
                    learning_rate_base: 0.07
# zmenšení počtu kroků výpočtu míry tréninku dle použitých počtu iterací sítě
                    total_steps: 50000
...
# zmenšení počtu iterací sítě díky rychlejší konvergenci k řešení
    num_steps: 50000
}
```

S takto připravenými daty a konfigurací bylo možno přistoupit k tréninku v kapitole 5.3.2. Soubor *pipeline.config* použitý k tréninku architektury *RetinaNet50* je dostupný v příloze C.

5.3.2 Proces učení detektoru

Učení bylo započato pomocí skriptu `model_main_tf2.py`, který se nacházel v hlavní složce balíčku TFOD2, přičemž z důvodu opakovaného použití byl implementován skript `train_cmd.py` generující příkaz pro vložení do příkazového řádku:

```
#Generuje prikaz pro trenink modelu
script_path = r"<cesta_ke_skriptu_model_main_tf2.py>"
model_dir_path = r"<cesta_ke_slozce_modelu_site>"
train_steps = r"50000"

#cesta k pipeline.config
pipeline_config_path = " --pipeline_config_path=" + model_dir_path + r"/
    pipeline.config"
#cesta k souboru ulozeneho modelu
model_dir = " --model_dir=" + model_dir_path + r"/model"
#pocet kroku uceni
num_train_steps = " --num_train_steps=" + train_steps

cmd = script_path + pipeline_config_path + model_dir + num_train_steps
print("python " + cmd)
```

Pokračovat v předešlém tréninku bylo možné specifikací parametru `-checkpoint_path`, který odkazoval na model uložený v průběhu tréninku. TensorFlow ukládal pouze pět nejnovějších modelů bez možnosti průběžného ukládání. Práce Martina Horáka [44] implementovala použitý skript `backup_model_files.py`, který periodicky kontroloval složku

s modely a nové soubory automaticky ukládal.

TFOD2 v průběhu učení ukládá a vyhodnocuje aktuální model a ukládá jej do souborů:

- *ckpt-23.data-00000-of-00001*
- *ckpt-23.index*

Tyto uložené soubory obsahovaly přesné hodnoty všech použitých parametrů sítě. Pro použití modelu bylo potřeba exportovat tento model do formátu *Protobuf*, který obsahuje jak definici použité sítě, tak i veškeré váhy a parametry. Konverze těchto uložených dat na model ve formátu *Protobuf* vykonával `exporter_main_v2.py`, kdy skript `freeze_model_cmd.py` generoval příkaz:

```
#Generuje prikaz pro export modelu
script_path = r"<cesta_ke_skriptu_exporter_main_v2.py>"
pipeline_path = r"<cesta_k_souboru_pipeline.config>"
trained_checkpoint_path = r"<cesta_k_natrenovanemu_modelu>"
output_path = r"<cesta_ke_slozce_pro_export>"

input_type = " --input_type=" + r"image_tensor"
pipeline_config_path = " --pipeline_config_path=" + pipeline_path
trained_checkpoint_dir = " --trained_checkpoint_dir=" +
    trained_checkpoint_path
output_dir = " --output_directory=" + output_path

cmd = script_path + input_type + pipeline_config_path +
    trained_checkpoint_dir + output_dir
print("python " + cmd)
```

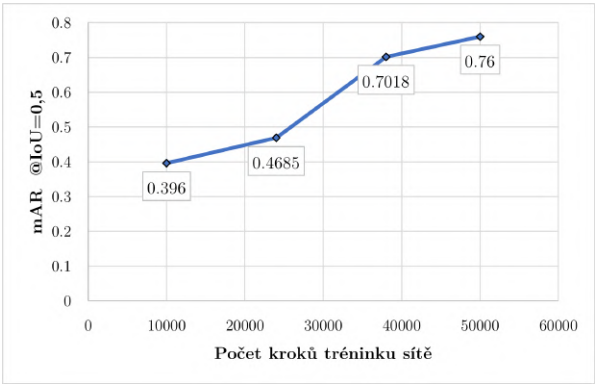
5.3.3 Evaluace detektoru

Aplikační rámec TFOD2 disponoval monitorovací webovou aplikací TensorBoard, která zaznamenávala metriky učení. Inicializace TensorBoard byla provedena příkazem

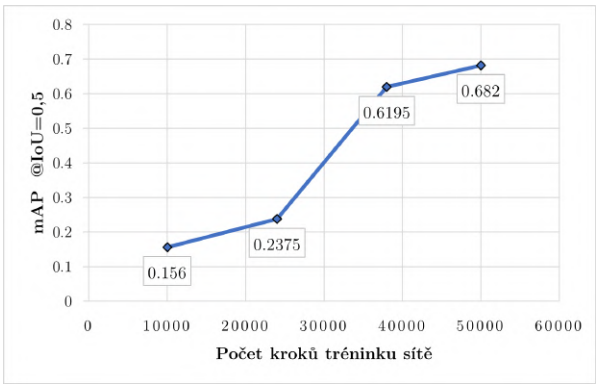
```
tensorboard --logdir<cesta_ke_slozce_trenovaneho_modelu>
```

Webová adresa `http://127.0.0.1:6000` umožnovala připojení k aplikaci pomocí webového prohlížeče. TensorBoard zobrazoval hodnotu účelové funkce, metriky COCO a velikost míry učení, kdy všechny tyto metriky byly generovány aplikačním rozhraním učení TFOD2.

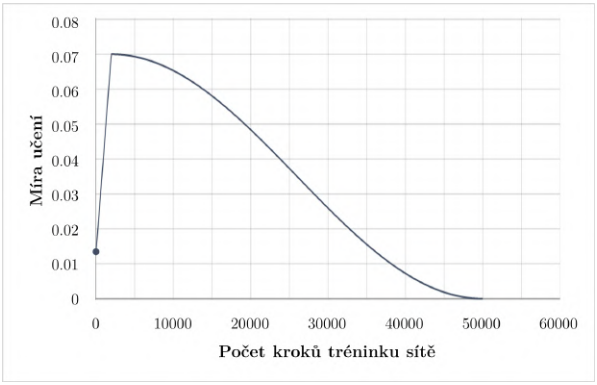
V obrázcích 5.3 až 5.6 je možno pozorovat vývoj metrik v průběhu učení. Evaluační sada byla z důvodu výpočetní náročnosti vyhodnocena pouze v uložených krocích sítě 10 000, 24 000, 38 000, 50 000. Trénink sítě trval osm hodin a výsledné evaluační metriky typu COCO jsou uvedeny v tabulce 5.2.



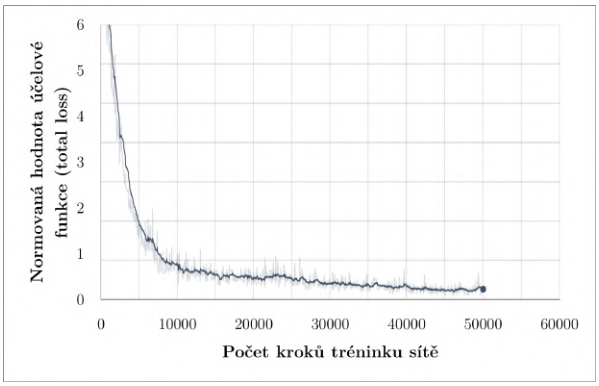
Obrázek 5.3: Tensorboard - mAR pro IoU=0,5, vývoj evaluační metriky v průběhu učení



Obrázek 5.4: Tensorboard - mAP pro IoU=0,5, vývoj evaluační metriky v průběhu učení



Obrázek 5.5: Tensorboard - mírá učení v průběhu tréninku



Obrázek 5.6: Tensorboard - celková normovaná hodnota účelové funkce v průběhu tréninku

Metrika	Maximální počet detekcí	Hodnota metriky
$mAP^{IoU=0,5:0,95}_{area=all}$	maxDets=100	0,682
$mAP^{IoU=0,5}_{area=all}$	maxDets=100	0,960
$mAP^{IoU=0,75}_{area=all}$	maxDets=100	0,828
$mAP^{IoU=0,5:0,95}_{area=small}$	maxDets=100	0,672
$mAP^{IoU=0,5:0,95}_{area=medium}$	maxDets=100	0,651
$mAP^{IoU=0,5:0,95}_{area=large}$	maxDets=100	0,694
$mAR^{IoU=0.5:0,95}_{area=all}$	maxDets=1	0,610
$mAR^{IoU=0,5:0,95}_{area=all}$	maxDets=10	0,740
$mAR^{IoU=0,5:0,95}_{area=all}$	maxDets=100	0,760
$mAR^{IoU=0,5:0,95}_{area=small}$	maxDets=100	0,707
$mAR^{IoU=0,5:0,95}_{area=medium}$	maxDets=100	0,705
$mAR^{IoU=0,5:0,95}_{area=large}$	maxDets=100	0,778

Tabulka 5.2: Metriky COCO[26] plně natrénovaného detektoru

Konkrétní interpretace metrik je následující[26]:

- $mAP^{IoU=0.5:0.95}$ je mAP zprůměrované skrze všech deset prahů IoU (0,50; 0,55; 0,60;

...; 0,95), jedná se o primární metriku použitou pro porovnávání způsobilosti sítě.

- kromě různých použitých prahů IoU je mAP vypočítáno i s využitím různých velikostí objektů, kdy *area=small* odpovídá malým objektům s rozměry do 32^2 pixelů, *area=medium* odpovídá středním objektům s rozsahem velikostí mezi 32^2 až 96^2 pixelů, *area=large* odpovídá velkým objektům s rozměry většími než 96^2 pixelů. *area=all* pak průměruje skóre těchto tří možností.
- hodnota $mAP^{IoU=0.5}_{area=all}$ = 0,960 tedy odpovídá 96% pravděpodobnosti nalezení objektů skrz všechny zmíněné oblasti s alespoň IoU=0,50.
- podobně je tak učiněno i u metriky mAR, kde se ovšem vypočítává i varianta s omezením maximálního počtu detekcí v obraze *maxDets*, kdy tento parametr nabývá hodnoty 1, 10 nebo 100.

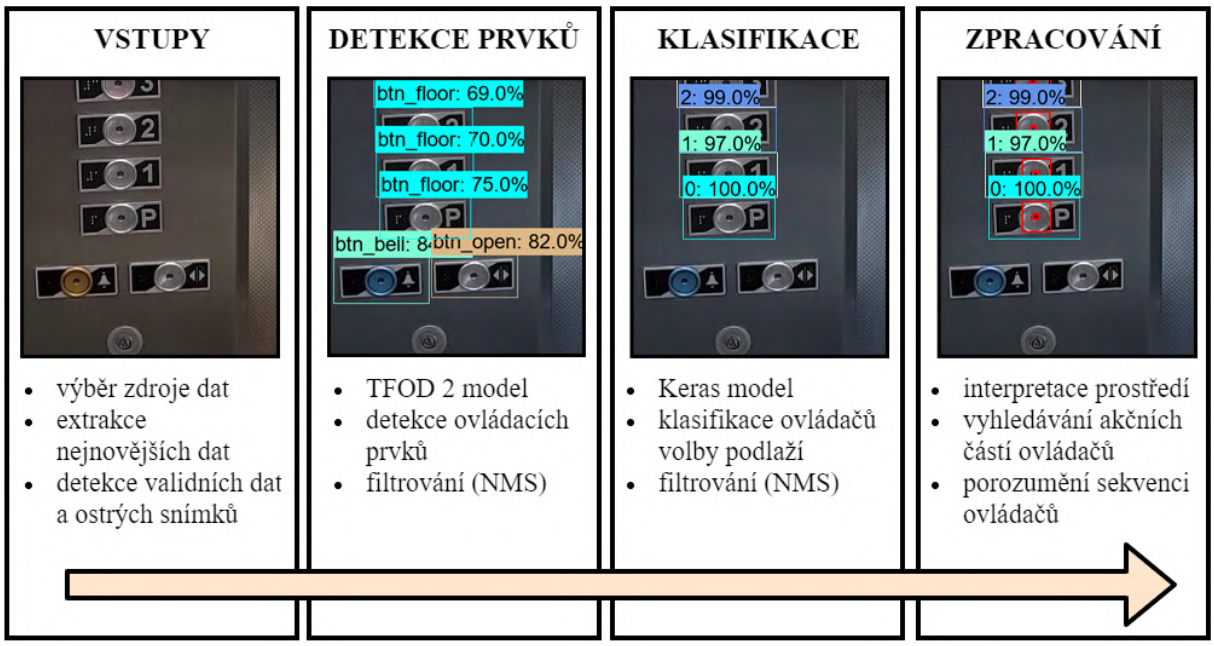
5.3.4 Trénink klasifikátoru číslic

Pomocí skriptu `extract_class_images_from_pascal_and_sort_by_attribute.py` byly z datové sady extrahovány instance ovládačů volby podlaží dle uvedené číslice, viz kapitola 4.3.2. Takto separové hodnoty v samostatných složkách (22 tříd, hodnoty číslic -2 až 19) sloužily jako klasifikační třídy a vytvářely datovou sadu pro trénink klasifikátoru.

Samotný proces trénování byl spouštěn skriptem `custom_training_keras.py` (k nalezení v digitální příloze **??**) za použití aplikačního rozhraní TensorFlow Keras. Skript byl schopen sestavit model, iniciovat učení, evaluovat na poskytnuté datové sadě a vizualizovat na zvoleném počtu vzorků. Model dosahoval 98% přesnosti určení třídy na předložených datech, přičemž tabulka záměn tříd je uvedena v příloze D.

6 Algoritmus metody

Šestá část diplomové práce se zabývá použitím detektoru a klasifikátoru z kapitoly 5 v naprogramované aplikaci. Názorné schéma algoritmu je nastíněno v obrázku 6.1. Algoritmus respektoval principy objektově orientovaného programování s abstraktními třídami tak, aby kterákoliv jeho část mohla být implicitně nahrazena jinou (při implementaci stejných metod).



Obrázek 6.1: Vizualizace průběhu metody

Třídy programu, které odpovídají jednotlivým fázím algoritmu, jsou vždy pojmenovány anglicky a jejich funkce je podrobněji popsána v jednotlivých podkapitolách. Jedná se o třídy:

- *InputFeed* – poskytování šarže vstupních dat
- *InputProcessing* – zpracování vstupních dat
- *ElementDetection* – detekce ovládacích prvků výtahu
- *ButtonClassification* – klasifikace ovládačů volby podlaží
- *OutputProcessing* – interpretace prostředí, nalezení akčních částí ovládačů
- *ButtonRelabeling* – přeznačení sekvence ovládačů
- *OutputVisualization* – vizualizace fází metody, ukládání výstupních obrazů

6.1 Definice parametrů

V rámci aplikace bylo použito nastavovacích parametrů, které ovlivňovaly běh aplikace, především určením použitých vstupů, lokalizaci modelů a výstupní vizualizaci. Mezi nejdůležitější patří následující:

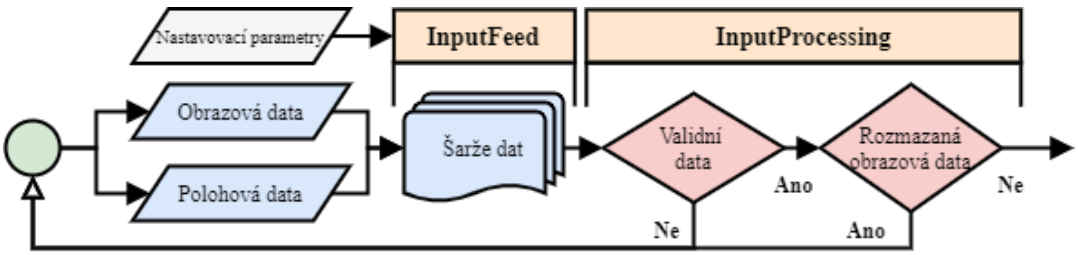
- *ImageInputMode*: výběr obrazového vstupu mezi synchronním nebo asynchronním čtením z videa, asynchronním vstupem obrazových dat z webkamery nebo čtení obrazů ze složky
- *DetectorElementsModelPath*: cesta k modelu typu *SavedModel* zajištující detekci ovládacích prvků výtahu
- *LabelMapPathDetection*: cesta k textovému souboru obsahující strukturu sloužící k mapování štítků typů ovládacích prvků dle provedené detekce
- *ClassificationFloorButtonModelPath*: cesta k modelu typu *SavedModel* zajištující klasifikaci kandidátů tlačítek volby pater výtahové kabiny
- *LabelMapPathClassification*: cesta k textovému souboru obsahující strukturu sloužící k mapování štítků (zde číslic) dle provedené klasifikace
- *VisualizationMode*: výběr mezi vizualizací v interaktivním okně, v pevném okně, nebo uložením do souborů

6.2 Vstupy

Vstupem algoritmu mohlo být dodané video, složka s obrázky nebo nahrávání obrazu z webkamery. Tento vstup byl doplněn informací označení polohy robotu, ve které se robotická platforma aktuálně nachází. Tato poloha byla v diplomové práci simulována konstantní veličinou specifikovanou pro daný typ vstupních dat. Algoritmus využíval následující označení polohy robotu:

- *CabinPanel* – robotická platforma se nachází v kabině výtahu čelně k vnitřnímu ovládacímu panelu
- *Cabin* – robotická platforma se nachází v kabině výtahu
- *FrontOfElevator* – robotická platforma se nachází před kabinou výtahu
- *Other* – označení pro jiná prostředí než tři výše zmíněná

Tyto vstupní data zajišťovala třída *InputFeed*, jejíž funkcí bylo dodávat šarže dat (obrazových a polohových) další části aplikace dle nastavených parametrů, jak je znázorněno v obrázku 6.2.



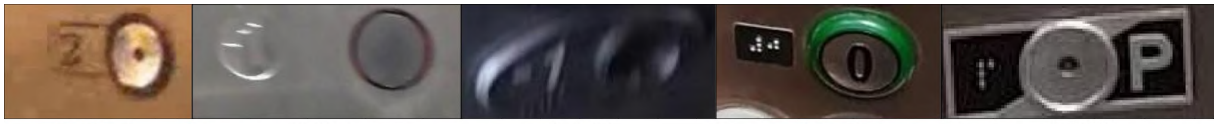
Obrázek 6.2: Diagram fází vstupu (InputFeed) a předzpracování dat (InputProcessing)

6.2.1 Předzpracování dat

Pro zajištění robustnosti metody byla zavedena metrika rozostřenosti a kontrola validního vstupu. Byl zaveden předpoklad rozostřeného vstupního obrazu v důsledku možnosti pohybu robota a předpoklad proměnného osvětlení způsobeného osvětlením z vnější části kabiny výtahu. Toto zmíněné osvětlení se měnilo při otevírání a zavírání kabinových dveří.

Byla implementována třída *InputProcessing* zodpovědná za předzpracování obrazových dat. Tato třída kontrolovala, zda-li byl obdržen nový snímek. Pokud ne, veškerá další logika byla přeskočena a nastával pokus znovu získat nový snímek. Poté co třída obdržela nový validní obraz, a to ve formě *NumPy*[45] pole, byla zkontrolována jeho velikost (rozměry pole: výška, šířka, počet barevných kanálů).

Byla zavedena metrika kontroly rozostřenosti vstupního obrazu. Na vstupní obraz byl aplikován Laplaceův operátor. Tímto byla získána druhá derivace vstupního obrazu, odpovídající velkým změnám intenzit obrazu. Velká variace těchto změn intenzit pak odpovídá zaostřenému obrazu, zatímco malá variace odpovídá rozostřenému obrazu. Každému vstupnímu obrazu přiděleno absolutní skóre rozostřenosti, které bylo implementováno jako invariantní vůči rozlišení vstupního obrazu. Dle experimentů s touto metrikou s použitím rozmazaných digitálních obrazů v datové sadě byl zvolen práh akceptovatelnosti vstupního obrazu. Vizualizace metriky rozostřenosti obrazu je uvedena v obrázcích 6.3 a 6.4.



Obrázek 6.3: Vizualizace metriky kontroly rozostřenosti obrazu - detail na různé ovládače volby podlaží doplněné hodnotou metriky

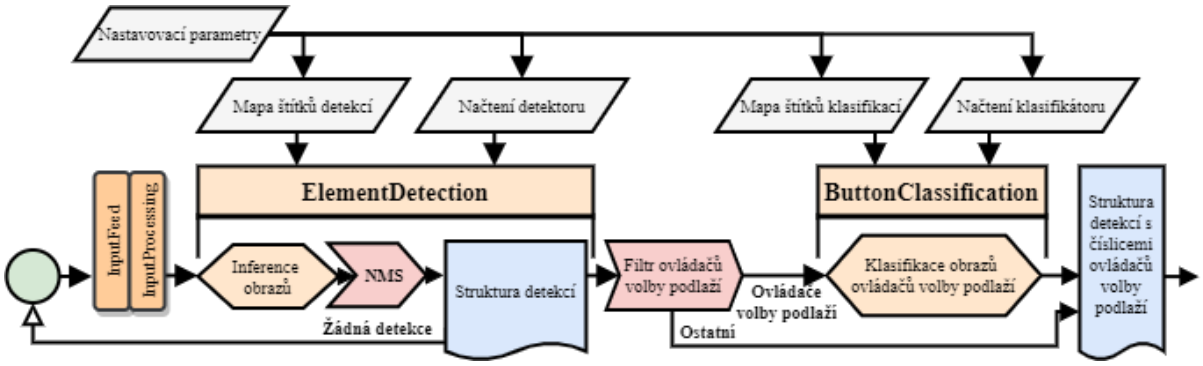
Obrázek 6.4: Vizualizace metriky kontroly rozostřenosti obrazu - detail na konkrétní ovládač volby podlaží v průběhu záznamu doplněný hodnotou metriky

Myšlenkou za touto selektivní metodou a volbou práhu je předpoklad nemožnosti detekce takových vstupních dat, které ani lidský zrak nedokáže s absolutní jistotou rozeznat. Postup dat skrze tuto fázi algoritmu je znázorněn v obrázku 6.2.

Pro kompenzaci variace osvětlení byl také proveden experiment se zpracováním digitálního obrazu pomocí algoritmu CLAHE[46], který vede k ekvalizaci histogramu získaného obrazu. Z důvodu velké změny vstupního obrazu vůči obrazovým datům v datové sadě se však výsledek detekce zhoršoval, a tato metoda vylepšení obrazového vstupu nebyla použita.

6.3 Využití neuronových sítí

V rámci inicializace algoritmu byly načteny modely a váhy detektoru výtahových prvků a klasifikátoru číslic tlačítek. Požadovaným výstupem této sekce byly detekce ovládacích prvků výtahu, doplněné o klasifikaci číslic podlaží kandidátů třídy ovládačů volby podlaží. Diagram této fáze je představen v obrázku 6.5.



Obrázek 6.5: Diagram fází detekce obrazových dat (ElementDetection) a klasifikace ovládačů volby podlaží (ButtonClassification), NMS - algoritmus non-maximum suppression

6.3.1 Detekce ovládacích prvků výtahu

Vytvořená programová třída *ElementDetection* se zabývala řádným načtením modelu z definované složky *DetectorElementsModelPath* společně s natrénovanými váhami. Třída podporuje načtení jakéhokoliv modelu typu *SavedModel*.

Po načtení byla k dispozici metoda *infer_image*, zajišťující inferenci jakéhokoliv dodaného obrazu ve formátu NumPy matice. Obraz byl převeden na tenzor a model provedl inferenci dopředným průchodem načtenou neuronovou sítí a navrátil strukturu detekcí obsahující:

- Počet detekcí v obrazu
- Ohraničující rámečky detekcí v dodaném digitálním obrazu
- Seznam indexů označujících třídu s nejvyšším skórem jistoty pro každou detekci
- Seznam skóre jistoty detekce třídy
- Normovaný seznam skóre jistoty všech tříd pro každou detekci

Pro filtraci slabých a překrývajících se detekcí bylo použito algoritmu *non-maximum suppression*, kdy detekce se skórem jistoty menším než 0,4 byly automaticky zahozeny a ohraničující rámečky s IoU větším než 50 % sloučeny v jeden rámeček podle rámečku s nejvyšším skórem jistoty. Pomocí této detekce byli navrženi kandidáti ovládacích prvků výtahu vstupního obrazu.

6.3.2 Klasifikace číslic ovládače volby podlaží

Po obdržení kandidátů ovládacích prvků výtahu z předchozí podkapitoly 6.3.1 byla z detekcí vyfiltrována třída *btn_floor* zastupující kandidáty oblastí ovládačů volby podlaží. Následně byly vytvořeny obrazy kandidátu ovládačů oříznutím vstupního obrazu na oblasti dle ohraničujících rámečků kandidátů ovládačů.

Třída *ClassifierKeras* načetla uložený *SavedModel* neuronové sítě z definované složky *ClassificationFloorButtonModelPath*, a poté inicializovala váhy načtené z téže složky.

Samotná klasifikace byla zajištěna voláním metody *infer_image*. Tato metoda přijímala vstupní obraz kandidáta tlačítka. Dodaný obraz byl opět převeden na tenzor a model provedl inferenci dopředným průchodem načtenou neuronovou sítí.

Klasifikátor navrátil normovaný seznam skóre jistoty všech tříd pro daný vstupní obraz. Pro usnadnění práce s výsledky byla zavedena struktura *classifications*, obsahující:

- Ohraničující rámeček ovládače z detekce
- Nejvyšší skóre jistoty
- Štítek odpovídající třídě nejvyššímu skóre jistoty (predikovaná číslice ovládače)[1]

6.4 Zpracování dat z neuronových sítí

Programová třída *OuputProcessing* byla zodpovědná za zpracování dat z neuronových sítí, konkrétně po získání dat popsaných v kapitole 6.3 byla provedena kontrola výstupu neuronových sítí vůči typu detekovaného prostředí výtahu.

Typ detekovaného prostředí	**Požadavky**
Dispej aktuálního podlaží kabiny	Maximálně 1 ovládač volby podlaží výtahu Alespoň 1 dispej aktuálního podlaží kabiny Poloha robotu *CabinPanel, Cabin, FrontOfElevator*
Ovládačová kombinace volby podlaží	Alespoň 2 a maximálně 25 ovládačů volby podlaží výtahu Alespoň 1 ovládač komunikace (ovládač ALARM / ovládač POMOC) Maximálně 1 dispej aktuálního podlaží kabiny Poloha robotu *CabinPanel, Cabin*
Ovláčová kombinace stanice	Alespoň 1 a maximálně 4 ovládače volby podlaží Detekce právě 1 instance ovládačové kombinace stanice Maximálně 1 displej aktuálního podlaží kabiny Poloha robotu *FrontOfElevator*
Ostatní	Nebyly splněny požadavky na žádný z výše uvedených typů prostředí

Tabulka 6.1: Typy detekovaného prostředí

Byl zaveden sémantický algoritmus rozlišující čtyři typy prostředí, jejichž název a požadavky jsou popsány v tabulce 6.1. Zmíněné polohy robotu byly již definovány v kapitole 6.2.

Při splnění všech vypsaných požadavků byl dané instanci detekcí přiřazen typ prostředí. Tyto typy prostředí se vzájemně vylučovaly.

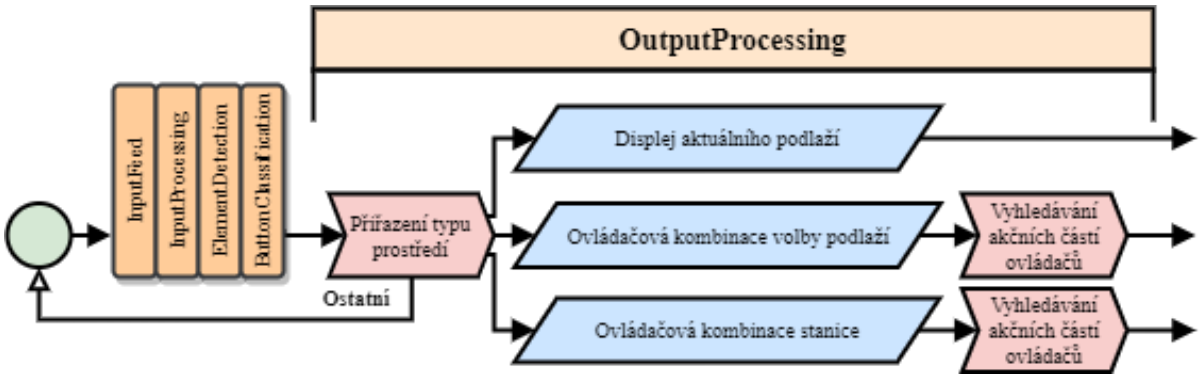
Při detekci prostředí typu *Displej aktuálního podlaží* nebyly učiněny další kroky zpracování těchto dat. Pro prostředí typu *Ostatní* byl výsledek detektoru zamítnut a algoritmus započal další iteraci začínající získáním nových vstupních dat.

Na základě typu detekovaného prostředí *Ovládačové kombinace volby podlaží*, respektive *Ovládačové kombinace stanice* byl implementován rozdílný způsob dalšího zpracování vizualizovaný v diagramu 6.6 a popsaný v podkapitolách 6.4.1, respektive 6.4.2.

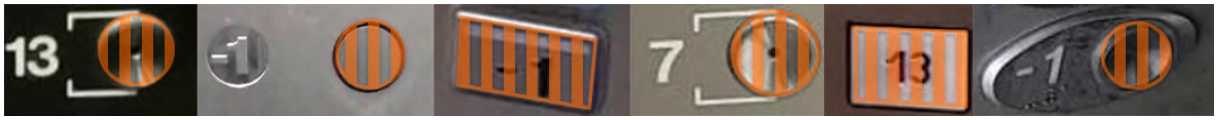
6.4.1 Interpretace prostředí typu ovládačové kombinace volby podlaží

Dle nasbíraných dat byla u tohoto prostředí předpokládána instance ovládačové kombinace volby podlaží jako uceleného bloku ovládačů kabiny v jedné oblasti, s možným výskytem displeje aktuálního podlaží kabiny.

[1] mapován pomocí textového soubouru poskytnutého z *LabelMapPathClassification*



Obrázek 6.6: Diagram zpracování dat z neuronových sítí (OutputProcessing)



Obrázek 6.7: Ukázka akčních částí ovládačů volby podlaží

Pro možnost použití mobilním robotem s možností změny podlaží stiskem ovládače volby stanice bylo nutné lokalizovat akční část ovládačů, což byla právě ta část detekovaného ovládače, jejíž stisk způsobil záznam příkazu. Tato akční část u ovládačů volby podlaží je představena v obrázku 6.7.

Předpokládalo se, že akční část byla dominantním rysem ovládače volby stanice s nejčastěji kruhovým, případně zaobleným obrysem a jeho velikost byla mezi 30–120 % výšky ohraničujícího rámečku získaného detekcí. Nalezení této akční části bylo zprostředkováno metodou vyhledávání kružnic v Houghově prostoru.[47] Každý obraz ovládače volby stanice byl převeden na černobílý obraz, provedena na něm operace Unsharp Mask[2] a nalezeny alespoň tři kružnice dle zadaných parametrů.

Obrázek 6.8: Vyhledávání akčních oblastí ovládačů volby podlaží výtahové kabiny, postupně zleva doprava: originální obraz ovládače z detekce prvků, převod na černobílý obraz a filtrování Unsharp Mask, nalezení kružnic v Houghově prostoru, detekovaná akční část ovládače

[2]metoda zostření obrazu využívají negaci rozmazaného vstupního obrazu

Akumulátor[3] použité metody vyhledávání kružnic byl iterativně inkrementován, dokud nebyly nalezeny alespoň tři kružnice, nebo akumulátor nedosáhl předem definovaného maxima. Tyto tři kružnice byly následně sjednoceny v jednu, a to metodou *non-maximum suppression*. Ukázka použití metody vyhledávání akční oblasti je demonstrována na obrázku 6.8.

6.4.2 Interpretace prostředí typu ovládačů kombinace stanice

Toto prostředí obsahovalo instanci ovládačové kombinace stanice, jakožto oblasti v blízkosti výtahových dveří s ovládači pro přivolání výtahové kabiny. U některých vzorků v datovém setu se i u těchto instancí objevovaly displeje aktuálního podlaží nebo směru kabiny.

Pro možnost přivolání výtahu robotickou platformou bylo nutné nalézt akční části ovládačů. Metoda vyhledávání akčních částí ovládačů byla přiblížena v předchozí podkapitole 6.4.1.

Jednotlivé vzory ovládačů stanice nebyl detektor explicitně naučen rozpoznávat, proto byla metodě vyhledávání akčních oblastí, v případě nenalezení akčních částí ovladačů stanice, předložena celá oblast instance ovládačové kombinace stanice, přičemž parametry metody byly mírně upraveny.

6.5 Porozumění sekvenci ovládačů

Nejdůležitějším detekovaným elementem byl (pro možnost využití na robotické platformě) ovládač volby podlaží. Byl vznesen předpoklad, že klasifikace ovládačů volby pater nebyla bezchybná, a že některá označení ovládačů nemusela být ve vstupním obraze dostatečně čitelná. Pro odhalení špatně rozeznaných štítků v prostředí typu ovládačové kombinace volby podlaží byla implementována metoda, která využívala následujících předpokladů vztahující se na ovládače volby podlaží v kabině výtahu:

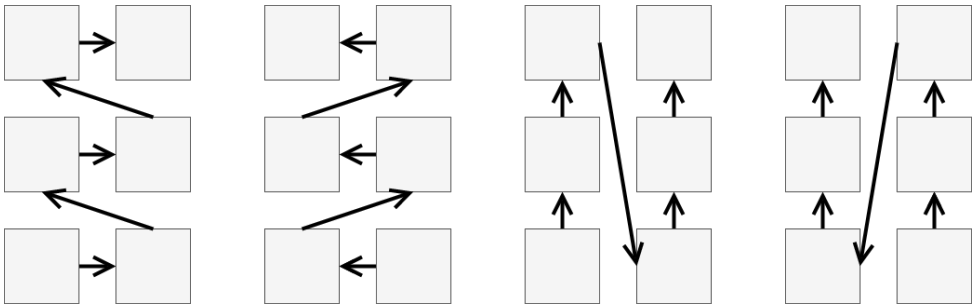
- Tlačítka byla seskupena v jednotný blok
- Tlačítka byla uniformně rozdělena do řádků a sloupců vyjma prvního a posledního řádku
- Velikosti rozestupů mezi dvěma sousedními tlačítky byly rovnoměrné
- Tlačítka byla řazena vzestupně či sestupně po řádcích či sloupcích, viz obrázek 6.9
- U takovéto sekvence ovládačů byl možný výskyt vynechání číslice (například řada číslic -2,-1, 1, 2, 3, ...)
- Všechna tlačítka výběru podlaží byla řádně detekována (byl znám celkový počet tlačítek)
- Poloha číslic tlačítek byla vždy na levé straně akčního členu tlačítka, případně přímo na akčním členu tlačítka, jak je požadováno v normě a uvedeno v tabulce 3.1

6.5.1 Implementace automatického přeznačení podlaží

Na základě zmíněných předpokladů a interpretace detekovaných dat prostředí ovládačové kombinace volby podlaží byla implementována třída *RelabelButton*. Jejím vstupem byly polohy a štítky označení ovládačů volby podlaží, doplněné průměrnou výškou a šířkou tlačítek.

[3] zde zastupoval metriku nedokonalosti hledaných kružnic

V první řadě bylo nutné zjistit způsob řazení tlačítek. Všechny zohledněné možnosti jsou znázorněny na obrázku 6.9.



Obrázek 6.9: Zohledněné možnosti řazení označení ovládačů volby podlaží, od nejčastějšího (vlevo) po nejméně časté (vpravo)

Algoritmus využil třídy *Template* (šablona), která vypočítala hodnosti všech způsobů řazení tlačítek následujícím algoritmem:

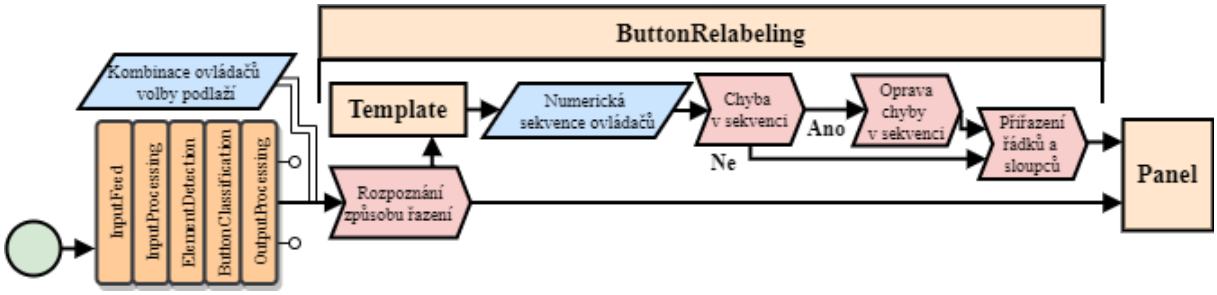
```
# Definice startovnich indexu listu.
if direction == 1:
    curr, foll = 0, 0
else:
    curr, foll = -1, -1
    rank = 0
# Iterace skrze list.
for _ in range(len(listed_numbers) - 1):
    foll += direction
    absdiff = abs(listed_numbers[foll]) - abs(listed_numbers[curr])
    if listed_numbers[curr] < listed_numbers[foll] and absdiff < 3:
        rank += 1  # Sekvence odpovida zvolenemu smeru.
    curr = foll
return rank
```

Byl zvolen způsob řazení ovládačů s nejvyšší hodností, načež bylo toto řazení šabloně přiřazeno. Na základě řazení byla vytvořena sekvence štítků označení ovládačů volby podlaží, která byla otestována algoritmem určeným pro identifikaci nesprávně označených štítků sekvence. Algoritmus respektoval uvedený předpoklad možného výskytu vynechání číslice podlaží inspirovaný předchozím řešením v práci [13].

Sekvence štítků ovládačů s odhalenými nesprávně označenými štítky byla opravena pomocí rekurzivního algoritmu. Při detekci nesprávně označeného štítku po vynechaném tlačítku podlaží bylo naopak využito klasifikací číslice z kapitoly 6.3.2 s horším skórem jistoty. Tento algoritmus pak porovnával nově vybraný štítek s nejblíže správně označeným tlačítkem. Algoritmus přeznačování štítků je rozvinut v příloze F.

Posledním krokem bylo přiřazení řádků a sloupců jednotlivým ovládačům dle jejich

polohy v obraze, což implementovala třída *Panel*. Celý postup je znázorněn v diagramu 6.10



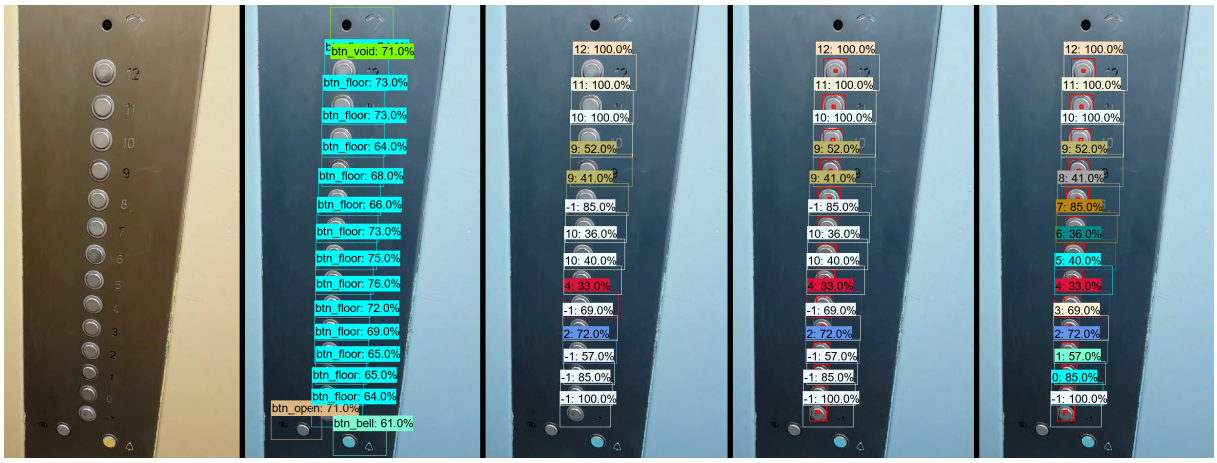
Obrázek 6.10: Diagram automatického přeznačení podlaží

6.5.2 Popis panelu ovládačů volby podlaží

Každé tlačítko bylo dále zastoupeno třídou *Button* obsahující jeho pozici v obraze, řadu, sloupec a štítek podlaží pomocí dat získaných metodami popsanými v předešlé kapitole. Zaštiťující třída *Panel* pak sjednocovala instance třídy *Button* v seznam těchto prvků a disponovala dále informacemi o typu řazení sekvence.

6.6 Fúze výstupů na vstup, vizualizace

Implementace třídy *OutputVisualization* zajišťovala zobrazování ohraničujících rámečků, dat a štítků na vstupní obrazy a ukládání obrazů do souborů. Disponovala zobrazením výstupu detekce a klasifikace společně s generickými metodami zobrazování tvarů a textu.



Obrázek 6.11: Sekvence obrazů vizualizace fází algoritmu automatické detekce ovládacích prvků výtahu, postupně zleva doprava: originální obraz vstupu, detekce ovládacích prvků, klasifikace ovládačů volby podlaží, nalezení akčních čáastí ovládačů volby podlaží, přeznačení sekvence číslic

U výsledné fúze výsledků pro kontrolu funkce automatické detekce ovládacích prvků výtahu bylo možné zvolit zobrazované typy prvků pomocí nastavitelných parametrů. Pro vizualizaci výstupu bylo možno volit mezi:

- *image_save* – vizualizace realizována skrze ukládání do souborů typu PNG
- *image_window* – vizualizace realizována pomocí neinteraktivního okna

Typy zobrazených prvků bylo možno kombinovat obdobným použitím nastavitelných parametrů, kde byl výběr z vizualizace detekcí, klasifikací tlačítek výběru pater, lokalizace akčních částí tlačítek podlaží a přivolávacího panelu a zobrazení aktuálního typu výtahového prostředí.

V sekvenci obrazů 6.11 jsou pro znázornění volby zobrazených prvků inkrementálně vizualizovány jednotlivé fáze algoritmu automatické detekce ovládacích prvků výtahu u prostředí typu ovládačové kombinace volby podlaží.

7 Ověření metody

Pro ověření metody detekce ovládacích prvků výtahových zařízeních, byly vytvořeny videozáznamy kombinací ovládačů volby podlaží a kombinací ovládačů stanice. Ověření metody probíhalo na osobním počítači bez grafické karty. Byla ignorována část předzpracování dat algoritmu pro demonstraci nesprávných výstupů na základě nedostatečně kvalitních vstupů. V této kapitole je demonstrováno použití metody na kombinace ovládačů, ostatní detekovaná prostředí jsou zahrnuta v digitální příloze.

7.1 Úspěšná detekce kombinace ovládačů

Metoda je schopna úspěšné detekce ovládacích prvků v obraze a rozlišuje jednotlivé třídy ovládacích prvků, přičemž ovládače volby podlaží doplňuje detekcí jejich akčních částí. Dle definované úspěšné detekce v podkapitole 3.2 vykazuje metoda v použitém videozaznámu lokalizaci správných oblastí prvků u alespoň 85 % všech ovládačů volby podlaží a alespoň 75 % všech instancí ostatních tříd při zvoleném IoU = 0,5.

Obrázek 7.1: Dobré detekce a klasifikace kombinací ovládačů volby podlaží - část 1.

Metoda úspěšně detekovala a klasifikovala kombinace ovládačů, viz obrázek 7.1. Další úspěšné detekce jsou uvedeny v obrázku 7.2. Jednotlivé dílčí obrázky v plném rozlišení jsou k dispozici v digitální příloze.

U těchto obrazů kombinací ovládačů volby podlaží byla patrná dobrá kvalita a kontrast obrazu, doplněný vhodným osvětlením, které zajišťovalo jasné oddělení ovládačů od pozadí s dobře čitelnými označeními ovládačů volby podlaží. Tyto kombinace ovládačů byly podobné kombinacím uvedeným v datové sadě.

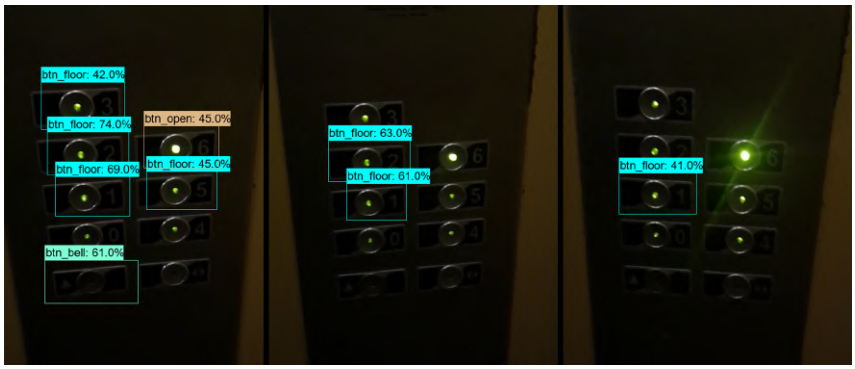
Obrázek 7.2: Dobré detekce kombinace ovládačů volby podlaží - část 2.

7.2 Omezení detekce kombinace ovládačů

Jak je patrné z obrázku 7.3, metoda přikládala detekovaným objektům v obraze nižší skóre jistoty čím více se zvětšoval úhel záběru obrazu kombinace ovládačů, až do ztráty instance ovládače volby podlaží. Toto omezení je možné kompenzovat vylepšením metody implementací transformace perspektivy, případně volbou adaptivního prahu eliminace prvků pro obtížné detekce.

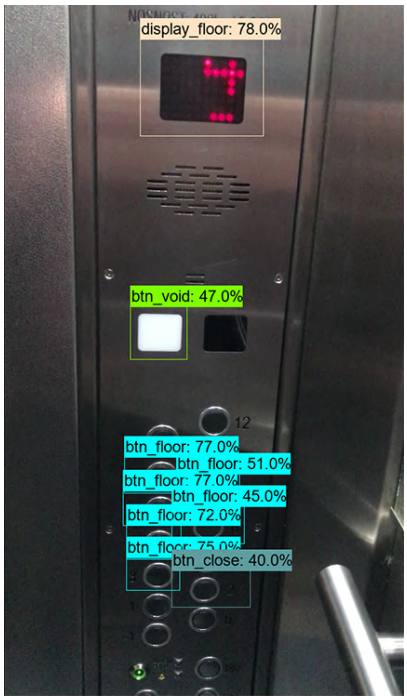
Obrázek 7.3: Ukázka omezení detekce kombinace ovládačů volby podlaží - perspektiva

Na obrázku 7.4 je poukázáno na nekvalitní vstupní obraz s nedostatečným kontrastem, kde řešením tohoto omezení by bylo použití jednolitého osvětlení s menší intenzitou pro zamezení silných odlesků v obraze.



Obrázek 7.4: Ukázka omezení detekce kombinace ovládačů volby podlaží - kontrast, osvětlení

Vhodnost použití metody na typech výtahových zařízeních, které nebyly zahrnuty v datové sadě, lze ověřit experimentem na daných obrazových datech. Příklad kombinace ovládačů volby podlaží, která nebyla součástí datové sady (ani nebyla podobného typu), a u které metoda nedokázala spolehlivě detekovat jednotlivé ovládací prvky je uvedena v obrázku 7.5



Obrázek 7.5: Ukázka omezení detekce kombinace ovládačů volby podlaží - neznámý typ kombinace ovládačů

7.3 Rychlost metody

Při nevyužití možnosti dedikované grafické karty na přenosném osobním počítači s procesorem Intel Core i5-5300U trvalo načtení detektoru a klasifikátoru celkem 20 sekund a vytěžovalo paměť na 1,5 GB, kdy samotná detekce ovládacích prvků trvala průměrně 3,42 sekundy a každá klasifikace tlačítka volby podlaží 0,189 sekund. Tyto časy načítání, detekce i klasifikace byly markantně (3–8x) kráceny při použití výkonné dedikované

grafické karty RTX 2070.

V předešlém řešení[13] byla využita předselekce významných oblastí dle vyhledaných hran v obrazu tak, aby detekce neuronovou sítí nemusela být prováděna na celé ploše vstupního obrazu. Od roku 2018 se také stávají dostupnějšími mikroprocesory zvané VPU[1], jenž jsou dedikované práci s algoritmy počítačového vidění a konvolučními neuronovými sítěmi. Dále je dostupná knihovna *TFLite*, jenž se zaměřuje na optimalizaci modelů na výkon pomocí jejich konverze, avšak pro modely sítí v TFOD2 není toto použití možné bez značných úprav architektur. Použití těchto možných vylepšení může přinést značné zrychlení detekce.

[1]Vision Processing Unit

8 Závěr

Práce se zabývala automatickou detekcí ovládacích prvků výtahů s použitím metod strojového učení, konkrétně použitím konvolučních neuronových sítí. Práce byla součástí celku zahrnujícího možnost použití metody na robotické platformě pro umožnění autonomní změny podlaží robotu, jenž disponuje autonomní navigací. Práce měla za cíl umožnit automatickou detekci ovládacích prvků výtahu z dodaných obrazových dat. Tento cíl je naplňován navrženým algoritmem, který implementoval schopnost rozpoznání ovládacích prvků výtahového prostředí, doplněn o vyhledávání akčních částí ovládačů a semantické porozumění kombinaci ovládačů volby podlaží.

První část práce se zabývala základy zpracování digitálního obrazu a možnostmi rozpoznávání objektů v obraze, přičemž obě témata byla vztažena k úloze detekce objektů v digitálním obraze. Na základě provedené rešerše předchozích řešení bylo rozhodnuto o použití konvolučních neuronových sítí pro úlohu automatické detekce ovládacích prvků výtahu. Dále se tato část zabývá principy konvolučních neuronových sítí ve vztahu k detekci objektů v digitálním obraze.

Obsáhlost řešeného problému vedla k upřesnění cílů v následující kapitole, kde byly přiblíženy restrikce úlohy, založené na získaných obrazových datech výtahových prostředí, definici rozpoznávaných ovládacích prvků a normě ČSN ISO 4190-5, která specifikovala požadavky na ovládací prvky výtahu.

Vhodnost použití metod strojového učení je velmi závislá na použité datové sadě, o které pojednává kapitola 4. Konkrétně se zabývala informacemi o sběru dat a typu pořízených záběrů s uvedeným vzorovým případem, načež bylo využito pořízených dat k anotaci. Byly definovány rozlišované prvky a úpravy datové sady na základě experimentů s neuronovými sítěmi. Samotná anotace probíhala v otevřeném softwaru *CVAT*, kde každému ovládacímu prvku v pořízených záznamech byl přiřazen ohraničující rámeček a třída. Dále byla datová sada doplněna hodnotami číslic vyskytujících se u ovládačů volby podlaží. Datová sada disponovala 32 916 anotovaných obrázků s celkem 131 803 označenými instancemi tříd uvedených v 4.3, načež byla exportována do standardizovaného typu *Pascal VOC*.

Pátá kapitola se zabývala vytvořením modelů detekce ovládacích prvků. Po probádání tohoto odvětví byl zvolen aplikační rámec *TensorFlow Object Detection API 2*, disponující vysokoúrovňovým přístupem k učení neuronových sítí a předtrénovanými modely na datové sadě *COCO*. Byla provedena informovaná selekce architektur neuronových sítí následovaná popisem práce se zvoleným aplikačním rámcem. Detektorem ovládacích prvků se stala síť architektury SSD ResNet50 V1 FPN 1024x1024(RetinaNet50), která byla doplněna klasifikátorem číslic ovládačů volby podlaží založené na architektuře EfficientNet-B0. Dále byly přiblíženy implementované skripty pro práci s datovou sadou a inferencí modelů.

V rámci tréninku a evaluace byla znázorněna optimalizace parametrů sítě, proces učení a evaluace vhodnosti modelu detekoru ovládacích prvků. Dále se kapitola zabývala tréninkem klasifikátoru číslic ovládačů volby podlaží. Bylo dosaženo mAP 96 % pro 0,5 IoU

detektoru na evaluačních datech a mAP 98 % klasifikátoru na tréninkových datech. Výstupy učení přinesly modely s dostatečnou přesností pro použití na definovanou úlohu automatické detekce ovládacích prvků výtahu.

Využití těchto modelů bylo rozebráno v následující kapitole, která uvedla aplikaci automatické detekce prvků, napsanou v jazyce Python formou objektově orientovaného programování. V první fázi algoritmus mohl načítat obrazová data z vícero zdrojů a zavrhovat rozmazané vstupní obrazy. Dále byl použit detektor pro extrakci instancí ovládacích prvků výtahu, načež byly detekované ovládače volby podlaží předloženy klasifikátoru, který jim přidělil číslici označení podlaží. Aplikace rozlišovala čtyři typy prostředí na základě detekcí a známé polohy pořízení záběru. Na základě tohoto prostředí byl volen odlišný způsob dalšího zpracování. Byly detekovány akční části ovládačů pro možnost stisku relevantních částí ovládačů a kombinace ovládačů volby podlaží byla zpracována algoritmem porozumění sekvenci ovládačů. Ten na základě definovaných předpokladů, pozicí a hodnot číslic detekovaných ovládačů volby podlaží dedukoval nejpravděpodobnější způsob řazení a rozložení ovládačů, načež se pokusil opravit chyby v číslování, pokud nějaké odhalil. Poslední část kapitoly implementovala vizualizaci dat aplikace a jejich fúzi na vstupní obraz.

Poslední část práce se věnovala ověření metody na vytvořené sekvenci obrazových dat s ověřením rychlosti a funkčnosti metody a jejích výstupů na běžném osobním počítači. Tato kapitola také poukazuje na možná vylepšení a úpravy metody pro zlepšení jejího chodu a funkcionality.

Zvolená metoda detekce splňuje požadavky na dostatečně přesnou a robustní metodu pro automatickou detekci ovládacích prvků výtahových zařízení pomocí zpracování digitálních obrazových dat. Na základě vytvořenené datové sady a implementované metody úspěšně detekuje ovládací prvky výtahových zařízení a splňuje zadané cíle práce.

Literatura

[1] THIPKHAM, Pitchaya. Image Processing Class #4 — Filters. Towards data science [online]. San Francisco: medium, 2018 [cit. 2021-04-30]. Dostupné z: https://medium.com/@boelsmaxence/introduction-to-image-processing-filters-179607f9824a

[2] KAUR, Dilpreet a Yadwinder KAUR. Various image segmentation techniques: a review. International Journal of Computer Science and Mobile Computing [online]. 2014, 3(5), 809–814 [cit. 2021-04-30]. ISSN 2320-088X. Dostupné z: https://www.ijcsmc.com

[3] ZHAO, Hengshuang, Jianping SHI, Xiaojuan QI, Xiaogang WANG a Jiaya JIA. Pyramid Scene Parsing Network [online]. arXiv, 2017 [cit. 2021-04-30]. Dostupné z: https://arxiv.org/abs/1612.01105v2

[4] MATCHA, Anil Chandra Naidu. A 2021 guide to Semantic Segmentation. Nanonets [online]. San Francisco, 2021 [cit. 2021-04-30]. Dostupné z: https://nanonets.com/blog/semantic-image-segmentation-2020/

[5] PERERA, Shehan Akalanka. A Comparison of SIFT , SURF and ORB. Medium [online]. San Francisco: Medium, 2018 [cit. 2021-04-30]. Dostupné z: https://medium.com/@shehan.a.perera/a-comparison-of-sift-surf-and-orb-333d64bcaaea

[6] VAHAB, Abdul, Maruti S NAIK, Prasanna G RAIKAR a Prasad S R. Applications of Object Detection System. International Research Journal of Engineering and Technology [online]. 2019, 6(4), 4186-4192 [cit. 2021-04-30]. ISSN 2395-0056. Dostupné z: https://www.irjet.net/archives/V6/i4/IRJET-V6I4920.pdf

[7] GANDHI, Rohith. Support Vector Machine — Introduction to Machine Learning Algorithms: SVM model from scratch. Towards data science [online]. San Francisco: Medium, 2018 [cit. 2021-04-30]. Dostupné z: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[8] KHANDELWAL, Renu. K-Nearest Neighbors(KNN). Data Driven Investor [online]. San Francisco: Medium, 2018 [cit. 2021-05-07]. Dostupné z: https://medium.datadriveninvestor.com/k-nearest-neighbors-knn-7b4bd0128da7

[9] JURAFSKY, Dan a James H. MARTIN. Speech and Language Processing: Appendix A: Hidden Markov Models. Stanford University [online]. Stanford, CA, 2020 [cit. 2021-05-01]. Dostupné z: https://web.stanford.edu/ jurafsky/slp3/A.pdf

[10] GUPTA, Prashant. Decision Trees in Machine Learning. Towards data science [online]. San Francisco: Medium, 2017 [cit. 2021-04-30]. Dostupné z: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

[11] MOURRI, Younes Bensouda, Kian KATANFOROOSH a Andrew NG. Neural Networks and Deep Learning. Coursera [online]. DeepLearning.AI [cit. 2021-05-07]. Dostupné z: https://www.coursera.org/learn/neural-networks-deep-learning

[12] KLINGBEIL, Ellen, Blake CARPENTER, Olga RUSSAKOVSKY a Andrew Y NG. Autonomous operation of novel elevators for robot navigation. 2010 IEEE International Conference on Robotics and Automation [online]. IEEE, 2010, , 751-758 [cit. 2021-05-01]. ISBN 978-1-4244-5038-1. Dostupné z: doi:10.1109/ROBOT.2010.5509466

[13] DONG, Zijian, Delong ZHU a Max Q.-H. MENG. An autonomous elevator button recognition system based on convolutional neural networks. 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO) [online]. IEEE, 2017, , 2533-2539 [cit. 2021-05-01]. ISBN 978-1-5386-3742-5. Dostupné z: doi:10.1109/ROBIO.2017.8324801

[14] JIANG, Shenlu, Wei YAO, Man-Sing WONG, Meng HANG, Zhonghua HONG, Eun-Jin KIM, Sung-Hyeon JOO a Tae-Yong KUC. Automatic Elevator Button Localization Using a Combined Detecting and Tracking Framework for Multi-Story Navigation. IEEE Access [online]. 2020, **8**, 1118-1134 [cit. 2021-05-01]. ISSN 2169-3536. Dostupné z: doi:10.1109/ACCESS.2019.2958092

[15] LIU, Jingya a Yingli TIAN. Recognizing Elevator Buttons and Labels for Blind Navigation. International Journal of Computer Applications [online]. 2020, **177**(35), 1-8 [cit. 2021-05-01]. ISSN 09758887. Dostupné z: doi:10.5120/ijca2020919835

[16] KARIM, Raimi. Illustrated: 10 CNN Architectures. Towards data science: Inside AI [online]. San Francisco: Medium, 2019 [cit. 2021-05-01]. Dostupné z: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e971

[17] HONG, Chenzimo. Deeplearning - Overview of Convolution Neural Network. Zybuluo [online]. 2018 [cit. 2021-05-01]. Dostupné z: https://www.zybuluo.com/hongchenzimo/note/1086311

[18] GONZALEZ, Rafael C. a Richard E. WOODS. Digital image processing. Fourth edition. New York: Pearson, 2018. ISBN 978-013-3356-724.

[19] MWITI, Derrick. A 2019 Guide to Object Detection. Heartbeat [online]. San Francisco: Medium, 2019 [cit. 2021-05-01]. Dostupné z: https://heartbeat.fritz.ai/a-2019-guide-to-object-detection-9509987954c3

[20] CS231n Convolutional Neural Networks for Visual Recognition. CS231n [online]. Stanford, CA [cit. 2021-05-01]. Dostupné z: https://cs231n.github.io/convolutional-networks/

[21] LAU, Suki. A Walkthrough of Convolutional Neural Network — Hyperparameter Tuning. Towards data science [online]. San Francisco: Medium, 2017 [cit. 2021-05-01]. Dostupné z: https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network-7f474f91d7bd

[22] STEWART, Matthew. Simple Guide to Hyperparameter Tuning in Neural Networks. Towards data science [online]. San Francisco: Medium, 2019 [cit. 2021-05-01]. Dostupné z: https://towardsdatascience.com/simple-guide-to-hyperparameter-tuning-in-neural-networks-3fe03dad8594

[23] NAUTIYAL, Dewang. Underfitting and Overfitting in Machine Learning. GeeksForGeeks [online]. 2020 [cit. 2021-05-01]. Dostupné z: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

[24] SHMUELI, Boaz. Multi-Class Metrics Made Simple, Part I: Precision and Recall. Towards data science [online]. San Francisco: Medium, 2019 [cit. 2021-05-02]. Dostupné z: https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2

[25] VARMA, Subir a Sanjiv DAS. Deep Learning. Sanjiv Ranjan Das blog [online]. 2018 [cit. 2021-05-02]. Dostupné z: srdas.github.io/DLBook/ImprovingModelGeneralization.html

[26] Detection Evaluation. COCO - Common Objects in Context [online]. [cit. 2021-05-02]. Dostupné z: https://cocodataset.org/#detection-eval

[27] MAVUDURU, Amol. Which deep learning framework is the best?: An in-depth comparison of Keras, PyTorch, and several others. Towards data science [online]. San Francisco: Medium, 2020 [cit. 2021-05-02]. Dostupné z: https://towardsdatascience.com/which-deep-learning-framework-is-the-best-eb51431c39a

[28] ČSN ISO 4190-5. Zřizování výtahů - Část 5: Ovládací prvky, signalizace a další příslušenství. Z1. Praha: Úřad pro technickou normalizaci, metrologii a státní zkušebnictví, 1992.

[29] KHANDELWAL, Renu. COCO and Pascal VOC data format for Object detection. Towards data science [online]. San Francisco: Medium, 2019 [cit. 2021-05-02]. Dostupné z: https://towardsdatascience.com/coco-data-format-for-object-detection-a4c5eaf518c5

[30] Computer Vision Annotation Tool (CVAT). GitHub - openvinotoolkit/cvat [online]. [cit. 2021-05-04]. Dostupné z: https://github.com/openvinotoolkit/cvat

[31] HAVELKA, Martin. Detekce aktuálního podlaží při jízdě výtahem. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce doc. Ing. Jiří Krejsa, Ph.D.

[32] Object Detection API with TensorFlow 2. GitHub - tensorflow/research [online]. [cit. 2021-05-04]. Dostupné z: https://github.com/tensorflow/models/ blob/master/research/object_detection/g3doc/tf2.md

[33] HUANG, Jonathan, Vivek RATHOD, Chen SUN et al. Speed/accuracy trade-offs for modern convolutional object detectors. ArXiv [online]. arXiv, 2016 [cit. 2021-05-04]. Dostupné z: https://arxiv.org/abs/1611.10012

[34] TSANG, Sik-Ho. Review: RetinaNet — Focal Loss (Object Detection). Toward data science [online]. San Francisco: Medium, 2019 [cit. 2021-05-07]. Dostupné z: https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4

[35] TSANG, Sik-Ho. Review: FPN — Feature Pyramid Network (Object Detection): Surpassing Single-Model Entries Including COCO Detection Challenges Winners, G-RMI and MultiPathNet. Toward data science [online]. San Francisco: Medium, 2019 [cit. 2021-05-04]. Dostupné z: https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610

[36] TSANG, Sik-Ho. Review: ResNet — Winner of ILSVRC 2015 (Image Classification, Localization, Detection). Toward data science [online]. San Francisco: Medium [cit. 2021-05-04]. Dostupné z: https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8

[37] A Beginners Guide To Tesseract OCR Using Pytesseract. Gitconnected [online]. San Francisco: Medium, 2020 [cit. 2021-05-04]. Dostupné z: https://levelup.gitconnected.com/a-beginners-guide-to-tesseract-ocr-using-pytesseract-23036f5b2211

[38] TAN, Mingxing. EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling. Google AI Blog [online]. 2019 [cit. 2021-05-04]. Dostupné z: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html

[39] ImageNet Large Scale Visual Recognition Challenge. ImageNet [online]. 2021 [cit. 2021-05-08]. Dostupné z: https://www.image-net.org

[40] KALINCHUK, Ivan. OCR with EfficientNet and PyTorch. Kaggle [online]. [cit. 2021-05-04]. Dostupné z: https://www.kaggle.com/ivankalinchuk/ocr-efficientdet-with-pytorch-part-1-3-training

[41] TFRecord and tf.train.Example. TensorFlow [online]. 2021 [cit. 2021-05-05]. Dostupné z: https://www.tensorflow.org/tutorials/load_data/tfrecord

[42] Protocol Buffers. Google Developers [online]. [cit. 2021-05-05]. Dostupné z: https://developers.google.com/protocol-buffers/

[43] TensorFlow 2 Detection Model Zoo. GitHub - tensorflow/models [online]. 2021 [cit. 2021-05-05]. Dostupné z: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

[44] HORÁK, Martin. Sémantický popis obrazovky embedded zařízení. Brno, 2020. Master's Thesis. Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Ing. Ilona Janáková, Ph.D.

[45] NumPy: The fundamental package for scientific computing with Python. NumPy [online]. 2005 [cit. 2021-05-10]. Dostupné z: https://numpy.org

[46] CLAHE Histogram Equalization - OpenCV. GeeksForGeeks [online]. 2020 [cit. 2021-05-10]. Dostupné z: https://www.geeksforgeeks.org/clahe-histogram-eqalization-opencv

[47] LEE, Socret. Lines Detection with Hough Transform: An algorithm to find lines in images. Toward data science [online]. San Francisco: Medium, 2020 [cit. 2021-05-10]. Dostupné z: https://towardsdatascience.com/lines-detection-with-hough-transform-84020b3b1549

Seznam zkratek a symbolů

IEEE Institute of Electrical and Electronics Engineers

GAN Generative adversarial network

SSD Single-shot detector

ROI Region of interest - oblast zájmu

ASM Attribute Selection Measure

IoU Intersection over union - průnik sjednocení

TP True positive

FP False positive

FN False negative

TN True negative

COCO Common objects in context

mAP Mean average precision

mAR Mean average recall

CVAT Computer Vision Annotation Tool

TFOD2 TensorFlow Object Detection API 2

CLAHE Contrast Limited Adaptive Histogram Equalization

NMS Non-maximum supression

Seznam obrázků

Seznam tabulek

Seznam příloh

A Mapa šťítků detekce ovládačů

```
item {  id: 1
        name: 'btn_void'}

item {  id: 2
        name: 'btn_floor'}

item {  id: 3
        name: 'btn_bell'}

item {  id: 4
        name: 'btn_call'}

item {  id: 5
        name: 'display_void'}

item {  id: 6
        name: 'display_floor'}

item {  id: 7
        name: 'panel_call_void'}

item {  id: 8
        name: 'panel_call_valid'}

item {  id: 9
        name: 'btn_open'}

item {  id: 10
        name: 'btn_close'}
```

B Příklad anotačního souboru XML

```
<annotation>
  <folder>frame</folder>
  <filename>0_frame_000006.jpg</filename>   # Název obrazu
  <source>
    <database>Unknown</database>
    <annotation>Unknown</annotation>
    <image>Unknown</image>
  </source>
  <size>                              ## Velikost obrazu
    <width>1920</width>
    <height>1080</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>                            ## Sekce detekovaného objektu
    <name>btn_floor</name>            # Jméno anotovaného objektu
    <occluded>0</occluded>
    <bndbox>                          # Ohraničující rámeček objektu
      <xmin>697.17</xmin>
      <ymin>314.8</ymin>
      <xmax>839.0</xmax>
      <ymax>421.79</ymax>
    </bndbox>
    <attributes>
      <attribute>
        <name>Value</name>
        <value>5.0</value>            # Hodnota číslice ovládače
      </attribute>
    </attributes>
  </object>
  <object>
  ...
  </object>
```

C Použitý soubor pipeline.config pro trénink detektoru

```
model {
  ssd {
    num_classes: 10
    image_resizer {
      fixed_shape_resizer {
        height: 1024
        width: 1024
      }
    }
    feature_extractor {
      type: "ssd_resnet50_v1_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 0.0004
          }
        }
        initializer {
          truncated_normal_initializer {
            mean: 0.0
            stddev: 0.03
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
      }
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
      }
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    box_predictor {
      weight_shared_convolutional_box_predictor {
        conv_hyperparams {
          regularizer {
            l2_regularizer {
              weight: 0.0004
```

```
67              }
68            }
69            initializer {
70              random_normal_initializer {
71                mean: 0.0
72                stddev: 0.01
73              }
74            }
75            activation: RELU_6
76            batch_norm {
77              decay: 0.997
78              scale: true
79              epsilon: 0.001
80            }
81          }
82          depth: 256
83          num_layers_before_predictor: 4
84          kernel_size: 3
85          class_prediction_bias_init: -4.6
86        }
87      }
88      anchor_generator {
89        multiscale_anchor_generator {
90          min_level: 3
91          max_level: 7
92          anchor_scale: 4.0
93          aspect_ratios: 1.0
94          aspect_ratios: 2.0
95          aspect_ratios: 0.5
96          scales_per_octave: 2
97        }
98      }
99      post_processing {
100       batch_non_max_suppression {
101         score_threshold: 1e-08
102         iou_threshold: 0.6
103         max_detections_per_class: 100
104         max_total_detections: 100
105         use_static_shapes: false
106       }
107       score_converter: SIGMOID
108     }
109     normalize_loss_by_num_matches: true
110     loss {
111       localization_loss {
112         weighted_smooth_l1 {
113         }
114       }
115       classification_loss {
116         weighted_sigmoid_focal {
117           gamma: 2.0
118           alpha: 0.25
119         }
120       }
121       classification_weight: 1.0
122       localization_weight: 1.0
123     }
124     encode_background_as_zeros: true
125     normalize_loc_loss_by_codesize: true
126     inplace_batchnorm_update: true
127     freeze_batchnorm: false
128   }
129 }
130 train_config {
131   batch_size: 2
132   data_augmentation_options {
133     random_image_scale {
134       min_scale_ratio: 0.8
135       max_scale_ratio: 1.5
136     }
137   }
138   sync_replicas: true
139   optimizer {
140     momentum_optimizer {
141       learning_rate {
142         cosine_decay_learning_rate {
143           learning_rate_base: 0.07
144           total_steps: 50000
145           warmup_learning_rate: 0.0122222
146           warmup_steps: 2000
147         }
148       }
149       momentum_optimizer_value: 0.9
```

```
150       }
151       use_moving_average: false
152     }
153     fine_tune_checkpoint: "<...>/model/ckpt-0"
154     num_steps: 50000
155     startup_delay_steps: 0.0
156     replicas_to_aggregate: 8
157     max_number_of_boxes: 100
158     unpad_groundtruth_tensors: false
159     fine_tune_checkpoint_type: "detection"
160     use_bfloat16: true
161     fine_tune_checkpoint_version: V2
162   }
163   train_input_reader {
164     label_map_path: "<...>/labelmap.pbtxt"
165     tf_record_input_reader {
166       input_path: "<...>/train.record-?????-of-00020"
167     }
168   }
169   eval_config {
170     metrics_set: "coco_detection_metrics"
171     use_moving_averages: false
172   }
173   eval_input_reader {
174     label_map_path: "<...>/labelmap.pbtxt"
175     shuffle: false
176     num_epochs: 1
177     tf_record_input_reader {
178       input_path: "<...>/eval.record-?????-of-00020"
179     }
180   }
```

D Tabulka záměn tříd klasifikátoru

Třída skutečná / Třída detekovaná	-1	-2	0	1	10	11	12	13	14	15	16	17	18	19	2	3	4	5	6	7	8	9
-1	3435	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
-2	0	1468	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	2275	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	7289	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0	0	0	0
10	0	0	0	0	2036	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	2036	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	1851	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	1676	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	1676	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	1676	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	2	1673	0	0	1	0	0	0	0	0	0	0	0
17	0	0	0	0	0	1	0	0	0	0	0	1675	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	1676	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	1184	0	0	0	0	0	0	0	0
2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	6770	0	0	0	0	0	0	0
3	0	0	0	2	0	0	0	0	0	0	0	0	0	0	1	6206	0	3	2	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4400	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4400	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3649	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3004	0	0
8	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	2034	0
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2035

Tabulka D.1: Tabulka záměn tříd klasifikátoru

E Algoritmus přeznačování štítků sekvence číslic ovládačů volby podlaží

```
"""Opraví sekvenci číslic ovládačů."""
seq_old = np.array(seq)
seq_is_correct = self.seq_is_correct.copy()
seq_numbers_corrected, seq_index_corrected = [], []
for i in range(len(seq) - 1):   # vynechání prvního a posledního členu, iterace ostatních
    seq_index = i
    if not seq_is_correct[seq_index]:
        found_valid_number = False
        valid_number_index = seq_index + 1

        while not found_valid_number:
            if valid_number_index >= len(seq_is_correct):
                print("Nelze nalézt číslici, sekvence neodpovídá své délce.")
                valid_number_index = valid_number_index - 1
                break
            if seq_is_correct[valid_number_index]:
                found_valid_number = True   # nalezena správná číslice
            elif not seq_is_correct[valid_number_index]:
                valid_number_index += 1     # inkrementace indexu objektu

        if (abs(seq[seq_index - 1] - seq[valid_number_index]))
        == (abs(valid_number_index - (seq_index - 1))):
            # TEST: rozdíl číslic je roven rozdílu pozic v sekvenci
            for i in range(seq_index, valid_number_index):
                number_changed = seq[i]
                # odvození nové číslice
                seq[i] = seq[seq_index - 1] + (seq_index - i) + 1
                seq_numbers_corrected.append(seq[i])
                seq_index_corrected.append(i)
                seq_is_correct[seq_index] = True
                print(f"Nahrazen index({i}) číslicí({number_changed}->{seq[i]})")

        elif (abs(seq[seq_index - 1] - seq[valid_number_index]))
        != (abs(valid_number_index - (seq_index - 1))):
            # TEST: rozdíl číslic není roven rozdílu pozic v sekvenci
            print("Detekováno přeskočení číslice.")
            jump_button = True
            pred = self.softmax_pred
```

```
            for proposal in range(3):
                # Nahrazení číslicí s nižším skórem jistoty
                if pred[seq_index][proposal]
                   == seq[valid_number_index] + (seq_index - valid_number_index)
                or pred[seq_index][proposal]
                     == seq[seq_index] - ((seq_index - 1) - seq_index):
                    proposal_rank = 1
                else:
                    proposal_rank = 0
                if proposal_rank == 1:
                    seq[i] = pred[seq_index][proposal]
                    print(f"Nahrazen index({i}) číslicí({seq[i]})")
                    break
# Vypsání provedených změn na sekvenci
print(f"Sekvence štítkyů{seq_old} \nopravena na sekvenci {np.array(seq)}")
```

F Digitální příloha