VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VYHLEDÁVÁNÍ TLAČÍTEK VE FORMULÁŘÍCH

BUTTON SEARCH IN FORMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **TOMÁŠ DYK**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Dr. Ing. PAVEL ZEMČÍK**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Dyk Tomáš, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Vyhledávání tlačítek ve formulářích**
Button Search in Forms
Kategorie: Zpracování obrazu
Zadání:

1. Prostudujte literaturu na téma detekce geometrických tvarů a textu se zaměřením na 2D struktury formulářů zobrazovaných na displejích počítačů.
2. Navrhněte postup detekce tlačítek a/nebo geometrických tvarů ve fotografiích formulářů snímaných kamerou nebo fotoaparátem z displeje počítače tak, aby bylo možné zjistit, kde jsou zejména tlačítka, a to kvůli testům zařízení
3. Navrhněte vhodný způsob implementace postupu a diskutujte jeho vlastnosti a možnosti. Současně navrhněte postup výběru nejvhodnějších fotografií tak, aby je bylo možno užívat v dokumentaci.
4. Implementujte postup a demonstrujte na vhodných příkladech funkčnost a vlastnosti implementovaného postupu.
5. Diskutujte dosažené výsledky a možnosti uplatnění a pokračování práce.

Literatura:

- Dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zemčík Pavel, prof. Dr. Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 3. června 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Diplomová práce popisuje systém pro automatickou detekci tlačítek z fotografie obrazovky. Systém je využit společně s robotickou rukou a kamerou, aby detekoval tlačítka na obrazovce vestavěných zařízení. Po detekci tlačítek na obrazovce algoritmus vydá příkaz robotickému rameni, aby klikl na pozici detekovaného tlačítka. Systematickým proklikáváním všech tlačítek algoritmus vytvoří navigační mapu, ve které jsou vidět všechny obrazovky daného zařízení. U každé obrazovky jsou uloženy pozice detekovaných tlačítek a cíl na jakou obrazovku odkazují.

Abstract

Diploma thesis describes a system for automatic detection of buttons from a screen photo. The system is used together with a robotic arm and a camera to detect buttons on the screens of embedded devices. After detecting buttons on the screen, the algorithm commands the robotic arm to click on the position of the detected button. The algorithm creates a navigation map in which all screens of the device are visible by systematically clicking through all buttons on all screens. For each screen, the positions of the detected buttons and the destination to which the screen points are stored.

Klíčová slova

Detekce, lokalizace, tlačítka, grafické rozhraní, počítačové vidění, neuronové sítě

Keywords

Detection, localization, buttons, graphical interface, computer vision, neural netwoks

Citace

DYK, Tomáš. *Vyhledávání tlačítek ve formulářích.* Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Dr. Ing. Pavel Zemčík

Vyhledávání tlačítek ve formulářích

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana profesora Pavla Zemčíka. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Dyk
3. června 2020

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce panu prof. Dr. Ing. Pavlu Zemčíkovi za jeho rady, podnětné připomínky a čas, který mi věnoval na konzultacích. Dále bych chtěl poděkovat panu Ing. Michalu Hradiši za jeho rady k vytvoření datasetu. V neposlední řadě bych chtěl poděkovat mé rodině za podporu při studiu a při psaní diplomové práce.

Obsah

Kapitola 1

Úvod

Při vývoji aplikací je podstatnou součástí i její testování. Díky němu je možné předejít problémům v reálném provozu aplikace. Testování však může být časově i finančně náročné. Projevuje se to hlavně u aplikací, které neumožňují automatické testování. Jedná se například o vestavěná zařízení, která je nutné manuálně testovat. Většina vestavěných zařízení totiž neposkytuje rozhraní, které by umožňovalo s tímto zařízením pracovat pomocí příkazů (softwarově).

Neustálý rozvoj nových technologií se projevil i v oblasti testování. Pro zařízení s dotykovou obrazovkou byl vyvinut systém, který umožní automatizaci jejich testování. Existující systém se v současné době skládá z robotického ramene a kamery, která snímá obrazovku testovaného zařízení. Robotické rameno se stylusem kliká na dotykový displej zařízení na zadané elementy. Očekávané chování aplikace se následně ověří pomocí kamery, která snímá obrazovku zařízení. Využitím umělé inteligence a zpracováním obrazu se následně ověří, že aplikace pracuje, jak se předpokládalo.

Nevýhodou tohoto systému je však potřeba manuálně definovat a nasnímat obrazovky grafického rozhraní nového zařízení a zaznačit pozice tlačítek na jednotlivých obrazovkách. Cílem této práce je tento krok zautomatizovat. Navržený systém analyzuje snímky pořízené kamerou, lokalizuje tlačítka na snímku obrazovky a následně zadá příkaz robotickému rameni, aby klikl na jedno z nalezených tlačítek. Na nově zobrazené obrazovce tyto kroky zopakuje. Výsledkem algoritmu tedy budou získané snímky ze všech obrazovek, na které se lze v zařízení proklikat. Ke každé obrazovce budou uloženy pozice všech tlačítek, které se na ní nachází a na jakou obrazovku dané tlačítko odkazuje. Na základě získaných dat algoritmus vytvoří „navigační mapu“ (orientovaný graf) zkoumaného zařízení.

V první části práce jsou uvedeny teoretické informace o využívaných algoritmech a o principu jejich fungování. V kapitole 2 jsou popsány některé algoritmy, které lze použít pro zjištění, zda se jedná o již analyzované snímky obrazovky. Kapitola 3 popisuje způsoby, jak detekovat a lokalizovat tlačítka na snímku obrazovky. Kapitola 4 uvádí některé z používaných algoritmů pro prohledávání stavového prostoru. Jeden z nich bude následně využit pro systematické proklikávání grafického rozhraní a hledání nových obrazovek (stavů). V kapitole 5 jsou popsány vlastnosti rozšíření pro internetový prohlížeč Google Chrome a jak se pro tento prohlížeč vyvíjí rozšíření.

V druhé části práce je popsán samotný návrh (6) a implementace (7) popisovaného algoritmu společně s dosaženými výsledky.

Kapitola 2

Podobnost fotografií

Tato kapitola popisuje metody, které lze využít pro určení podobnosti dvou fotografií. Nejsou zde uvedeny všechny metody, ale pouze metody, které byly využity v mé práci. Metody lze rozdělit do dvou skupin. Metody SIFT a SURF jsou založeny na vyhledání deskriptorů na prvním snímku a nalezení jejich korespondujících deskriptorů na druhém snímku. Na základě nalezeného množství korespondujících deskriptorů a jejich vzájemných vzdáleností je určena podobnost mezi těmito snímky. Dále lze tyto metody použít pro detekci a lokalizaci objektů nebo při vytváření panoramatických fotografií z několika fotek. Do druhé skupiny patří metody SSIM a PSNR. Výsledkem těchto metod je přímo poměr udávající podobnost fotografií. Tyto metriky jsou vypočítány operacemi mezi fotografiemi, tedy například sčítáním nebo odčítáním jednotlivých pixelů na fotografií, na které byli aplikovány různé filtry. Tento typ metod se nejčastěji používá pro porovnání komprimačních metod fotografií.

2.1 Metoda scale invariant feature transform

Metoda Scale Invariant Feature Transform (SIFT) slouží pro nalezení význačných bodů a výpočtu deskriptorů. Dříve publikované metody pro výpočet deskriptorů nezvládali pracovat s klíčovými body v různých měřítkách. Tento nedostatek zvládla vyřešit právě metoda SIFT. Každý z nalezených význačných bodů je invariantní vůči změně velikosti, rotaci, translaci a je částečně invariantní proti změně jasu.

SIFT využívá specifický typ obrazové pyramidy. Obrazová pyramida je způsob reprezentace fotografie v různých rozlišeních. Skládá se z několika vrstev, kde každá vrstva pyramidy odpovídá jednomu rozlišení fotografie. Pyramida je vytvořena tak, že je fotografie opakovaně rozostřena a zmenšena. Metoda SIFT využívá pro rozostření gaussův filtr a jednotlivé vrstvy pyramidy počítá jako rozdíly dvou různě rozostřených fotografií. Tímto výpočtem se získá rozdíl takzvaných *Difference of Gaussians*, ze kterých se skládá pyramida pro metodu SIFT.

Lokalizace klíčových bodů spočívá v porovnání každého pixelu fotografie s osmi pixely z jeho okolí. Jestliže má porovnávaný pixel nejvyšší nebo nejnižší hodnotu, porovná se s pixely ve vyšší a nižší vrstvě pyramidy. Jestliže i v těchto úrovních je minimální nebo maximální hodnota, je tento bod považován za klíčový. Deskriptor pro konkrétní klíčový bod se vypočítá z okolí bodu, na základě histogramu orientovaných gradientů.

Každá fotografie je popsána několika deskriptory. U fotografie o velikosti 512x512 pixelů je nalezeno přibližně 1000 deskriptorů. Porovnávání deskriptorů ze dvou různých fotografií

a hledání odpovídajících deskriptorů je komplikované, jelikož je každý reprezentován více rozměrným vektorem. SIFT k tomuto problému využívá prohledávací metodu *best-bin-first*. Jedná se o aproximační algoritmus, který hledá předem daný počet nejpodobnějších bodů ve více rozměrném prostoru. Je založen na prohledávací metodě *k-d tree*, která umožňuje popisování bodů v k-rozměrném prostoru. Metoda SIFT přiřadí deskriptorům pro větší měřítka dvojnásobnou váhu než deskriptorům z vrstev pyramidy s menším měřítkem. Díky tomu je prohledávání ještě efektivnější.[7]

2.2 Metoda speeded-up robust features

Speeded-Up Robust Features, zkráceně SURF, je také metoda pro nalezení význačných bodů a výpočtu deskriptorů fotografie. Metoda je invariantí vůči rotaci a změně velikosti. Detekce význačných bodů je založena na využití Hessovy matice a integrálního obrazu. Integrální obraz je specifický způsob reprezentace obrazu. Hodnota každého pixelu je rovna součtu hodnot všech předchozích pixelů ve vertikálním a horizontálním směru k bodu se souřadnicemi (0,0), který se nachází v levém horním rohu. Využitím této reprezentace dochází k velkému snížení doby výpočtu, jelikož je potřeba pouze tří operací pro vypočítání součtu intenzit pixelů uvnitř libovolně velkého obdélníku na fotografii. Snížení času výpočtu je nejvíce patrné při počítání s konvolučními filtry velkých rozměrů.

Hessova matice je využita z důvodu velké přesnosti. SURF detekuje body na místech, kde dosahuje hodnota determinantu svého maxima. Hodnota determinantu je využita i pro určení měřítka. Hessova matice má tvar

$$\mathcal{H}(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \tag{2.1}$$

kde x je daný bod, σ je měřítko, $L_{xx}(x,\sigma)$ je konvoluce druhé derivace Gausovy funkce $\frac{\partial^2}{\partial x^2}$ nad obrazem I v bodě x a obdobně pro $L_{xy}(x,\sigma)$ a $L_{yy}(x,\sigma)$. Druhé derivace Gausovy funkce jsou následně ořezány a diskretizovány, čímž se zrychlí výpočet. Diskretizace a ořezání se však projeví v horších výsledcích u obrazu, který je rotován po lichých násobcích $\frac{\pi}{4}$. Tento jev se projevuje u všech detektorů založených na Hessově matici.

Hledání korespondujících význačných bodů na dvou rozdílných fotografiích je obtížné, jelikož není zaručeno, že jsou objekty a jejich příznaky foceny ve stejném měřítku. Z tohoto důvodu je nutné pracovat s více měřítky. Většina metod pracuje s obrazovými pyramidami (jako například metoda Scale Invariant Feature Transform viz 2.1). Díky využití integrálního obrazu není nutné opakovaně aplikovat filtry na předchozí vypočítanou vrstvu, ale stačí přímo aplikovat takzvaný *box filter* o potřebné velikosti.

Výpočet deskriptoru je rozdělen do dvou fází. V první fázi se vypočítá orientace význačných bodů. Směr je odvozen z hodnoty odezvy Haarových vlnek z horizontálního a vertikálního směru kruhového okolí bodu. Velikost kruhového okolí je závislá na detekovaném měřítku. Vypočítané odezvy Haarových vlnek jsou reprezentovány jako body, kde souřadnice x odpovídá odezvě z horizontálního směru a souřadnice y vertikálnímu směru. Z těchto bodů je následně vypočítán dominantní směr daného význačného bodu. Ve druhé fázi se kolem význačného bodu vytvoří čtverec natočený dle vypočítaného dominantního směru a o velikosti zjištěného měřítka. Čtverec se rozdělí na 4x4 menších čtverců. Pro každý z nich se vypočítají odezvy Haarových vlnek v obou směrech (směr se vztahuje k natočení čtverce) a odezvy s absolutními hodnotami. Každá podoblast je popsána čtyřrozměrným vektorem. Výsledný deskriptor každého význačného bodu je tedy popsán vektorem o velikosti 64. Hod-

noty z Hessovy matice, které byly již dříve vypočítány při lokalizaci deskriptorů, se využijí i pro nalezení korespondujících význačných bodů. [2]

2.3 Metrika vycházející z poměru maximální hodnoty signálu a šumu

Peak Signal-to-Noise Ratio (PSNR) udává poměr mezi maximální hodnotou signálu a šumu. PSNR se vyjadřuje

$$\begin{aligned} MSE &= \frac{1}{m*n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j)-K(i,j)]^2 \\ PSNR &= 10*\log_{10}\frac{MAX_I^2}{MSE} \end{aligned} \tag{2.2}$$

kde I je originální fotografie o rozměru $m*n$, K je zašuměný obraz (porovnávaný obraz), MAX_I je maximální možná hodnota pixelu ve fotografii I. Při počítání MSE barevného obrázku se vypočítá MSE nad jednotlivými kanály pixelu, které jsou následně zprůměrovány.

Z rovnice vyplývá, že v případě porovnání dvou identických fotografií bude hodnota MSE rovna nule. V tomto případě by bylo $PSNR$ nedefinované. Tato metrika bere v potaz pouze numerické porovnávání a nebere v potaz strukturální informace ve fotografii.[8]

2.4 Index strukturální podobnosti

Index strukturální podobnosti je v angličtině známý pod pojmem Structural SIMilarity index (SSIM). Pomocí něj lze vypočítat objektivní ohodnocení kvality dvou fotografií, kde je jedna fotografie referenční (neupravený originál) a druhá fotografie je zkomprimovaná nebo upravená metodami pro zpracování obrazu.

SSIM lze využít pro dynamickou kalibraci obrazu při streamování nebo pro porovnání kvality kompresních metod.

Metrika SSIM se při porovnávání kvality zaměřuje na podobné věci jako člověk. Na rozdíl od metod MME nebo PSNR, se ve výpočtu mimo jiné zaměřuje na strukturální informace v obraze a tyto informace porovnává s druhou fotografií. Další složky, které jsou při výpočtu podobnosti použity, jsou kontrast a jas.

Na fotografii se nejdříve vypočítá hodnota μ_x na základě vzorce 2.3. Hodnota μ_x reprezentuje průměrnou úroveň jasu fotografie x. Tato hodnota je následně odečtena od každého pixelu a vypočítá se kontrast fotografie dle vzorce 2.4. Strukturální informace je následně reprezentována hodnotou s_x, která se vypočítá na základě vzorce 2.5.

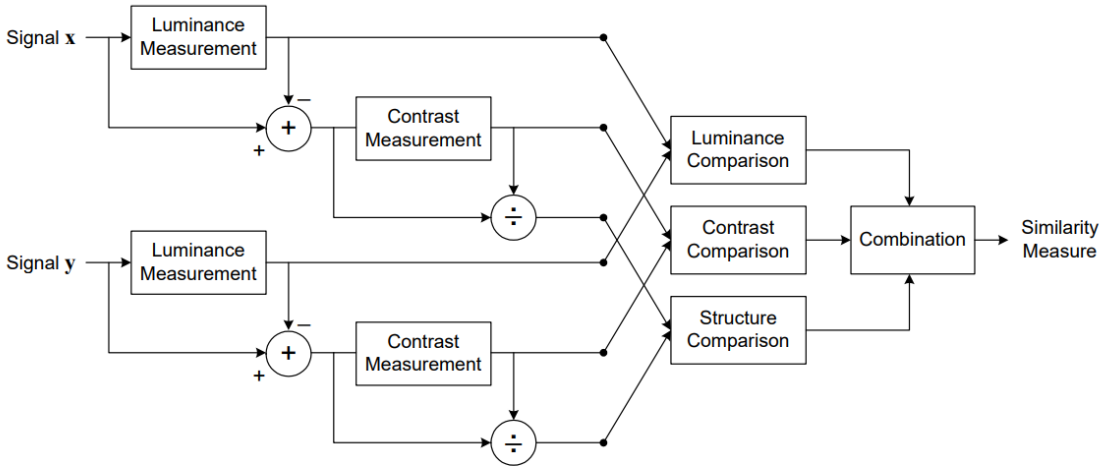
$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.3}$$

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\mu_x)^2\right)^{1/2} \tag{2.4}$$

$$s_x = \frac{x_i-\mu_x}{\sigma_x} \tag{2.5}$$

Stejné operace se provedou i u druhé fotografie a jednotlivé hodnoty jasu, kontrastu a strukturálních informací se aplikují podle vzorce 2.6, kde C_1 a C_2 jsou konstanty pro zamezení nestabilních výsledků v případech, kdy se $\mu_x^2 + \mu_y^2$ nebo $\sigma_x^2 + \sigma_y^2$ blíží nule. Postup porovnávání je názorně vidět na obrázku 2.1. [15]

$$\begin{aligned} \sigma_{xy} &= \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x) * (y_i - \mu_y) \\ SSIM(x,y) &= \frac{(2\mu_x\mu_y + C_1) * (2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1) * (\sigma_x^2 + \sigma_y^2 + C_2)} \end{aligned} \tag{2.6}$$



Obrázek 2.1: Schéma výpočtu metriky SSIM dvou fotografií[1]

[1]Obrázek byl převzat z [15].

Kapitola 3

Detekce tlačítek pomocí neuronové sítě

Pro detekci a lokalizaci objektů se nejčastěji používají neuronové sítě. Neuronové sítě umožňují lepší zobecnění naučených vzorů a díky tomu dosahují lepších výsledků. Nově navržené metody a modely neuronových sítí se testují na obecných datasetech. Mezi nejpoužívanější datasety patří COCO[1] a Pascal VOC[2]. Jako hlavní metrika pro určení přesnosti sítě se používá mean average precision (`mAP`). Tato metrika udává průměrnou hodnotu `AP` pro jednotlivé třídy v datasetu. Určuje, jak přesně síť určila pozici objektu a také v kolika případech síť objekt vůbec nedetekovala.

V této sekci jsou popsány základní informace o používaných modelech a jejich principu fungování. Jsou zde uvedeny i základní modely neuronových sítí, ze kterých vychází většina novějších architektur. Je zde popsán i jeden z nejnovějších modelů od výzkumníků z Google – EfficientDet. Pro detekci tlačítek lze použít i metody založené na deskriptorech jakou jsou například SIFT a SURF, které jsou již popsány v sekci 2. Vypočítané deskriptory se například použijí jako vstup do natrénovaného modelu, který určí pravděpodobné pozice hledaných objektů.

3.1 Neuronové sítě

Neuronové sítě jsou inspirovány nervovou soustavou živých organismů, které se skládají ze vzájemně propojených nervových buněk. Tyto nervové buňky se nazývají neurony. Existuje několik modelů umělého neuronu. Obecně lze model neuronu zapsat rovnicí 3.1, která spočívá v sečtení všech vstupních hodnot x_i vynásobených jejich příslušnými vahami w_i. Následně je k součtu přičten θ práh neuronu a výsledná hodnota je použita jako parametr aktivační funkce $f(x)$, která určí výstupní hodnotu y. Modely se liší využitím odlišných aktivačních funkcí. Nejčastěji se používá sigmoidální, skoková, hyperbolická nebo radiální aktivační funkce.

$$y = f(\sum_{i=1}^{N}(w_i * x_i) + \theta) \tag{3.1}$$

Shlukováním neuronů do vrstev a jejich propojováním mezi vrstvami se vytvářejí komplexnější modely, které se nazývají umělé neuronové sítě. Typicky lze takové sítě rozdělit na

[1] http://cocodataset.org/
[2] http://host.robots.ox.ac.uk/pascal/VOC/

vstupní, skrytou a výstupní část. Vstupní a výstupní část je reprezentována jednou vrstvou neuronů. Skrytá část může obsahovat několik vrstev. U počátečních vrstev lze pozorovat shluky neuronů, jejichž funkce se podobá hranovým detektorům určitého směru. Data z dalších vrstev skryté části neuronové sítě však bývají pro člověka neinterpretovatelné.

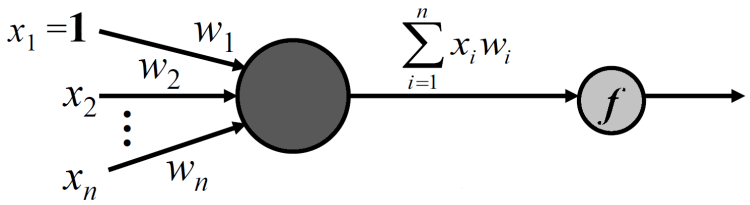
Umělé neuronové sítě pracují na principu dopředného šíření informací. Neuronům ve vstupní vrstvě se předají vstupní data. Podle rovnice 3.1 se vypočítají výstupní hodnoty všech neuronů z této vrstvy. Tato data dále postupují do následující vrstvy, ve které se tento proces opakuje, dokud data neprojdou všemi vrstvami sítě. Jednotlivé vrstvy neuronů jsou mezi sebou propojeny tak, že každý neuron z jedné vrstvy je spojen se všemi neurony následující vrstvy. Tyto propojení lze chápat jako synapse spojující neurony v mozku. Každé spojení mezi neurony má vlastní váhu, která se upravuje při učení neuronové sítě.

Neuronové sítě se trénují na datasetu. Dataset je množina dvojic vstupních dat a očekávaných výsledků. Dataset by měl být rozdělen do dvou části – trénovací a testovací část. Neuronová síť je učena na trénovací části datasetu a po každém cyklu učení se zkontroluje její úspěšnost na testovací části datasetu. Tímto způsobem se zvětší schopnost generalizace a zamezí se přetrénování sítě. Přetrénování sítě znamená, že síť dosahuje perfektních výsledků na trénovacím datasetu, ale u vstupů, se kterými se nesetkala, má špatné výsledky.

Při učení neuronové sítě je na vstupní vrstvu přiveden jeden ze vstupů z trénovací části datasetu. Síť vypočítá výstupní hodnotu, která je porovnána s očekávaným výsledkem. Jestliže se výsledky neshodují, jsou pomocí metody backpropagation upraveny váhy mezi jednotlivými neurony tak, aby pro daný podnět síť produkovala požadovaný výsledek. Váhy neuronů jsou upravovány pouze v určitém měřítku.

3.2 Perceptron

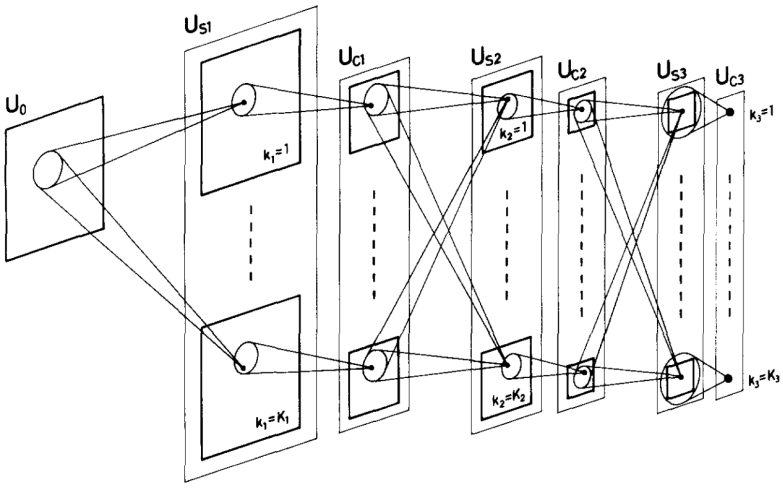
Perceptron je základní reprezentace neuronu. Byl vymyšlen v roce 1957 Frankem Rosenblattem. Pomocí tohoto modelu, kde je použit jeden neuron, lze vyřešit pouze lineárně separabilní problémy. Schéma modelu je znázorněno na obrázku 3.1. Proměnné x_2 až x_n reprezentují vstupní hodnoty pro neuron, w_1 až w_n jsou váhy pro konkrétní vstup, y je výstupní hodnota perceptronu a $f(x)$ je sigmoidální aktivační funkce. Při učení perceptronu jsou upravovány jednotlivé váhy w_1 až w_n tak, aby výstup perceptronu y byl roven požadovanému výstupu. Chování perceptronu lze zapsat rovnicí 3.1, která spočívá v sečtení všech vstupních hodnot vynásobených jejich příslušnými váhami. Následně je použit vážený součet jako parametr aktivační funkce, která určí výstupní hodnotu y. [16]



Obrázek 3.1: Model perceptronu

3.3 Konvoluční neuronové sítě

Konvoluční neuronové vrstvy byly také inspirovány mozkem živočichů, konkrétně propojením neuronů ve zrakové kůře mozku. Na základě výzkumu v této oblasti v roce 1980 vydal Kunihiko Fukushima článek[3], ve kterém představil model pojmenovaný neocognitron. Model se skládá ze dvou různých typů buněk uspořádaných do vrstev – C-buňky a S-buňky. Vzájemné propojení jednotlivých vrstev neocognitronu je znázorněno na obrázku 3.2. S-buňky slouží pro extrakci příznaků, jako jsou například hrany nebo rohy objektů v určitém směru. Ve vyšších vrstvách se extrahují rozsáhlejší příznaky. C-buňky slouží pro korekci pozic. C-buňky získají příznaky z předchozích S-buněk, a i když bude příznak na rozdílných pozicích (z rozdílných výstupů S-buněk), výstup C-buňky bude i tak stejně aktivní. Každá vrstva je ještě rozdělena do dvojdimenzionálních podvrstev.



Obrázek 3.2: Schéma propojení vrstev v neocognitronu[3]

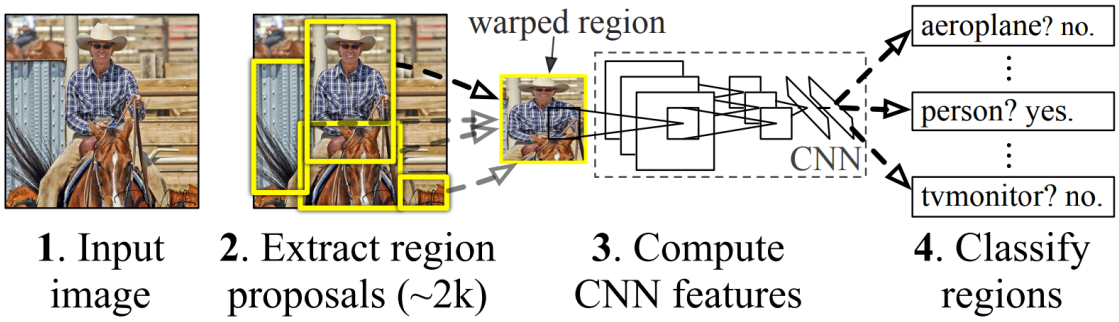
Z modelu neocognitron později vznikly konvoluční neuronové sítě, které jsou dnes nejpoužívanější při práci s obrazem. Tento typ sítě obsahuje alespoň jednu konvoluční vrstvu. Každý neuron z této vrstvy bere vstup pouze z malého okolí předchozí vrstvy. Stejně jako u neocognitronu se v prvních vrstvách neuronové sítě extrahují základní příznaky hran nebo rohů na malých oblastech. Následně se získaná data podvzorkují a postupují do dalších konvolučních vrstev. Další konvoluční vrstvy berou v potaz větší oblasti fotografie a pracují nad příznaky z větších oblastí a jsou schopny reprezentovat komplexnější informace.

3.4 Architektura neuronové sítě Region-based Convolutional Network

Chování sítě **Region-based Convolutional Network** (R-CNN) lze rozdělit do tří částí. V první části sítě se vygenerují možné oblasti, na kterých by se mohl vyskytovat některý z hledaných objektů. Síť vygeneruje přibližně 2000 regionů na zadané fotografii. Jednotlivé regiony se transformují do čtvercového tvaru. Ve druhé části se pomocí konvolučních neuronových sítí vyextrahují příznaky dané oblasti o předem specifikované velikosti. Ve třetí části se pomocí metody podpůrných vektorů určí odpovídající třída objektu v nalezeném

[3]Obrázek byl převzat z [3]

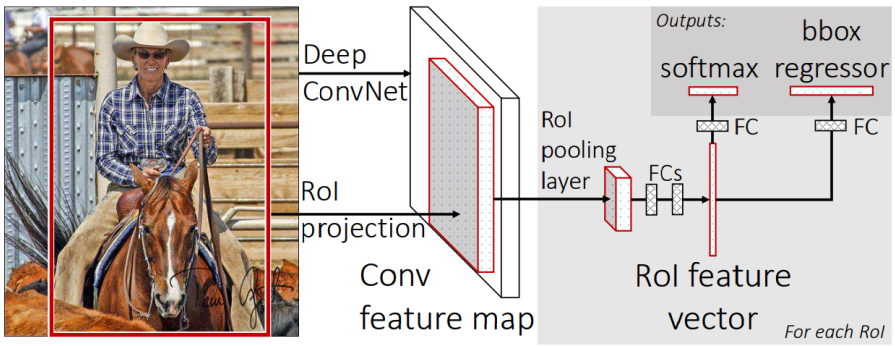
regionu, případně se rozhodne, že se v dané oblasti žádný z hledaných objektů nenachází. Její schéma je vidět na obrázku 3.3.



Obrázek 3.3: Schéma procesu detekce objektů v neuronové síti R-CNN[4]

Tato síť dosahuje dobrých výsledku při lokalizaci a určení třídy objektu. Její velkou nevýhodou je její nízká rychlost při učení i při predikci pozice. [5]

Stejným autorem byla v roce 2015 představena architektura **Fast Region-based Convolutional Network** (Fast R-CNN), která tento nedostatek vylepšuje. V modelu Fast R-CNN je fotografie přivedena na vstup konvolučních sítí, který vytvoří mapu příznaků. Následně je pro každý návrh regionu extrahován vektor příznaků o předem dané velikosti. Získané vektory příznaků se předají plně propojeným vrstvám. Tyto vrstvy pomocí funkce softmax určí třídu objektu a ohraničující box. Přesné schéma fungování sítě lze vidět na obrázku 3.4.



Obrázek 3.4: Architektura neuronové sítě Fast R-CNN[5]

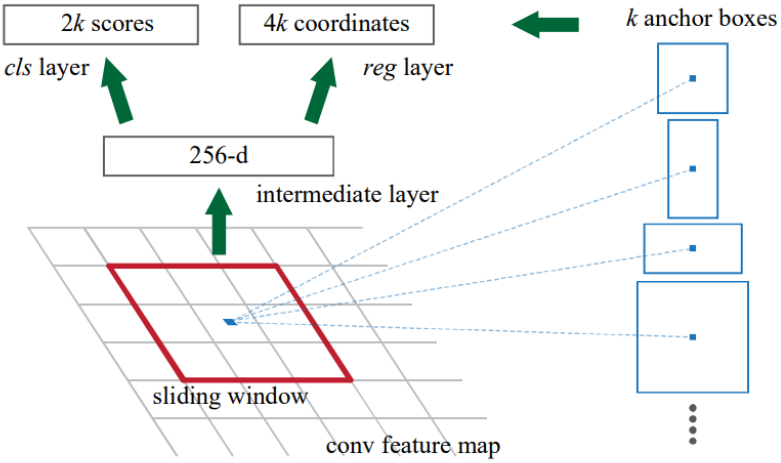
Jak vyplývá z předchozího odstavce, hlavní rozdíl oproti původní verzi R-CNN je změna pořadí při extrakci příznaku. V této síti se nejprve provede extrakce příznaků pro celou fotografii. Takto vypočítané příznaky se pak analyzují v určitých oblastech. Díky tomuto způsobu síť dosahuje mnohonásobně rychlejšího trénování i predikci. [4]

Nejnovější architektura nese označení **Faster R-CNN**. V této verzi byl vylepšen způsob návrhu kandidátních oblastí. Předchozí modely R-CNN používali k tomuto účelu selektivní vyhledávací algoritmus. V modelu Faster R-CNN je místo něj použita speciální plně propojená konvoluční neuronová síť – region proposal network (RPN). Síť RPN pracuje hlavně

[4]Obrázek byl převzat z [5].
[5]Obrázek byl převzat z [4].

na grafické kartě a umožňuje sdílení vah pro celou fotografii. Díky využití této sítě se několikanásobně snížil čas potřebný pro detekci možných pozic.

Faster R-CNN využívá dva moduly. První modul obsahuje plně propojenou neuronovou síť pro navrhování možných oblastí (RPN). Region proposal network funguje na principu posuvného okna. Aplikuje se na vyextrahovanou mapu příznaků z konvolučních vrstev. Část obrazu pod posuvným oknem je předána dvěma plně propojeným konvolučním vrstvám pro klasifikaci a predikci boxu. Výstupem RPN sítě je sada pravděpodobných obdélníkových regionů, na kterých se vyskytuje objekt, a hodnota určující, že je v dané oblasti opravdu jeden z hledaných objektů. Schéma tohoto modulu je vidět na obrázku 3.5. Druhý modul obsahuje upravenou síť Fast R-CNN, která je popsána dříve v této sekci. Úprava spočívá pouze v přijímaném vstupu, jelikož v této verzi jsou na vstup předány již navržené regiony. Konvoluční vrstvy pro extrakci mapy příznaků jsou pro oba moduly společné, ale moduly jsou trénovány odděleně.



Obrázek 3.5: Architektura sítě Region proposal network (RPN) použité v neuronové sítí Faster R-CNN pro nalezení pravděpodobných pozic hledaných objektů[6]

Síť Faster R-CNN je trénována pomocí specifického trénovacího algoritmu, který umožňuje trénování společných konvolučních vrstev odděleně. Algoritmus se skládá ze 4 kroků. V prvním kroku je natrénována samotná RPN síť. Ve druhém kroku se natrénuje druhý modul – síť Fast R-CNN, která využívá návrh regionů ze sítě RPN z prvního kroku algoritmu. V dalším kroku se dotrénuje RPN síť s již pozměněnými konvolučními vrstvami, které se již nemění. Poslední krok algoritmu spočívá v dotrénování sítě v druhém modulu.

3.5 Architektura neuronové sítě You Only Look Once

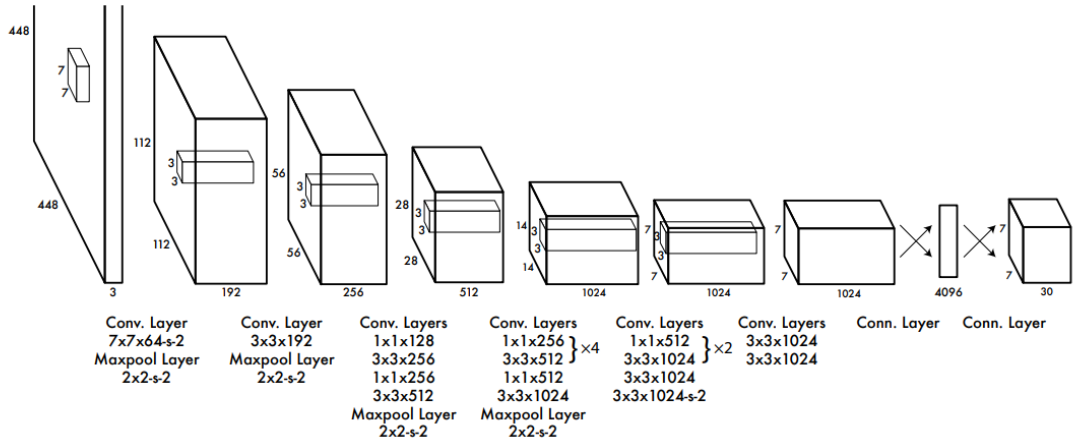
You only look once, zkráceně YOLO, využívá jedinou konvoluční neuronovou síť, která vyhledá a zároveň klasifikuje všechny druhy objektů na fotografii. YOLO analyzuje fotografii jako celek na rozdíl od techniky posuvného okna nebo technikám založených na regionálních návrzích. Tímto způsobem nedochází k chybám u pozadí, jelikož nahlíží na obraz ve větším kontextu a bere v potaz příznaky z celého obrázku. Síť YOLO dokáže lépe generalizovat z naučených dat. Při testování, kde síť byla natrénována na fotografiích a následně testována

[6]Obrázek byl převzat z [12].

na uměleckých dílech, dosáhlo YOLO podstatně lepších výsledků než R-CNN. Nevýhodou této sítě je však menší přesnost v lokalizaci objektu. Nejvíce se nepřesnosti projevují u malých objektů, které jsou blízko u sebe. Dále síť YOLO špatně detekuje objekty, u kterých se změní poměr stran.

Síť YOLO rozdělí vstupní fotografii do mřížky. Každá buňka mřížky odhadne několik ohraničujících boxů a pro každý z nich i skóre udávající, jestli se v boxech vyskytuje některý z objektů a také jak je pravděpodobně přesný predikovaný box. Každá buňka z mřížky obsahuje pět predikcí – x, y, w, h, c. Souřadnice x, y reprezentují střed ohraničujících boxů, w a h určují šířku a výšku boxu a c určuje, jak moc si je síť jistá predikcí, že se v navrženém boxu skutečně vyskytuje objekt.

Architektura sítě YOLO je inspirována architekturou GoogLeNet, která se využívá pro klasifikaci obrázků. Vstupní obrázek nejprve projde přes 24 konvolučních vrstev, po kterých následují dvě plně propojené vrstvy, následně pak redukční a konvoluční vrstva. Celé schéma je znázorněno na obrázku 3.6. Konvoluční vrstvy vyextrahují příznaky z fotografie. Tyto příznaky se pak využijí v plně propojených vrstvách pro určení pravděpodobných míst, na kterých se nachází hledané objekty. [9]



Obrázek 3.6: Architektura neuronové sítě YOLO[7]

V roce 2016 byla navržena vylepšená síť pojmenovaná YOLOv2 a systém YOLO9000, který zvládá detekovat přes 9000 různých objektů v reálném čase. Autoři se snažili zachovat stejnou hloubku sítě jako u původního modelu YOLO, aby se zachovala velká rychlost při predikci. V základní verzi sítě také změnili rozlišení vstupní fotografie z původních 448x448 na 416x416. Síť byla vylepšena přidáním dávkové normalizace pro všechny konvoluční vrstvy, použitím většího rozlišení klasifikátoru (zvětšení na 448x448 oproti původním 224x224). Dále byly odstraněny plně propojené vrstvy a místo nich se pro predikci ohraničujících boxů využívají takzvané `anchor boxes`. Díky tomu se zvýšil počet predikcí boxů z původních 98 na více než 1000 ohraničujících boxů na fotografii. YOLOv2 přináší podstatné vylepšení původního modelu a vylepšuje jeho přesnost. [10]

Další verze sítě byla představena v roce 2018. Model YOLOv3 přináší pouze menší vylepšení oproti modelu YOLOv2. Objekty jsou predikovány na základě binární křížové entropie (binary cross-entropy loss) oproti původní funkci softmax. Tato změna umožňuje lépe určit třídu u objektů, které mohou spadat i do jiných tříd (například třída člověk

[7]Obrázek byl převzat z [9].

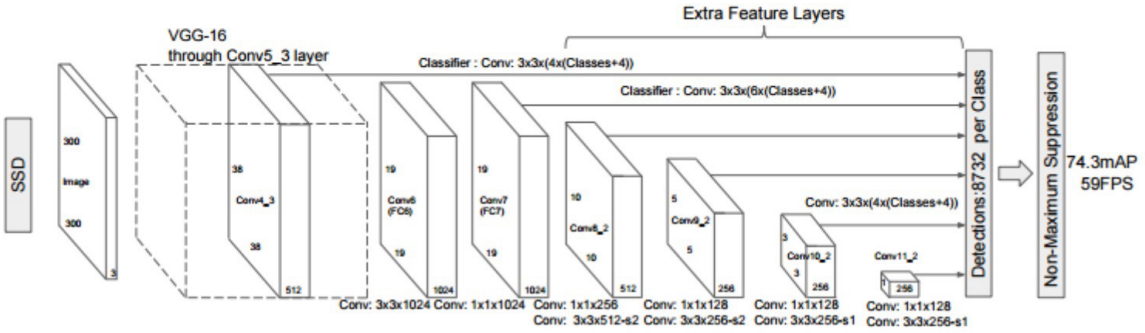
a muž). Změna také nastala u části sítě obstarávající extrakci příznaků. Přidalo se sem několik nových vrstev. Síť YOLOv3 tedy obsahuje celkem 53 konvolučních vrstev. Díky tomu dosahuje síť ještě větší přesnosti na úkor snížení rychlosti predikce. [11]

3.6 Architektura neuronové sítě Single Shot MultiBox Detector

Single shot multiBox detector využívá hluboké neuronové sítě. SSD síť pracuje s výchozími ohraničujícími rámečky, na kterých by se mohl vyskytovat některý z hledaných objektů, pro různé poměry stran a různá měřítka příznakových map. Oproti jiným architekturám tedy využívá výchozí pozice rámečků, menší konvoluční filtry. SSD dosahuje poměrně velké přesnosti i na vstupech s menším rozlišením. Například na datasetu VOC2007 dosahuje podobné úspěšnosti jako architektury uvedené v této kapitole. Síť dosahuje vyšší rychlosti detekování. Toho je dosaženo tím, že SSD nepotřebuje část pro výpočet pravděpodobných míst, ve kterých by mohl být některý z hledaných objektů, jelikož jsou tyto pozice předem definované. Díky tomu je jednodušší její natrénování.

SSD je založen na dopředných konvolučních neuronových sítích. Pro všechny výchozí ohraničující rámečky se vypočítá skóre určující, jestli se v něm nachází daný objekt. Výpočet se provede pro každý z hledaných objektů. Dále se vypočítá úprava výchozí pozice rámečku, tak aby se dosáhlo lepšího skóre, tedy aby box lépe odpovídal pozici objektu. Dále jsou vypočítané hodnoty zkombinované s výsledky z více velikostních map příznaků, aby byla zajištěna detekce objektů různých velikostí. Následně je aplikována funkce pro potlačení ne-maxim (non-maximum suppression).

Základ sítě tvoří architektura VGG-16. Další část sítě se stará o více velikostní mapu rysů, konvoluční detektory a předem definované ohraničující boxy. Přesné schéma struktury SSD je vidět na obrázku 3.7.[6]



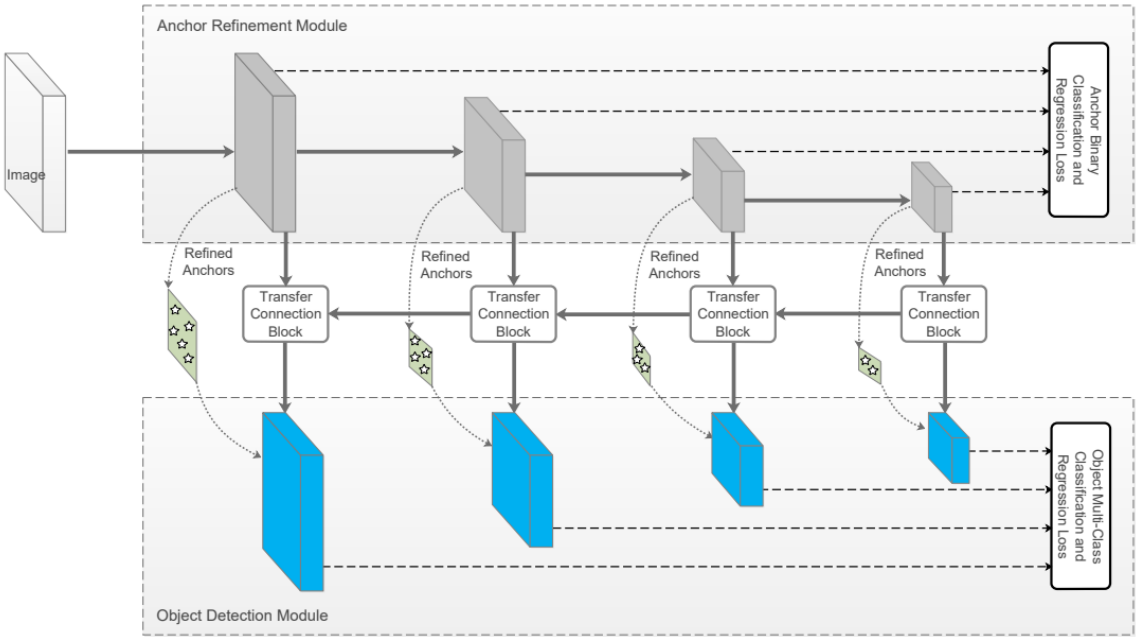
Obrázek 3.7: Architektura neuronové sítě Single shot multibox detector[8]

3.7 Architektura neuronové sítě RefineDet

RefineDet se snaží zkombinovat výhody jedno fázové detekce (Single Shot MultiBox Detector) jako je efektivita a rychlost a výhody dvoufázové detekce pro dosahování větší přesnosti (R-CNN, YOLO). RefineDet se skládá ze dvou vzájemně propojených modulů. V první části

[8]Obrázek byl převzat z [6].

(anchor refinement module) se redukují nalezená místa, která neobsahují žádné z hledaných objektů. To způsobí, že se nemusí prohledávat a analyzovat zbytečně další místa navíc. U správně nalezených míst se upraví poloha boxu a jeho velikost, aby druhý modul (object detection module) měl lepší výchozí data a produkoval lepší pozice a velikosti ohraničujících boxů. Druhý modul určí, o jakou třídu objektu se přesně jedná. Moduly jsou propojeny pomocí propojovacího bloku (transfer connection block), který slouží k přenosu příznaků z *anchor refinement module* do *object detection module*. Další funkcí tohoto bloku je zpracování příznaků ve větším kontextu díky vazbě mezi příznakovými mapami různých velikostí. Pro jejich zpracování v daném propojovacím bloku je nutné nejprve sjednotit jejich velikost. To je docíleno využitím dekonvolučních operátorů. Dále je propojovací blok tvořen konvolučními bloky a rektifikovanou lineární jednotkou. Přesné schéma tohoto bloku je znázorněno na obrázku 3.8. [17]

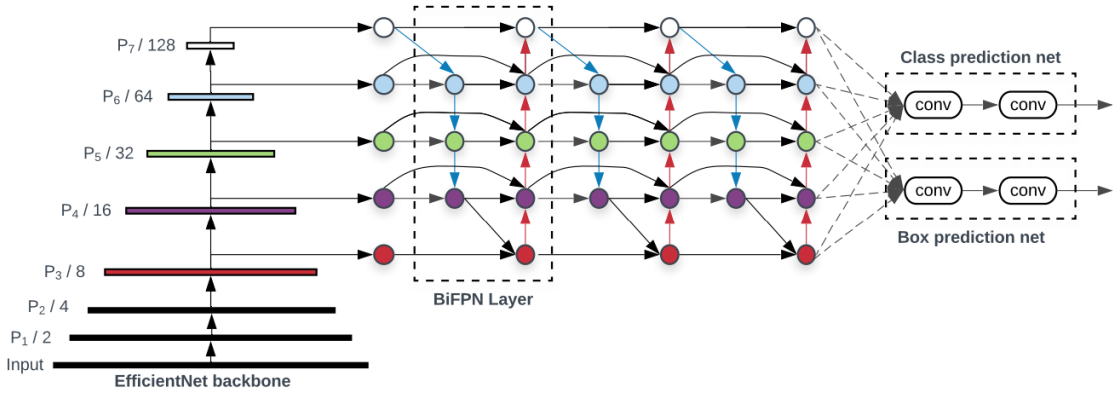


Obrázek 3.8: Architektura neuronové sítě RefineDet[9]

3.8 Architektura neuronové sítě EfficientDet

EfficientDet byl vyvinut ve výzkumném týmu Google Brain. Síť dosáhla zatím nejlepších výsledků v datasetu COCO. Hlavní myšlenkou této architektury je efektivně pracovat s více dimenzionálními příznakovými mapami v různém měřítku. EfficientDet vychází z klasifikační sítě EfficienNet[13], která byla vyvinuta stejnou skupinou. Tato síť je využita pro extrakci příznaků. Příznaky jsou následně zpracovány nově navrženou sítí BiFPN. Síť BiFPN pracuje s pěti úrovněmi příznaků. Jednotlivé úrovně jsou v různém měřítku. Jsou opakovaně spojovány využitím metody top-down feature fusion a bottom-up feature fusion. Upravené příznaky jsou následně předány síti pro predikci ohraničujícího boxu objektu a síti, která určí, o jaký objekt se jedná. Architektura této sítě je vidět na obrázku 3.9. [14]

[9]Obrázek byl převzat z [17].
[10]Obrázek byl převzat z [14].



Obrázek 3.9: Architektura neuronové sítě EfficientDet[10]

Kapitola 4

Prohledávání stavového prostoru

Tato kapitola shrnuje poznatky o metodách prohledávající stavový prostor. Vzhledem k omezenému rozsahu práce zde nejsou uvedeny všechny metody prohledávání stavového prostoru, ale pouze metody související s touto diplomovou prací.

4.1 Základní pojmy stavového prostoru

Stavový prostor se definuje jako dvojice (S, O), kde S je množina stavů, O je množina operátorů. Úloha, která se v daném stavovém prostoru má řešit, se pak definuje jako dvojice (s_0, G), kde s_0 označuje počáteční stav, který patří do množiny stavů, a G označuje množinu koncových stavů. Mezi jednotlivými stavy lze přecházet aplikováním určitých operátorů. V každém stavu je ale pouze určitá podmnožina operátorů, které lze v daném stavu využít. Cílem úlohy daného stavového prostoru je nalézt nejideálnější cestu do některého koncového stavu z množiny G.

Stavový prostor lze reprezentovat několika způsoby. Nejčastěji se pro zobrazení používá orientovaný graf nebo strom. Uzly v grafu představují jednotlivé stavy ve stavovém prostoru a hrany jsou pak aplikovatelné operátory mezi jednotlivými uzly (stavy). Úloha pak řeší problém nalezení nejefektivnějšího průchodu grafem od počátečního uzlu ke koncovému uzlu. V případě stromové reprezentace stavového prostoru je počáteční uzel kořen stromu.

K prozkoumání stavového prostoru lze využít klasické metody prohledávání stavového prostoru. Hlavní rozdíl mezi těmito metodami je, že při prozkoumávání stavového prostoru známe pouze počáteční stav a na základě postupného aplikování operátoru na změnu stavu se získávají a objevují nové stavy stavového prostoru.

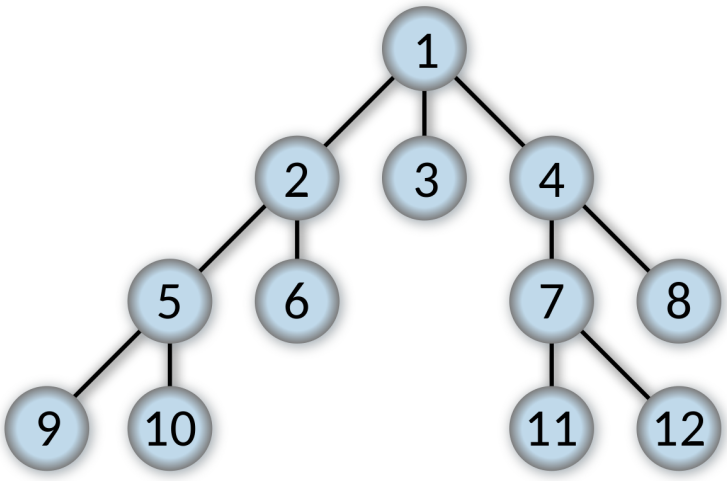
Metody prohledávání stavového prostoru znají celý stavový prostor. Lze je rozdělit do tří kategorií – slepé, informované a metody lokálního prohledávání. Slepé metody nemají žádné dodatečné informace o stavovém prostoru, které by umožnily lepší a efektivnější naplánování průchodu prostorem. Z tohoto důvodu bývají tyto metody pomalejší, jelikož pouze systematicky prohledávají stavový prostor. Jednotlivé slepé metody se mezi sebou liší hlavně pořadím procházení jednotlivých uzlů, jelikož žádná další data, které by mohly využít pro plánování, nemají. Informované metody mají navíc heuristickou funkci, která umožňuje ohodnotit jednotlivá spojení mezi uzly. Díky tomu metody získají další informaci, na základě které mohou vybrat nejvhodnější operátor, který budou aplikovat. Metody lokálního prohledávání analyzují pouze okolí aktuálního stavu. Tento typ metod se využívá především v optimalizačních úlohách.

Jednotlivé metody prohledávání stavového prostoru se posuzují na základě několika kritérií. Jedním z kritérií je časová složitost dané metody, která vypovídá o počtu kroků potřebných pro nalezení ideální cesty. Dále se posuzuje prostorová složitost, tedy paměť potřebná pro výpočet cesty. Dalším kritériem je kvalita získaného řešení. Zde se posuzuje například úplnost metody nebo optimálnost řešení. Metoda je úplná jestliže nalezne alespoň jedno z existujících řešení. Metoda je optimální právě tehdy, když metoda najde řešení, které je nejlepší (například se jedná o nejkratší nebo nejrychlejší cestu). Každá optimální metoda musí tedy být i úplná.

Diplomová práce se zaměřuje na exploraci všech dosažitelných obrazovek, o kterých robot nemá žádné dodatečné informace, proto zde budou popsány hlavně některé neinformované metody. Stavy v tomto případě odpovídají všem obrazovkám a operátory jsou reprezentovány tlačítky na jednotlivých obrazovkách. Řešenou úlohou je pak nalezení všech stavů (obrazovek) pomocí aplikování operátorů (klikáním na dostupná tlačítka na obrazovce). Tato kapitola popisuje principy nejpoužívanějších neinformovaných metod, jako jsou prohledávání do šířky, prohledávání do hloubky, prohledávání do hloubky s omezením a iterativní prohledávání do hloubky. [16]

4.2 Prohledávání do šířky

Metoda prohledávání do šířky, zkráceně BFS z anglického pojmenování breadth-first search, je jednou z nejzákladnějších metod prohledávání stavového prostoru. Vyvinul ji již v roce 1945 německý informatik Konrad Zuse. Jak již název napovídá, metoda spočívá v postupném prohledávání stavového prostoru do šířky, tedy po jednotlivých úrovních. Úroveň stavu je dána nejmenším možným počtem potřebných kroků, které je nutné vykonat, od počátečního uzlu. Algoritmus se snaží nejdříve prozkoumat všechny stavy na stejné úrovni. Jestliže algoritmus nenalezne cílový stav, začne prohledávat další úroveň. Takto se postupuje až do stavu, kdy algoritmus prohledá všechny stavy, nebo dokud nenalezne cílový stav. Postup prohledávání stavového prostoru touto metodou je znázorněn na obrázku 4.1.



Obrázek 4.1: Znázornění průchodu grafem stavového prostoru pomocí metody prohledávání do šířky (hodnoty v jednotlivých stavech reprezentují pořadí, ve kterém jsou tyto stavy analyzovány)[1]

Při implementaci metody BFS se využívá datová struktura FIFO fronta, jelikož její způsob přidávání a odebírání prvků přesně odpovídá požadovaným vlastnostem této metody. Algoritmus této metody začíná inicializací fronty OPEN, ve které budou uloženy ještě nezpracované stavy, a přidá do ní počáteční uzel. Dále vytvoří seznam CLOSE, který bude obsahovat již zpracované stavy. Algoritmus následně vybere první prvek z fronty OPEN a začne s jeho zpracováním. Zkontroluje, jestli se nejedná o koncový stav. Pokud ano, ukončí algoritmus a vrátí cestu z počátečního stavu k nalezenému koncovému stavu. V opačném případě vygeneruje všechny dosažitelné stavy z tohoto stavu, které nejsou ve frontě OPEN ani v seznamu CLOSE. Generování nových stavů probíhá aplikováním všech dostupných operátorů pro tento stav. Vygenerované stavy přidá do fronty OPEN a aktuálně zpracovávaný stav přidá do seznamu CLOSE. Algoritmus následně pokračuje opět vybráním prvního prvku z fronty OPEN, porovnání s koncovým stavem a generováním dalších stavů. Algoritmus končí v případě, že je fronta OPEN prázdná nebo dosáhne požadovaného koncového stavu.

Výše popsaný algoritmus popisuje upravenou metodu prohledávání do šířky s přidáním seznamu CLOSE. BFS se v praxi používá pouze s touto modifikaci, jelikož díky ní nedochází ke zbytečnému generování již zpracovaných stavů. Díky této úpravě se sníží časová i prostorová složitost.

Metoda BFS je optimální i úplná. Všechny vygenerované uzly musejí být uloženy v paměti, a proto je paměťová složitost stejná jako časová složitost. Při stromové reprezentaci stavového prostoru a použitím modifikace se seznamem CLOSE odpovídá časová složitost $O_{BFS}(b^d)$, kde b odpovídá maximálnímu počtu bezprostředních následníků všech vygenerovaných uzlů (tzv. faktor větvení) a symbol d je hloubka stromu, ve které se nachází koncový stav. Obecně ji lze zapsat jako $O_{BFS}(|V| + |E|)$, kde V jsou všechny vygenerované stavy a E jsou všechny aplikovatelné operace na stavech z množiny V. [16]

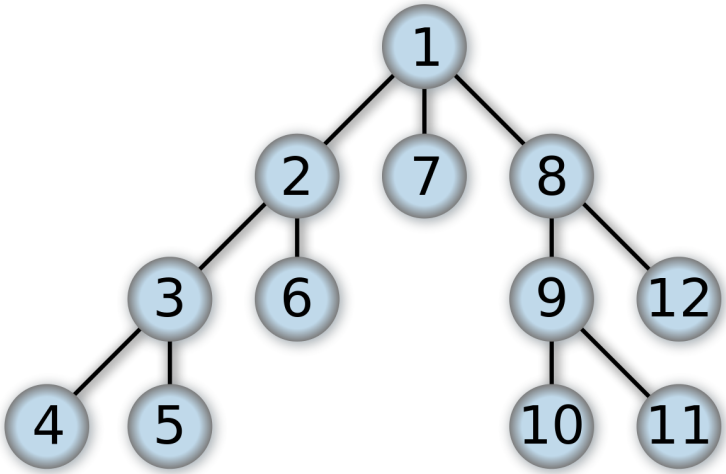
4.3 Prohledávání do hloubky

Metoda prohledávání do hloubky, anglicky depth-first search (DFS), pracuje na principu backtrackingu. Metoda nejprve prohledává stavový prostor až ke koncovému stavu. Při jeho dosažení dojde k navracení k předchozímu stavu, jehož ostatní vygenerované stavy nebyly zatím prozkoumány. U takového stavu se vybere jeden z dosud neprozkoumaných stavů a začne se analyzovat tato část stavového prostoru. Na obrázku 4.2 je graficky znázorněn průběh průchodu stavovým prostorem pomocí metody DFS. Tato metoda má ale problémy při prohledávání nekonečného stavového prostoru. Tento problém řeší další metody, které jsou postavené na této metodě jako například prohledávání do hloubky s omezením nebo prohledávání do omezené hloubky s postupným zanořováním, které budou popsány v následujících podkapitolách.

Typ metod, které jsou založené na backtrackingu, využívají datovou strukturu zásobník. Algoritmus metody prohledávání do hloubky se používá hlavně v modifikované verzi, která snižuje prostorovou i časovou složitost. Modifikovaná verze algoritmu nejprve inicializuje zásobník OPEN a přidá do něj počáteční stav. Po dokončení inicializace, algoritmus začne se samotným prohledáváním stavového prostoru. Vybere stav z vrcholu zásobníku OPEN, zkontroluje, jestli je tento stav koncový, případně zastaví analýzu stavového prostoru a vrátí

[1] Obrázek byl převzat z https://commons.wikimedia.org/wiki/File:Breadth-first-tree.svg (poslední aktualizace 29. března 2008 13:08 [cit. 11. 5. 2020]).

[2] Obrázek byl převzat z https://commons.wikimedia.org/wiki/File:Depth-first-tree.svg (poslední aktualizace 29. března 2008 11:38 [cit. 11. 5. 2020]).



Obrázek 4.2: Znázornění průchodu grafem stavového prostoru pomocí metody prohledávání do hloubky (hodnoty v jednotlivých stavech reprezentují pořadí, ve kterém jsou tyto stavy analyzovány)[2]

cestu z počátečního stavu k nalezenému koncovému stavu. V opačném případě vygeneruje pomocí aplikovatelných operátorů všechny dosažitelné stavy z vybraného stavu. Vygenerované stavy se zkontrolují, zda již nejsou uloženy v zásobníku OPEN nebo zda se nejedná o předky vybraného uzlu. Stavy, které splňují předcházející podmínky, jsou následně umístěny do zásobníku OPEN. Algoritmus dále opakovaně vybírá další stavy z vrcholu zásobníku, které analyzuje a rozgenerovává, dokud zásobník nebude prázdný nebo algoritmus nenalezne hledaný koncový stav.

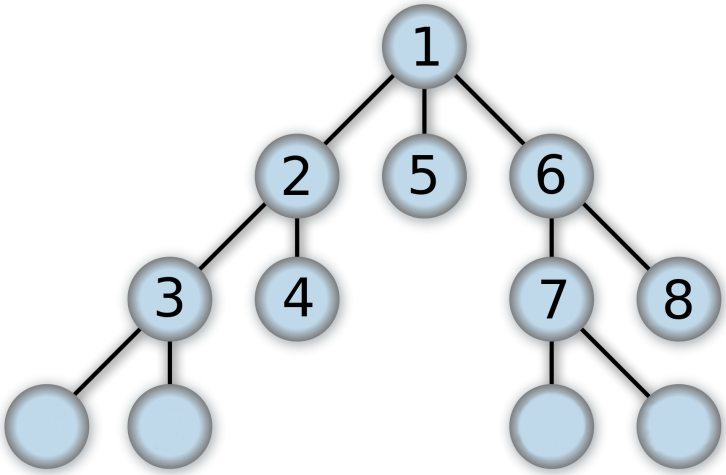
Rozdíl oproti základní verzi je v posledním kroku algoritmu. V původní verzi se přidávají do zásobníku OPEN všechny vygenerované stavy bez kontroly, zda je nově vygenerovaný stav již v zásobníku OPEN nebo se nejedná o předky nového stavu.

Modifikovaná metoda prohledávání do hloubky je úplná, ale není optimální. Původní verze není ani optimální ani prostorová, což je dalším důvodem proč se původní verze nepoužívá. Modifikovaná metoda dosahuje stejné časové náročnosti jako metoda prohledávání do šířky – tedy $O_{DFS}(b^{m/b})$, kde b je maximální počet bezprostředních následníků všech vygenerovaných stavů a m odpovídá maximálnímu počtu možných stavů. Prostorová složitost se liší od časové a odpovídá $O_{DFS}(m)$. [16]

4.4 Prohledávání do hloubky s omezením

Prohledávání do hloubky s omezením, neboli depth-limited search (DLS), vychází z metody prohledávání do hloubky popsané v předchozí podkapitole. Výhodou této metody je, že ji lze použít i pro prohledávání nekonečného stavového prostoru, jelikož prohledává stavový prostor pouze do předem definované hloubky. Po jejím dosažení metoda pokračuje v další části stavového prostoru nebo případně ukončí prohledávání. Tento způsob však má i svoje nevýhody. V případě, že se zvolí maximální hloubka prohledávání menší než hloubka, ve které se nachází koncový stav, tato metoda nenalezne hledaný stav. Tento problém řeší metoda prohledávání stavového prostoru do omezené hloubky s postupným zanořováním

popsaná v následující sekci (4.5). Postup prohledávání pomocí metody DLS je znázorněn na obrázku 4.3.



Obrázek 4.3: Znázornění průchodu grafem stavového prostoru pomocí metody prohledávání do hloubky s omezením s nastavenou hloubka prohledávání do 3. úrovně (hodnoty v jednotlivých stavech reprezentují pořadí, ve kterém jsou tyto stavy analyzovány)

Algoritmus funguje téměř stejně jako u metody prohledávání do šířky. Také používá zásobník, do kterého se při inicializaci vloží počáteční stav. Vybere z něj vrchol, u kterého zkontroluje, zda není koncovým stavem, případně rozgeneruje dosažitelné stavy, které následně analyzuje. Rozdíl oproti metodě DFS se projevuje u rozhodování, které uzly se rozgenerují. Jestliže je aktuálně zpracovávaný stav v hloubce, která je větší než předem určená maximální hloubka, tak se pro tento stav dále negenerují jeho následníci. V této části stavového prostoru skončí prohledávání a pokračuje se analýzou dalšího stavu z vrcholu zásobníku. Při generování dosažitelných stavů se opět kontroluje, zda se již nevyskytují v zásobníku stavů nebo se nejedná o předky aktuálního stavu, aby se dosáhlo lepší časové i prostorové složitosti.

Prohledávání do hloubky s omezením není úplné ani optimální. Tyto vlastnosti jsou zapříčiněny hlavně tím, že je možné nastavit maximální hloubku prohledávání nižší, než ve které se vyskytuje koncový stav. Prostorová složitost této metody je $O_{DLS}(b * l)$, kde l představuje maximální prohledávanou hloubku stromu a b je faktor větvení. Časová složitost pak odpovídá $O_{DLS}(b^l)$. [16]

4.5 Prohledávání do omezené hloubky s postupným zanořováním

Metoda prohledávání do omezené hloubky s postupným zanořováním, v angličtině známá pod názvem iterative deepening depth-first search (zkráceně IDS), vylepšuje metodu DLS, která je popsaná v předchozí sekci (4.4). Konkrétně opravuje její nevýhodu s nutností zvolit maximální prohledávanou hloubku. Ve většině případů není předem známo, v jaké hloubce se nachází hledaný stav.

Prohledávání do omezené hloubky s postupným zanořováním využívá algoritmu prohledávání do hloubky s omezením. Algoritmus inicializuje maximální hloubku prohledávání na

hodnotu 1 a zavolá algoritmus DLS. V případě, že algoritmus v dané hloubce nenalezne koncový stav, předá řízení zpět algoritmu IDS. Ten inkrementuje maximální povolenou hloubku prohledávání a opět zavolá algoritmus DLS pro danou hloubku. Jestliže algoritmus DLS skončí úspěšně, algoritmus IDS vrátí nalezenou cestu algoritmem DLS. Takto algoritmus inkrementuje hloubku, dokud jedno z volání DLS nenalezne hledaný koncový stav nebo dokud nebudou expandovány a zanalyzovány všechny stavy celého stavového prostoru.

Metoda prohledávání do omezené hloubky s postupným zanořováním dosahuje prostorové složitosti $O_{IDS}(b * d)$ a časové složitosti $O_{IDS}(b^d)$, kde d reprezentuje hloubku koncového stavu a b představuje faktor větvení. Dále tato metoda splňuje kritéria úplnosti i optimálnosti. [16]

Kapitola 5

Rozšíření pro internetový prohlížeč Google Chrome

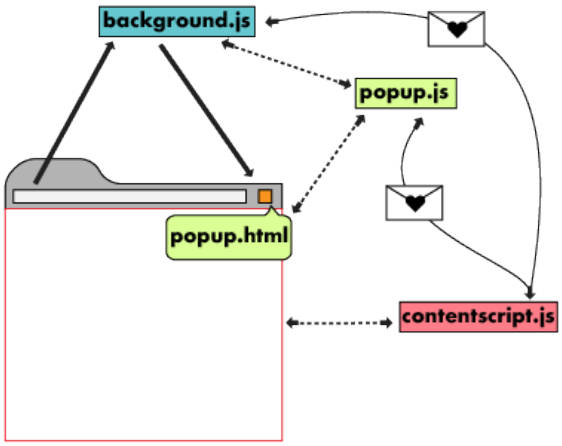
Jedná se o miniaplikace, které rozšiřují základní funkcionalitu webového prohlížeče Google Chrome. Umožňují modifikovat uživatelské rozhraní a webové stránky. Díky tomu může rozšíření například kontrolovat gramatiku při psaní na webu, blokovat reklamy, usnadnit přístup k emailovému klientu a mnoho dalšího. Vydané aplikace lze stáhnout na webové stránce chrome web store[1], kde je k dispozici popis daného rozšíření, hodnocení od uživatelů a počet stáhnutí.

Google chrome rozšíření jsou událostmi řízené programy. Události jsou vyvolány manuálně uživatelem při práci s webovou stránkou nebo prohlížečem. Rozšíření se skládá z několika typů souborů. Každé rozšíření musí obsahovat soubor `manifest.json`, ve kterém je specifikováno jméno rozšíření, popis, verze, oprávnění a cesty k JavaScript souborům s logikou aplikace. Položky *manifest_version*, *name*, *version* jsou povinné.

Hlavní logika miniaplikace se programuje pomocí jazyka JavaScript. Rozšíření má tři typy skriptů – *background script*, *content script* a *popup script*. Background skript běží na pozadí a monitoruje události prohlížeče. Na základě těchto událostí pak provádí definovanou činnost. Content skript je vložen do stránky a je spouštěn v rámci jejího kontextu. Skript může číst data z této stránky a měnit je využitím standardu DOM (Document Object Model). Je proveden hned při vložení skriptu a nemůže uchovávat perzistentní informace, jelikož při znovu načtení stránky je do ní opět vložen původní skript. Popup skript se provede po kliknutí na ikonu rozšíření v navigačním panelu prohlížeče. Vzhled vyskakovacího okna se upravuje pomocí HTML a CSS.

Jednotlivé soubory rozšíření mezi sebou komunikují pomocí zpráv nebo ukládáním proměnných v rámci úložiště prohlížeče Google chrome. Způsob komunikace je znázorněn na obrázku 5.1.[1]

[1] https://chrome.google.com/webstore
[2] Obrázek byl převzat z [1].



Obrázek 5.1: Obecné schéma způsobu předávání zpráv mezi skripty v rozšíření pro internetový prohlížeč Google Chrome[2]

Kapitola 6

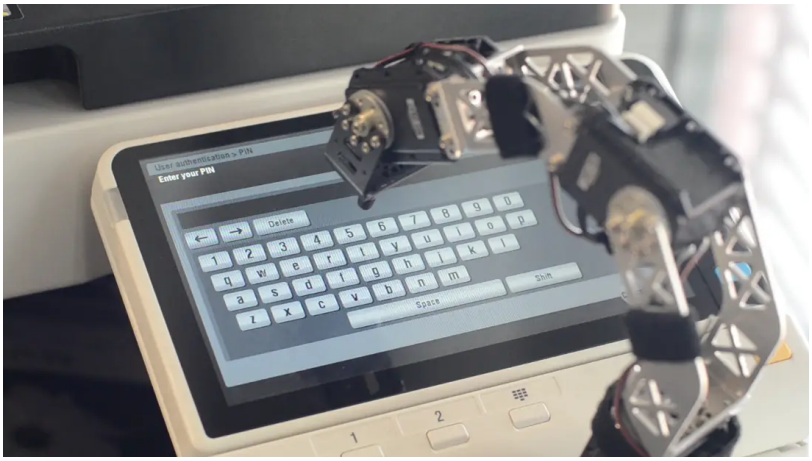
Návrh řešení

V této kapitole je uveden popis samotného navrženého systému. Nejprve je zde zhodnocen aktuální stav řešení robota pro testování vestavěných zařízení a jsou zde vytyčeny základní cíle, kterými se tato práce zabývá. Dále je zde popsán návrh algoritmů pro porovnání a nalezení všech dostupných obrazovek grafického rozhraní daného zařízení, způsob, jakým bude systém detekovat a lokalizovat tlačítka z fotografie a také způsob získání datasetu pro trénovní neuronové sítě.

6.1 Analýza a koncepce systému

Aktuální stav řešení

Při testování vestavěných zařízení je nutné fyzicky klikat na tlačítka, jelikož většina výrobců neposkytuje rozhraní pro softwarovou komunikaci mezi vestavěným zařízením a jinými aplikacemi. Z tohoto důvodu vyvinula společnost Y-Soft řešení pro automatizaci testování vestavěných zařízení s dotykovým displejem. Společnost Y-Soft vytvořila robotické rameno se stylusem, které umožňuje ovládat tento typ zařízení. Nad robotickým ramenem je upevněna kamera, která poskytuje zpětnou vazbu tomuto systému. Na základě porovnání očekávaného stavu (snímku displeje z databáze) a aktuálního stavu (právě zobrazená obrazovka na displeji) se rozhodne, zda aplikace funguje správně. Robotické rameno je vidět na obrázku 6.1.



Obrázek 6.1: Robotické rameno využívané pro automatické testování vestavěných zařízení

Společnost Y-Soft se zaměřuje na vývoj softwaru pro správu tisku (*Y-Soft SafeQ*). Primárně je tedy robot využíván na testování správného fungování nových aktualizací tohoto softwaru na různých typech tiskáren.

Tento systém však vyžaduje před prvním použitím robota na neznámém zařízení provést zdlouhavý proces inicializace pro dané zařízení. Proces inicializace spočívá v nasnímání potřebných obrazovek, které budou sloužit jako referenční stav, se kterým se budou porovnávat obrazovky při testování. Na každé obrazovce se musí také zaznačit pozice jednotlivých tlačítek.

Princip funkcionality navrhovaného systému

Cílem této práce je navrhnout a implementovat systém, který automaticky inicializuje nové zařízení, které se bude testovat. Systém tedy musí proklikat a analyzovat všechny dosažitelné obrazovky grafického rozhraní daného zařízení. Na každé obrazovce následně detekuje a lokalizuje tlačítka, které postupně prokliká.

O daném zařízení nebude mít systém žádné informace. Operátor robota pouze manuálně zkalibruje robotické rameno, aby správně klikalo na dotykovou obrazovku, a zadá velikost obrazovky zařízení. Vstupem algoritmu bude ořezaná nasnímaná fotografie obrazovky zařízení. Algoritmus lokalizuje tlačítka na dané fotografii a zadá povel robotickému ramenu, aby kliklo na vypočítanou pozici tlačítka. Systém následně zjistí, zda se změnila obrazovka zařízení, což indikuje, že se jedná o navigační tlačítko. Určí také, jestli již danou obrazovku analyzoval, a uloží závislost mezi stisknutým tlačítkem a nově zobrazenou obrazovkou. Tímto způsobem se postupně proklikají všechna detekovaná tlačítka na všech nalezených obrazovkách.

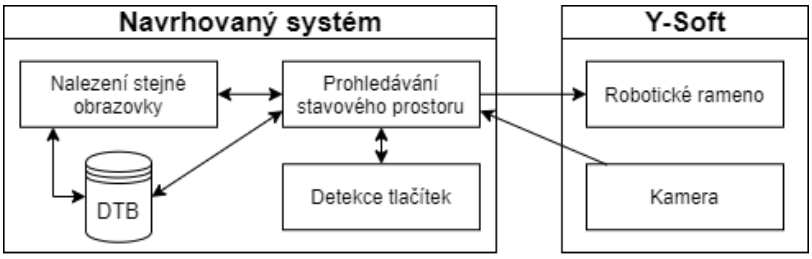
Výsledkem algoritmu tedy budou získané snímky ze všech obrazovek, na které se lze v zařízení proklikat. Ke každé obrazovce budou také uloženy pozice všech tlačítek, které se na ní nachází a informace, na jakou obrazovku dané tlačítko odkazuje. Na základě získaných dat algoritmus vytvoří „navigační mapu“ (orientovaný graf) zkoumaného zařízení.

„Navigační mapa“ bude následně sloužit pro inicializaci robotické ruky na novém zařízení, typicky na novém modelu tiskárny nebo na tiskárně od nového výrobce. Díky vytvořené „navigační mapě“ bude proces inicializace rychlejší a jednodušší, jelikož nebude nutné manuálně označovat pozice tlačítek na konkrétních obrazovkách a nastavovat jejich funkci (co se stane po kliknutí). Díky vytvořené „navigační mapě“ bude také možné najít nejkratší cestu mezi úvodní a požadovanou obrazovkou.

Návrh řešení systému

Navrhovaný systém lze rozdělit do čtyř samostatných částí. První část má za úkol detekci a lokalizaci tlačítek na snímku obrazovky. Další část se zabývá průběžným ukládáním dat navigační mapy. Pro tuto část jsou tedy implementovány skripty pro ukládání a načítání příslušných záznamů vytvářené navigační mapy do/z databáze. V další části je implementován algoritmus pro porovnávání dvou obrazovek, na základě kterého se rozhodne, zda se jedná o stejnou obrazovku. Případně algoritmus dohledá z databáze nejpodobnější obrazovku. Poslední část se stará o samotnou analýzu grafického rozhraní vestavěného zařízení. V této části je implementována hlavní logika celého systému, která komunikuje a řídí ostatní části systému. Takto rozdělené části lze i samostatně testovat a analyzovat jejich úspěšnost. To umožňuje snížit dobu potřebnou na otestování celého systému a zredukovat chyby při jeho kompletaci.

Systém bude navíc využívat rozhraní pro ovládání robotického ramene a získávání dat z kamery. Rozhraní je dostupné v rámci balíku funkcí robota, které s ním obstarávají komunikaci. Celkové propojení a vzájemné závislosti mezi jednotlivými částmi systému jsou znázorněny na obrázku 6.2.



Obrázek 6.2: Schéma rozdělení navrhovaného systému společně se zaznačenou vzájemnou komunikací mezi jednotlivými částmi systému a robotem společnosti Y-Soft

6.2 Vytvoření datasetu

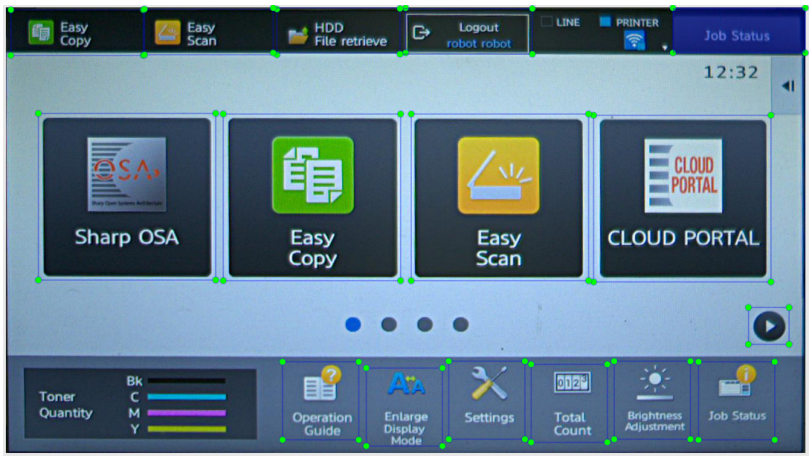
Při tvorbě datasetu jsem se rozhodl využít snímky obrazovek tiskáren, které mně poskytla firma Y-Soft. Jednalo se celkem o 3185 snímků obrazovek z odlišných modelů tiskáren od různých výrobců. Získané snímky ale nebyly dostatečně obecné, jelikož obsahovaly pouze grafické rozhraní tiskáren. Z tohoto důvodu jsem se rozhodl pořídit ještě další snímky, které by obsahovaly grafické rozhraní i z dalších zařízení, abych mohl vytvořit obecnější model, pro lokalizaci tlačítek.

K poskytnutým snímkům jsem vytvořil anotace všech tlačítek, které se na daných obrazovkách vyskytují. Pro tento účel jsem využil program LabelImg[1]. Jedná se o jednoduchý anotační nástroj sloužící k označování pozic na fotografii, na kterých se nachází hledaný objekt. Anotační soubory jsem ukládal ve formátu PascalVOC. Příklady některých manuálně anotovaných snímků obrazovek jsou vidět na obrázku 6.3

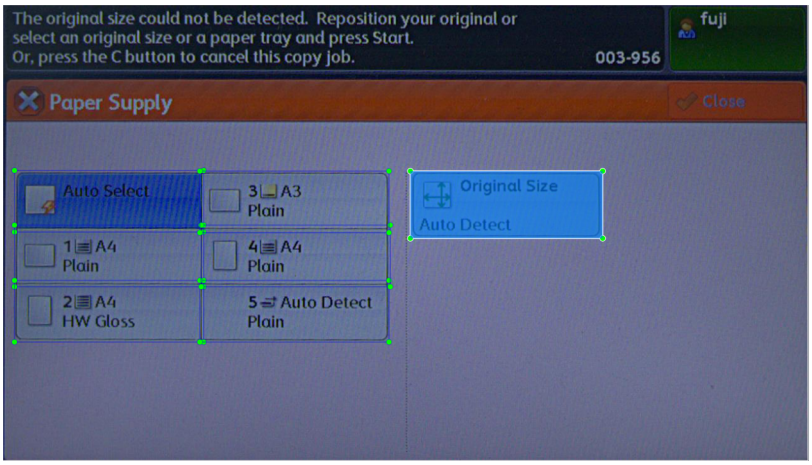
Aby bylo možné použít vyvíjený systém pro libovolná zařízení (nejenom tiskárny), doplnil jsem dataset o další typy snímků s tlačítky. Ty jsem se rozhodl získat dolováním dat z internetových stránek, jelikož každá stránka obsahuje několik tlačítek s různým vzhledem. Za tímto účelem jsem tedy navrhl algoritmus, který vytvoří snímek webové stránky a k němu i anotační soubor s pozicemi tlačítek. Algoritmus je implementován jako rozšíření pro internetový prohlížeč Google chrome. Na základě html kódu dané stránky se vyhledají všechna tlačítka a pro každé z nich se vypočítá jejich poloha vzhledem k oknu prohlížeče, ze kterého byl pořízen snímek. U každého tlačítka se ještě určí, zda je možné, aby ho neuronová síť detekovala pouze na základě vzhledu. Zkontrolují se tedy následující podmínky, které jsem vymezil pro detekci rozpoznatelnosti. Zjišťuje se, zda má daný objekt (tlačítko) viditelné ohraničení nebo jestli se v daném objektu vyskytuje obrázek, ikona nebo piktogram. Dále se testuje, jestli se liší pozadí daného html objektu a pozadí rodičovského objektu. Pozice nalezených tlačítek, která splňují alespoň jednu z podmínek viditelnosti, se následně uloží do souboru v anotačním formátu *PascalVOC*. Jakmile je webová stránka zpracována, program automaticky simuluje kliknutí na jedno z nalezených tlačítek, které odkazují na webovou stránku ve stejné doméně, a pro nově načtenou stránku provede opět

[1]LabelImg je dostupný na webové stránce https://github.com/tzutalin/labelImg

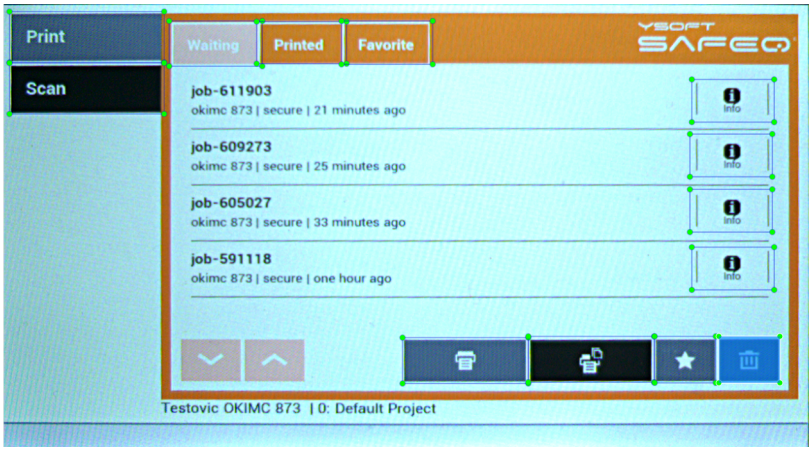
analýzu webové stránky a uložení anotačního souboru. Takto program postupuju, až do té doby, dokud neprojde celou strukturu webové stránky v dané doméně.



(a)



(b)

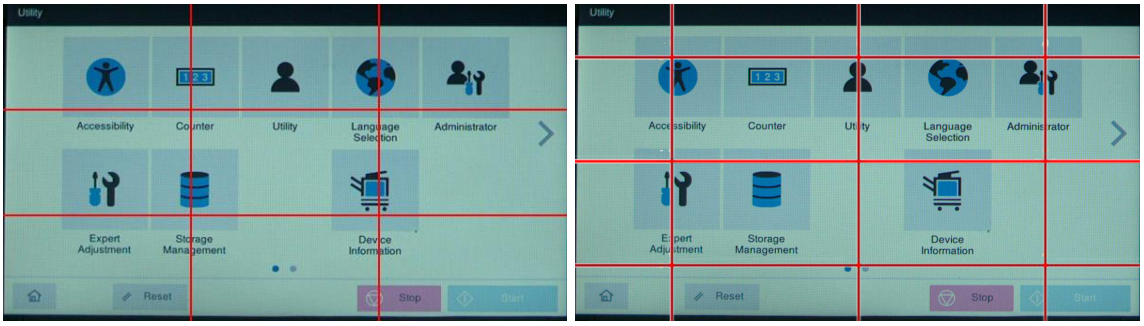


(c)

Obrázek 6.3: Příklady anotovaných snímků obrazovek tiskáren pro trénování neuronové sítě

6.3 Detekce stejných obrazovek

Detekování stejných obrazovek je klíčové pro to, aby robot zjistil, že se proklikal na obrazovku, kterou již někdy zpracoval. Pro tento úkol jsem se rozhodl použít jeden z algoritmů popsaných v sekci 2. Problém je, že velká část obrazovky bude vždy stejná, jelikož bude robot v jednom běhu pracovat pouze s jedním zařízením. Výrobci mají zpravidla obrazovky velmi podobné, aby se zákazníci nemuseli učit rozeznávat nové grafické rozhraní při přechodu na novější typ zařízení. Výrobci koncipují svá grafická rozhraní tak, aby měla stejné barevné schéma a stejné rozložení ovládacích prvků. Tento fakt komplikuje detekci, zda se jedná o stejnou nebo rozdílnou obrazovku. Algoritmus musí správně rozhodnout i na základě nepatrných změn na snímku obrazovky. Nepatrné změny mohou být například změna slova nebo věty na obrazovce, zobrazení malého vyskakovacího nebo dialogového okna a podobně. Pro lepší rozpoznání těchto detailů jsem se rozhodl rozdělit fotografii na předem daný počet částí. Každou z nich následně porovnat s korespondující částí z druhé fotografie. Takto se vypočítá podobnost pro každou část. Výsledná hodnota podobnosti bude nejmenší hodnota podobnosti z jednotlivých částí. Způsob rozdělení fotografie je znázorněn na obrázku 6.4. Algoritmus bude sloužit i pro rozhodnutí, zda zařízení zaznamenalo kliknutí robotického ramene a rozlišení typu tlačítka – akční tlačítko nebo tlačítko klávesnice, které přidá na obrazovku navíc jeden znak (způsobí pouze malou změnu).



(a) (b)

Obrázek 6.4: Navržený postup rozdělení fotografie pro zjištění nejmenší hodnoty podobnosti jednotlivých oblastí (nejprve se použije rozdělení fotografie znázorněno v (a) a následně na základě (b))

Vstupní parametr navrhovaného algoritmu bude snímek obrazovky, na které se robot aktuálně nachází. Z databáze se následně načtou snímky všech navštívených obrazovek a pro každou z nich se vypočítá index podobnosti. Jestliže jeho hodnota překročí předem určený práh pro stejné snímky, robot již aktuální obrazovku navštívil. Algoritmus následně uloží do databáze vazbu mezi tlačítkem, které odkazovalo na tuto obrazovku, a nalezenou obrazovkou.

6.4 Detekce tlačítek

Pro samotnou detekci a lokalizaci tlačítek na snímku obrazovky jsem se rozhodl využít neuronové sítě. Neuronové sítě poskytují dostatečnou schopnost generalizace natrénovaných dat, což je optimální v případě, když bude síť detekovat tlačítka neznámých zařízení. Dalším důvodem je jednoduchá úprava a optimalizace natrénovaného modelu i po nasa-

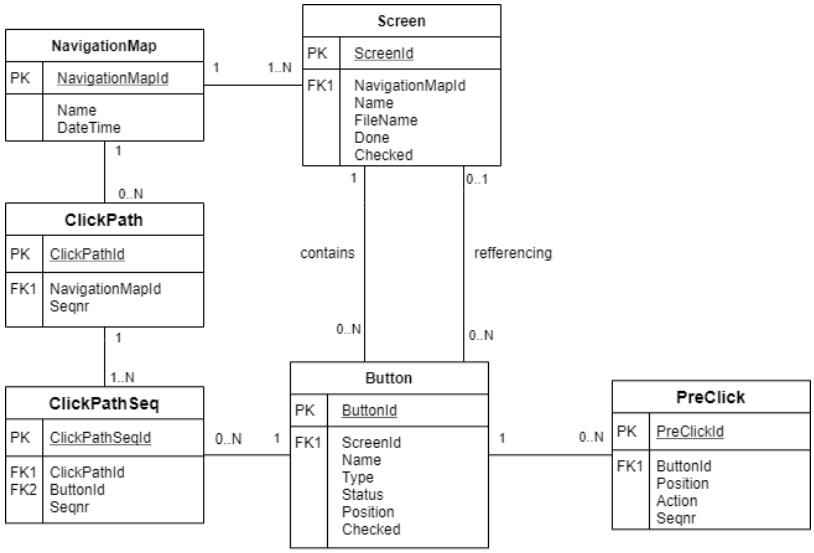
zení systému do reálného provozu. V případě, že model vůbec tlačítko nedetekuje nebo jej špatně lokalizuje, uživatel manuálně zaznačí pozici tlačítka a síť dotrénuje pro nový a dosud neznámý vzhled tlačítka.

Pro detekci tlačítek na základě fotografie obrazovky zatím nejsou dostupné žádné výsledky úspěšnosti neuronových sítí. Rozhodl jsem se tedy vyzkoušet a natrénovat modely popsané v sekci 3.1. Tyto modely jsem zvolil, jelikož dosahují dobrých výsledků na obecných datasetech, kde je cílem detekovat různé druhy objektů na fotografii. Jedná se například o datasety *COCO* nebo *Pascal VOC*. Vybrané neuronové sítě jsem tedy natrénoval na mnou vytvořeném datasetu z webových stránek a následně síť dotrénoval na datasetu se snímky obrazovek tiskáren, jelikož bude výsledný systém testován právě na tiskárnách. Následně porovnám výsledky natrénovaných modelů a nejlepší z nich bude nasazen do systému. Ve výsledcích budu hlavně porovnávat přesnost predikovaného ohraničujícího boxu oproti reálné pozici tlačítka a poměr nedetekovaných tlačítek. V tomto případě není problém, když bude model detekovat tlačítka, které na snímku nejsou. Při následné analýze kliknutím na takto detekovanou pozici „tlačítka“, na které se reálně žádné tlačítko nevyskytuje, systém vyhodnotí jako neakční, tedy tlačítko, které neslouží pro navigaci mezi obrazovkami a nebude tedy dále analyzováno.

6.5 Schéma databáze

Navigační mapa se ukládá do databáze. Přesné schéma celé databáze a závislostí jednotlivých tabulek jsou znázorněny na obrázku 6.5. Navržená databáze se skládá z 6-ti tabulek. Hlavní tabulka v sobě ukládá všechny vytvořené navigační mapy. Obsahuje datum a čas vytvoření mapy a její název. Tabulka uchovávající data o nalezených obrazovkách obsahuje název, který při kontrole operátor změní na více vypovídající název, cestu k fotografii obrazovky z kamery robota a pomocné atributy, které využívá algoritmus proklikávání obrazovek. Dále je v databázi tabulka s informacemi o nalezených tlačítkách na jednotlivých obrazovkách. Tabulka obsahuje opět vygenerovanou hodnotu atributu pro název tlačítka, typ tlačítka a propojení s obrazovkou, na kterou tlačítko odkazuje a také na které obrazovce se dané tlačítko nachází. Typ tlačítka udává, zda se jedná o akční tlačítko, které odkazuje na některou obrazovku, nebo tlačítko, které způsobí pouze změnu na dané obrazovce (například tlačítko z klávesnice, které do obrazovky navíc doplní pouze jeden znak). Pomocné atributy slouží pro uchování informací, jestliže dané tlačítko/obrazovka byla již plně zpracována. Tlačítko se nastaví jako zpracované, když odkazuje na již zpracovanou obrazovku nebo pokud se jedná o neakční typ tlačítka. Obrazovka se označí jako zpracovaná, pokud všechna tlačítka, které se na ní nachází, jsou označena jako zpracovaná. Další pomocný atribut slouží pro rozlišení, zda byla správnost daného tlačítka nebo obrazovky již zkontrolována operátorem robota.

Dále jsou v databázi uloženy prerekvizity pro určitá tlačítka. Prerekvizitami se rozumí kroky, které je nutné udělat před samotným kliknutím na konkrétní tlačítko. Tyto kroky se vykonají, aby se dané tlačítko stalo aktivní a mohl na něj robot kliknout. Typicky se prerekvizity budou muset zadávat například pro tlačítko Login, aby mohl robot proklikat i část grafického rozhraní pro přihlášené uživatele. Záznamy do této tabulky musí doplnit manuálně operátor robota. Může se například jednat o sérii kliknutí na tlačítka klávesnice pro zadání uživatelského jména a hesla, které budou zadány pozicemi, na které má robotické rameno kliknout. Schéma tabulky jsem proto navrhl tak, že obsahuje atributy – pořadové číslo operace, souřadnice, kam má robot kliknout a vazbu na tlačítko, ke kterému se váže tato prerekvizita.



Obrázek 6.5: Schéma navržené databáze pro ukládání navigační mapy společně s informacemi o průběhu vytváření

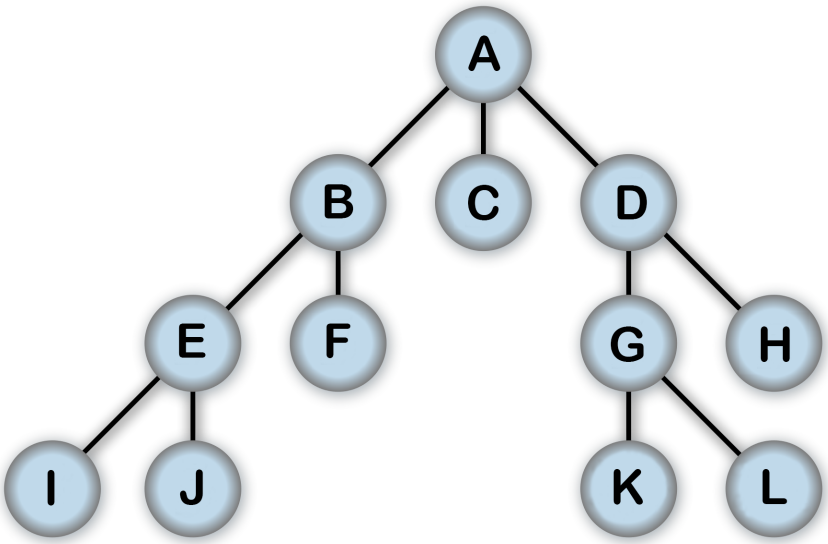
6.6 Algoritmus pro proklikávání obrazovek

Po dohodě se zadavateli projektu jsem se rozhodl nevyužít žádný z běžných algoritmů pro prohledávání stavového prostoru popsaných v sekci 4, jelikož žádný z nich nevyhovoval požadavkům pro tento projekt. Hlavním argumentem byl fakt, že postupné proklikávání všech tlačítek může mít vliv na vzhled některých obrazovek. Pořízené snímky obrazovek by pak mohly způsobovat komplikace při klasickém běhu robota, jelikož by algoritmus uložil jinak vypadající snímek obrazovky. Při porovnání aktuální obrazovky na zařízení a snímku obrazovky z databáze by pak mohlo robotické rameno špatně vyhodnotit tyto obrazovky, čímž by selhal běh testování robotem. Z tohoto důvodu jsem se rozhodl adaptovat algoritmus pohledávání do hloubky na zadaný problém.

Hlavní změna spočívá v navracení po nalezení posledního uzlu – v tomto případě obrazovka, na které už nejsou žádná neproklikaná tlačítka. V případě algoritmu pohledávání do hloubky by se navrátilo na předchozí obrazovku a z ní by algoritmus pokračoval kliknutím na další tlačítko. Adaptovaný algoritmus se po nalezení posledního uzlu navrátí až na domovskou obrazovku, ze které znovu začne proklikávání na poslední známou obrazovku, na které se vyskytují ještě neproklikané tlačítka. Princip algoritmu je popsán na obrázku 6.6. Navržený způsob procházení stavového prostoru lépe odpovídá způsobu proklikávání jednotlivých obrazovek při běžném provozu robotického ramene, které využívá výstup tohoto algoritmu.

6.7 Zobrazení výsledků

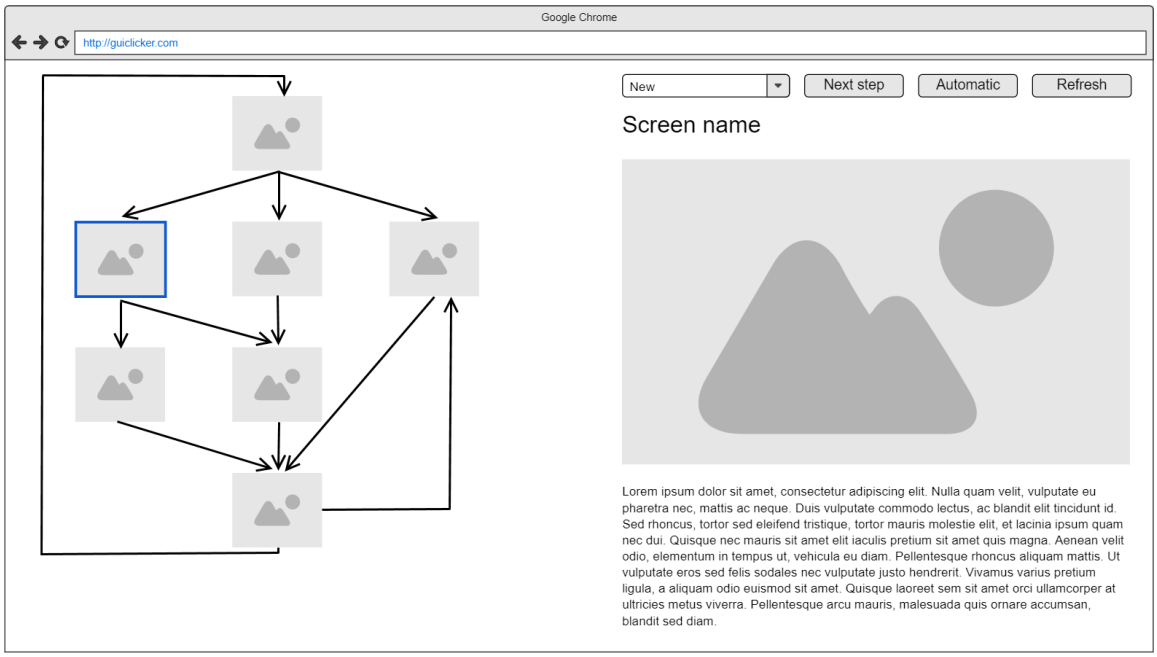
Dále jsem se rozhodl vytvořit webovou aplikaci, která umožňují sledovat průběžný stav vytváření navigační mapy. Po dokončení generování navigační mapy bude aplikace sloužit i pro její vizualizaci. Aplikace bude také sloužit pro ovládání robota a algoritmu, například umožní přepnout režim robota z automatického módu na debugovací, ve kterém lze



Obrázek 6.6: Průchod grafem stavového prostoru pomocí navržené metody probíhá v posloupnosti $A \rightarrow B \rightarrow E \rightarrow I \rightarrow A \rightarrow B \rightarrow E \rightarrow J \rightarrow A \rightarrow B \rightarrow F \rightarrow A \ldots$

postupně krokovat klikání robotického ramene na jednotlivá tlačítka a zobrazit detekované tlačítko ze získaného snímku obrazovky.

Pro tuto webovou aplikaci jsem navrhl jednoduchý a intuitivní vzhled. V levé části obrazovky se nachází vytvořený graf navigační mapy. Jednotlivé uzly grafu jsou znázorněny miniaturou snímku obrazovky pro snazší orientaci v grafu. U miniatury bude i vygenerovaný název obrazovky, který bude možné manuálně případně změnit operátorem robota. Návaznost tlačítek je vyobrazena pouze orientovanými hranami mezi obrazovkou, na které se tlačítko nachází, a obrazovkou, na kterou odkazuje. Webová aplikace také umožní zobrazení snímků jednotlivých obrazovek. Kliknutím na některý uzel se v pravé části zobrazí snímek odpovídající obrazovky v plné velikosti. Na tomto snímku budou znázorněny i detekovaná tlačítka a informace na jakou obrazovku odkazují. V pravé části obrazovky budou základní tlačítka webové aplikace pro nastavení režimu robotického ramene – automatický a debugovací mód. Dále zde bude možnost vybrat z databáze již dokončenou navigační mapu a vizualizovat ji pomocí navrhované aplikace. Přesný návrh designu webové aplikace lze vidět na obrázku 6.7.



Obrázek 6.7: Návrh designu webové aplikace pro zobrazení vygenerované navigační mapy

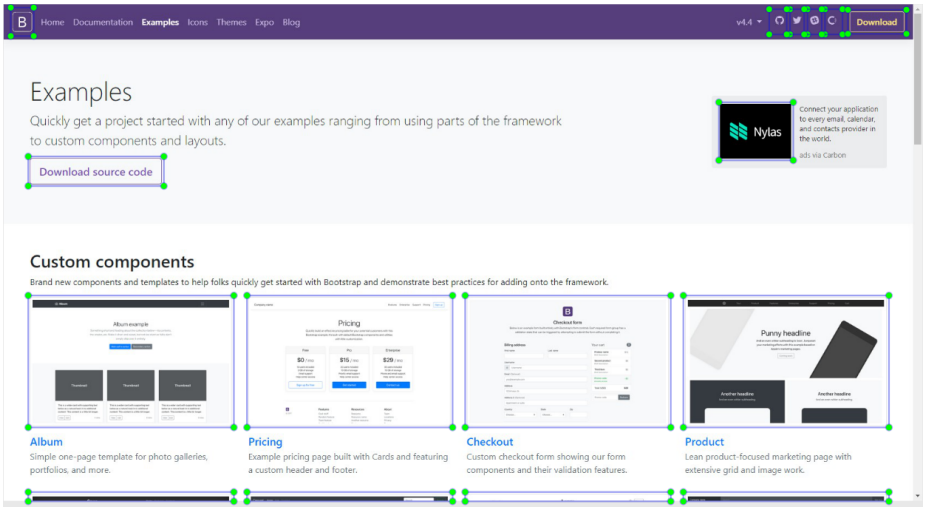
Kapitola 7

Implementace

Hlavní část aplikace jsem implementoval v programovacím jazyku C#. Trénování a testování modelů pro detekci a lokalizaci tlačítek byly implementovány v programovacím jazyce Python s využitím knihovny Tensorflow a Keras.

7.1 Rozšíření pro Google chrome

Algoritmus popsaný v sekci 6.2 jsem naimplementoval jako rozšíření pro internetový prohlížeč Google chrome. Rozšíření jsem naprogramoval pomocí jazyka javascript. Chování algoritmu jsem rozdělil do dvou částí – background a content. Tyto části odpovídají klasickému rozdělení rozšíření, které jsem popsal v sekci 5. Záměrně jsem vynechal skript obstarávající logiku vyskakovacího okna, jelikož rozšíření funguje automaticky a samo proklikává webové stránky. V následujících kapitolách jsou podrobněji popsány implementační detaily souborů s logikou aplikace. Ukázka vygenerovaných snímků webových stránek pomocí rozšíření pro Google Chrome lze vidět na obrázku 7.1.



Obrázek 7.1: Snímek webové stránky se zaznačenými anotacemi vytvořených pomocí mnou implementovaného rozšíření pro Google Chrome

Content skript

Content skript je vložen přímo do načtené stránky. Při vložení skriptu se zanalyzuje html struktura webové stránky a najdou se v ní všechny elementy typu `a`, `button`, `li` a `span`. Tyto typy elementů se nejčastěji používají jako odkazy. Nalezené elementy se postupně prochází a u každého z nich se rozhodne, zda je možné jej detekovat softwarově (například neuronovou sítí). Zkoumají se podmínky definované v sekci 6.2. Zkontrolují se hodnoty příslušných atributů – `borderWidth`, `backgroundColor`. Případně se ověří, zda některý z podelementů neobsahuje html element typu `img`, `svg` nebo `i`.

Dále je u potencionálního elementu zkontrolován CSS atribut `cursor`, jestli odpovídá očekávané hodnotě `pointer`. U navigačních menu na začátku stránky se často stává, že obsahují rozbalovací menu. Položky v rozbalovacím menu se zobrazí až po najetí kurzoru na určitý element. Aby algoritmus nedetekoval elementy, které sice jsou v html struktuře, ale nejsou na snímku webové stránky zobrazeny, je přidána kontrola viditelnosti. Algoritmus tedy porovná atribut `display` u daného elementu a všech rodičovských elementů, aby ověřil jeho viditelnost. V dalším kroku je ověřeno, zda element odkazuje na webovou stránku ve stejné doméně a jestli neodkazuje na soubor ke stažení.

V případě, že element úspěšně projde výše uvedenými podmínkami, vytvoří se kolem tohoto elementu ohraničující obdélník, pomocí javascript funkce `getBoundingClientRect()`. Vytvořený obdélník se zvětší o několik pixelů, což napomůže lepšímu trénování. Následně se do finálního listu připojí struktura s informacemi o ověřeném odkazu, tedy vypočítaná pozice ohraničujícího boxu a hodnota html atributu `id` daného elementu. V případě, že element nemá nastaven atribut `id`, je mu vygenerována nová unikátní hodnota `id` a vložena do html struktury webové stránky.

Po zpracování všech odkazů je list uložen do úložiště `chrome.storage`, ze kterého může číst data i background skript. Content skript informuje background skript, že dokončil analýzu stránky, zasláním notifikační zprávy pomocí chrome API. Následně content skript čeká na přijetí zprávy od background skriptu. Zpráva obsahuje hodnotu `id` elementu, pomocí kterého jej najde v html struktuře a simuluje na něm kliknutí, tedy přesměrování na stránku, na kterou odkazuje.

Background skript

Background skript slouží pro pořizování a ukládání snímků obrazovek a ukládání detekovaných odkazů ve formátu *PascalVOC*. Stará se také o automatické procházení webu. Při automatickém procházení webových stránek je nutné mít přehled o již navštívených a zpracovaných stránkách. K tomuto účelu jsem implementoval pole, které uchovává URL adresy navštívených stránek, seznam detekovaných odkazů a informaci o tom, zda byla webová stránka, na kterou odkazuje, již zpracována.

Pomocí funkce `chrome.tabs.captureVisibleTab()` je vytvořen snímek viditelné části webové stránky, který je převeden do formátu JPEG a uložen do počítače do složky `Downloads`. Protože se většina navigačních prvků vyskytuje v horní části webové stránky, je zbytečné vytvářet snímek i zbytku stránky, jelikož se v těchto částech vyskytují většinou texty nebo obrázky.

Následně je vytvořen anotační soubor k získanému snímku webové stránky, do kterého skript vygeneruje ohraničující boxy, které vytvořil content skript, a metadata o tomto snímku. Anotační soubor se také uloží do složky `Downloads`. Po dokončení stahování background skript informuje content skript, aby simuloval kliknutí na první odkaz z listu.

V případě, že byla webová stránka již analyzována a v počítači je uložen anotační soubor i snímek obrazovky, skript zašle oznámení content skriptu, aby simuloval kliknutí na další odkaz. Hodnota `id` odkazu se určí na základě pole již proklikaných odkazů dané webové stránky. Jestliže byly analyzovány již všechny odkazy na stránce, je zaslána content skriptu přímo URL adresa stránky, na které ještě nebyly prozkoumány všechny odkazy.

Manifest

V souboru manifest jsou uloženy informace o tomto rozšíření a povolení, které potřebuje pro správné fungování. Rozšíření jsem pojmenoval *Button dataset maker*. Mezi potřebné povolení patří například `storage` pro ukládání pole odkazů, `Downloads` pro stahování vygenerovaného datasetu a snímku webové stránky, `desktopCapture` pro pořízení snímku stránky a `tabs` pro získání URL adresy analyzované stránky. Dále se zde nastaví seznam URL adres, na kterých bude rozšíření aktivní – tedy bude vytvářet anotační soubory a snímky webu. Implementované rozšíření jsem použil hlavně na webových stránkách nejpoužívanějších CSS knihoven, jako je například knihovna Bootstrap[1].

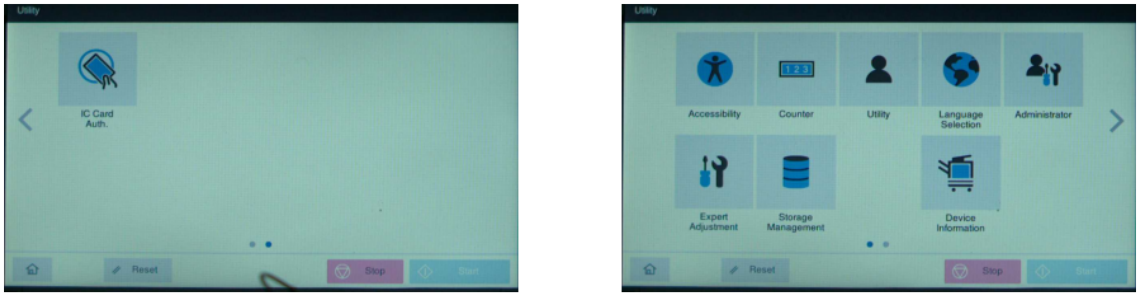
7.2 Detekce stejných obrazovek

Analyzoval jsem schopnost rozpoznat rozdílné obrazovky na získaném datasetu. Hlavně jsem se soustředil na dataset obsahující snímky obrazovek tiskáren, jelikož u tohoto typu bude nejkomplikovanější rozlišit jednotlivé snímky obrazovek. Tento dataset jsem rozdělil do kategorií na základě výrobců tiskáren. V dané kategorii mohou být zastoupeny i snímky obrazovek z různých typů a verzí tiskáren.

Na základě výsledků porovnání algoritmů podobnosti fotografií jsem se rozhodl použit metriku Structural SIMilarity index. Tato metoda dokáže reflektovat i malé změny na snímcích, což se potvrdilo i při testování jednotlivých metod. Implementoval jsem tedy výpočet metriky SSIM indexu v programovacím jazyku C# pro detekci již navštívených obrazovek. Výpočet je implementovaný ve funkci `GetSSIM` v rámci hlavní třídy programu `Clicker`. Třída je podrobněji popsána v sekci 7.5.

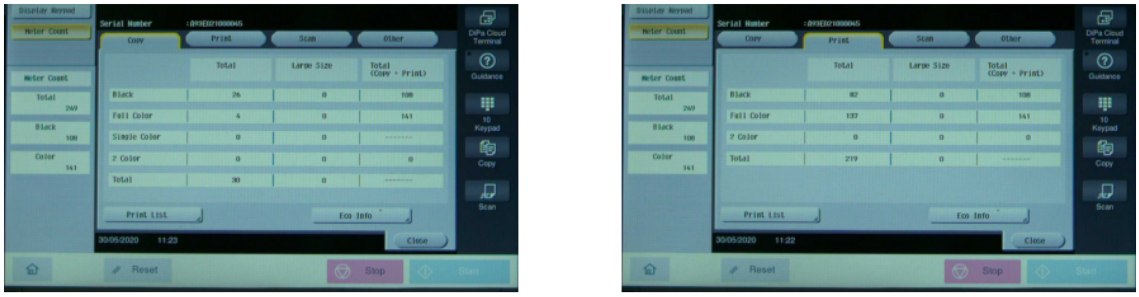
Metriku SSIM jsem implementoval na základě vzorců popsaných v kapitole 2.4. Pro výpočty nad fotografiemi jsem využil knihovnu *OpenCvSharp*, což je knihovna umožňující používat původní knihovnu OpenCV v programovacím jazyku C#. Hodnotu prahu určující, zda se jedná o stejnou obrazovku, jsem zvolil 0,91 a hodnotu 0,97 pro určení, zda robot klikl na obrazovku (zda se stylus dotkl obrazovky a zařízení to zaznamenalo). Z experimentů se ukázalo, že takto zvolené hodnoty poskytují dostatečnou rezervu pro toleranci šumu z kamery u stejných snímků, ale již při změně textu některých slov na obrazovce dokáže detekovat, že se jedná o jinou obrazovku. Na obrázku 7.2 jsou znázorněny výsledky porovnání některých snímků na základě implementované metriky SSIM. Hodnota *MinSSIM* udává nejnižší hodnotu SSIM rozdělení obrazu na 16 menších částí.

[1] https://getbootstrap.com



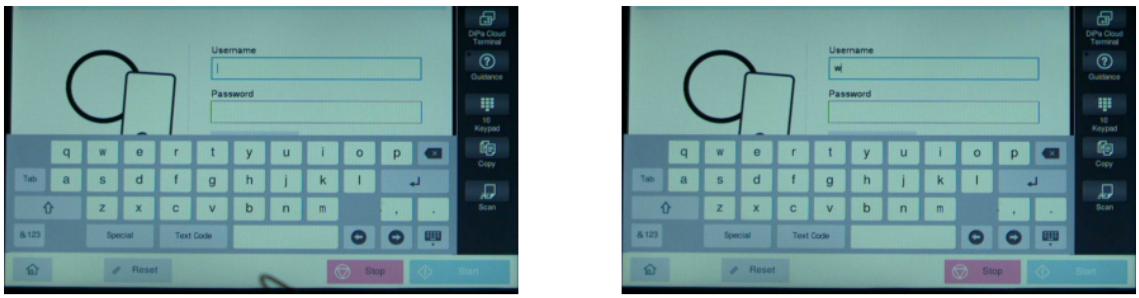
SSIM: 0.88, MinSSIM: 0.75

(a)



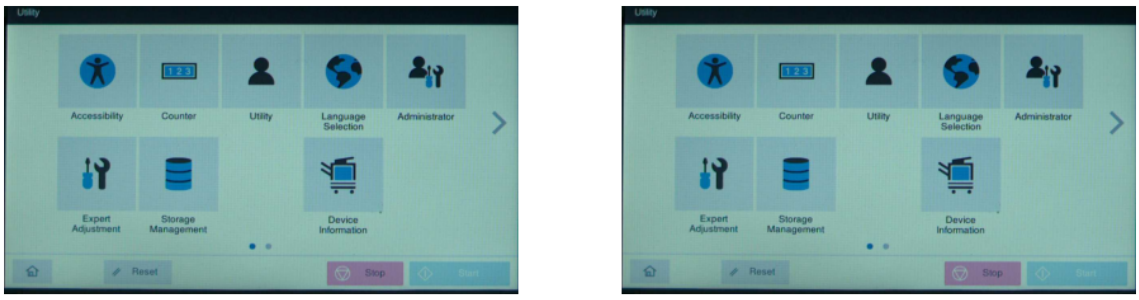
SSIM: 0.95, MinSSIM: 0.82

(b)



SSIM: 0.97, MinSSIM: 0.91

(c)



SSIM: 1.00, MinSSIM: 0.99

(d)

Obrázek 7.2: Porovnání snímků obrazovek tiskáren na základě metriky SSIM (hodnota *SSIM* je vypočítána z celé fotografii a hodnota *MinSSIM* je nejmenší z vypočítaných SSIM jednotlivých částí fotografie)

7.3 Detekce tlačítek

Detekování tlačítek je zajištěno pomocí neuronové sítě. Nejprve jsem natrénoval všechny modely popsané v sekci 3.1 na mnou vytvořených datasetech. Neuronové sítě jsem trénoval formou *cross validation*. Dataset jsem tedy rozdělil na několik částí. Jednu z nich jsem vůbec nezapojil do trénování, jelikož jsem chtěl porovnat i výsledky modelů na datech, se kterými se daný model vůbec nesetkal. Další část jsem zvolil jako testovací a zbylé části jako trénovací množinu. Tyto části jsem postupně měnil, díky čemuž se posílila schopnost neuronové sítě generalizovat naučená data a bylo snazší určit nejlepší moment pro ukončení trénování. Výsledky jednotlivých architektur jsou analyzovány v podsekci 7.3.

Převod mezi požadovanými anotačními formáty

Jednotlivé třídy modelů a jejich trénovací algoritmy využívají různé formáty anotačních souborů. Z tohoto důvodu jsem musel doimplementovat i skripty pro převod mezi těmito anotačními formáty. Skripty jsem implementoval v programovacím jazyce Python.

Můj vytvořený dataset je uložen v anotačním formátu Pascal VOC, který má xml strukturu. Implementoval jsem tedy konvertory z tohoto formátu do anotačního formátu darknet pro modely z třídy YOLO a další formát pro model SSD na základě definované syntaxe.

Faster R-CNN, SSD

Modely Faster R-CNN a SSD jsem natrénoval pomocí knihovny Tensorflow. Na stránce https://github.com/tensorflow/models/tree/master/research/object_detection, která je spravována výzkumným týmem Tensorflow, jsou k dispozici trénovací algoritmy i s několika implementovanými architekturami modelů, mezi které patří například i architektury Faster R-CNN a SSD.

YOLO, YOLOv2, YOLOv3

Modely ze třídy YOLO jsou dostupné na webu https://pjreddie.com/darknet/yolo/. Tento web spravuje autor publikací o architekturách jednotlivých modelů YOLO. Poskytuje zde také ke stažení architektury těchto modelů společně s algoritmy pro jejich natrénování a výpočet jejich predikcí. Algoritmy jsou implementovány v programovacím jazyce C.

Algoritmy pro trénování všech verzí modelů YOLO využívají specifický anotační formát. Ke každé fotografii je vygenerován speciální soubor, který na každém řádku obsahuje specifikaci jednoho ohraničujícího boxu na dané fotografii. Box je specifikován pomocí 5-ti hodnot oddělených mezerou ve formátu – „$c\ x\ y\ w\ h$“. Na prvním místě je třída objektu, který je v daném ohraničujícím boxu. Hodnoty pozice a velikosti ohraničujícího boxu jsou normalizovány do rozmezí od 0 do 1. Jsou tedy reprezentovány desetinným číslem a jsou vztaženy relativně k velikosti fotografie. Dvojice x, y reprezentují střed ohraničujícího boxu a w, h udávají šířku a výšku tohoto boxu.

RefineDet, EfficientDet

Autoři práce RefineDet[17] poskytly implementovanou architekturu tohoto modelu na svém Github účtu na https://github.com/sfzhang15/RefineDet. Model je implementován v programovacím jazyce Python s využitím frameworku Caffe.

$mAP_{0.5}$	Trénování	Dataset zařízení výrobců, kteří byli zastoupeni v trénovacím datasetu	Dataset s úplně neznámými tiskárnami
Faster R-CNN	81,9	74,4	61,4
SSD	68,1	62,9	38,9
YOLOv3	79,8	71,5	54,6
RefineDet	82,6	71,3	57.3
EfficientDet	85,4	72,8	57.8

Tabulka 7.1: Dosažené výsledky zkoumaných neuronových sítí na datasetu obsahující snímky obrazovek různých modelů tiskáren.

Architektura modelu EfficientDet je také dostupná na webové stránce Github na https://github.com/google/automl/tree/master/efficientdet. Model je implementován v programovacím jazyce Python.

Oba modely poskytují také algoritmy pro natrénování dané architektury na obecných datasetech *COCO* a *Pascal VOC*. Po menších úpravách těchto algoritmů je možné tyto modely natrénovat i na vlastním datasetu v anotačním formátu *Pascal VOC*.

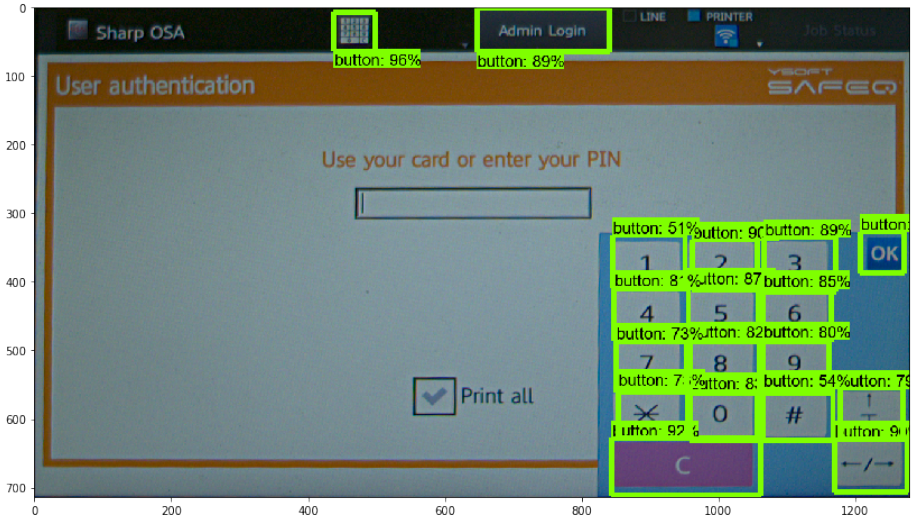
Výsledky neuronových sítí

Část datasetu, která nebyla vůbec použita pro trénování neuronových sítí, byla záměrně zvolena tak, aby obsahovala různé skupiny snímků. V první skupině snímků byly snímky obrazovek z tiskárny od výrobce, který byl zastoupen i v trénovací části datasetu, ale konkrétní řada tiskárny v trénovací části nebyla. Další skupinou byly snímky, na kterých jsou tiskárny od zcela neznámého výrobce tiskáren. V poslední skupině jsou pak snímky obrazovek z tiskáren, na kterých se neuronová síť učila.

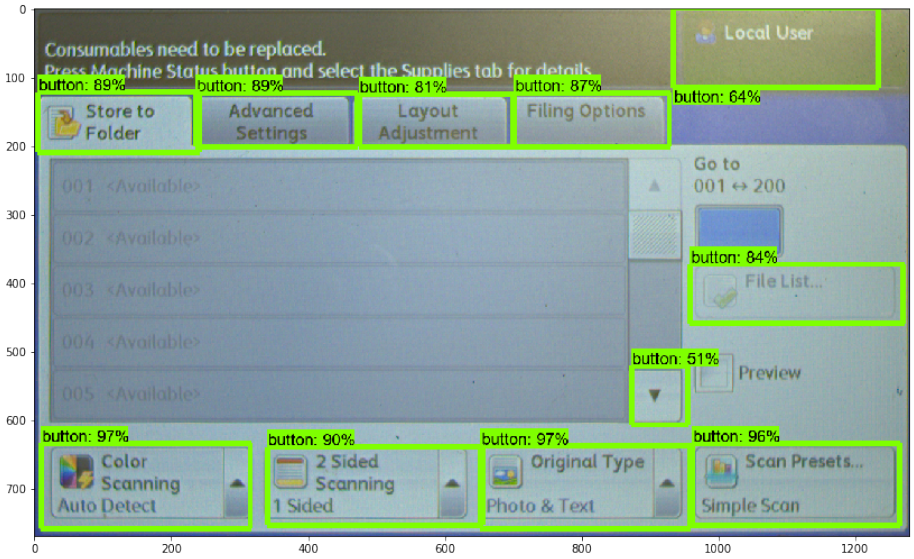
Každou z těchto kategorií jsem samostatně otestoval a vypočítal pro ni výslednou hodnotu mAP, na základě které jsem tyto modely porovnával. Výsledky jsou zobrazeny v tabulce 7.1.

Ze získaných výsledků vyplývá, že nejméně úspěšnou neuronovou sítí byl model SSD, který na každém datasetu dosáhl nejhorší úspěšnosti o poměrně značný rozdíl. Ostatní modely dosáhly podobné úspěšnosti okolo $80 mAP_{0.5}$. Na trénovací části datasetu dosáhl nejlepší úspěšnosti model EfficientDet. Na ostatních částech datasetu ji však překonal model Faster R-CNN. Jelikož se tento systém bude využívat právě pro inicializaci nových typů zařízení, se kterými se neuronová síť ještě nesetkala, rozhodl jsem se využít v systému model Faster R-CNN.

Nevýhodou tohoto modelu je však čas potřebný pro výpočet výsledku neuronové sítě, jelikož je větší než u ostatních modelů. V tomto případě to však není až takový problém. Očekává se totiž, že se systém bude spouštět přes noc, aby mohl robot proklikat všechny obrazovky a až další den se zkontroluje vytvořená navigační mapa. Tento nedostatek navíc kompenzuje větší přesností v detekci tlačítek, což je pro implementovaný systém podstatnější faktor. Detekované pozice tlačítek na snímcích obrazovky z testovacího datasetu jsou vidět na obrázku 7.3.



(a)



(b)

Obrázek 7.3: Příklady snímků obrazovek s detekovanými tlačítky pomocí neuronové sítě Faster R-CNN

Implementace do systému

Na základě získaných výsledků jsem se rozhodl zakomponovat do systému neuronovou síť Faster R-CNN. Natrénovaný model jsem převedl do univerzálního formátu s názvem *protobuf* (.pb). Pro výpočet výstupu neuronové sítě v programovacím jazyce C# jsem využil framework *TensorFlowSharp*. Tento framework umožňuje pracovat s již vytvořenými modely v tomto programovacím jazyce. Primárně je určen pouze na výpočet výstupu neuronových sítí, ale umožňuje i jejich dodatečné dotrénování.

Po načtení natrénovaného modelu neuronové sítě je převedena vstupní fotografie z datového typu `Mat` do formátu požadovaného na vstupu neuronové sítě na základě implemen-

tovaných utilit. Pomocí funkce `sessionRunner.AddInput().Fetch()` se provede výpočet nad načteným modelem a funkce vrátí výstup sítě.

Výstupem jsou detekované pozice v podobě pole. První položka obsahuje detekované pozice. Pozice jsou uchovány ve formě $[x_1, y_1, x_2, y_2]$, kde $[x_1, y_1]$ jsou souřadnice levého horního rohu a $[x_2, y_2]$ jsou souřadnice udávající polohu pravého spodního rohu tlačítka. Pro každou nalezenou pozici je uvedena i třída objektu (druhá položka pole), který se na dané pozici pravděpodobně nachází, a hodnota jak moc si je síť jistá touto predikcí (třetí položka pole).

Algoritmus vybere jen pozice, které dosáhly skóre pravděpodobnosti větší než 50%. Tento výstup je následně převeden na pole tlačítek, které je tvořeno třídami `ButtonModel`. Instancím jsou nastaveny výchozí hodnoty, jako jsou například stav, který je nastaven na `New`, atribut `Checked` je nastaven na `false` a další. Na základě výstupu sítě jsou nastaveny vypočítané pozice jednotlivých tlačítek a každému z nich je přiřazena obrazovka, na které bylo tlačítko detekováno. Takto vytvořené pole je pak následně předáno na další zpracování a je přiřazeno do `ScreenModel` obrazovky, pro kterou byla tato tlačítka detekována.

7.4 Databáze

Databázi jsem implementoval na základě přístupu code first. Tento způsob spočívá ve vytvoření tříd s atributy, které reprezentují sloupce v databázi. Atributy se speciálními vlastnostmi, jako například primární nebo cizí klíč, se označí speciální anotací. Pomocí knihovny Entity framework se na základě vytvořených tříd a jejich atributů vygeneruje migrace, která se následně převede na sekvenci SQL příkazů. Sekvence příkazů se následně provede nad zvolenou databází a vytvoří se požadované tabulky. Tabulky jsem ukládal do databázového systému PostgreSQL.

Code first přístup také zjednodušuje práci s načítáním dat z databáze. Díky tomu je možné načíst uložená data z databáze přímo do vytvořených tříd v programu a ihned je používat. Převod mezi modely zajišťují mapovací funkce, které upraví potřebné atributy a případně je i přetypují. Modely a mapovací funkce jsem implementoval například pro entitu reprezentující vlastnosti obrazovky. V databázi je uložena pouze cesta k fotografii obrazovky. Díky implementovanému modelu a mapovací funkci je možné převést načtenou entitu z databáze přímo do modelu, který již bude obsahovat fotografii snímku obrazovky.

Implementoval jsem tedy 6 tříd na základě návrhu popsaného v sekci 6.5 s požadovanými atributy. Vytvořil jsem model pro tabulku `Screen`, který je využíván v programu při analýze obrazovek. Při načtení entity z databáze se převede entita `screen` na korespondující model. Mapovací funkce načte obrázek na základě cesty k souboru z databáze a převede jej do datového typu `Mat`, jelikož se v projektu pracuje s knihovnou OpenCV. V případě, že se ukládá model s fotografií obrazovky do databáze, převede se model na entitu. Při převádění pomocí mapovací funkce se mimo jiné uloží zachycený snímek obrazovky do úložiště počítače a do entity se místo snímku obrazovky uloží pouze cesta k uloženému snímku.

Dále jsem implementoval repozitáře, které zajišťují komunikaci mezi databází a vytvořenými třídami entit. Implementoval jsem tedy třídy `ClickPathRepository`, `ButtonRepository`, `ScreenRepository` a `GUIRepository`. Každá z nich se stará o komunikaci mezi korespondující tabulkou. Obsahují metody pro uložení dat do databáze, úpravu existujících záznamů a několik výběrových příkazů, jako jsou například výběr všech záznamů z dané tabulky nebo výběr záznamu na základě identifikačního čísla.

7.5 Prohledávání obrazovek vestavěného zařízení

Logika proklikávání je implementována v programovacím jazyce C#. Je zakomponována do třídy `Clicker`, která obsahuje mimo jiné i implementaci metriky SSIM na určení podobnosti obrazovek popsané v sekci 7.2. Dále je zde implementována hlavní logika algoritmu proklikávání obrazovek vestavěného zařízení vycházející z algoritmu pohledávání do hloubky s modifikací popsanou v návrhové části v sekci 6.6.

Pro spuštění proklikávání stavového prostoru vestavěného zařízení se využívají metody `AutoStep` nebo `NextStep`. Obě metody jsou téměř stejné. Jediný rozdíl je, že funkce `NextStep` po analýze aktuální obrazovky skončí a vrátí aktuální stav vygenerované mapy. Tato funkce se tedy primárně využívá pro krokování vytvořeného algoritmu a případně hledání chyb v jeho logice. Naproti tomu funkce `AutoStep` automaticky analyzuje a proklikává jednotlivé obrazovky, na kterých se zrovna nachází, dokud nezpracuje všechny dosažitelné obrazovky vestavěného zařízení.

Implementovaný algoritmus začíná vždy z domovské obrazovky (home screen). Z tohoto důvodu je nutné v každé iteraci cyklu zajistit, aby se grafické rozhraní vestavěného zařízení dostalo z jakékoli obrazovky na domovskou obrazovku. Většina vestavěných zařízení mají pro tuto funkci zabudované hardwarové tlačítko nebo je přímo na obrazovce speciální tlačítko, díky kterému se lze dostat na domovskou obrazovku. Toto tlačítko bývá umístěno vždy na stejné pozici. Přístup k domovské obrazovce je uložen a musí být předem zadán do tabulky s prerekvizitami v podobě posloupnosti akcí, které je nutné provést. Typicky však bývá potřebná pouze jedna akce – kliknutí na určitou pozici.

Plně automatický běh proklikávání obsahuje hlavní cyklus, který je prováděn, dokud nejsou proklikány všechny tlačítka na všech nalezených obrazovkách. Tento cyklus obsahuje ještě jeden vnořený cyklus, který je opakován, dokud na aktuální obrazovce stále existují tlačítka, která nemají status `Done`. Zajišťuje tedy, aby se algoritmus dostal na „koncovou“ obrazovku, na které se nachází pouze tlačítka se stavem `Done`.

V každé iteraci vnořeného cyklu se vybere tlačítko z aktuální obrazovky, které má nastavený status na `Analyzing`. Jestliže na dané obrazovce ještě není žádné analyzované tlačítko, vybere se první tlačítko ze seznamu dostupných tlačítek pro danou obrazovku a nastaví se jeho status na `Analyzing`. Následováním tlačítek se stavem `Analyzing` se robot dostane až k obrazovce, která byla analyzována v předchozím běhu. Po vybrání tlačítka zašle algoritmus instrukce robotickému rameni, aby klikl na příslušnou pozici, na které se nachází analyzované tlačítko. Toto tlačítko se také přidá do seznamu `ClickPath`, aby mohl být po dokončení vnitřního cyklu analyzován tento běh a upravil se stav u jednotlivých tlačítek.

Po kliknutí na tlačítko se pomocí kamery získá aktuální snímek obrazovky. Na základě metriky SSIM se porovná se snímkem předchozí obrazovky a na základě toho se rozhodne, zda se jedná o stejnou obrazovku. V takovém případě se jedná o tlačítko, které způsobí pouze změnu na dané obrazovce (typicky psací/klávesnicové tlačítko). V opačném případě se jedná o tlačítko, které někam odkazuje. Na základě metriky SSIM se tedy porovnají všechny dosud nalezené snímky obrazovek a nalezne se obrazovka s největším SSIM skóre, která je větší než předem definovaná konstanta. Pokud SSIM skóre nepřesahuje definovanou konstantu, vytvoří se nová instance třídy `Screen`. Instance se přidá do navigační mapy a pomocí neuronové sítě se na ní detekují tlačítka. Následně se stisknutému tlačítku nastaví vazba na příslušnou obrazovku a jeho typ. Zkontroluje se, zda se algoritmus nedostal na obrazovku, která se již nachází v `ClickPath` a pokud ano, přestane se provádět zanořený cyklus. Díky tomu algoritmus detekuje smyčky v grafickém rozhraní a nezacyklí se. V opačném případě

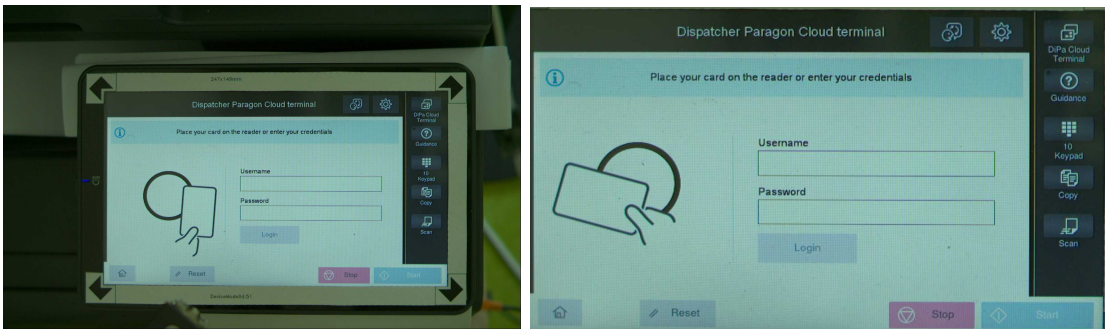
algoritmus iteruje v zanořeném cyklu, dokud se na aktuálně zpracovávané obrazovce v dané iteraci nachází dosud nezpracovaná tlačítka.

Po dokončení vnořeného cyklu, algoritmus upraví stav stisknutých tlačítek v `ClickPath` a obrazovek, na kterých se daná tlačítka vyskytují. Seznam `ClickPath` je procházen od konce. Poslednímu tlačítku je vždy nastaven status na `Done`. U dalších tlačítek v pořadí se tento stav nastaví pouze v případě, že mají ostatní tlačítka na obrazovce, na které dané tlačítko odkazuje, nastaven stav `Done`. To znamená, že všechna tlačítka na obrazovce byla již analyzována a obrazovky, na které odkazují, mají status `Done`. Proto již není nutné tyto obrazovky navštěvovat a dále analyzovat.

Pomocí předem načtené posloupnosti akcí se robotické rameno prokliká na domovskou obrazovku a výše popsaný algoritmus, který se nachází v hlavním cyklu, se provede pro všechny dosud nezpracované obrazovky a tlačítka.

Komunikace s robotickým ramenem je zajištěna pomocí Nuget package `Ysoft.RQA.RemoteControl`. Robotické rameno je ovládáno pomocí funkce `TapCoordinates`, které jsou předány argumenty adresa robotického ramene a souřadnice x, y, z, na které má kliknout. Souřadnice jsou zadány v milimetrech od horního rohu obrazovky. Výstupem neuronové sítě jsou normalizované souřadnice do rozmezí od 0 do 1. Jsou tedy dány desetinným číslem vztaženým relativně k velikosti fotografie. Z toho důvodu je nutné nalezené pozice tlačítek neuronovou sítí ještě vynásobit rozměry obrazovky vestavěného zařízení. Rozměry obrazovky je nutné předem zadat pro správnou funkčnost systému. Souřadnice z je při klikání na displej nastavena na hodnotu 0, jelikož se stačí displeje pouze dotknout. Při klikání na hardwarové tlačítko se nastaví souřadnice z na zápornou hodnotu, jelikož je nutné „prokliknout“ dané tlačítko.

Obraz z kamery je zprostředkován funkcemi z Nuget package `Ysoft.RQA.Camera`. Funkce vrací přímo snímek obrazovky. Je tedy zajištěna detekce obrazovky z fotografie a příslušné oříznutí obrazu. Obrazovka je detekována díky markerům, které jsou nalepeny na okraji obrazovky (viz obrázek 7.4).



(a) Neupravený pohled z kamery

(b) Oříznutý snímek (a) na základě markeru okolo displeje tiskárny

Obrázek 7.4: Ukázka snímků obrazovky tiskárny pořízené kamerou

7.6 Zobrazení výsledků

Webovou aplikaci pro zobrazení výsledků jsem také implementoval v jazyce C#. Využil jsem k tomu frameworku DotVVM. Na frontendovou část webu jsem použil knihovnu Bo-

otstrap[2], která poskytuje responzibilitu a alespoň základní úpravu vzhledu jednotlivých HTML komponent. Dále jsem využil javascript knihovnu Vis.js[3] pro zobrazení grafu navigační mapy. Pro vytvoření grafu navigační mapy jsem využil dynamického navázání na komponenty z backendové části aplikace. To bylo nutné, jelikož se graf navigační mapy vytváří na základě dat, které se postupně mění tím, že se doplňují nově nalezené obrazovky. Pro toto navázání jsem využil javascriptového frameworku Knockout.js[4].

Framework DotVVM je založen na konceptu MVVM – Model View ViewModel. Jak již název napovídá, lze jej rozdělit do tří částí. První část, pojmenované jako Model, obsahuje popis dat a samotná data, se kterými aplikace pracuje. V tomto případě tato část odpovídá implementovaným modelům a entitám popsaných v sekci 7.4. Další část se jmenuje View, která představuje frontendovou část aplikace – její vzhled. Frontendová část se ve frameworku DotVVM implementuje pomocí jazyka DOThtml. Zajišťuje správné zobrazení a naformátování dat, které jsou na ni navázány. Poslední částí je ViewModel, který propojuje obě části dohromady. Určuje, jaká data se mají na stránce zobrazit, definuje funkce jednotlivých tlačítek a tak dále. Tyto vlastnosti jsou zajištěny pomocí takzvaného bindingu. Změna hodnoty ve ViewModelu se hned projeví i změnou v části View nebo se kliknutí na tlačítko na webové stránce převede pomocí této technologie na volání navázané funkce.

Nově jsem tedy musel implementovat části View a ViewModel. Jelikož se webová aplikace bude skládat pouze z jedné webové stránky, implementoval jsem pouze jedno View. Dodržoval jsem návrh webu popsaný v sekci 6.7. Navigační mapa se tedy vygeneruje pomocí knihovny Vis.js. V implementovaném konfiguračním skriptu je určen vzhled uzlů, hran a typ grafu – `Network`. U vzhledu jsem hlavně upravil zobrazování náhledu obrazovky v grafu a zobrazení hran jako orientované. Dále jsem doimplementoval chování pro kliknutí na uzel pomocí javascript události `OnClick`, aby se zobrazily dodatečné informace o konkrétní obrazovce společně s její fotografií a s detekovanými tlačítky.

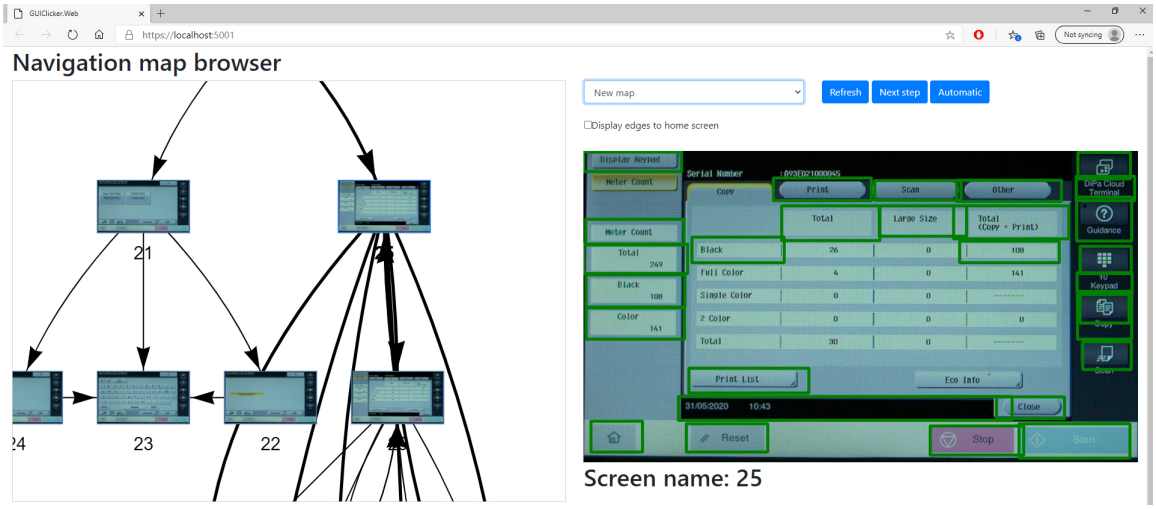
Viewmodel je navázán na třídu `Clicker` popsanou v sekci 7.5. Tlačítka jsou navázána na metody `NextStep` a `AutoStep`, které pouze volají stejnojmenné metody z třídy `Clicker`. Díky nim je ovládán algoritmus proklikávání. Dále ViewModel obsahuje atributy `Nodes` a `Edges` v potřebném formátu pro framework Vis.js společně s metodami převádějícími aktuální stav navigační mapy z databáze (případně třídy `Clicker`) do podporovaného formátu ve ViewModelu. Výsledný vzhled implementované webové aplikace je vidět na obrázku 7.5.

7.7 Testování celkové funkcionality systému

Při testování celkové funkcionality jsem měl omezené možnosti využití robotické ruky. Většina robotů byla využita na prioritnější operace (testování nových aktualizací systému YSoft SafeQ). Dále v období Covid-19 pracovala většina pracovníků z domova a využívala roboty na vzdálené ovládání tiskáren. Proto jsem v rámci diplomové práce provedl testování kompletního systému pouze na 4 tiskárnách.

Kompletní systém byl testován na tiskárnách *Konica Minolta C3350i, Ricoh MP C3004, Xerox AltaLink C8055, Sharp MX 3060N*. Snímky obrazovek z modelů *Konica Minolta C3350i, Ricoh MP C3004, Sharp MX 3060N* nebyly využity k trénování detektoru tlačítek, ale při trénování byly využity snímky obrazovek tiskáren od stejného výrobce. Obrazovky z tiskárny *Xerox AltaLink C8055* spadaly do části datasetu určeného pro trénování.

[2] https://getbootstrap.com/
[3] https://visjs.org/
[4] https://knockoutjs.com/

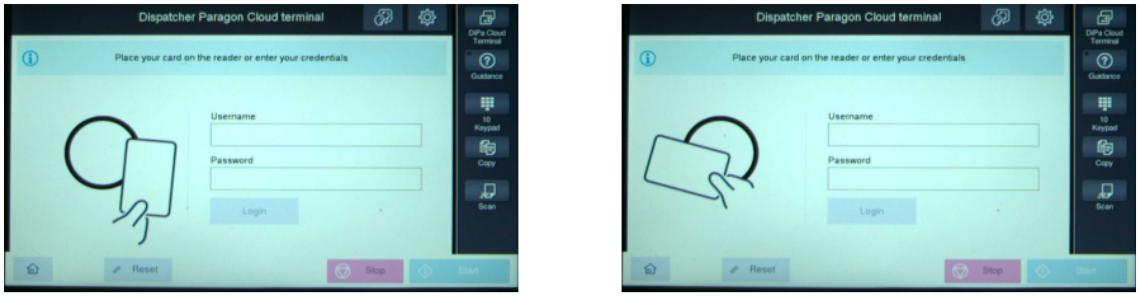


Obrázek 7.5: Vzhled webového rozhraní pro komunikaci s implementovaným systémem

Při testování jsem se primárně zaměřil na schopnost detekovat tlačítka. Úspěšnost zvolené neuronové sítě je 74,4 $mAP_{0.5}$ pro tiskárny od výrobců, na kterých byla síť trénována. Podrobnější výsledky detektoru jsou uvedeny v kapitole 7.3. Dále jsem posuzoval úspěšnost porovnávání jednotlivých obrazovek. Zvolená metrika SSIM se osvědčila a správně detekovala všechny obrazovky. Jediný problém nastal u porovnání obrazovek s klávesnicí. Po kliknutí na tlačítko klávesnice s nevýrazným písmenem (například „i“ nebo „l“) byla vypočítaná hodnota SSIM mimo zvolený interval a kliknutí nebylo zaznamenáno. Příklad porovnání tohoto typu obrazovek je vidět na obrázku 7.6. U tiskárny *Konica Minolta C3350i* nastal ještě problém u přihlašovací obrazovky, jelikož na ní byl animovaný obrázek. Animace znemožnila správně určit podobnost mezi obrazovkami, jelikož záleželo, v jakém momentu systém pořídil snímek obrazovky. Tato situace je vidět na obrázku 7.7 společně s vypočítanou hodnotou SSIM. V rámci testování jsem tento problém vyřešil manuálním přihlášením a následně spuštěním systému. Všechny tiskárny obsahovaly přihlašovací obrazovku. Bylo zde tedy možné otestovat i správnou funkcionalitu manuálně nastavených prerekvizit pro přihlášení.

Obrázek 7.6: Porovnání snímku obrazovky po zadání písmene „i“ pomocí metriky SSIM

Grafické rozhraní tiskáren obsahovalo v průměru 246 obrazovek. Do průměru nejsou započítány jazykové mutace jednotlivých obrazovek. Před prvním během byla systému předem nastavena pouze pozice tlačítka zajišťující navigaci na domácí obrazovku a instrukce



SSIM: 0.95, MinSSIM: 0.70

Obrázek 7.7: Porovnání snímku obrazovky s animací pomocí metriky SSIM

pro přihlášení do tiskárny. Pouze díky těmto informacím dokázal systém odhalit v průměru 192 obrazovek. Zbylé obrazovky neodhalil, jelikož nedetekoval tlačítka, která odkazovala na obrazovky v daném podstromu navigační mapy. Tento počet je však velmi ovlivněn tím, na které obrazovce síť nedetekovala určité tlačítko. V případě, že síť nedetekovala tlačítko na úvodní obrazovce, velká část grafického rozhraní nebyla prozkoumána.

Robotické rameno kliká vždy na střed nalezené pozice tlačítka. Díky tomu nedocházelo ke špatným kliknutím na tlačítko. Tedy v případě i méně přesné detekce pozice tlačítka, se robotické rameno trefí na tlačítko a systém může dále pokračovat.

Kapitola 8

Závěr

Cílem této práce bylo vytvořit systém, který postupně uloží snímky všech dosažitelných obrazovek, pozice tlačítek, které se na nich vyskytují, a také na které obrazovky tato tlačítka odkazují. Na základě získaných dat následně vytvořit navigační mapu grafického prostředí vestavěného zařízení. Navigační mapa slouží k automatické inicializaci robota, který automaticky testuje dané zařízení. Tento cíl byl splněn. Nejprve bylo nutné nastudovat literaturu související s detekcí a lokalizací tlačítek z fotografie obrazovky. Získané informace jsem využil při řešení této práce. Některé metody a postupy jsem popsal v podkapitole 3.1. Algoritmus prohledávání obrazovek jsem popsal v podkapitole 6.6. Vycházel jsem z nastudované literatury prohledávání stavového prostoru popsané v kapitole 4.

Implementovaný systém je složen z několika funkčních komponent – detekce tlačítek, algoritmus prohledávání obrazovek a detekce již analyzovaných obrazovek.

Komponenta pro automatickou detekci tlačítek využívá neuronovou síť, která určí, na kterých pozicích se pravděpodobně tlačítka nachází. Díky schopnosti generalizace u neuronových sítí je možné, aby systém detekoval i tlačítka na snímcích, na kterých nebyl trénován. Další výhodou je jednoduché dodatečné dotrénování sítě v případě, že síť chybně detekovala některá tlačítka. To je důležité pro využití systému v reálném prostředí a na různých zařízeních. Z natrénovaných neuronových sítí dosáhla nejlepších výsledků architektura Faster R-CNN. Přesné výsledky trénovaných sítí jsou popsány v kapitole 7.3.

Neuronové sítě jsem trénoval na ručně vytvořeném datasetu snímků tiskáren. Pro zlepšení schopnosti generalizace neuronových sítí jsem navíc implementoval aplikaci, která automaticky rozšíří dataset o nové typy tlačítek, které získává z webových stránek. Aplikace na rozšíření datasetu o nové typy tlačítek je implementována jako rozšíření do prohlížeče Google Chrome. Aplikace projde webové stránky v dané doméně. Na každé webové stránce pořídí snímek obrazovky a z html struktury vytvoří anotační soubor se zaznačenými pozicemi tlačítek.

V budoucnu bude implementovaný systém integrován do balíku funkcí robotického ramene a do webového informačního systému, pomocí kterého se kontroluje stav robota a ovládá se robotické rameno. Díky tomu bude snadnější spouštět mnou implementovaný systém pro počáteční inicializaci robota. Integrace do webového systému také umožní využít již existující funkce, které se používají pro úpravu atributů obrazovek a tlačítek.

Literatura

[1] *Google chrome extension* [online]. 2018 [cit. 2019-12-28]. Dostupné z: https://developer.chrome.com/extensions/overview.

[2] Bay, H., Ess, A., Tuytelaars, T. a Van Gool, L. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.* USA: Elsevier Science Inc. červen 2008, roč. 110, č. 3, s. 346–359. Dostupné z: https://doi.org/10.1016/j.cviu.2007.09.014. ISSN 1077-3142.

[3] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*. Apr 1980, roč. 36, č. 4, s. 193–202. Dostupné z: https://doi.org/10.1007/BF00344251. ISSN 1432-0770.

[4] Girshick, R. Fast R-CNN. In: *The IEEE International Conference on Computer Vision (ICCV)*. December 2015.

[5] Girshick, R. B., Donahue, J., Darrell, T. a Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*. 2013, abs/1311.2524. Dostupné z: http://arxiv.org/abs/1311.2524.

[6] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. et al. SSD: Single Shot MultiBox Detector. In: Leibe, B., Matas, J., Sebe, N. a Welling, M., ed. *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, s. 21–37. ISBN 978-3-319-46448-0.

[7] Lowe, D. G. Object recognition from local scale-invariant features. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. 1999, s. 1150–1157 vol.2.

[8] Poobathy, D. a Chezian, R. Edge Detection Operators: Peak Signal to Noise Ratio Based Comparison. *International Journal of Image, Graphics and Signal Processing*. Září 2014, roč. 6, s. 55–61.

[9] Redmon, J., Divvala, S. K., Girshick, R. B. a Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. 2015, abs/1506.02640. Dostupné z: http://arxiv.org/abs/1506.02640.

[10] Redmon, J. a Farhadi, A. YOLO9000: Better, Faster, Stronger. *CoRR*. 2016, abs/1612.08242. Dostupné z: http://arxiv.org/abs/1612.08242.

[11] Redmon, J. a Farhadi, A. YOLOv3: An Incremental Improvement. *CoRR*. 2018, abs/1804.02767. Dostupné z: http://arxiv.org/abs/1804.02767.

[12] Ren, S., He, K., Girshick, R. a Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M. a Garnett, R., ed. *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, s. 91–99. Dostupné z: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf.

[13] Tan, M. a Le, Q. V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *CoRR*. 2019, abs/1905.11946. Dostupné z: http://arxiv.org/abs/1905.11946.

[14] Tan, M., Pang, R. a Le, Q. V. *EfficientDet: Scalable and Efficient Object Detection*. 2019. Dostupné z: https://arxiv.org/pdf/1911.09070v1.pdf.

[15] Wang, Z., Bovik, A. C., Sheikh, H. R. a Simoncelli, E. P. Image Quality Assessment: From Error Visibility to Structural Similarity. *Trans. Img. Proc.* IEEE Press. duben 2004, roč. 13, č. 4, s. 600–612. Dostupné z: https://doi.org/10.1109/TIP.2003.819861. ISSN 1057-7149.

[16] Zbořil, F. a Zbořil, F. *Základy umělé inteligence*. 2012. Studijní opora, 5. vydání.

[17] Zhang, S., Wen, L., Bian, X., Lei, Z. a Li, S. Z. Single-Shot Refinement Neural Network for Object Detection. *CoRR*. 2017, abs/1711.06897. Dostupné z: http://arxiv.org/abs/1711.06897.