VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

DETEKCE POVRCHOVÝCH VAD A ZNAČENÍ SMD SOUČÁSTEK

VISUAL SURFACE INSPECTION OF SMD COMPONENTS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Adrián Zsidek

AUTHOR

VEDOUCÍ PRÁCE — Ing. Peter Honec, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Adrián Zsidek ***ID:*** 221031

Ročník: 2 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Detekce povrchových vad a značení SMD součástek

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit detektor a klasifikátor laserového značení SMD tantalových kondenzátorů na balícím stroji. K dispozici je obrazová databáze součástek.

1. Seznamte se se základy zpracování obrazu, statistického a strukturního rozpoznávání a strojového učení.
2. Proveďte rešerši metod vhodných pro klasifikaci obrazu vzhledem k charakteru pořízených dat z reálné lokality.
3. Anotujte dodaný dataset snímků obsahujících povrchové defekty a potisk.
4. Vytvořte vhodné rozhraní pro knihovnu (DLL) detektoru a klasifikátoru.
5. Otestujte vhodné metody pro detekci povrchových vad a klasifikaci potisku s přihlédnutím k provozu v reálném čase.
6. Zhodnoťte spolehlivost.

DOPORUČENÁ LITERATURA:

HLAVAC V., SONKA M., BOYLE R.: Image Processing, Analysis, and Machine Vision, ISBN 978-0495082521

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 5.8.2024

Vedoucí práce: Ing. Peter Honec, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

Abstrakt

Táto diplomová práca popisuje vývoj komplexného systému výstupnej vizuálnej kontroly SMD tantalových kondenzátorov. Využíva sa strojového učenia aj tradičných prístupov, ich kombináciou sa vyhodnocujú povrchové vady a popis. Porovnáva a vyhodnocuje použité metódy, s ohľadom na presnosť aj rýchlosť vyhodnotenia. Uceluje vyvinuté algoritmy do knižnice a demonštruje ich funkčnosť pomocou grafického rozhrania.

Klíčová slova

spracovanie obrazu, detekcia objektov v reálnom čase, detekcia vád, YOLO, segmentácia objektov, rozpoznávanie znakov,

Abstract

This master's thesis describes a complex detection system for visual final inspection of SMD components. It makes use of both machine learning and traditional methods, whose combination it uses to detect surface faults and process the print. Compares and evaluates the used methods, considering their accuracy and speed of execution. Encloses these algorithms in a library and demonstrates their function using a graphical interface.

Keywords

image processing, real-time object detection, fault detection, YOLO, object segmentation, optical character recognition

Bibliografická citace

ZSIDEK, Adrián. *Detekce povrchových vad a značení SMD součástek*. Brno, 2024. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/160136. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Peter Honec.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	**Adrián Zsidek**
VUT ID studenta:	**221031**
Typ práce:	**Diplomová práce**
Akademický rok:	**2023/2024**
Téma závěrečné práce:	**Detekce povrchových vad a značení SMD součástek**

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dnes: 5. sprna 2024 ----------

podpis autora

Poděkování

Ďakujem vedúcemu diplomovej práce Ing. Petrovi Honcovi Ph.D. za účinnú metodickú, pedagogickú, a odbornú pomoc a ďalšie cenné rady pri spracovaní mojej diplomovej práce. Ďakujem pánovi Ing. Pavlovi Škodovi za ochotu odpovedať na moje otázky a potreby ohľadom datasetu.

V Brně dne: 5. sprna 2024 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ **9**

ÚVOD **11**

1. ROZBOR ÚLOHY **12**

1.1 CIEĽ PRÁCE 12
1.2 DATASET 12
1.2.1 Druhy vád kondenzátorov *12*
1.2.2 Pracovná verzia datasetu *15*
1.3 ROZDELENIE NA DIELČIE ÚLOHY 15
1.3.1 Vývoj algoritmov *15*
1.3.2 Implementácia do DLL knižnice *16*
1.3.3 Grafické rozhranie *16*

2. ZÁKLADY SPRACOVANIA OBRAZU **17**

2.1 OBRAZ Z POHĽADU POČÍTAČOVÉHO VIDENIA 17
2.1.1 Digitálny obraz a reťazec spracovania obrazu *17*
2.2 PREDSPRACOVANIE - ZÁKLADNÉ ÚPRAVY OBRAZU 17
2.2.1 Jasové a geometrické transformácie *18*
2.2.2 Konvolúcia *18*
2.2.3 Vyhladzovanie šumu *19*
2.2.4 Detekcia hrán *19*
2.3 SEGMENTÁCIA 21
2.3.1 Segmentácia prahovaním *21*
2.3.2 Segmentácia z obrazu hrán *21*
2.3.3 Segmentácia založená na regiónoch *23*

3. SEGMENTÁCIA KONDENZÁTOROV **24**

3.1 TESTY A „NAIVNÉ“ RIEŠENIE 24
3.1.1 Testy filtrov šumu *24*
3.1.2 Testy hranových filtrov *26*
3.1.3 Vyhodnotenie stĺpcových súčtov *26*
3.2 SEGMENTÁCIA PRAHOVANÍM OBRAZU 28
3.3 SEGMENTÁCIA VYUŽITÍM ZNALOSTÍ O POLOHE 29
3.3.1 Znalosti *29*
3.3.2 Popis algoritmu *30*
3.3.3 Poznámky k algoritmu *32*
3.4 DOLADENÝ SEGMENTAČNÝ ALGORITMUS 33
3.4.1 Parametre *35*
3.4.2 Zhodnotenie algoritmu *35*

4. DETEKCIA POVRCHOVÝCH VÁD **36**

4.1 NEURÓNOVÁ SIEŤ YOLO 36
4.1.1 YOLO(v1) *36*
4.1.2 YOLOv8 ..... *37*
4.1.3 Vyhodnocovacie metriky [14] *38*

4.2 UČENIE NA DATASETE SMD500 ... 39
4.2.1 Značkovanie a príprava datasetu ... *39*
4.2.2 Používanie YOLOv8 ... *41*
4.2.3 YOLOv8 nano s smd500 ... *42*
4.2.4 YOLOv8 medium s smd500 ... *43*
4.3 UČENIE NA DATASETE SMD1500 ... 44
4.3.1 Rozšírenie datasetu ... *44*
4.3.2 YOLOv8 nano ... *45*
4.3.3 YOLOv8 medium ... *46*
4.3.4 YOLOv8 nano so zväčšeným vstupným obrázkom ... *48*
4.3.5 YOLOv8 nano s upravenou augmentáciou ... *49*
4.4 ZHODNOTENIE VÝSLEDKOV ... 51
4.4.1 Úspešnosť ... *51*
4.4.2 Návrhy na zlepšenie ... *53*
4.4.3 Rýchlosť ... *53*

5. VYHODNOCOVANIE POPISU ... 54

5.1 POKUSY S MASKOVANÍM ... 54
5.1.1 Tvorenie masky ... *54*
5.1.2 Aplikovanie masky a zhodnotenie ... *55*
5.2 POROVNÁVANIE VZOROV ... 56
5.2.1 ImageInfo ... *56*
5.2.2 Popis algoritmu ... *57*
5.2.3 Obmedzenia algoritmu ... *58*
5.2.4 Vyhodnotenie úspešnosti ... *59*
5.2.5 Navrhované vylepšenia algoritmu ... *61*
5.3 ALTERNATÍVNY ALGORITMUS SEGMENTÁCIE POPISU ... 61

6. KNIŽNICA DLL A GRAFICKÉ ROZHRANIE ... 62

6.1 KNIŽNICA SMDCLASSIFIERLIB ... 62
6.1.1 Trieda smdSample ... *62*
6.1.2 Trieda yoloInstance ... *63*
6.2 UŽÍVATEĽSKÉ ROZHRANIE SMDCLASSIFIERCLIENT ... 63
6.2.1 Ovládanie ... *64*

7. ZÁVER ... 65

LITERATURA ... 66

SEZNAM OBRÁZKŮ

1.1 Ukážka špiny 12
1.2 Ukážka praskliny 13
1.3 Ukážka odštiepku 13
1.4 Ukážka diery 13
1.5 Ukážka ulomeniny 14
1.6 Ukážka vád popisu 14
2.1 Reťazec spracovania obrazu [2] 17
2.2 Grafické znázornenie konvolúcie obrazu s maskou [2] 18
2.3 Porovnanie filtrov šumu: priemerovací a Gaussov [2] 19
2.4 Porovnanie detektorov hrán 1. derivácie [6] 20
2.5 Laplacov operátor štvor a osem-okolia [6] 20
2.6 Jednoduché prahovanie [7] 21
2.7 Prahovanie obrazu hrán [7] 22
2.8 Príklad vývoja aktívnych kontúr [7] 22
2.9 Demonštrácia level-set [7] 22
2.10 Ukážka sledovania hranice [7] 23
2.11 Segmentácia štiepením a spojovaním [8] 23
3.1 Porovnanie filtrov šumu na stĺpcových súčtoch 25
3.2 Porovnanie stĺpcového súčtu s a bez filtru šumu 26
3.3 Porovnanie hranových filtrov na stĺpcových súčtoch 27
3.4 Ukážka nejasností ohľadom správnej hrany 27
3.5 Znázornenie postupu pri segmentácii prahovaním. 28
3.6 Výsledky metódy segmentácie prahovaním. 29
3.7 Ohraničenie oblasti v ktorej sa vyskytujú zvislé hrany kondenzátorov 30
3.8 Dielčie kroky orezania bočných oblastí obrázkov 31
3.9 Dielčie kroky orezania vrchných a spodných častí obrázkov 32
3.10 Výrazne posunuté kondenzátory 33
3.11 Chyba segmentácie 33
3.12 Medzivýsledky upraveného algoritmu segmentácie 34
3.13 Napravená segmentácia 34
4.1 Prístup zjednotenia v YOLO[8] 36
4.2 Prístup zjednotenia v YOLO[11] 37
4.3 Architektúra YOLOv8 [13] 38
4.4 Intersection over union 38
4.5 MATLAB Image Labeler 40
4.6 Matica zámen 1 42
4.7 Matica zámen 2 43
4.8 Krivka precision-recall 2 44
4.9 Precíznosť pri značkovaní 45
4.10 Krivka precision-recall 3 45
4.11 Matica zámen 3 46
4.12 Krivka precision-recall 4 47
4.13 Matica zámen 4 47
4.14 Matica zámen 5 48
4.15 Krivka precision-recall 5 49
4.16 Matica zámen 6 50

4.17 Krivka precision-recall 6 50
4.18 Krivka úplnosti 3 51
4.19 Potenciálne chyby pri značkovaní 1 52
4.20 Potenciálne chyby pri značkovaní 2 52
4.21 Potenciálne chyby pri značkovaní 3 53
5.1 Priemerný obrázok a jeho prahovanie 54
5.2 Identifikácia polohy znakov pre riadok 1 55
5.3 Ukážka úspešného a neúspešného maskovania 55
5.4 Prahovaný vzorok a hľadaný znak 57
5.5 Výsledok spracovania popisu 58
5.6 Vysoká zhoda „0“ napriek skreslenému tvaru 59
5.7 Chyba nájdenia znaku 1 59
5.8 Chyba nájdenia znaku 2 60
5.9 Chyba nájdenia znaku 3 60
5.10 Príklad zlého prahovania popisu. 61
5.11 Ukážka výsledku neimplementovaného algoritmu. 61
6.1 Ukážka vzhľadu smdClassifierClient 63

Úvod

Práca je zameraná na vytvorenie nástroja na detekciu vád pri výstupnej kontrole kondenzátorov, využitím počítačového videnia. Úlohou je vyhodnotenie povrchových vád obalu a popisu kondenzátoru. Výstupom má byť knižnica, obsahujúca tieto algoritmy.

Práca je okrem úvodu s rozborom úlohy a malým množstvom teórie rozdelená na tri hlavné časti. Najprv sa vysvetľuje návrh segmentačného algoritmu, oddeľujúceho kondenzátor od pozadia. Využíva sa základných metód spracovania obrazu.

Detekcia vád je založená na neurónovej sieti, vykonávajúcej detekciu objektov. Predstavuje mocný model YOLOv8 a vyhodnocuje vykonané pokusy pre dosiahnutie čo najlepších výsledkov.

Posledným hlavným blokom je vyhodnocovanie popisu kondenzátoru. Tu sa tiež vysvetľuje vývoj algoritmov a vyhodnocuje ich úspešnosť. Tieto algoritmy sú tiež založené na tradičných metódach, bez hlbokého učenia.

V práci sa hovorí tiež o značkovaní datasetov pre detekciu objektov, o vývoji DLL knižnice a grafického užívateľského rozhrania.

1. ROZBOR ÚLOHY

Na začiatok práce by som chcel venovať kapitolu hlbšiemu rozboru úlohy, aby čitateľ lepšie rozumel problematike a tým pádom aj vytvoreným riešeniam a nimi dosiahnutým výsledkom.

1.1 Cieľ práce

Cieľom práce je vytvoriť súbor algoritmov, ktoré budú zamerané vizuálnu výstupnú kontrolu SMD tantalových kondenzátor vo výrobe. Kontrola zahŕňa detekciu nežiaducich povrchových vád krytu a vyhodnotenie správnosti a kvality popisu kondenzátoru. Toto všetko sa musí implementovať s ohľadom na rýchlosť spracovania. Tieto algoritmy majú byť zabalené do knižnice typu DLL(dynamic link library). Pre vizuálnu ukážku funkcie, je tiež cieľom vytvoriť interaktívne grafické rozhranie, ktoré bude schopné načítavať obrázky, spracovať ich, a zobraziť výsledky.

1.2 Dataset

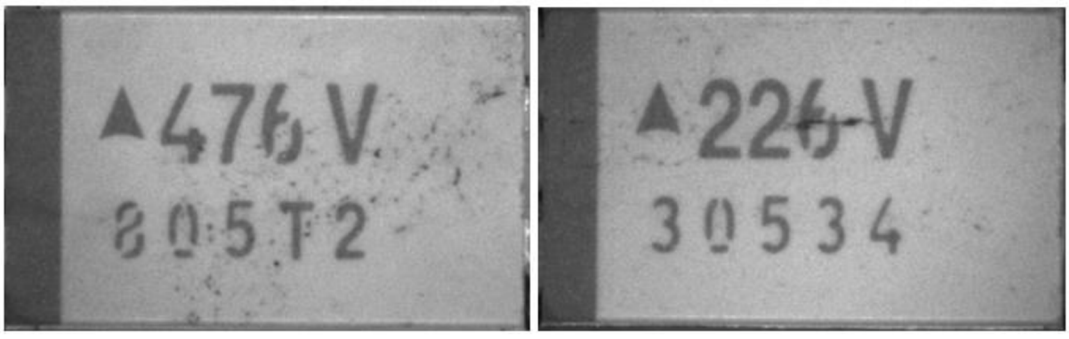
Obrázky kondenzátorov boli sprístupnené vedúcim diplomovej práce. Jednalo sa o dve dávky, prvá o počte približne 5700 vzorkov, a následne druhá dávka o počte približne 37000 vzorkov. Prvá dávka po pokusoch s učením neurónovej siete neobsahovala dostatok potrebných dát(vysvetlené neskôr, v kapitole 4), a preto boli sprístupnené veškeré dostupné obrázky v druhej dávke, pre rozšírenie učiaceho datasetu.

1.2.1 Druhy vád kondenzátorov

Na kondenzátoroch sa môžu objaviť rôzne typy vád, ktoré je potrebné od seba rozlíšiť. Nasledujúci zoznam popisuje jednotlivý druhy detailne:

1. Špina

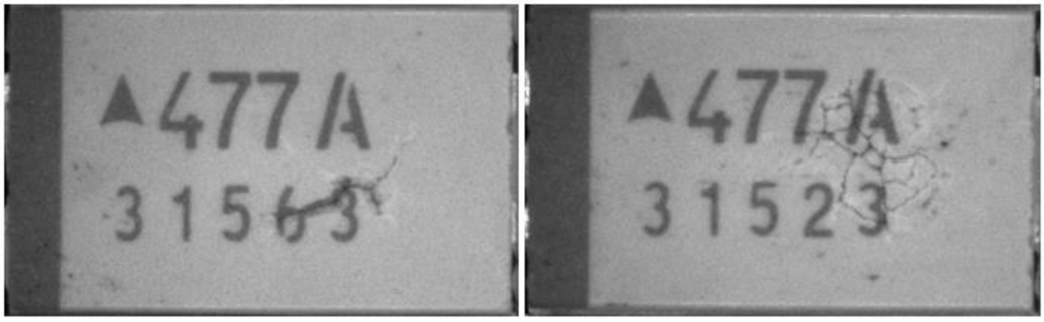
Jedná sa o prach a inú nečistotu na povrchu kondenzátoru. Vada má čisto estetický charakter a nemá negatívny vplyv na funkciu súčiastky.



Obrázok 1.1 Ukážka špiny

2. Prasklina

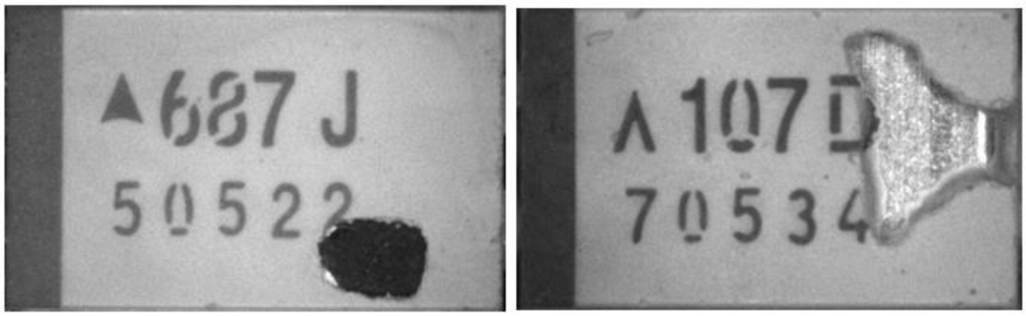
Nežiadúca vada, väčšinou podlhovastého a úzkeho tvaru.



Obrázok 1.2 Ukážka praskliny

3. Odštiepok

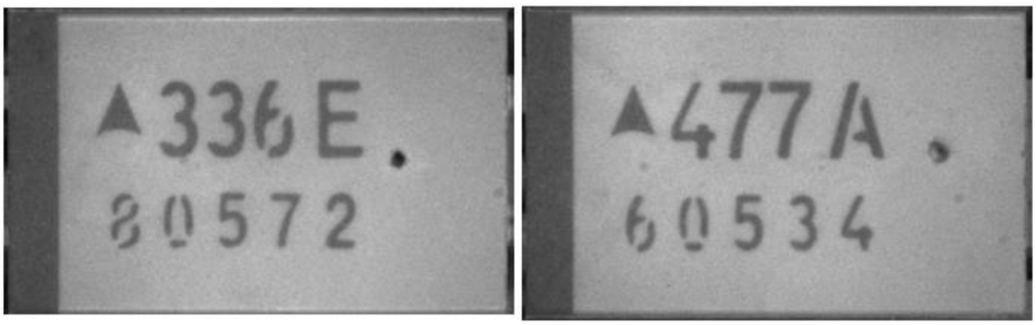
Nie najkrajší preklad pomenovania z pôvodného názvu „odštěpek“, so snahou zachovania podobnosti. Jedná sa o väčší odštiepený kus povrchovej hmoty kondenzátoru. Farba sa môže líšiť na základe odhaleného materiálu – čierna je anóda, strieborná kontakt. Jedná sa o nežiadúcu vadu.



Obrázok 1.3 Ukážka odštiepku

4. Diera

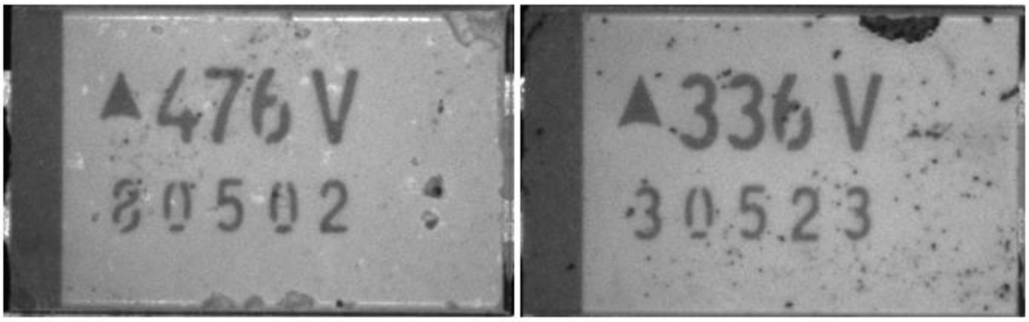
Drobný, prevažne kruhovitý otvor na povrchu kusu. Farba sa môže líšiť v závislosti na hĺbke a je nežiadúcou vadou.



Obrázok 1.4 Ukážka diery

5. Ulomenia

Ulomenina na okraji kondenzátoru, je nežiadúcou chybou.



Obrázok 1.5 Ukážka ulomeniny

6. Vady popisu

Popis môže mať viac druhov vád, ale pre kompaktnosť budú spojené do jednej kategórie. Ako prvé, môže chýbať časť polaritného pruhu. Ďalej sa môžu zlievať alebo chýbať niektoré časti jednotlivých znakov. Celý popis môže byť posunutý horizontálne alebo vertikálne. Znaky môžu byť zdvojené. Znaky môžu byť zamenené za nesprávne. Niekedy môže chýbať celý popis.

Obrázok 1.6 Ukážka vád popisu

1.2.2 Pracovná verzia datasetu

Po pretriedení spomínaných približne 37000 obrázkov sa našlo približne 1500, ktoré obsahovali nežiadúce vady. Tieto boli anotované, podľa obsahu prasklín, dier, odštiepkov a prasklín. Anotácia je využitá iba pre učenie neurónovej siete, ktorá má za úlohu vyhodnotiť práve iba nežiadúce vady.

Dataset je rozdelený na trénovacie, validačné a testovacie dáta, v pomere 70:20:10. Bolo to vykonané náhodným výberom, avšak pre každú triedu vád zvlášť, takže by mal byť pomer v jednotlivých častiach podobný.

K datasetu bolo pridaných okolo 200-250 obrázkov pozadia. Jedná sa o normálne fotky kondenzátorov, ktoré neobsahujú nežiadúce vady. Môžu však obsahovať špinu. Tieto boli pridané tak, aby tvorili v trénovacom aj validačnom datasete asi 10% obrázkov. Do testovacieho datasetu bolo pridaných viac, výsledne tvoria asi 25-50%. Vady sú pomerne zriedkavé, aspoň to vyplýva z rozloženia pri triedení, takže to bližšie reprezentuje reálne dáta. Taktiež sa tým lepšie overí, či neurónová sieť nebude nachádzať vady, kde nemá. A do tretice, pridáva to širšie možnosti vyskúšania spracovania popisu, v grafickom rozhraní.

1.3 Rozdelenie na dielčie úlohy

Z dôvodu komplexnosti zadania, bolo pre prehľadnosť klasifikačného systému, spracovanie rozdelené na menšie úlohy.

1.3.1 Vývoj algoritmov

Vývoj algoritmov v prostredí MATLAB*. Výuka počítačového spracovania obrazu v rámci štúdia prebiehala práve v tomto prostredí, a preto sa mi s ním dobre pracuje. Má mnoho výhod, ktoré zjednodušujú tzv. debugovanie, ako napríklad jednoduché zobrazovanie medzivýsledkov, prístup k premenným v prerušenom kóde a možnosť s nimi pracovať pomocou konzole počas prerušenia (napr. pri využití „breakpointu“). Toto je osobne oproti vývoju, napríklad priamo v C, značne priateľskejšie. Algoritmy spracovania boli rozdelené do troch primárnych častí:

Segmentácia kondenzátoru je prvým krokom spracovania. Oddeľuje kondenzátor od pozadia a pripravuje ho tým na ďalšie spracovanie. Je dôležitá napríklad preto, aby sa neobjavovali falošné chyby, identifikované na vodiacich kostrách alebo na z-časti-viditeľných vedľajších kondenzátoroch.

Detekcia vád slúži na identifikáciu nežiadúcich porúch na povrchu kondenzátoru. Cieľom je vytvoriť nástroj, ktorý vie tieto vady nájsť, a umožňuje zobraziť ich polohu používateľovi graficky, na kondenzátore.

Vyhodnotenie popisu je finálnym štádiom spracovania. Vyšetruje sa tu správnosť a kvalita popisu. Predpokladaným výstupom je určitá metrika, vyhodnocujúca kvalitu textu číselne, a je schopná vyhodnotiť, či sú vylaserované správne znaky.

*Poznámka: časť vývoja, primárne okolo neurónovej siete, prebiehala v prostredí Python.

1.3.2 Implementácia do DLL knižnice

Hotové algoritmy sú preložené a mierne upravené, aby sedeli do jazyka C#. Využíva sa verejne dostupných knižníc, ktoré umožňujú prácu s obrazmi a neurónovými sieťami. Hlavnými sú OpenCVSharp, knižnica zabaľujúca (wrapper) známu knižnicu OpenCV pre možnosť využitia v C#, a YoloDotNet, s podobným účelom implementácie rozhrania v C# pre prácu s neurónovou sieťou YOLOv8, ktorá je využívaná v práci.

1.3.3 Grafické rozhranie

Pre demonštráciu vytvorených algoritmov je vytvorené grafické rozhranie pomocou Windows Forms, na platforme .NET 8, v jazyku C#.

2. ZÁKLADY SPRACOVANIA OBRAZU

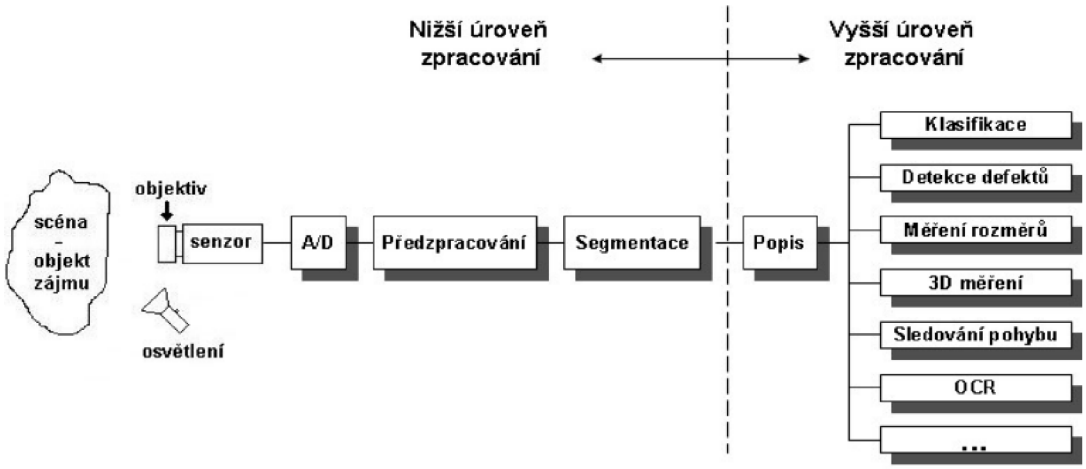
2.1 Obraz z pohľadu počítačového videnia

2.1.1 Digitálny obraz a reťazec spracovania obrazu

Obraz, z inžinierskeho hľadiska, považujeme za viacrozmerný signál. Typicky sa pracuje s dvojrozmerným obrazom, ale v niektorých prípadoch sa s veľkou výhodou využijú vyššie rozmery. Príkladom sú temporálne sekvencie, sledujúce vývoj obrazu v čase, alebo obrazy s hlbšou informáciou o jednotlivých pixeloch ako hyperspektrálne obrázky [1].

V reťazci spracovania obrazu, obrázok 2.1, je prvým krokom snímanie. Túto úlohu koná senzor. Z pohľadu spracovania signálu, senzor vykonáva vzorkovanie. Výsledkom je rozdelenie do pixelov, ktoré nesú informáciu o jase v danom bode v priestore. Tento krok tiež udáva rozlíšenie obrazu.

Vzorkovaný obraz je poslaný do analógovo-digitálneho prevodníku, ktorý kvantuje spojitý jas zo senzoru do diskrétnych úrovní, ktoré udávajú bitovú hĺbku vzorkov. Po vzorkovaní a kvantizácii vzniká digitálny obraz.



Obrázok 2.1 Reťazec spracovania obrazu [2]

Po získaní digitálneho obrazu sa postupuje na samotné spracovania. Do tejto oblasti patrí predspracovanie, segmentácia, a popis. Táto práca je zameraná práve na vymenované časti reťazca, takže sú viac popísané v nasledujúcich podkapitolách.

2.2 Predspracovanie - základné úpravy obrazu

Predspracovaním obrazu sa snažíme upravovať obraz tak, aby lepšie vyhovel ďalšiemu spracovaniu. To, ako sa upraví, závisí nie len na vlastnostiach obstaraného obrazu, ale aj na metódach použitých pri spracovaní, pre ktoré obraz pripravujeme.

2.2.1 Jasové a geometrické transformácie

Jasové transformácie sú určené na zmenu hodnôt v jase podľa nejakého pravidla. Rozlíšenie a bitová hĺbka obrazu ostávajú nezmenené. Príkladom využitia je korekcia nerovnomerného osvetlenia. Jasové transformácie sa rozdeľujú na globálne, lokálne, alebo bodové. [3]

Globálnou jasovou transformáciou je metrika využívajúca informáciu z celého obrazu a na základe tejto informácie opravuje jas v jednotlivých bodoch(napr. Fourierova transformácia). Lokálnou transformáciou sú operácie, ktoré využijú malé okolie bodu pre úpravu jeho jasu. Typicky je to lokálna filtrácia šumu, alebo tvorenie hranového obrazu. Bodová jasová transformácia upravuje jas na základe hodnoty daného bodu. Tieto úpravy sa často vykonávajú pomocou prevodovej funkcie jasu, ktorá môže byť ale nemusí byť rovnaká pre každý bod (parametrizácia na základe polohy). [3]

Geometrické transformácie upravujú súradnice obrazových bodov a zaobstarávajú následnú interpoláciu jasových hodnôt. Príkladmi využitia sú korekcia skreslenia(radiálne či tangenciálne), rotácia obrazu, či úprava mierky. [4]

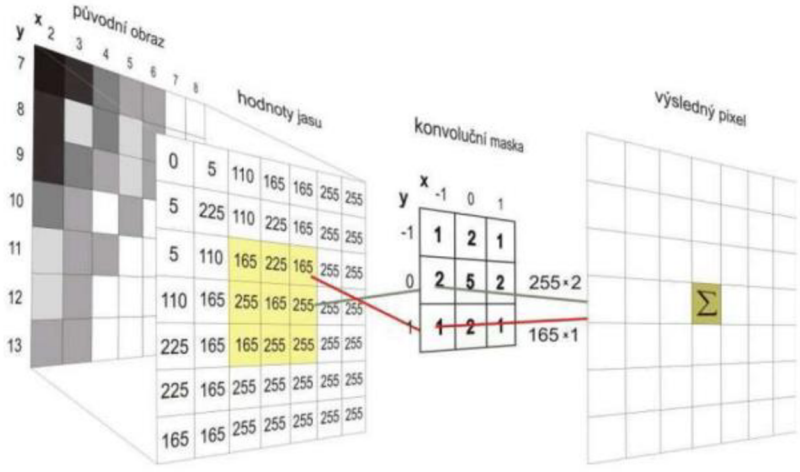
2.2.2 Konvolúcia

Konvolúcia sa využíva v spracovaní obrazu pri aplikácii rôznych druhov filtrov, tiež nazývaných masky alebo kernely. Jas výsledného bodu je rovný lineárnej kombinácii jasov bodov v okolí na vstupnom obraze s váhovými koeficientmi. Matematický popis konvolúcie je vyjadrený vzťahom

$$f(i,j) = \sum_{mí-M/2}^{M/2} \sum_{n=j-N/2}^{N/2} h(m-i, n-j)g(m,n)\,, \tag{2.1}$$

kde $f(i,j)$ je hodnota výslednej jasovej funkcie v bode (i,j), g je vstupný obraz, h je konvolučnej masky, a (M,N) sú rozmery tejto masky. [2]

Na obrázku 1.2 je možné vidieť grafické znázornenie pre lepšie pochopenie.



Obrázok 2.2 Grafické znázornenie konvolúcie obrazu s maskou [2]

2.2.3 Vyhladzovanie šumu

Na vyhladzovanie šumu je možné použiť viac metód. Jednoduchou nenáročnou metódou sú lokálne filtre. Tieto môžu byť lineárne a nelineárne. Ďalej je možné použiť výpočtovo náročnejšiu 2D Fourierovu transformáciu, a filtrovať obraz vo frekvenčnej oblasti. Pre statické scény, sa s výhodou dá použiť časová filtrácia šumu. [2]

Táto práca sa zaoberá primárne s lokálnymi filtrami šumu, takže sa detailnejšie popíšu iba tieto.

Lineárne filtre spočívajú v lineárnej kombinácii hodnôt filtračnej masky a obrazu, pre ich výpočet sa s obľubou používa konvolúcia. Najjednoduchším typom je priemerovací filter. Ako naznačuje názov, počíta priemernú hodnotu jasu vo vybranom okolí pixelu. Toto okolie môže byť 3x3, 5x5, 7x7 pixelov, či väčšie. S veľkosťou filtru rastie potlačenie šumu ale zároveň sa potláčajú detaily obrazu. [2]

Lineárne filtre môžu mať ľubovoľnú váhu príspevku jednotlivých pixelov, dôležité je, aby celková suma masky bola rovná 1. Často používaným príkladom masky s upravenými váhami je filter s Gaussovým rozložením. Z charakteru rozloženia vyplýva, že stred kernelu má väčší príspevok ako okraj. [2]

Na obrázku 2.3 je porovnanie priemerovacieho a Gaussovho filtru.

Priemer

$$\frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Gauss

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Obrázok 2.3 Porovnanie filtrov šumu: priemerovací a Gaussov [2]

Nelineárne filtre vkladajú na výstup niektorú hodnotu z vybraného okolia. Veľkosť okolia má rovnaký efekt ako pri lineárnych. Príkladmi nelineárnych filtrov sú využívajúce medián, modus, minimum a maximum.

2.2.4 Detekcia hrán

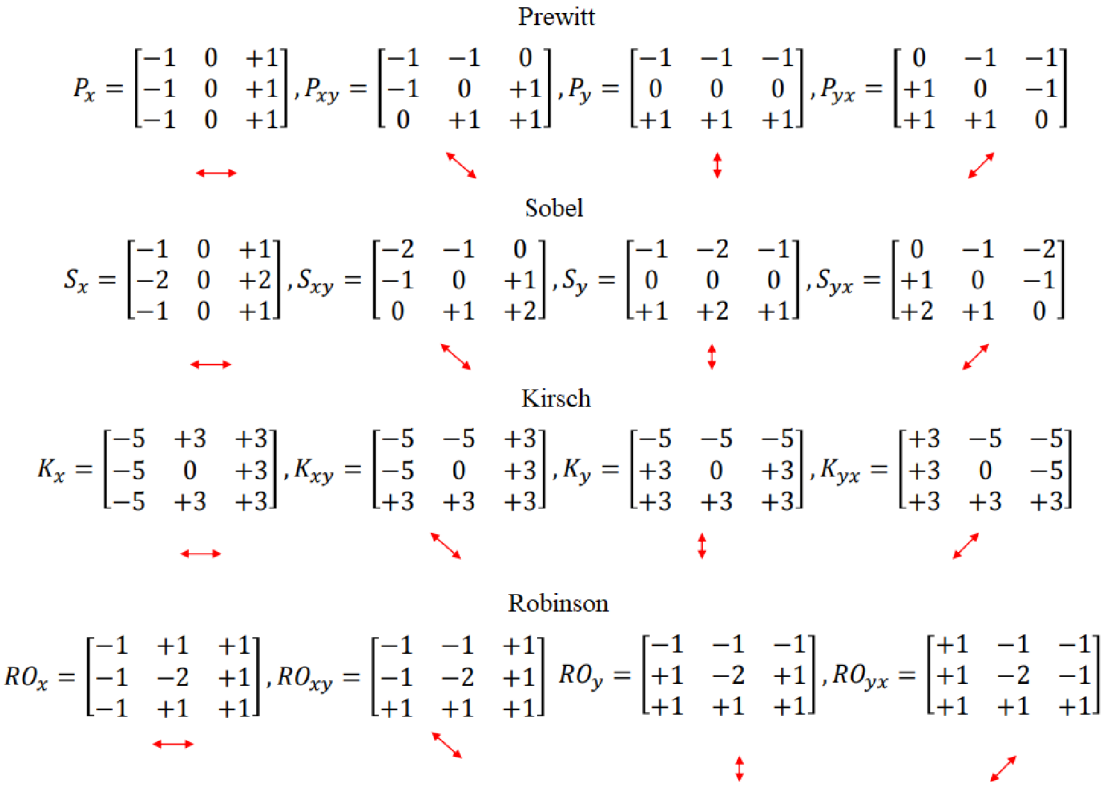
Závisí na uhle pohľadu, detekcia hrán môže byť zaradená do predspracovania, aj segmentácie obrazu. Rozhodol som sa zaradiť túto operáciu do predspracovania, pretože ak plánujeme pracovať s hranami obrazu, tak zobrazenie týchto hrán je prípravou, predspracovaním, pre ďalšie operácie. Niektoré zdroje uvádzajú zaradenie do segmentácie [3].

Hranami v obraze sú výrazné zmeny intenzity jasovej funkcie. Tieto môžu nastať pri prechodoch medzi homogénnymi oblasťami, alebo pri lokálnych výkyvoch ako čiary či

body. Intenzita týchto lokálnych výkyvov sa dá aproximovať diferenciou v smere jej zmeny, ktorú je možné zapísať maticovo. Toto vedie k využitiu konvolúcie pre výpočet obrazu hrán. [6]

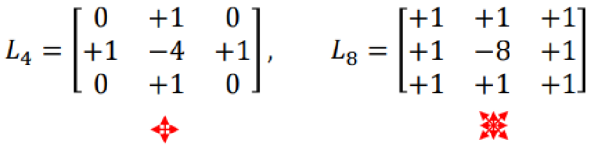
Maticový zápis hranových detektorov má vždy sumu rovnú 0. Hranové detektory sa delia do dvoch tried – s 1. a 2. deriváciou. Pri detektoroch prvej derivácie sa hľadá extrém ich výstupnej jasovej funkcie, toto je miesto kde sa nachádzajú hrany. Dôležitým faktom je, že tieto operátory zvýraznia hrany vždy iba v niektorom smere, podľa zostavenia matice. Príkladmi týchto detektorov sú Prewitt, Sobel, Kirch, Robinson. [6]

Ich porovnanie je na obrázku 2.4.



Obrázok 2.4 Porovnanie detektorov hrán 1. derivácie [6]

Hranové detektory druhej derivácie vytvoria vo výstupnej jasovej funkcii prechod nulou. Ich výhodou je, že jedna maska zobrazuje hrany vo všetkých smeroch. Pre aproximáciu výpočtu druhej derivácie sa využíva výhradne Laplacovho operátoru (zobrazený na obrázku 2.5), vo variante štvor-okolia a osem-okolia. [6]



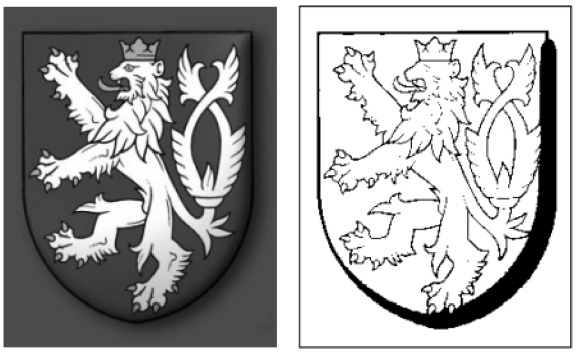
Obrázok 2.5 Laplacov operátor štvor a osem-okolia [6]

2.3 Segmentácia

Po pripravení obrazu predspracovaním nasleduje jeho segmentácia. Celkový obraz takmer nikdy neobsahuje iba to, čo nás zaujíma. Pozadie a externé elementy nechceme vyhodnocovať. Úlohou segmentácie je oddeliť oblasti záujmu od zvyšku obrazu, prípadne rozdeliť celý obraz na ucelené oblasti.

2.3.1 Segmentácia prahovaním

Prahovanie obrazu je rozhodovanie o hodnote pixelu, na základe predurčeného prahu. Najjednoduchšou verziou prahovania je binárne rozdelenie pixelov podľa toho, či sú nad alebo pod prahom (obrázok 2.6). [7]



Obrázok 2.6 Jednoduché prahovanie [7]

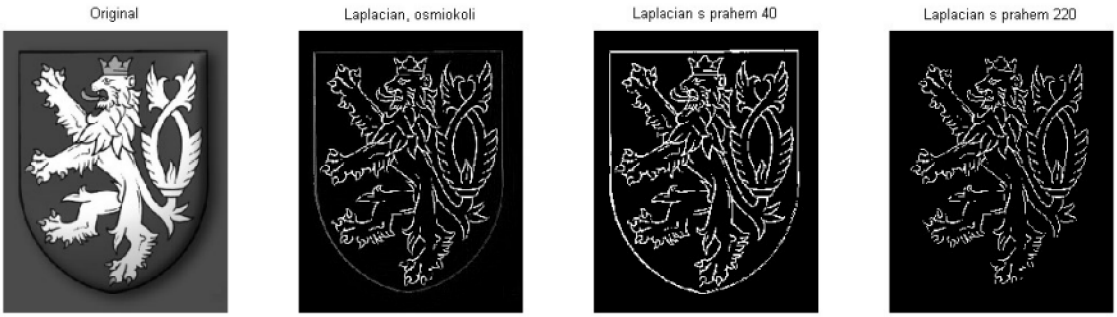
Prahovanie môže prebiehať s viacerými prahmi, môže sa zachovať pôvodný jas v niektorej podmienke prahovania (horné a dolné, tiež čiastočné prahovanie), adaptívne prahovanie, meniace prah v závislosti na polohe v obraze. [7]

Prah sa určuje buď experimentálne, z histogramu, zo štatistík (stredná hodnota, medián,...), percentuálneho podielu pixelov (z kumulatívneho histogramu), či z globálnej znalosti ako farby hľadanej oblasti. [7]

2.3.2 Segmentácia z obrazu hrán

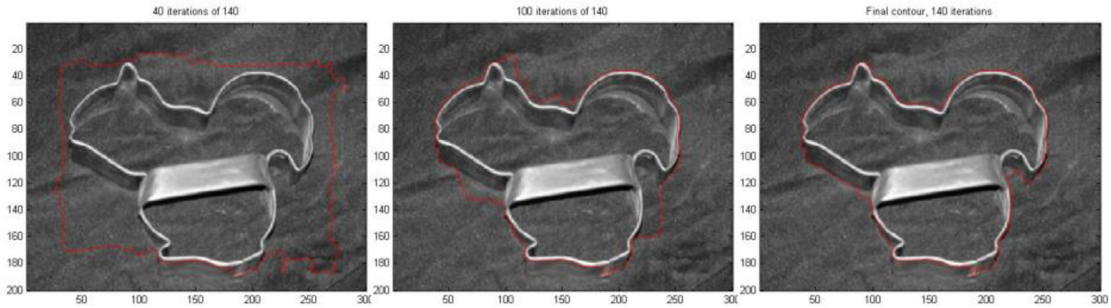
Na segmentáciu pomocou hrán sa využíva obrazu hrán, vytvoreného hranovými detektormi. Tiež existujú algoritmy na sledovanie hraníc medzi dvomi rôznymi homogénnymi oblasťami. Pre spresnenie je možné použiť apriórnu informáciu o polohe, farbe a pod. [7]

Prvou možnosťou segmentácie pomocou obrazu hrán je jeho prahovanie [7]. Takto získavame napríklad binárny obraz, v ktorom sú zachytené pixely dostatočne výrazných hrán (udané voleným prahom). Na obrázku 2.7 je znázornenie prahovania obrazu hrán vytvoreného detektorom Laplace.



Obrázok 2.7 Prahovanie obrazu hrán [7]

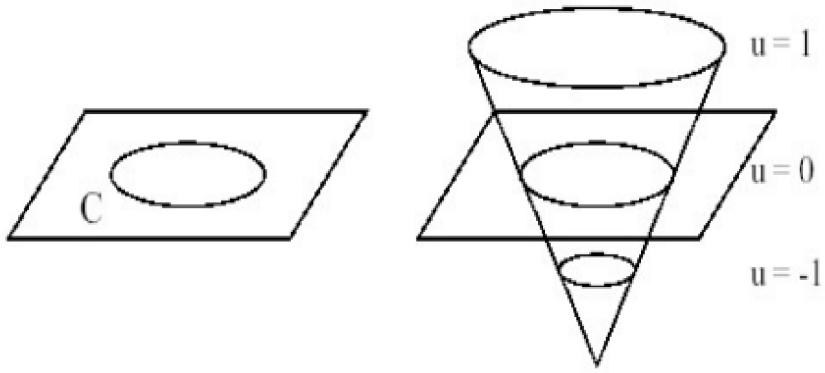
Ďalšou metódou sú aktívne kontúry. Je to iteratívna metóda, spočíva v postupnom tvarovaní kontúry až k hrane objektu. Je založená na minimalizácii energie, a existuje viac spôsobov výpočtu tohto kritéria. Metóda je graficky znázornená na obrázku 2.8. [7]



Obrázok 2.8 Príklad vývoja aktívnych kontúr [7]

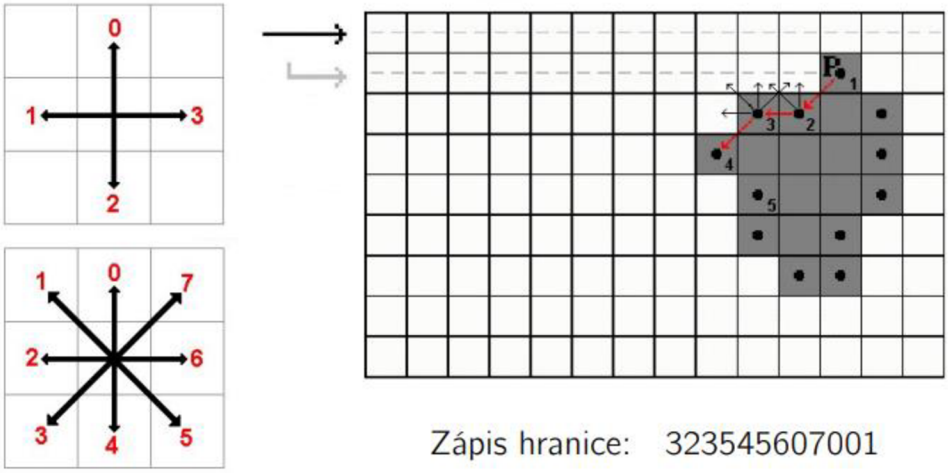
Podobnou metódou k aktívnym kontúram je level-set. Kontúry pri tejto metóde ale meníme na základe level-set funkcie, ktorá odpovedá napr. ihlanu. Posúvaním sa po výške vybraného objektu je možné meniť tvar krivky. [7]

Vizuálna ukážka je na obrázku 2.9.



Obrázok 2.9 Demonštrácia level-set [7]

Iným prístupom k segmentácii je sledovanie hranice. Využíva sa ak nemáme informáciu o polohe, iba napr. farbe objektu/hranice. Hranica objektu je nájdená jeho obkrúžením (štvor či osem-okolie). Ukážka na obrázku 2.10. [7]



8

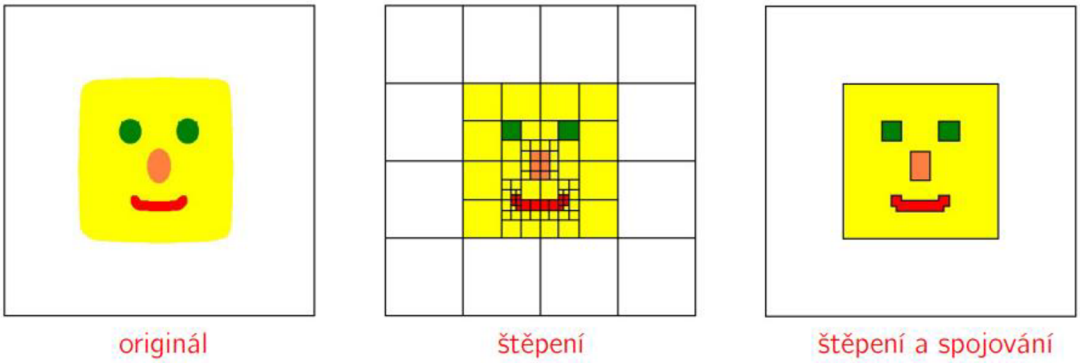
Obrázok 2.10 Ukážka sledovania hranice [7]

2.3.3 Segmentácia založená na regiónoch

Vytvorením regiónov sa snažíme získať rozdelenie do oblastí homogénnych vo vybranom parametri (jas, farba, poloha,...). [8]

Na základe podobnosti sa môžu oblasti obrazu spojovať alebo štiepiť. Tieto dve metódy môžu byť implementované v spojení i samostatne. Pri spojovaní sa vyhodnocuje podobnosť dvoch oblastí na základe vybraného kritéria, a pri dostatočnej blízkosti sa tieto oblasti spájajú. Začať sa teda môže pri jednotlivých pixeloch. Pri štiepení sa daná oblasť rozdelí, napr. na polovicu, až kým každá jednotlivá časť nie je homogénna. Začiatok je teda napr. celý obraz. [8]

Na obrázku 2.11 je znázornenie segmentácie, postup je štiepenie a následne spojovanie.



Obrázok 2.11 Segmentácia štiepením a spojovaním [8]

3. SEGMENTÁCIA KONDENZÁTOROV

V kapitole je popísaný postup vývojových krokov, ktorý viedol ku konečnému riešeniu segmentovania kondenzátorov od pozadia.

3.1 Testy a „naivné“ riešenie

Vývoj začal jednoduchým nápadom – po filtrácii šumu a vytvorení obrazu hrán sa využije riadkového súčtu (amplitúdovej projekcie) v horizontálnom a vertikálnom smere, tým sa nájde poloha hrán kondenzátoru, a máme hotovo.

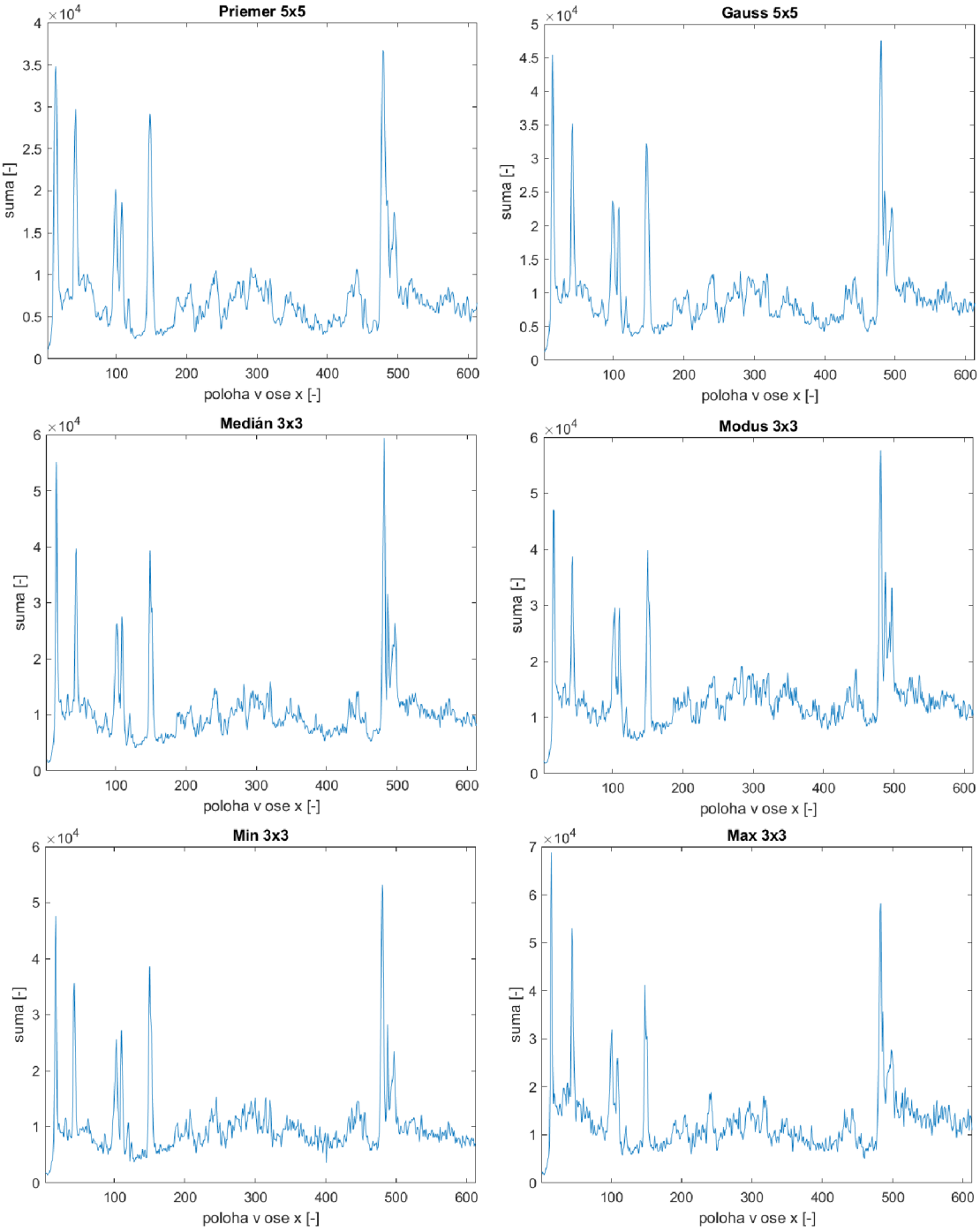
Aby sa lepšie spoznali vlastnosti obrázkov, počas vývoja sa uskutočnil malý súbor testov pre porovnanie rôznych typov filtrov šumu a hranových filtrov. Testy prebehli zobrazením výsledkov rôznych operácií a vizuálnym vyhodnotením (nenavrhovala sa žiadna metrika „vhodnosti“). Pre väčšinu rozhodovania sa použili výsledné priebehy amplitúdovej projekcie vertikálnych hrán, pretože práve tie budú využité v ďalších krokoch spracovávajúceho reťazca. Ako informácia pre čitateľa, na zobrazenie grafov bola použitá fotka z testovacej časti datasetu „1000608248.tif“.

3.1.1 Testy filtrov šumu

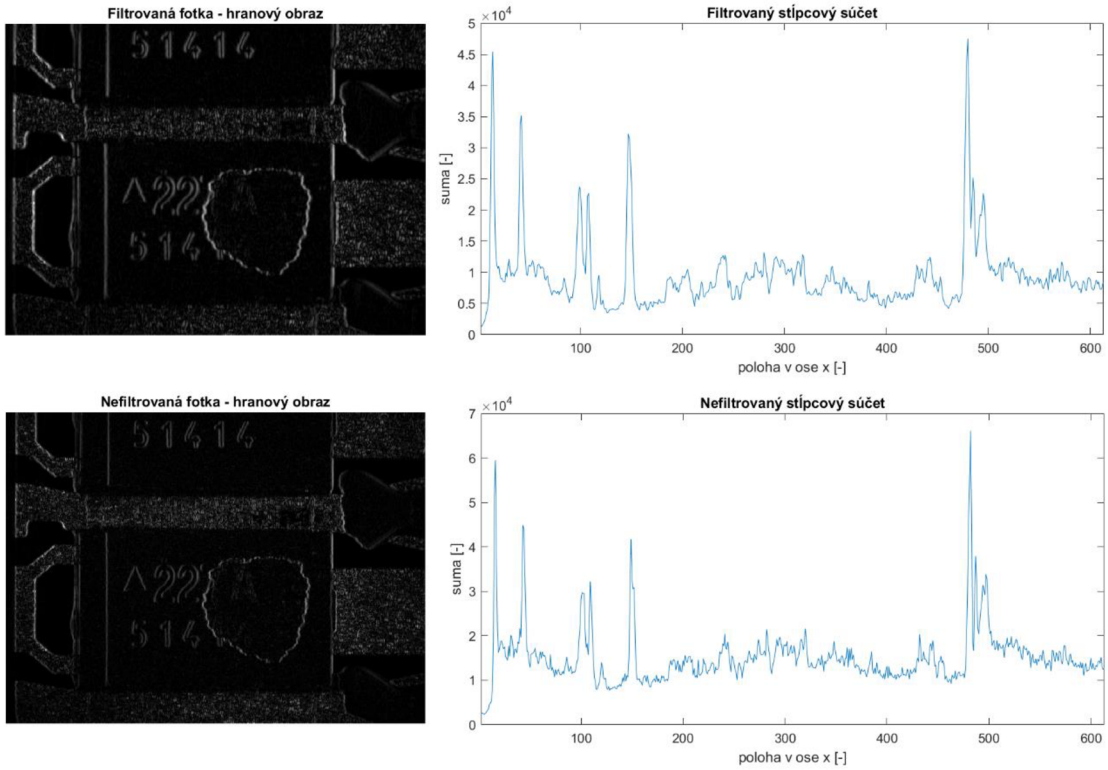
Filtrácia šumu je dôležitým krokom pri spracúvaní akýchkoľvek signálov, preto sa testovala ako prvá. Pre získanie obrazu hrán sa zatiaľ použil Sobelov filter. Amplitúdová projekcia, v tomto prípade stĺpcový súčet, je vytvorený veľmi jednoduchým sčítaním všetkých hodnôt v danom stĺpci.

Na obrázku 3.1 vidíme, že ak aj rozdiely existujú, nie sú dosť výrazné. Musí sa brať do úvahy fakt, že toto je výsledok iba jedného z mnohých obrázkov, takže kvôli variabilite záleží na malých detailoch menej. Zadanie práce poznamenáva kladenie dôrazu na časovú náročnosť. Operácia modus vytvára potrebu triedenia, takže sa ihneď vylučuje. Medián je tiež zložitejšou operáciu, vylučuje sa. Min a max filtre šumu sú ekvivalentom morfologickej erózie a dilatácie. Podľa subjektívneho názoru, toto môže skresľovať výsledky, preto sa tiež vylúčia. Gaussov filter môže často priniesť výhody, avšak teraz iba na rozdiel priemerujúceho filtru vykonáva zbytočné násobenie, bez výrazného pozitívneho efektu. V prípade použitia sa teda obracia na klasický filter pomocou dosadenia priemeru okolia.

Fotky nie sú takmer vôbec znečistené šumom, a ako ukáže obrázok 3.2, čo sa týka tvaru stĺpcového súčtu, filtre nemajú veľký prínos. Špičky, ktoré odpovedajú hranám, sú stále veľmi výrazné oproti zvyšku fotky. Vo finálnom riešení sa teda pravdepodobne nepoužije filter šumu.



Obrázok 3.1 Porovnanie filtrov šumu na stĺpcových súčtoch



Obrázok 3.2 Porovnanie stĺpcového súčtu s a bez filtru šumu

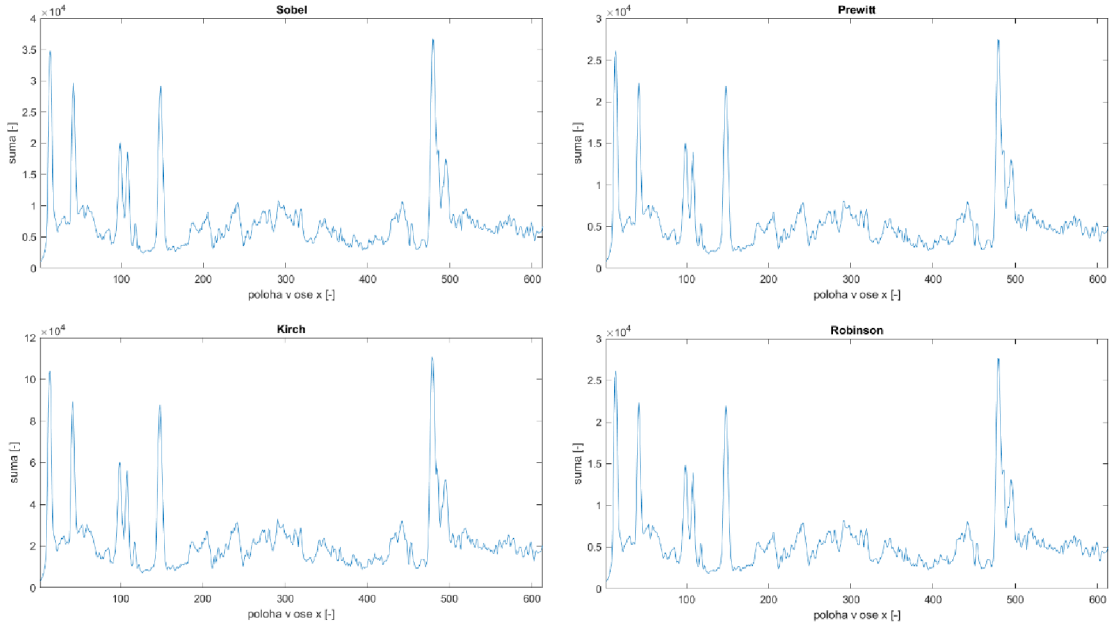
3.1.2 Testy hranových filtrov

Obraz hrán je ďalším krokom v spracovaní. Keďže sú kondenzátory približne zarovnané s hranami fotiek, vybrali sa filtre prvej derivácie, pre hľadanie konkrétne vertikálnych hrán. Obrázok 3.3 (ďalšia strana) ukazuje, že odlišnosti sú zase len minimálne. Z výpočtového hľadiska je najjednoduchším filtrom Prewitt. Pozn.: pre tieto testy bol použitý filter šumu, testy na jeho významnosť boli vykonané až neskôr.

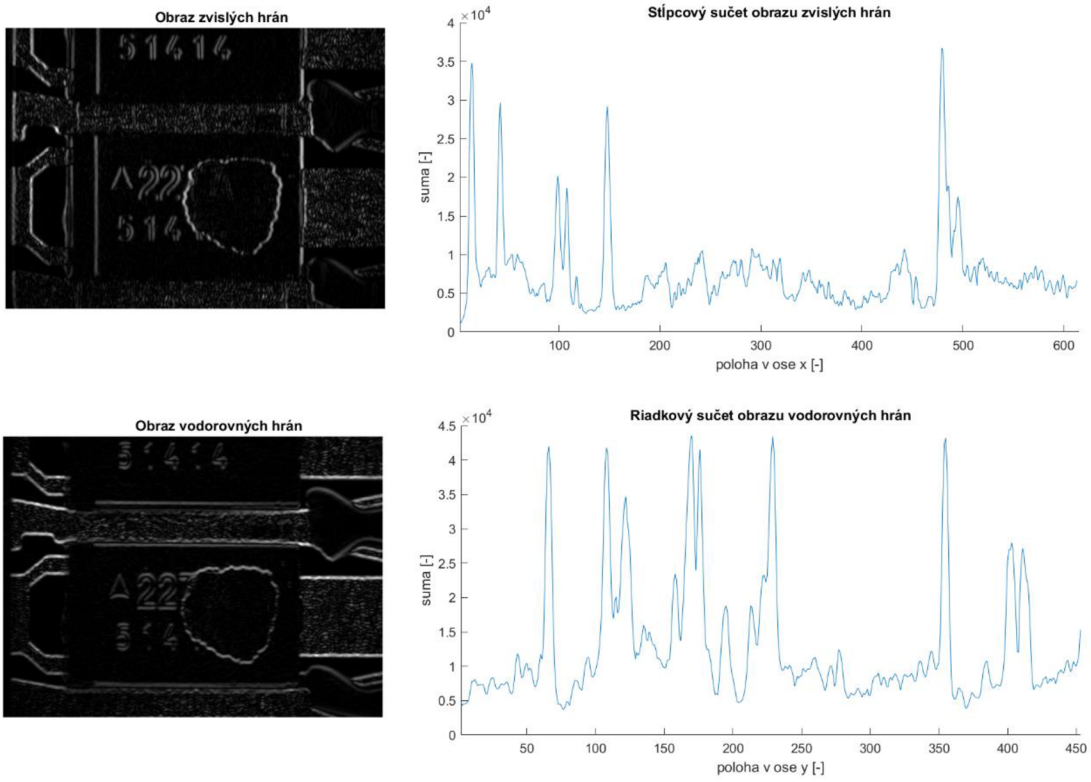
3.1.3 Vyhodnotenie stĺpcových súčtov

Na predchádzajúcich obrázkoch z tejto kapitoly je možné vidieť viac komplikácií s jednoduchými stĺpcovými súčtami. Na fotkách je viac výrazných hrán, ako aj na popise kondenzátoru, tak z vodiacich kostier.

Núka sa najprv nájsť buď zvislé alebo vodorovné hrany, orezať, a až potom hľadať tie druhé. Obrázok 3.4 (ďalšia strana) však ukazuje, že toto nie je spoľahlivým riešením, ak by sa hľadali prosté extrémy. Najsilnejšia zvislá hrana vľavo patrí vodiacej kostre, kým vodorovné hrany celého obrázku sú podobnej intenzity. Kombináciu týchto problémov sa nachádza na väčšine fotiek. Niektoré dostupné fotky sú takto úspešne segmentovateľné, ale nespoľahlivo. Zdá sa, že aj keby sa riešenie doladilo, malo by istý náhodný charakter a žiada sa navrhnúť robustnejší algoritmus.



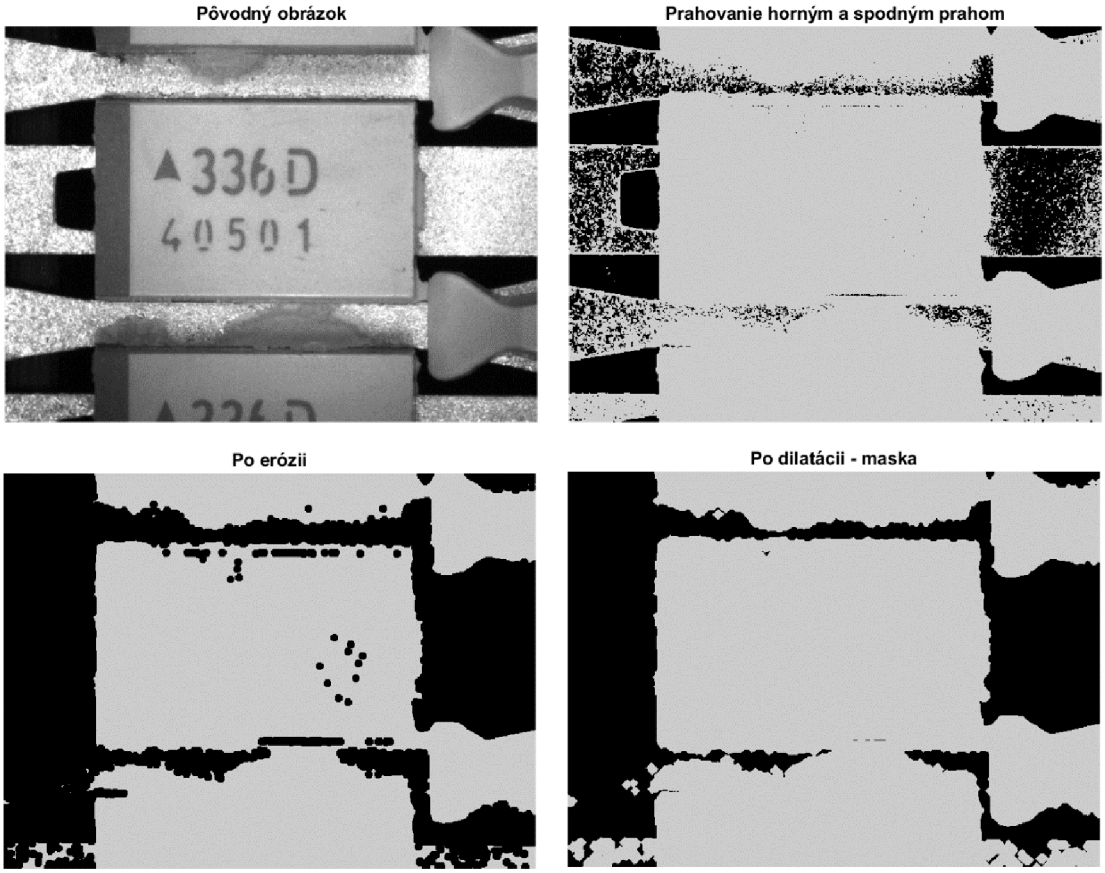
Obrázok 3.3 Porovnanie hranových filtrov na stĺpcových súčtoch



Obrázok 3.4 Ukážka nejasností ohľadom správnej hrany

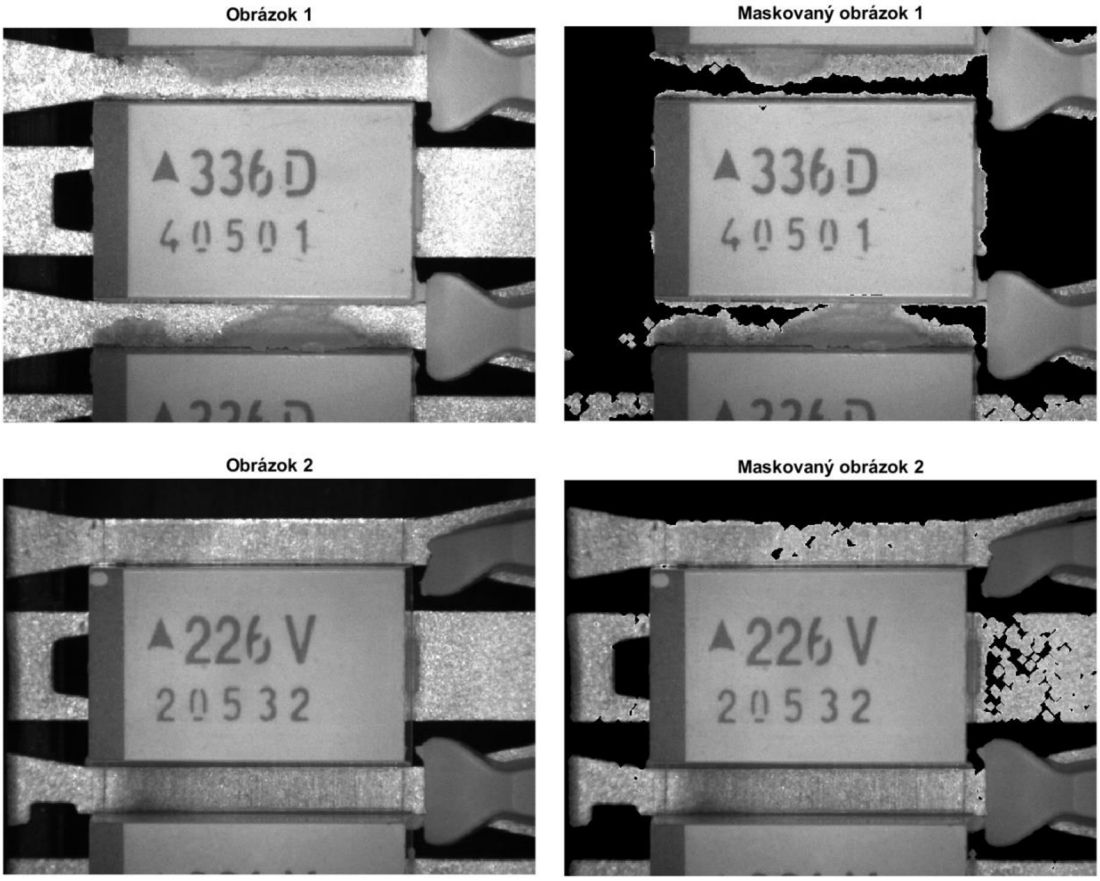
3.2 Segmentácia prahovaním obrazu

Úplne iný prístup k úlohe. Je založený na prahovaní samotnej fotky, pomocou horného a dolného prahu – kondenzátor má na väčšine plochy približne polovicu maximálnej intenzity jasu. Následne sú vykonané morfologické operácie erózia a dilatácia. Erózia spôsobí zmazanie malých zhlukov pixelov - vymazanie malých fragmentov. Následná dilatácia uzavrie nespojité oblasti, vzniknuté pri prahovaní či práve erózii. Týmito operáciami získame binárnu masku, ktorú použijeme na obraz. Cieľom je odstránenie problémového lead-framu z obrazu pomocou novovytvorenej masky. Jednotlivé kroky spracovania sú zobrazené na obrázku 3.5.



Obrázok 3.5 Znázornenie postupu pri segmentácii prahovaním.

Metóda sa zdala sľubná, avšak veľká variácia v jase medzi jednotlivými fotkami a tým spôsobená podobnosť jasu lead-framu a povrchu kondenzátoru túto metódu vyškrtli (obrázok 3.6).



Obrázok 3.6 Výsledky metódy segmentácie prahovaním.

3.3 Segmentácia využitím znalostí o polohe

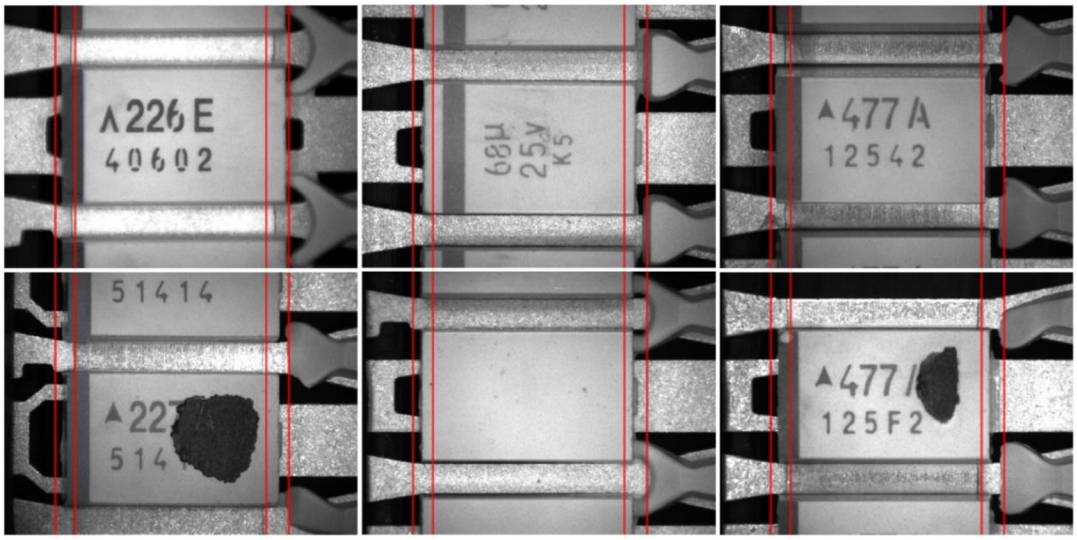
Návrat k pôvodnému prístupu – amplitúdová projekcia hrán. Tentoraz sa algoritmus obohatil o isté znalosti, ktoré výrazne zlepšili jeho úspešnosť. Musí sa dodať, že tieto vlastnosti sú pozorované na malom datasete, a v tomto čase mi nie sú známe podmienky vytvorenia fotiek(presnosť polohy,...). Možnosť ladiť parametre algoritmu necháva istý priestor pre odstránenie prípadných chýb, ak by k nim neskôr došlo.

3.3.1 Znalosti

Zoznam pozorovaných faktov z datasetu:

1. Poloha kondenzátorov, čo sa týka posunu vľavo a vpravo je málo odlišná.
2. Stred obrázku sa nachádza vždy vo vnútri oblasti kondenzátoru.

Ďalšou dôležitou zmenou je využitie prahovania hranového obrazu.

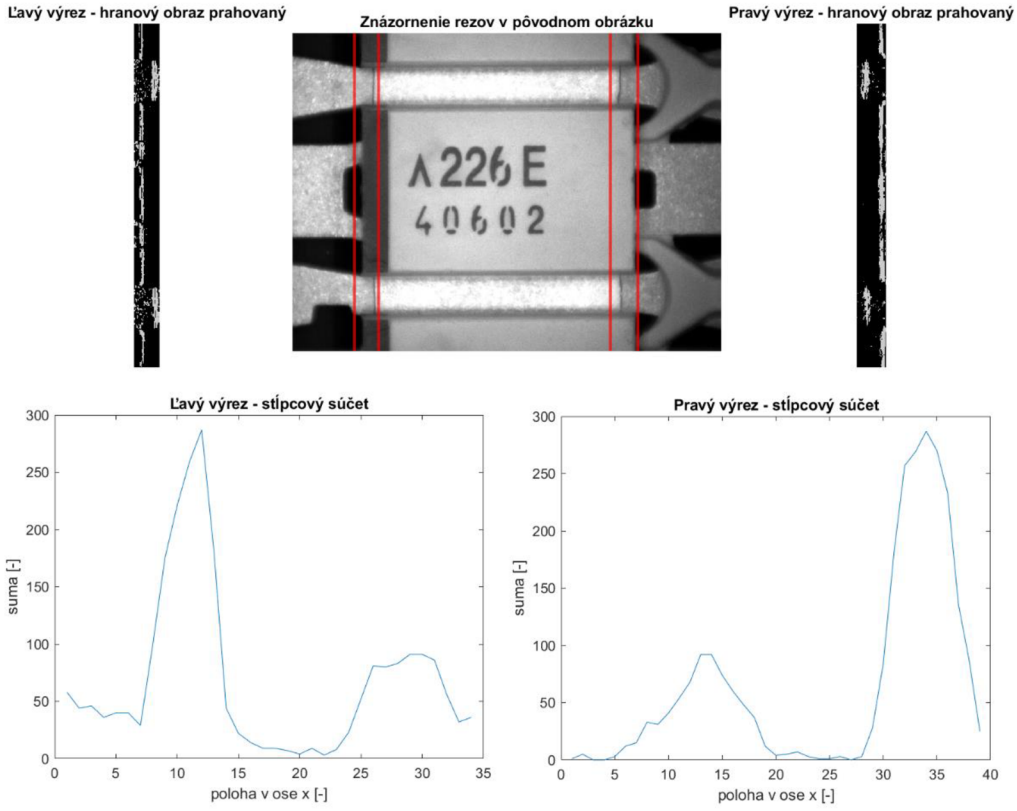


Obrázok 3.7 Ohraničenie oblasti v ktorej sa vyskytujú zvislé hrany kondenzátorov

3.3.2 Popis algoritmu

Cieľom pridaných úprav je obmedziť vplyv vodiacich kostier na spracovanie. Prvým krokom je vytvorenie dvoch úzkych výrezov z obrázku, na ktorom sa budú nachádzať iba zvislé hrany kondenzátorov a ich blízke okolie. Poloha rezov je zadaná pevne, je možné ju zmeniť úpravou parametru vo funkcii (v kóde, nie pri volaní).

Získané výrezy sa spracujú s hranovým filtrom Prewitt na zvislé hrany. Oblasť je nastavená tak, aby sa v nej nenachádzali iné významné hrany. Napriek tomu, kvôli robustnosti je pridané prahovanie obrazu hrán, prahom je dvojnásobok jeho priemernej hodnoty. Ďalej je prevedený stĺpcový súčet. Index jeho maxima je použitý na orezanie okrajov obrázku. Týmto krokom sme nie len z časti oddelili kondenzátor, ale zároveň sme odstránili vodiacu kostru, ktorá by vo väčšine prípadoch neumožňovala, alebo aspoň skomplikovala, ďalšie kroky spracovania. Znázornenie krokov je na obrázku 3.8.



Obrázok 3.8 Dielčie kroky orezania bočných oblastí obrázkov

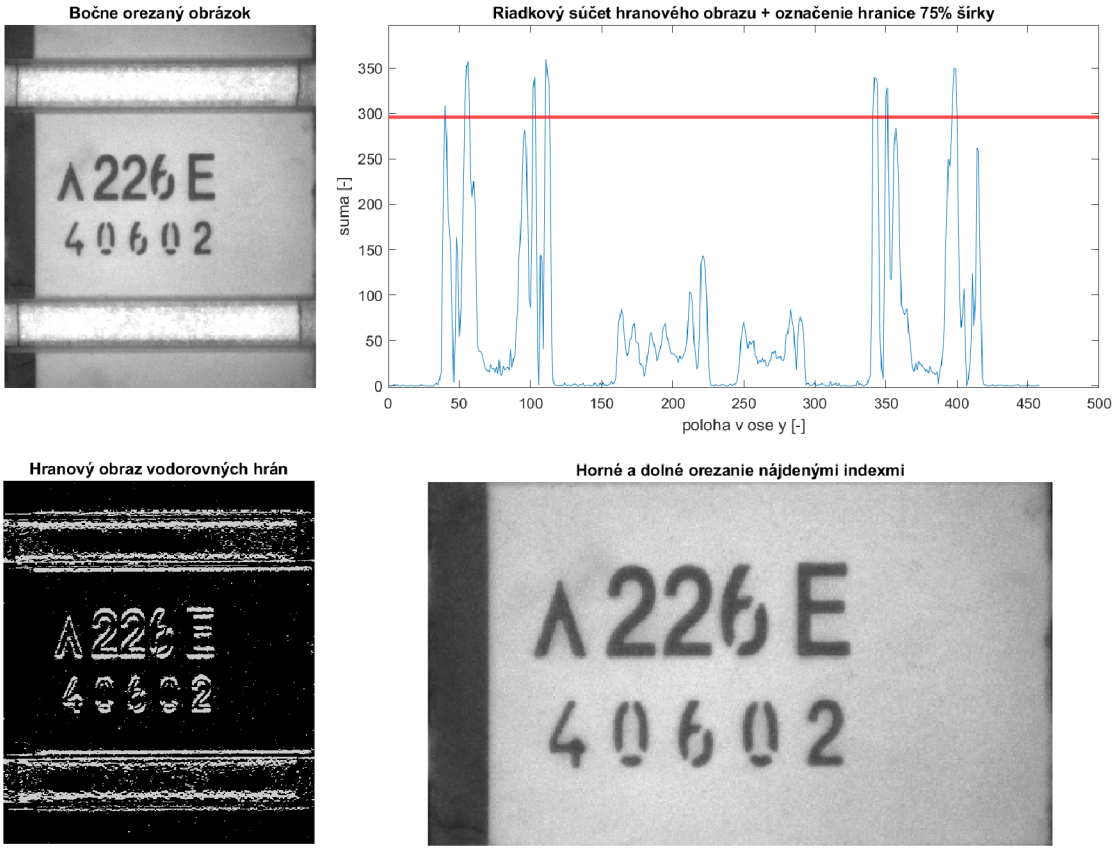
Výstupný orezaný obrázok je použitý na ďalšie spracovanie. Z obrázku sa vytvorí ďalší obraz hrán, tentoraz však vodorovných, tiež pomocou filtru Prewitt. Pokračuje sa prahovaním, s prahom dvojnásobku priemeru. V tejto fáze je tento krok dôležitejší, než pri zvislých hranách, text totiž môže tvoriť významnejšie hrany, a je vhodné ich čo najviac potlačiť.

Následne sa spočíta riadkový súčet. Vďaka prahovaniu, vyjadruje prakticky samotnú dĺžku hrán. Je evidentné, že hrany kondenzátoru by mali byť svojou dĺžkou veľmi blízko šírke orezaného obrázku, aj keby boli mierne prerušené kvôli špine a podobne. Kondenzátorov sa však často objaví viac na jednom obrázku, i keď iba čiastočne. Je potrebné odlíšiť správnu hranu.

Ak počítame s faktom, že stred obrázku je vo vnútri oblasti kondenzátoru, tak môžeme z vhodne pripraveného riadkového súčtu nájsť správne hrany tým, že by mali byť prvé, na ceste od stredu obrázku smerom k spodku a vrchu.

Postupných prechodom od stredu riadkového súčtu, symbolicky stred obrazu, smerom k počiatku sa kontroluje šírka hrán. Ak sa nájde hodnota, ktorá je väčšia než 75% šírky orezaného obrázku, algoritmus skončí a nájdený index sa považuje za polohu hrany kondenzátoru. Spodná hrana je nájdená identickým spôsobom, lenže sa iteruje od stredu ku koncu riadkového súčtu.

Medzivýsledky sú zobrazené na obrázku 3.9.



Obrázok 3.9 Dielčie kroky orezania vrchných a spodných častí obrázkov

3.3.3 Poznámky k algoritmu

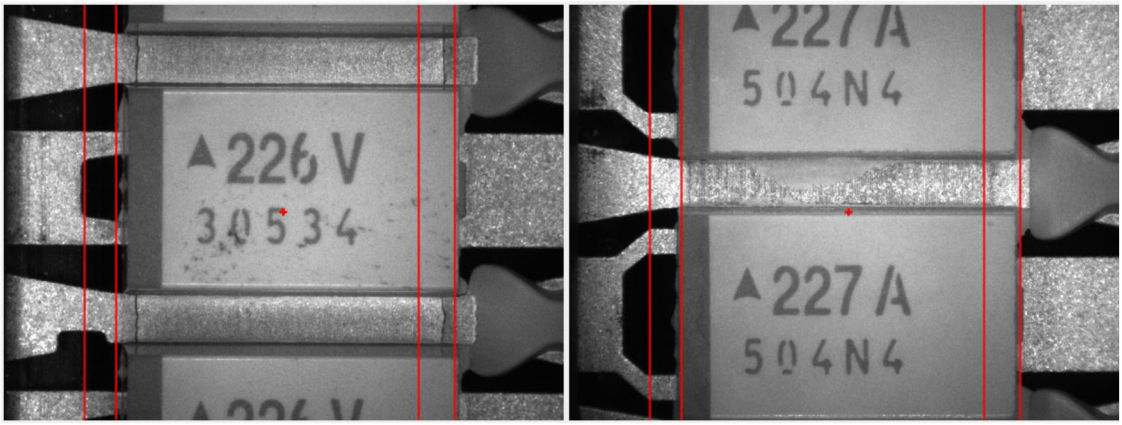
Pre prípadné ladenie algoritmu sú na začiatku kódu pripravené premenné nasledovných parametrov:

- left1 – ľavý okraj ľavej oblasti pri zvislých výrezoch
- left2 – pravý okraj ľavej oblasti pri zvislých výrezoch
- right1 – ľavý okraj pravej oblasti pri zvislých výrezoch
- right2 – pravý okraj pravej oblasti pri zvislých výrezoch
- meanCoefficient – násobkový koeficient priemeru, na vytvorenie prahu pre obraz hrán
- percentOfWidth – percento hraničnej šírky vodorovnej hrany pre ukončenie jej hľadania v riadkovom súčte
- offset – nastavením tejto premennej je možné zväčšiť výsledný obrázok algoritmu o voliteľný počet pixelov, vo všetkých smeroch rovnako

Čo sa týka času, segmentácia je iba malou súčasťou finálneho algoritmu, ale trvá 6 milisekúnd na obrázok. Doba sa môže líšiť, pretože vývoj a testovanie prebehlo v prostredí MATLAB a niektoré funkcie môžu byť optimalizované lepšie (alebo horšie) ako cieľová platforma.

3.4 Doladený segmentačný algoritmus

Algoritmus z podkapitoly 3.3 bol vyvinutý, ešte keď bol dostupný iba prvotný dataset o 20 vzorkoch. Keď sa testoval po nadobudnutí väčšieho datasetu, ukázalo sa, že sa výnimočne môže vyskytnúť výrazne posunutý kondenzátor, vo vertikálnom aj horizontálnom smere. Na obrázku 3.10 sú zobrazené takéto prípady, aj so zvýraznenými podmienkami funkcie algoritmu(oblasť hrany kondenzátoru, a stred obrázku nad kondenzátorom). Výstupy algoritmu pre pravý vzorok na obrázku 3.11, ľavý obrázok z má stred presne na hrane kondenzátoru, tým pádom vznikne úzky pás, pretože algoritmus sa ihneď ukončí.



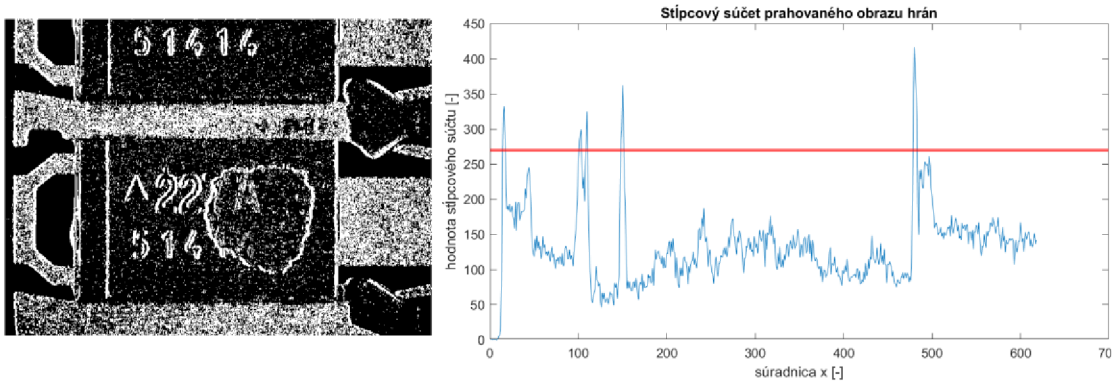
Obrázok 3.10 Výrazne posunuté kondenzátory

Obrázok 3.11 Chyba segmentácie

Ďalším faktorom je poloha vnútornej hrany polaritného pruhu. Jeho šírka sa môže mierne meniť a preto je potrebné nájsť jeho hranu dynamicky, pri spracovávaní textu popisu totiž pomáha oddeliť polaritný pruh (viac v kapitole o 5).

Segmentačný algoritmus bol teda trochu upravený, aby sa pokúsilo zlepšiť úspešnosť segmentovania a aby signalizoval polohu hrany polaritného pruhu. Miesto pevných oblastí, v ktorých sa hľadajú horizontálne hrany, je nový spôsob ich hľadania podobný hľadaniu vertikálnych hrán. Na celý obrázok je aplikovaný kernel Sobel, na zvislé hrany.

Následne sa prevedie prahovanie, a stĺpcový súčet. Medzivýsledok zobrazený na obrázku 3.12.



Obrázok 3.12 Medzivýsledky upraveného algoritmu segmentácie

Zvýraznená červená hranica je pevný parameter, určujúci vhodnú šírku hrany. Pre pravú hranu, iteruje sa po hodnotách stĺpcového súčtu od stredu obrázku po pravý okraj. Prvá dostatočne veľká hrana je výslednou. Pre ľavú hranu je to symetrické, avšak s malou úpravou. Prvá hrana na ktorú kód narazí je polaritný pruh. Zapamätá si jeho pozíciu a pokračuje ďalej. Následne nájde ľavú hranu kondenzátoru.

Čo sa týka horizontálnych hrán, funguje to ako v predchádzajúcom algoritme, alebo ako s pravou hranou. Výstupom je orezaný kondenzátor, s voliteľným offsetom. Je pridaný výstup na polohu hrany polaritného pruhu, vždy od kraju orezaného obrázku, započítaný je teda aj offset.

Nový algoritmus je menej citlivý na horizontálny posun, avšak nutnou podmienkou pre jeho funkciu je stále poloha stredu obrázku nad kondenzátorom. Znamená to, že z príkladov na obrázku 3.10 spracuje iba ľavý, znázornené na obrázku 3.13. Takto výrazne posunuté kondenzátory sú veľmi výnimočné v oboch prípadoch, takže sa algoritmus ďalej neupravil.

Obrázok 3.13 Napravená segmentácia

3.4.1 Parametre

Novému algoritmu vypadlo pár parametrov, zostali nasledovné:

- verticalEdgesThrsh – hodnota pre prahovanie obrazu vertikálnych hrán pri hľadaní bočných hrán kondenzátoru
- percentageOfWidth – percento šírky horizontálnej hrany z šírku okrajovo orezaného obrázku pre ukončenie ich hľadania

Offset je tiež parametrom využitým v tomto algoritme, ale v zdrojovom kóde je presunutý do premenných triedy smdSample, ktorá zahrňuje rôzne kroky spracovania a premenné súvisiace s jedným vzorkom. Toto bolo nutné preto, aby mohli všetky metódy využívať hodnotu offsetu.

3.4.2 Zhodnotenie algoritmu

Vyhodnotenie úspešnosti by som rozdelil na dve časti – formálnu a neformálnu. Ako formálne vyhodnotenie, ručne sa pomocou algoritmu orezali všetky obrázky z testovacej časti datasetu (220). Na žiadnom sa neobjavil problém s výsledkom.

Neformálne, dodal by som pár poznámok. Problémový kondenzátor v pravo z obrázku 3.10 je veľmi okrajovým prípadom, stretol som sa možno s 10 príkladmi z 37000, kde by sa stred obrázku nenachádzal nad kondenzátorom. Keďže samotný kondenzátor takmer nie je na obrázku celý, predpokladal by som, že je to zlyhanie systému, ktorý spúšťa kameru.

Ďalším drobným detailom je, že hrany kondenzátorov sú občas v rozmedzí pár pixelov nejasné. Niektoré sú pod malým uhlom a vidieť ich steny, alebo sa nachádza malá lesklá oblasť pri okraji. Z tohto dôvodu, a z princípu algoritmu, vzniká prísne orezanie z vnútornej strany rozhrania okraja. Tento detail sa nezdá byť problémový, a aj keby, je vyriešený pridaním offsetu, ktorý je nastavený na 20 pixelov, z dôvodu lepšej prehľadnosti ulomenín na okraji, či už pre používateľa alebo neurónovú sieť.

4. DETEKCIA POVRCHOVÝCH VÁD

V tejto kapitole je vysvetlený postup vytvorenia detektoru povrchových vád na kondenzátoroch.

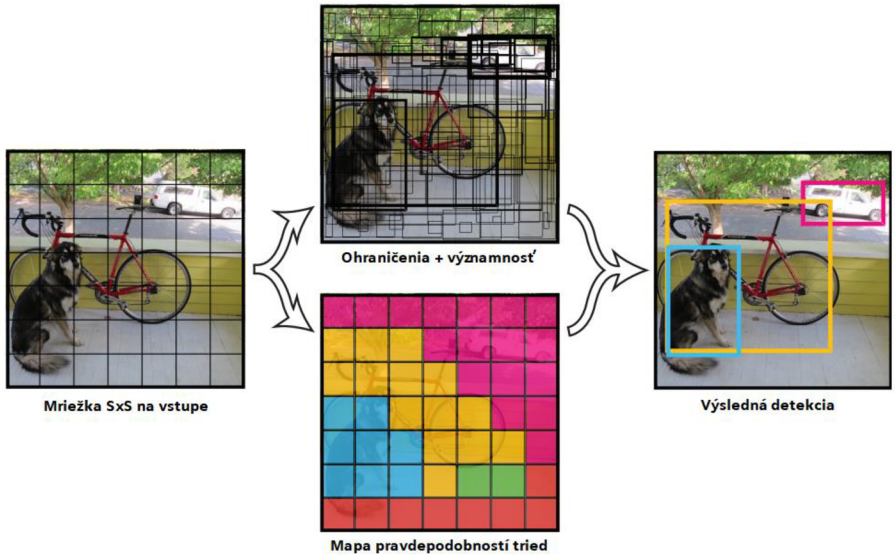
4.1 Neurónová sieť YOLO

Počas konzultácií s vedúcim práce som bol nasmerovaný k ceste hlbokého učenia, čo sa týka vyhodnocovania chýb kondenzátorov. Bola mi doporučená konkrétne architektúra YOLO.

4.1.1 YOLO(v1)

YOLO – You Only Look Once – je sieť vykonávajúca detekciu objektov v reálnom čase. V dobe prvotného vyvinutia, urobila veľký pokrok v smere zvýšenia rýchlosti a presnosti detekcie. Autori porovnávali YOLO s inými existujúcimi architektúrami, ktoré podľa mňa zapadajú do dvoch kategórií – presnosť na úkor rýchlosti (R-CNN) a rýchlosť na úkor presnosti (DPM). Rôzne prevedenia YOLO boli pri rovnakej presnosti násobne rýchlejšie ako iný architektúry. [8]

YOLO je založené na zjednotení dvoch komponentov detekcie objektov. Prvým je rozdelenie obrázku na mriežku SxS oblastí a odhadnutie tried obsiahnutých v nich. Druhým je odhad ohraničení a významnosti, ku ktorému sa využívajú príznaky z celého obrázku. Predikujú sa všetky triedy zároveň. Znázornenie princípu je možné vidieť na obrázku 4.1. [8]



Obrázok 4.1 Prístup zjednotenia v YOLO[8]

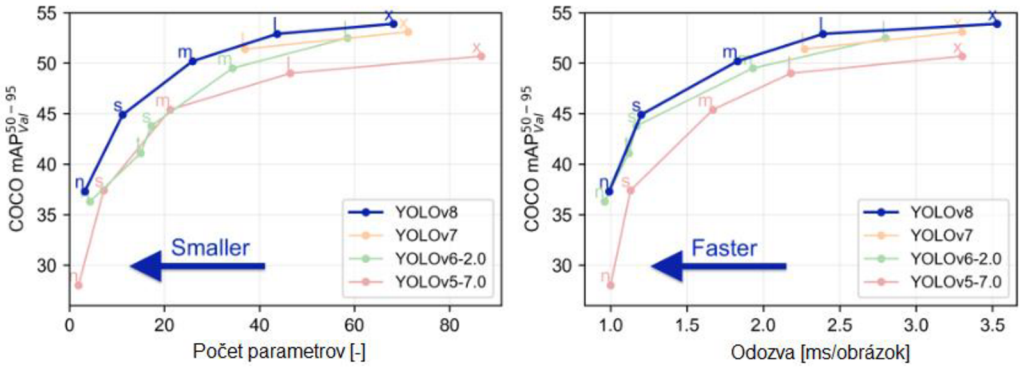
YOLO bolo po prvej verzii z roku 2016 ďalej vyvíjané, a do dnešnej doby stále je v aktívnom vývoji. Pôvodný autor odstúpil od vývoja v oblasti počítačového videnia po verzii YOLOv3, kvôli obavám zo zneužitia výsledkov jeho práce pre vojenské účely [9].

4.1.2 YOLOv8

YOLOv8 je vyvinuté spoločnosťou Ultralytics, ktorá sa podieľala na vývoji predchádzajúcich verzií. Dátum vydania je január roku 2023. Nie je najnovšou verziou, avšak tým Ultralytics sa postaral o to aby inštalácia a práca so sieťou bola veľmi jednoduchá, pretože v tomto smere bolo veľa sťažností zo strany komunity, primárne z inžinierskej strany, ktorá by chcela technológiu využiť v praxi. Predchádzajúce verzie tiež nemali oficiálny vedecký článok, na čo si sťažovala vedecká komunita. YOLOv8 má priniesť aj toto, avšak v čase písania tejto práce som ho nedohľadal. [10]

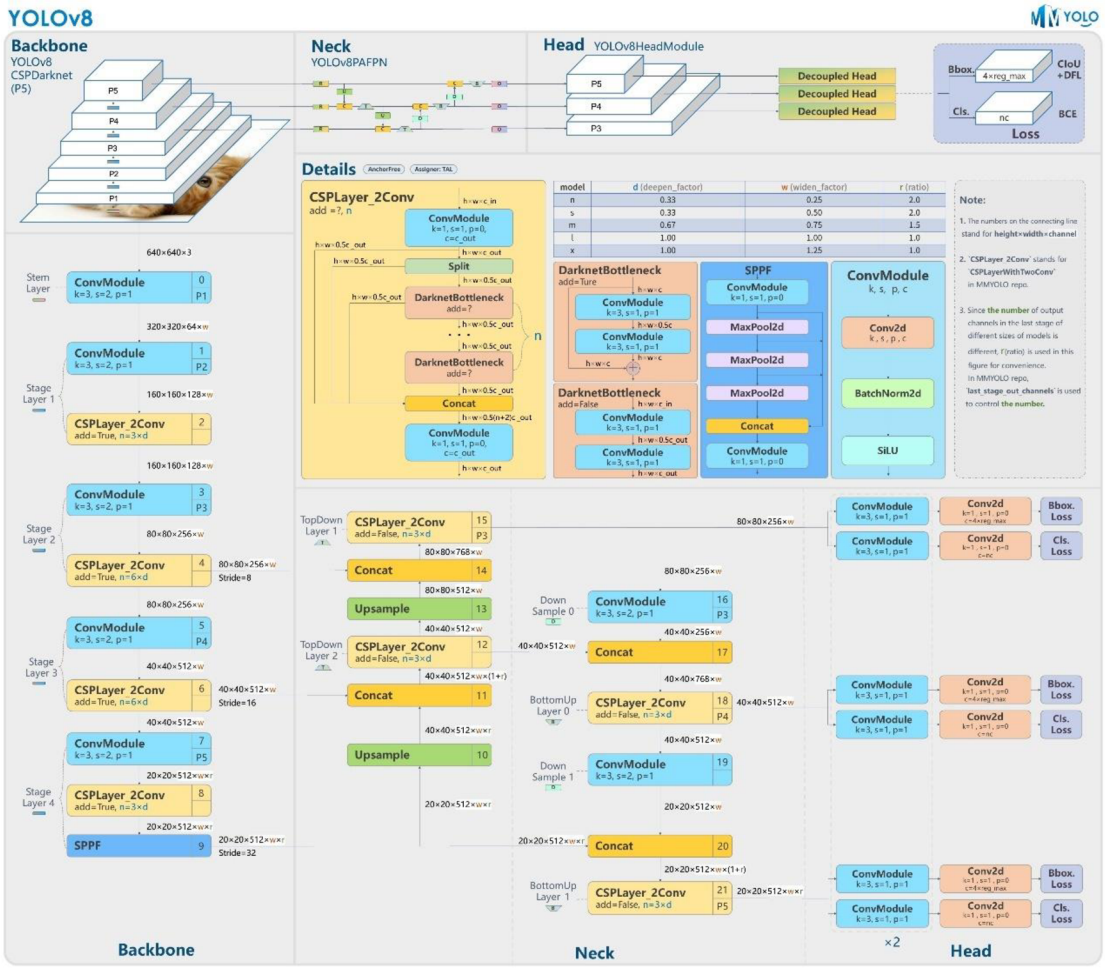
Vďaka ľahkej dostupnosti a širokým možnostiam pomoci (napríklad cez Discord server Ultralytics, kde sa často stane, že pomáhajú ľudia z Ultralytics), je táto verzia výhodnou voľbou.

YOLOv8 má až 5 rôznych dostupných veľkostí, a to nano, small, medium, large, extra-large. V závislosti na veľkosti, je možné očakávať inú presnosť a časovú odozvu. Na obrázku 4.2 sú znázornené výsledky na štandardnom datasete COCO, v porovnaní s inými YOLO modelmi. Rýchlosť odozvy je meraná na grafickej karte NVIDIA A100, a optimálnym formátom modelu pre grafické karty, nazývanej TensorRT.



Obrázok 4.2 Potenciál modelu YOLOv8[11]

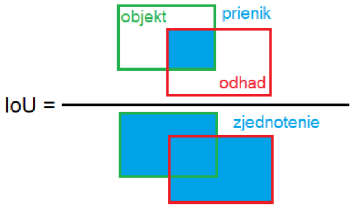
YOLOv8 je veľmi komplikovaný model, a vysvetlenie všetkých vlastností, použitých metód v rámci neho by bolo zdĺhavé, aj z dôvodu, že by bolo možno potrebné vysvetliť zmeny počas celého vývoja série. Myslím si, že takto detailný popis nezapadá do zamerania tejto práce. Z dôvodu nedostupnosti vedeckého článku, v ktorom je všetko spísané pohromade, ohľadom tohto modelu, sa tento proces ešte viac komplikuje. Ak má čitateľ záujem, pridal som na obrázku 4.3 detailný popis architektúry siete. Ak má záujem o prehľad zmien v rôznych verziách YOLO až YOLOv8, odporúčam článok [12], prípadne [13] pre YOLOv8 konkrétne.



Obrázok 4.3 Architektúra YOLOv8 [13]

4.1.3 Vyhodnocovacie metriky [14]

Pre vyhodnotenie modelov na detekciu objektov sa používajú špecifické metriky, ktoré je vhodné vysvetliť. Prvou je Intersection over Union, značené ako IoU, preložené ako prienik nad zjednotením. Vyjadruje pomer prieniku predikovaného ohraničenia objektu a skutočného ohraničenia (označeného), k ich zjednoteniu. V tomto prípade je najlepšie grafické znázornenie – obrázok 4.4. Pre vyhodnotenie ako správna identifikácia, musí byť IoU väčšie ako určitý prah, často 0,5.



Obrázok 4.4 Intersection over union

Precision, preložené ako presnosť, je metrika, ktorá vyjadruje správnosť predikcií triedy. Počíta sa podľa vzorca

$$Precision = \frac{TP}{TP + FP} , \tag{4.1}$$

kde TP sú správne identifikácie (true positive) a FP sú chybné identifikácie (false positive).

Recall, preložené ako úplnosť, popisuje schopnosť nájsť všetky prípady danej triedy. Počíta sa podľa vzorca

$$Recall = \frac{TP}{TP + FN} , \tag{4.2}$$

kde TP sú správne identifikácie (true positive) a FN sú nenájdené objekty (false negative).

F1 skóre, je kombináciou týchto dvoch metrík, a počíta sa podľa vzorca

$$F1 = 2 * \frac{precision * recall}{precision + recall} . \tag{4.3}$$

Poslednou, a primárnou metrikou je Average Precision, AP, priemerná presnosť, prípadne mean average precision, mAP. AP sa pre jednu danú triedu, počíta ako plocha pod krivkou závislosti presnosti na úplnosti v rozsahu 0 - 1 (precision = f(recall)). Ak to vykonáme pre prah IoU 0,5, nazveme výsledný parameter AP@0,5. V prípade vyhodnocovania detektorov objektov je často využitá metrika AP@[0,5:0,95], ktorá tvorí priemer metrík AP pre 11 rôznych prahov IoU, v rozsahu 0,5 až 0,95, s krokom 0,05. Ak sa zoberie priemer týchto metrík cez všetky hľadané triedy objektov, dostávame mAP, pre odpovedajúci prah IoU.

Metriky mAP a AP majú často zamenené značenia, čo ma neprekvapuje, keďže sa napríklad v prípade mAP@[0,5:0,95] jedná o „priemer priemeru priemeru“. Chcem primárne upozorniť na to, že v kontexte YOLO sa používa značenie mAP50-95 pre jednotlivé triedy, aj priemer všetkých tried, a to platí aj pre mAP50.

4.2 Učenie na datasete smd500

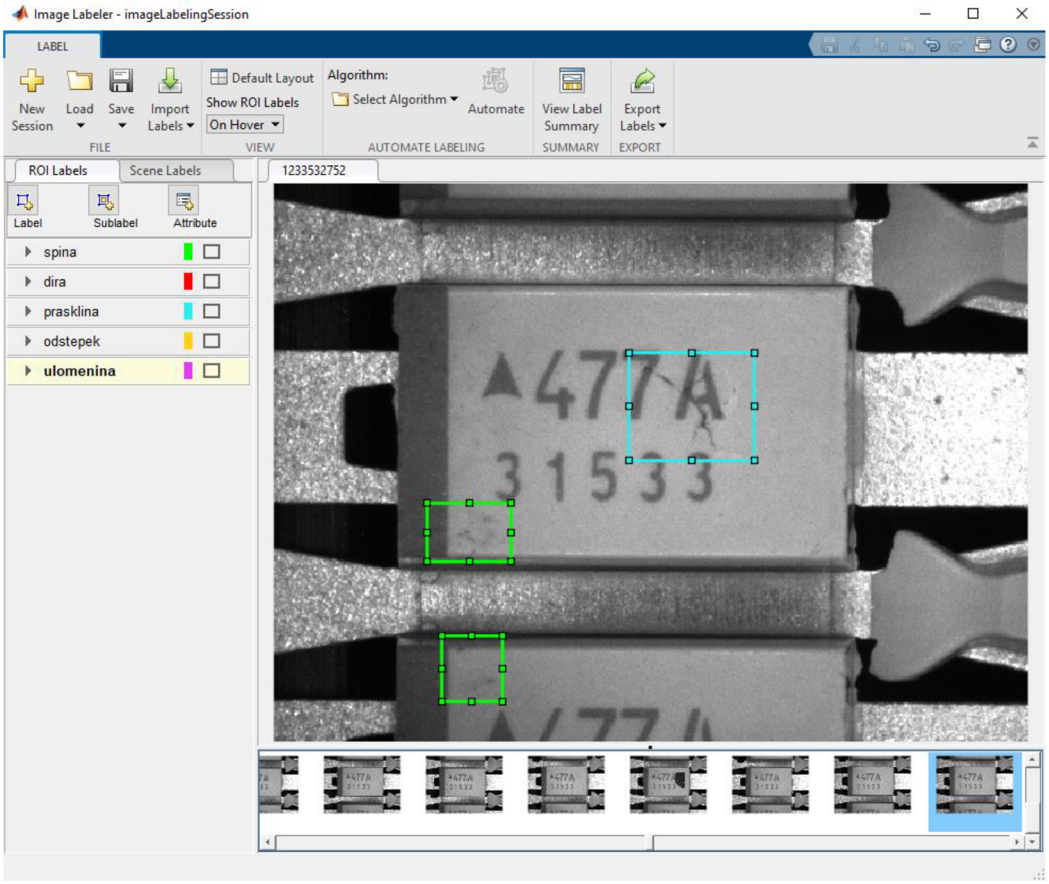
Prvé pokusy s učením modelu YOLOv8 prebiehali na datasete s pracovným názvom smd500. Prvé väčšie množstvo obrázkov, ktoré mi bolo sprístupnené, bolo približne o počtu 5700 kusov. Po pretriedení podľa obsahu nežiadúcich povrchových vád, a doplnení s extra špinou a pozadím (obrázkov bez detekovateľných objektov), sa dataset zredukoval na 557 obrázkov.

4.2.1 Značkovanie a príprava datasetu

Detekované objekty odpovedajú vadám na kondenzátoroch, popísaným v podkapitole 1.2.1. Pre smd500, sa označovalo 5 tried, a to špina, odštiepok, ulomenina, prasklina

a diera. Pôvodným zámerom bolo detekovať špinu, avšak kondenzátor by mal byť označený ako vadný kus, iba ak obsahuje jednu zo zvyšných štyroch vád.

Značkovanie prebiehalo v MATLAB aplikácie Image Labeler. Tu sa vytvorili označenia pre triedy, a načítali sa fotky do programu. Pri sieťach YOLO sa väčšinou odporúčajú iné značkovacie nástroje, ale mnoho z nich je online, a prišlo mi rozumnejšie použiť offline nástroj miesto nahrávania stoviek fotiek na server. Ukážka prostredia je viditeľná na obrázku 4.5.



Obrázok 4.5 MATLAB Image Labeler

Vytvorené značky je možné exportovať do workspace, alebo uložiť do súboru. Nevýhodou je, že Image Labeler nepodporuje export do formátu potrebného pre YOLOv8. Formát z Image Labeler je, že obrázok odpovedá riadku v matici, kým triedy odpovedajú stĺpcom. Číselné vyjadrenie polohy a rozmeru je určenie x a y súradnice ľavého horného rohu, následne výška a šírka, v počte pixelov. Pre model YOLOv8, je potrebné pre fotku „xyz.tif“ vytvoriť súbor textový súbor „xyz.txt“. Tento súbor musí obsahovať informáciu o ohraničeniach objektov na danom obrázku v nasledujúcom formáte:

```
0 0.769355 0.440217 0.103226 0.506522
0 0.506452 0.617391 0.541935 0.147826
1 0.737097 0.865217 0.045161 0.056522
2 0.260484 0.382609 0.053226 0.282609
```

Každý riadok odpovedá jednému ohraničeniu objektu. Prvé číslo v riadku vyjadruje triedu objektu. Dve nasledujúce čísla určujú stredový bod ohraničenia, kým ďalšie dve určujú šírku a výšku. Hodnoty pre určenie polohy a rozmerov sú v rozsahu 0 až 1, a sú relatívnou jednotkou z veľkosti odpovedajúcich strán (rozlíšenia) obrázku. Pre vytvorenie požadovaného formátu uloženia ohraničení objektov sa napísal krátky MATLAB kód, ktorý spracuje premennú obsahujúcu označenia na obrázkoch a vytvorí k ním príslušný súbor.

Dataset bol následne rozdelený do troch častí: trénovací, validačný a testovací dataset. Rozdelený bol v pomere 70:20:10, približne, na základe generovania náhodnej hodnoty. YOLOv8 pri učení pracuje s trénovacím a validačným datasetom. Testovací je použitý na vyhodnotenie úspešnosti, na nových dátach. Pre YOLOv8 je potrebné vytvoriť .yaml súbor, ktorý popisuje umiestnenie datasetu, štruktúru dát a indexy odpovedajúce menám tried. Súbor vyzerá takto:

```
path: C:\(tu doplnit cestu)\dataset_500
train: train\images
val: val\images
test: test\images

names:
  0: spina
  1: dira
  2: prasklina
  3: odstepek
  4: ulomenina
```

Štruktúra súborov v rámci priečinku datasetu je potom nasledovná:

```
dataset_500\
      train\
            images\
                  123.tif
            labels\
                  123.txt
      val\
            images\
                  456.tif
            labels\
                  456.txt
      test\
            images\
                  789.tif
            labels\
                  789.txt
```

4.2.2 Používanie YOLOv8

Inštalácia YOLOv8 je veľmi jednoduchá. Pomocou nástroja pip sa nainštaluje knižnica ultralytics do prostredia Python aj s ostatnými potrebnými knižnicami, na ktorých závisí (dependencies). Doporučujem si knižnicu PyTorch nainštalovať správne ešte pred týmto krokom, hlavne ak sa plánuje využiť grafická karta na učenie.

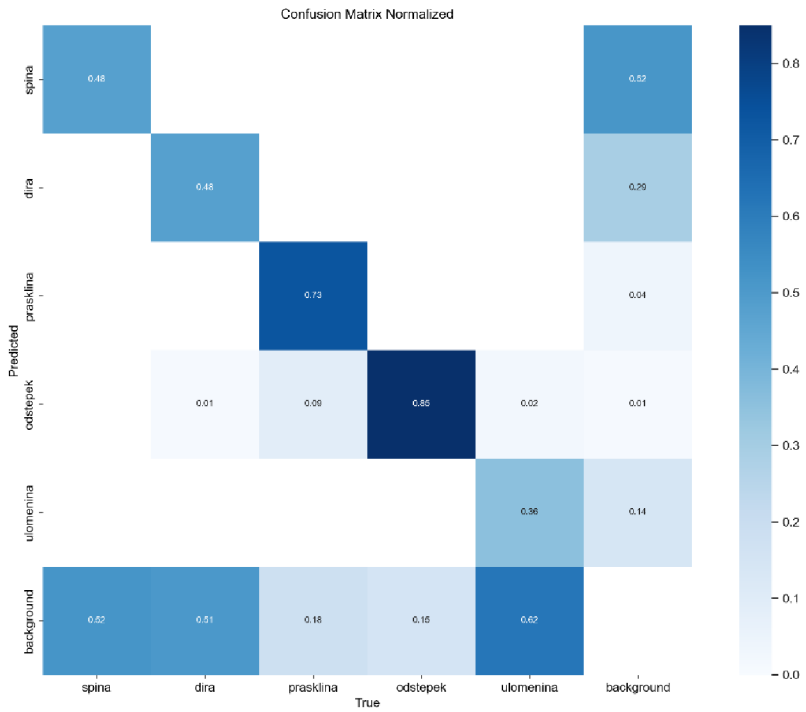
Od tohto momentu je všetko pripravené. Na učenie sa využije jednoduchého kódu:

```
from ultralytics import YOLO

model = YOLO("yolov8n.pt")

if __name__ == '__main__':

    model.train(data="smd_500.yaml", epochs=300, pretrained=True)

    model.val(data="smd_500.yaml", split="test")
```

Prvý riadok jednoducho načíta do prostredia knižnicu. Na druhom riadku sa načíta do pamäte model YOLOv8 nano, ktorý je predtrénovaný. Metóda train vykoná celé učenie, je možné nastaviť parametre ako použitý dataset, počet epochov, a mnoho iných. Metóda val slúži na vyhodnotenie modelu, v tomto prípade na testovacej časti datasetu. Obidve metódy majú radu grafických výstupov, ktoré sa ukážu pri prezentovaní výsledkov. Podmienka if je potrebná pre zaručenie správnej funkcie pri využití PyTorch CUDA s grafickou kartou.

4.2.3 YOLOv8 nano s smd500

Ako prvý pokus učenia sa vybrala najmenšia veľkosť modelu. Učenie sa automaticky zastavilo po 226 epochoch, najlepšia verzia bola po 126 epochoch. Toto je spôsobené parametrom patience, ktorý zastaví učenie, ak sa sieť nezlepší. V tomto prípade trval epoch približne 10 sekúnd. Po učení sa vykonala validácia na testovacom datasete. Dosiahnuté výsledky sú prezentované pomocou normalizovanej matice zámen na obrázku 4.6.

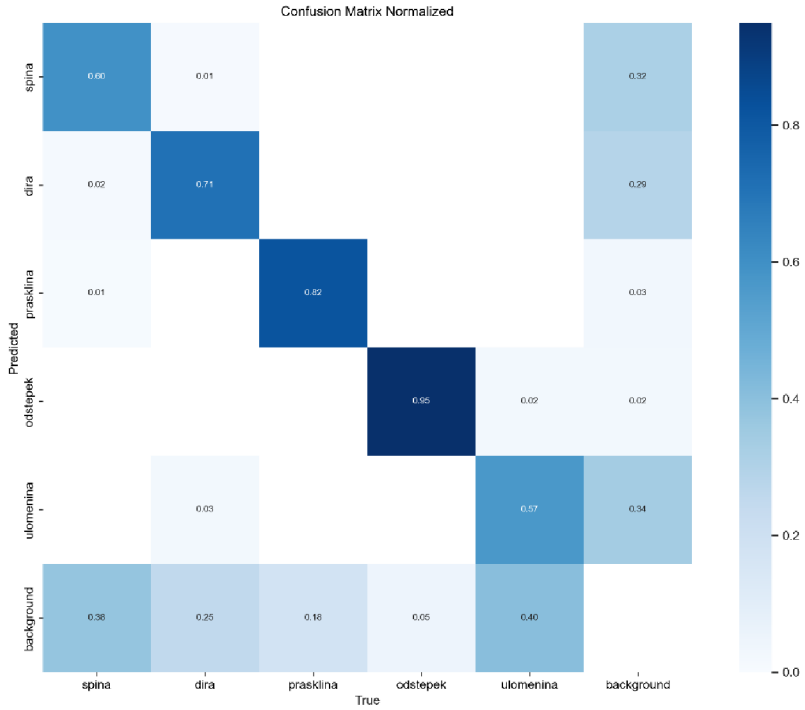


Obrázok 4.6 Matica zámen 1

V matici zámen je možné vidieť, že výsledky sú pre rôzne triedy značne odlišné. Odštiepky dosahujú dosť presných hodnôt, a praskliny za nimi zaostávajú iba mierne. Zvyšné triedy sú veľmi nepresné. Model napríklad väčšinu ulomenín vôbec nenájde. Tiež identifikuje veľké množstvo pozadia ako špinu, chybne.

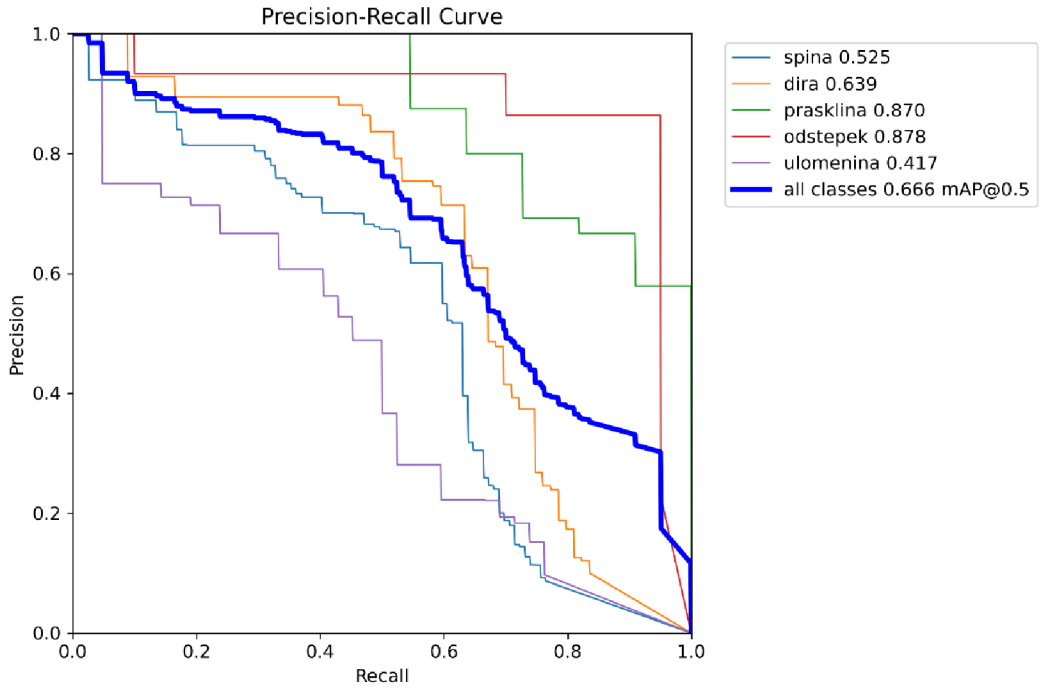
4.2.4 YOLOv8 medium s smd500

So zámerom zlepšiť výsledok detekcie, sa vykonalo učenie s väčšou verziou modelu, YOLOv8 medium. Učenie dobehlo do konca, 300 epochov, avšak model sa už ustálil. V tomto prípade trval 1 epoch asi 2:30 minút. Matica zámen po validácii na testovacom datasete je zobrazená na obrázku 4.7.



Obrázok 4.7 Matica zámen 2

Väčší model mal, ako sa dalo čakať, pozitívny efekt na úspešnosť detekcie. Viditeľne zlepšil odhad pre každú triedu. Model však stále neodhaduje vačšinu kritických tried dostatočne presne. Aby bolo možné pre čitateľa lepšie porovnať výsledky modelu s ďalšími pokusmi, je pridaná aj krivka presnosti-úplnosti(precision-recall curve), na obrázku 4.8. Je očíslovaná dvojkou, napriek tomu, že je prvá, aby bolo jasné, že patrí k matici zámen 2.



Obrázok 4.8 Krivka precision-recall 2

4.3 Učenie na datasete smd1500

Po hľadaní informácií, ako zlepšiť model, sa rýchlo došlo k tomu, že dataset obsahuje malé množstvo dát.

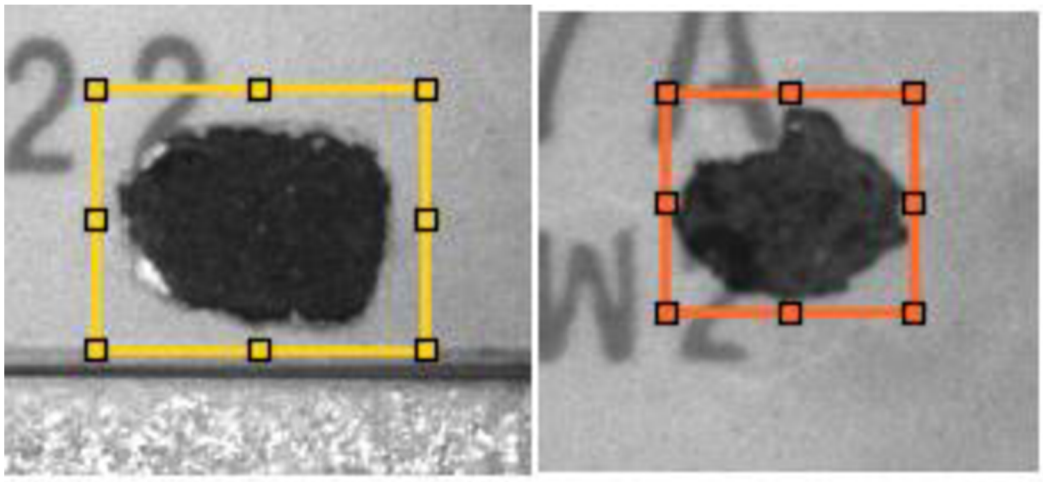
4.3.1 Rozšírenie datasetu

Pre rozšírenie datasetu mi bol sprístupnený celý súbor fotiek, o počte cca 37000. Neotestovalo sa prekrytie s predchádzajúcou várkou, ale ak by toto mali byť všetky fotky tak by predchádzajúca várka mala byť obsiahnutá v novej.

Mierne zmenený prístup – špina sa už nepovažuje za detekované objekty, ale za pozadie. To dáva zmysel v súlade s úvahou, že sa nepovažuje za problém, a taktiež uľaví veľa záťaže pri značkovaní. Značkované triedy sú teda odštiepky, diery, praskliny a ulomeniny.

Čo sa týka počtu, z 37000 obrázkov sa vytriedilo okolo 1300, ktoré obsahovali nežiadúce vady. Pri značkovaní sa dbalo nato, aby ohraničenia boli čo najpresnejšie okolo objektov. Porovnanie medzi voľnejším starým prístupom a precíznejším novým prístupom je znázornené na obrázku 4.9. Po označení sa obrázky rozdelili do trénovacieho, validačného a testovacieho datasetu, opakovane v pomere 70:20:10. Čo sa týka pozadia, obsahujúceho špinu aj bez špiny, vybralo sa asi 250 obrázkov. Boli rozdelené tak, aby v trénovacom a validačnom datasete tvorili približne 10% (okolo 100 a 25 obrázkov), zvyšok sa doplnil do testovacieho datasetu. Toto bude klásť vyššie nároky

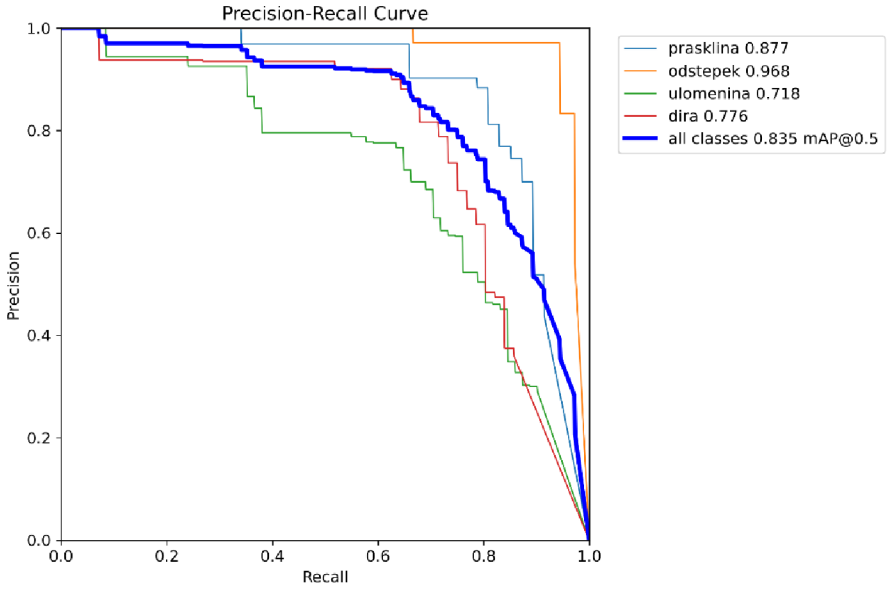
na model pri testovaní, čo sa týka FP. Nový dataset má teda celkovo 1560 obrázkov, preto pracovný názov smd1500.



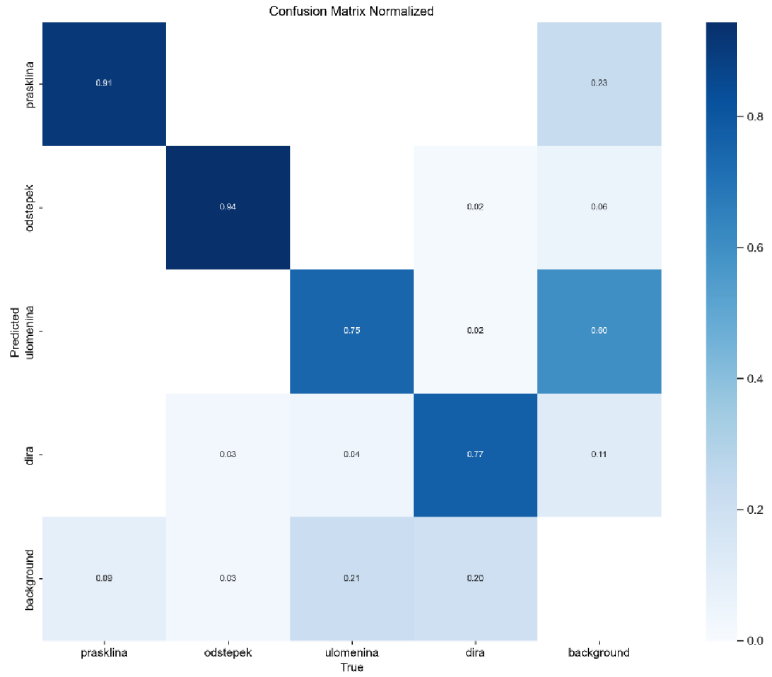
Obrázok 4.9 Precíznosť pri značkovaní

4.3.2 YOLOv8 nano

Ako prvé sa vyskúšal nový dataset s modelom nano. Objavil sa nastaviteľný parameter pre učenie zvaný „batch“, ktorý umožňuje nastaviť percento pamäte grafickej karty, ktoré sa má využívať pri učení. Implicitná hodnota je konštantná veľkosť obrázkov v rámci jedného batch, čo nemusí byť optimálne. Pre toto učenie sa použil batch 0,7 a na jeden epoch potreboval algoritmus 25 sekúnd. Dobehol 300 epochov, bol však v ustálenom stave. Po validácii na testovacom datasete sa dosiahlo výsledkov na obrázkoch 4.10 a 4.11.



Obrázok 4.10 Krivka precision-recall 3



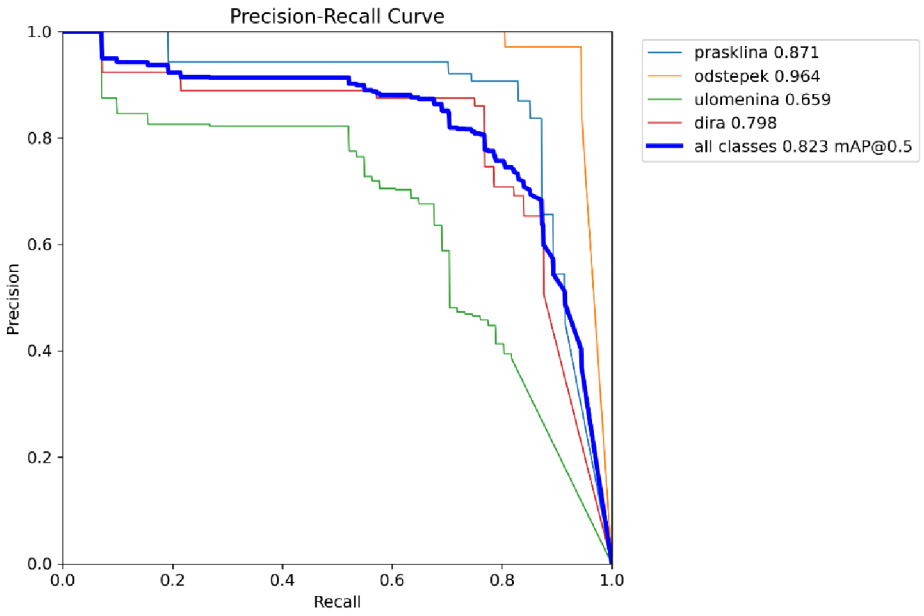
Obrázok 4.11 Matica zámen 3

Matica zámen má prehodené poradie tried, ale je možné vidieť, že presnosť klasifikácie sa výrazne zlepšila, aj keď sa porovná s hlbším modelom medium pre smd500. Čo sa týka krivky precision-recall, je tiež možné pre všetky triedy vidieť výrazný posun smerom k AuC(plocha pod krivkou) rovné 1.

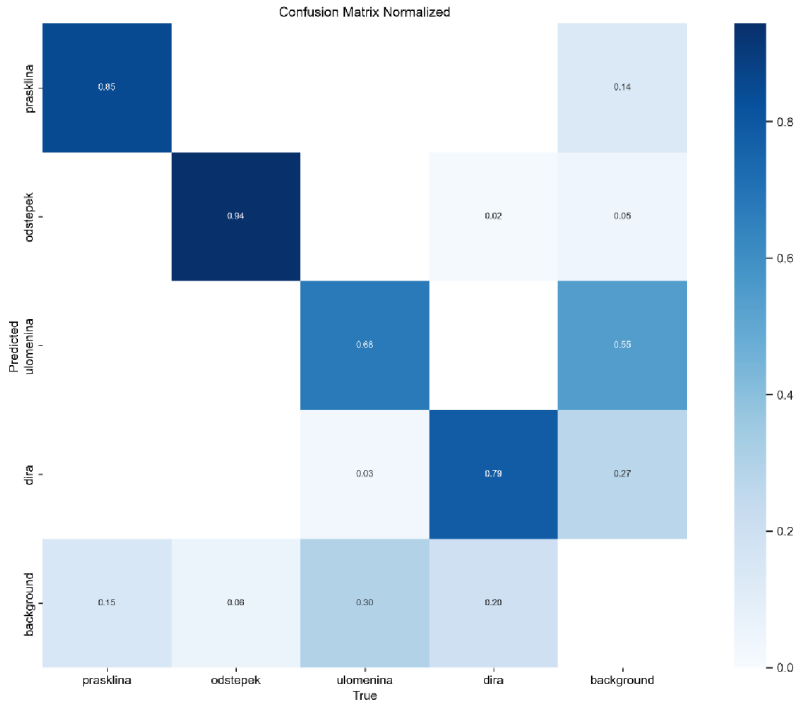
4.3.3 YOLOv8 medium

Ďalším krokom je zase len pozrieť sa na výsledky hlbšej siete. Batch 0,7 umožnil epochy trvajúce 1:40, čo je s trojnásobným datasetom výrazne rýchlejšie ako pri smd500(2:30). Algoritmus sa zastavil pri epochu 265, s najlepším výsledkom na epochu 215. Výsledky validácie na testovacom datasete viditeľné na obrázkoch 4.12 a 4.13.

V tomto prípade nastal zaujímavý úkaz. Výsledky sa pre jednotlivé triedy trochu zhoršili. Výraznejšie je to pre ulomeniny, kde AP@0.5 pokleslo z 0,718 na 0,659.



Obrázok 4.12 Krivka precision-recall 4



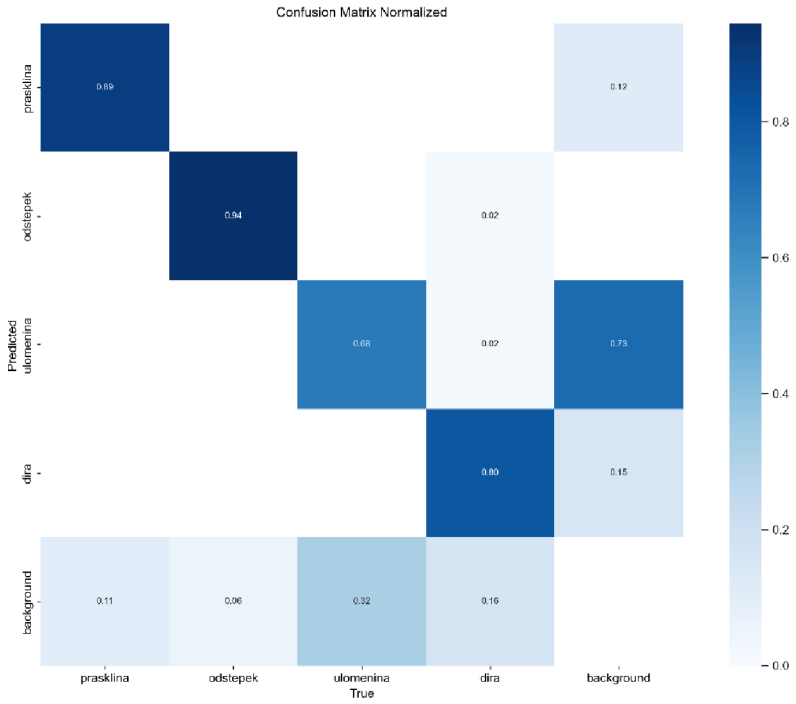
Obrázok 4.13 Matica zámen 4

4.3.4 YOLOv8 nano so zväčšeným vstupným obrázkom

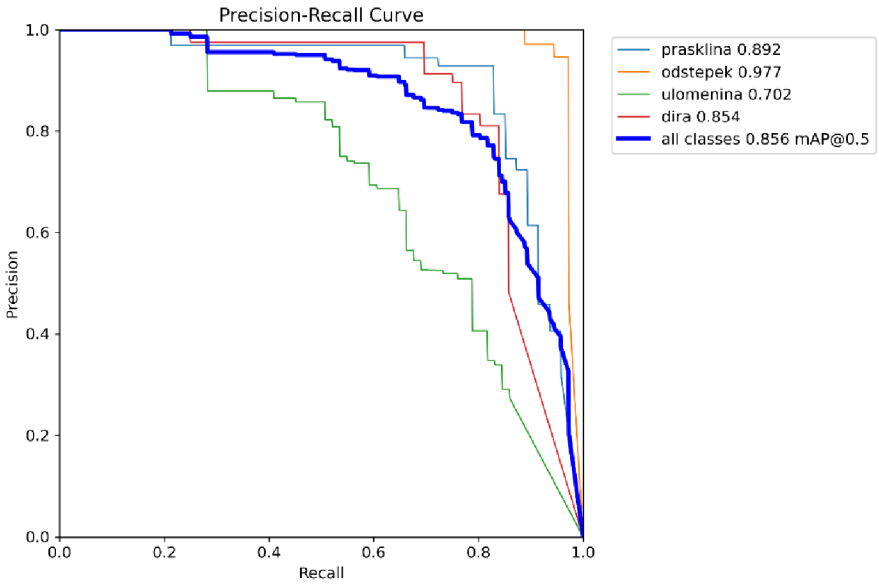
V rámci ďalšieho pokusu sa zväčšila veľkosť vstupného obrázku siete zo základných 640x640 na 1280x1280. Dôvodom je, že veľká časť hľadaných objektov je malých rozmerov, a takto by sa mohlo zlepšiť rozlišiteľnosť.

Učenie sa ukončilo po 225 epochoch, s najlepším výsledkom po 175. Nezapamätal som si dobu jedného epochu, ale viem, že veľkosť obrázkov mala silný vplyv na ňu, a učenie bolo rádovo tak zdĺhavé ako učenie YOLOv8 medium. Výsledky opäť validácie na testovacom datasete viditeľné na obrázkoch 4.14 a 4.15.

Aj v tomto prípade je možné pozorovať mierne zhoršenie oproti obyčajnému nano modelu, hlavne pre triedu ulomenín.



Obrázok 4.14 Matica zámen 5



Obrázok 4.15 Krivka precision-recall 5

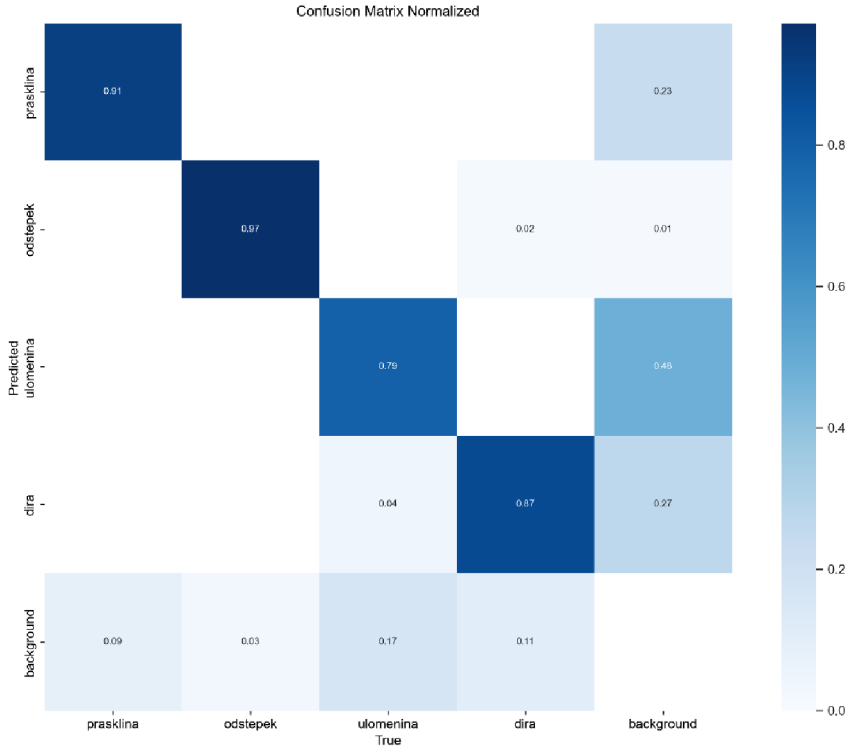
4.3.5 YOLOv8 nano s upravenou augmentáciou

Metóda train umožňuje augmentáciu vstupných dát. V základe je použité otočenie okolo vertikálnej osi „fliplr“, variácia vo všetkých parametroch farebného modelu hsv, translácia, škálovanie, vymazávanie častí obrázku a orezávanie obrázku.

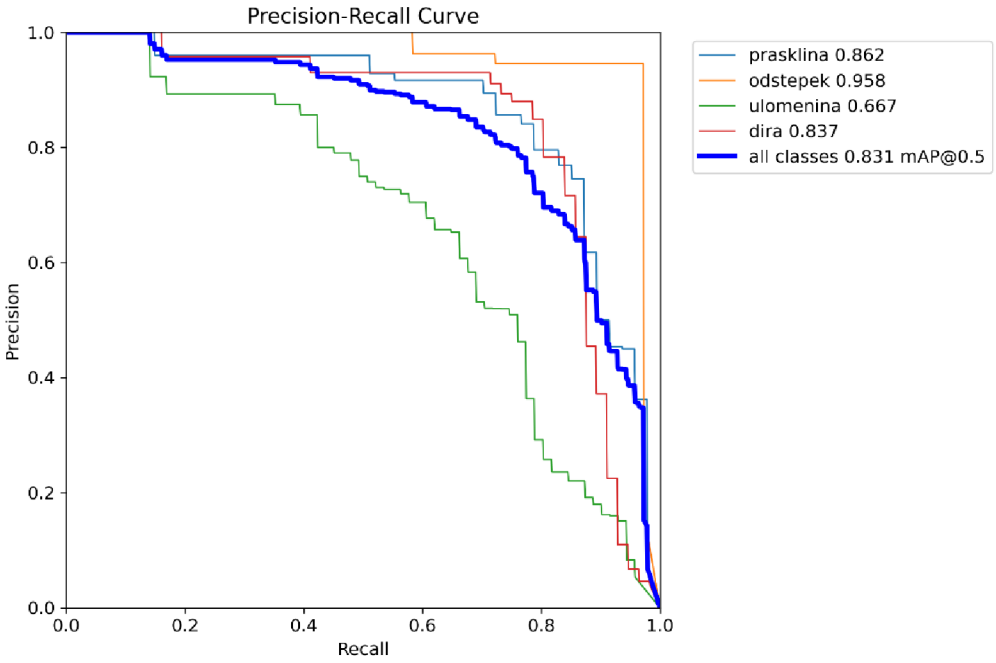
Pre prispôsobenie augmentácie k datasetu, sa upravili parametre. Variácia hsv sa ponechala iba v zložke value, keďže sa jedná o šedo tónové obrázky. Vynulovalo sa škálovanie, pretože vzdialenosť objektov od kamery sa nemení. Pridalo sa otočenie okolo horizontálnej osi „flipud“. Pridal sa „mixup“, ktorý zmieša dva obrázky a ich označenie, a mal by zlepšiť generalizáciu. Pridal sa „copy_paste“, ktorý kopíruje objekty z jedného obrázku na druhý a je užitočný pre zvýšenie počtu inštancií objektov.

Učenie sa ukončilo po 156 epochoch, s najlepším po epochu 106. Toto je výrazne zrýchlené učenie oproti obyčajnému nano. Toto však môže byť spojené aj s náhodným charakterom učenia a jeho predĺženia (presnejšie nezastavenia) kvôli inkrementálne väčšej presnosti. Výsledky sú pozorovateľné na obrázkoch 4.16 a 4.17.

Pre praskliny a odštiepky sa nestala žiadna výrazná zmena. Pre diery výrazne narástol počet TP, ale narástol aj počet FP z pozadia. Toto zlepšilo hodnotu AP@0,5 z krivky precision-recall. Pre ulomeniny počet FP z pozadia mierne poklesol. Toto zhoršilo hodnotu AP@0,5.



Obrázok 4.16 Matica zámen 6

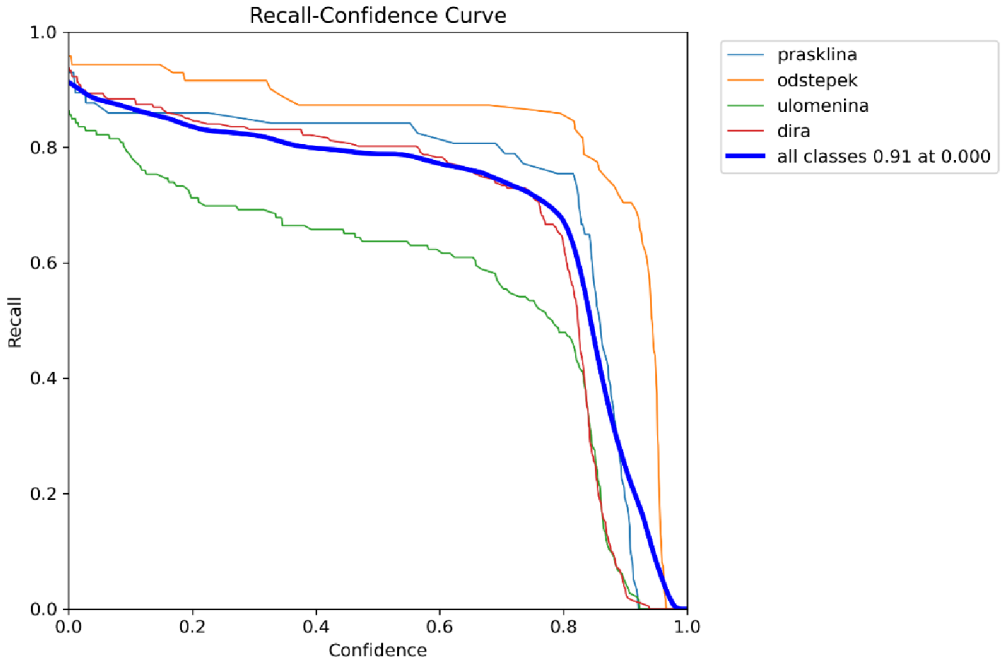


Obrázok 4.17 Krivka precision-recall 6

4.4 Zhodnotenie výsledkov

4.4.1 Úspešnosť

Pokusy s učením modelu YOLOv8 nepriniesli dokonalé výsledky. Jediná trieda, ktorú bol model schopný detekovať s vysokou presnosťou sú odštiepky. Táto trieda je veľmi výrazná a má ľahkú rozoznateľnosť oproti ostatným. Triedu prasklín by som zaradil na druhé miesto, veľmi blízko za odštiepkami. Po väčšom skoku sú diery, a po ďalšom skoku sú ulomeniny. Znázornil by som krivku úplnosti (recall), závislú na významnosti (confidence), pre obyčajné YOLOv8 nano na smd1500. Obrázok 4.18.



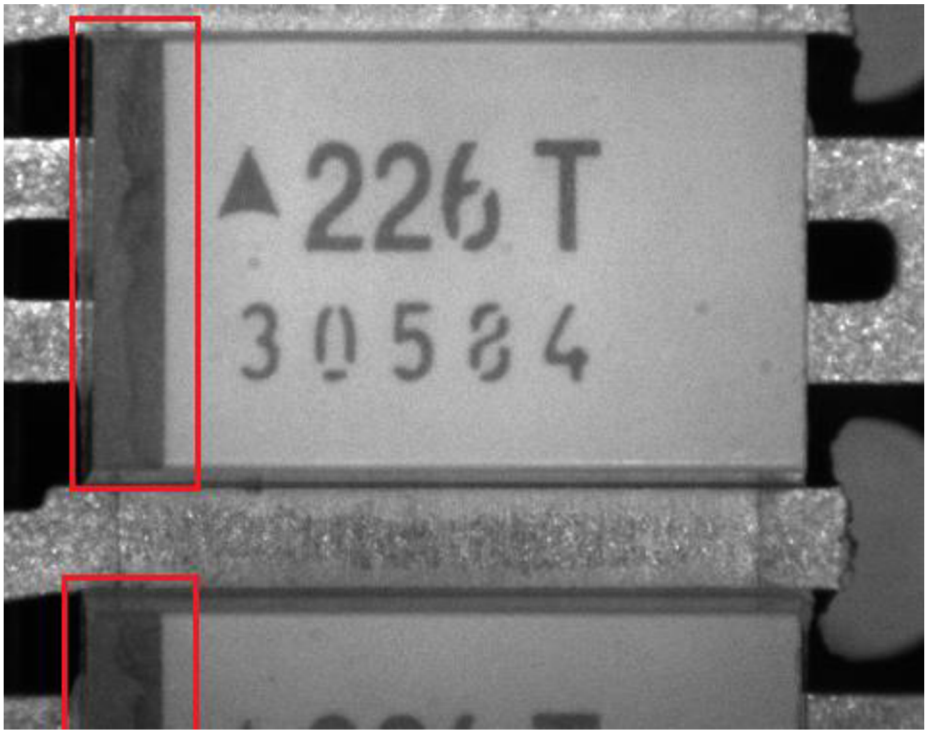
Obrázok 4.18 Krivka úplnosti 3

Úplnosť vyjadruje schopnosť detekovať všetky označené objekty, tuto konkrétne v závislosti na významnosti. Je možné pozorovať, že veľkosť plochy pod krivkou jednotlivých tried odpovedá predstavenému zoradeniu, preto si myslím, že sa problém týka tejto metriky. Ak spojíme pozorovania s maticou zámen, usúdil by som, že je problém so značkovaním datasetu. Pomer TP (diagonála matice) ku FN (riadok background), je pomerne dobrý, oproti finálnym výsledkom. Dosť to kazia FP (stĺpec background), pre ulomeniny napríklad veľmi výrazne. Usúdil by som z toho, že model je schopný dosť dobre nájsť objekty, avšak často im chýba označenie.

Keďže som značkoval dataset, predchádzajúce úvahy viem celkom presne napasovať do problémov s ktorými som sa stretol. Rozoznávanie vád je pre tento prípad náročné aj pre človeka. Povedal by som že sa dá ľahko niektoré vady zameniť, alebo ich nebrať za významné. Myslím, že by bolo nutné určiť pevné hranice pre vady. Ak by som mal tento

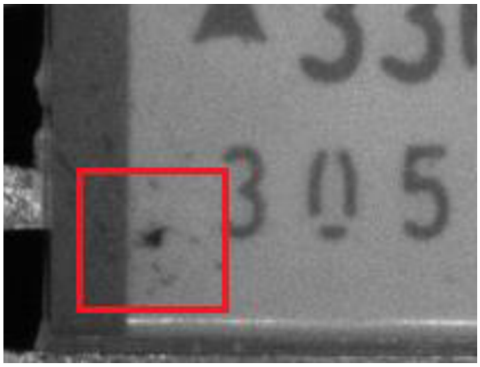
systém robiť v praxi, snažil by som sa o to prispôsobiť hranicu požiadavkám zákazníka, s jeho spoluprácou.

Pre praskliny je to veľmi jasné – ak sa nachádzajú v polaritnom pruhu, je ľahké si ich pomýliť so špinou. V tomto prípade si však myslím, že ide o neoznačenie týchto prasklín vo vedľajších kondenzátoroch na obrázku, príklad na obrázku 4.19.



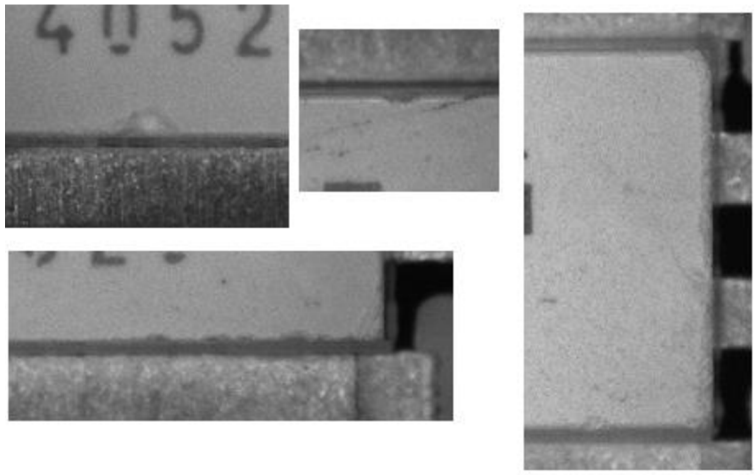
Obrázok 4.19 Potenciálne chyby pri značkovaní 1

Pri dierach ju určite problémom, že sa malé diery podobajú väčším kusom špiny. Toto z časti spôsobuje, že môže vzniknúť nesprávne označenie, a tiež si ich môže pomýliť model. V niektorých prípadoch viem, že mi chýbala presná referencia, aby som sa rozhodol. Príkladom je čierna škvrna na obrázku 4.20.



Obrázok 4.20 Potenciálne chyby pri značkovaní 2

Ulomeniny sú, vyplývajúc aj z dosiahnutých výsledkov, najviac ovplyvnené. Príklady uvedené v kapitole 1 sú jasné veľké ulomeniny, týchto je však pomenej. Niekedy sa jedná o neistotu spôsobenú veľkosťou, niekedy jasnosťou. Mnoho kondenzátorov má, ako som si pre seba nazval, „ohlodané“ hrany, ktoré sú mierne odreté, ale nie sú na nich výrazné ulomené oblasti. Väčšinou nejde o to, že by sa prehliadli potenciálne vady, ale že je ťažké spoľahlivo rozhodnúť. Týmto sa vnáša do učenia chyba. Pár príkladov uvedených na obrázku 4.21.



Obrázok 4.21 Potenciálne chyby pri značkovaní 3

4.4.2 Návrhy na zlepšenie

Aby som zhrnul poznatky z kapitoly 4.4.1, myslím si, že by sa malo previesť vytvorenie lepšej referencie obrázkov, ktoré sa majú označiť ako vadné. V kombinácii s pozornejším značkovaním si myslím, že by sa výsledky mali výrazne zlepšiť. Jedným z prístupov by mohlo byť viacnásobné prechádzanie a kontrola značiek. Prípadne by sa mohlo navzájom skontrolovať viac ľudí, pracujúcich na značkovaní, ale toto sa zdá byť mimo možnosti diplomovej práce.

V rámci učenia je možné upraviť IoU(IoU=0,7), a váhovacie koeficienty pre kladenie zámeru na presnosť ohraničenia, presnosť klasifikácie, či pozornosť smerom k malým detailom, ktoré upravujú chybovú funkciu.

4.4.3 Rýchlosť

Pre dosiahnutie najvyššej rýchlosti, je možné modely exportovať do formátu engine. Tento formát je vlastne formát NVIDIA TensorRT, a je vytvorený tak, aby bol maximálne optimalizovaný na grafickú kartu.

Pre YOLOv8nano, bolo s grafickou kartou GTX1060 dosiahnuté času 0,8 milisekúnd na predspracovanie, 6 milisekúnd na odozvu modelu, a 2 milisekundy na post-processing.

Pre YOLOv8medium bolo dosiahnuté času 0,7 milisekúnd na predspracovanie, 26,8 milisekúnd na odozvu modelu, a 1,8 milisekundy na post-processing.

5. VYHODNOCOVANIE POPISU

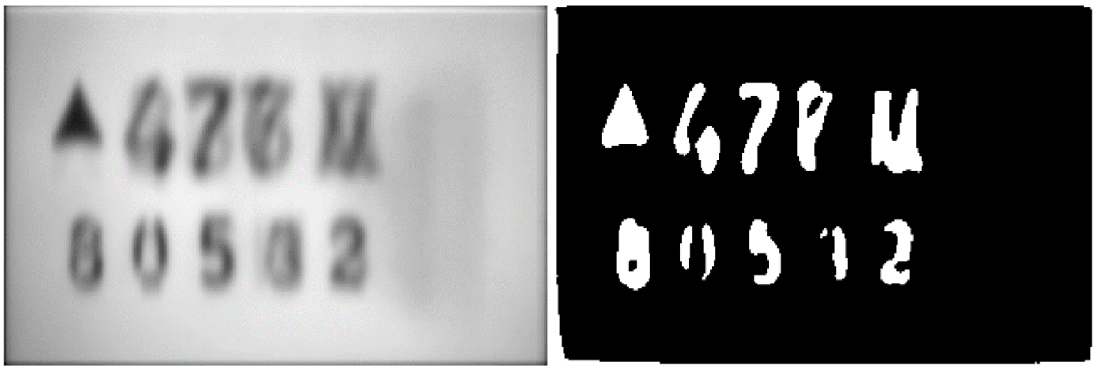
Kapitola popisuje algoritmy, ktoré majú za úlohu spracovať popis na kondenzátoroch.

5.1 Pokusy s maskovaním

Prvou úlohou pri spracovaní textu na segmentovanom kondenzátore je jeho oddelenie od pozadia, čiže opakovaná segmentácia. Napadlo mi, že ak je popis na správnom mieste, znaky by sa v rámci kondenzátoru mali nachádzať vždy na rovnakom mieste. Toto ma viedlo vyskúšať segmentáciu pomocou maskovania.

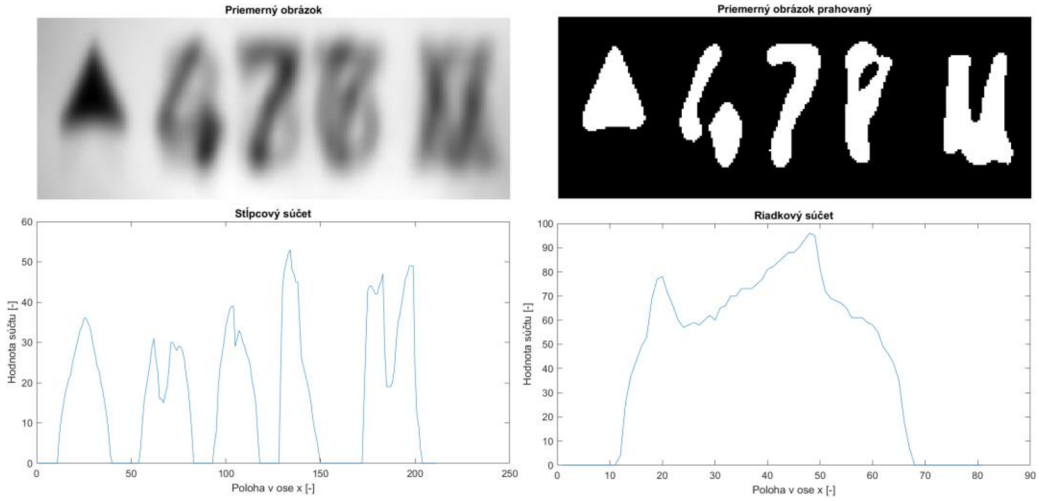
5.1.1 Tvorenie masky

Princíp nápadu je veľmi jednoduchý, spočíta sa priemerný obrázok väčšieho množstva orezaných kondenzátorov. V tomto štádiu som mal vytvorený učiaci dataset o približne 500 obrázkoch, tak som použil práve tieto. Priemerný obrázok je v ďalšom kroku prahovaný, znázornené na obrázku 5.1.



Obrázok 5.1 Priemerný obrázok a jeho prahovanie

Prahovaný obrázok sa manuálne určeným rozsahom orezal tak, aby vznikli dva menšie, obsahujúce jednotlivé riadky. Následne sa vykonali na oba riadkové a stĺpcové súčty. Z nich je možné vyčítať hranice jednotlivých riadkov, a v nich jednotlivých znakov(obrázok 5.2). Malým ručným doladením sú tieto hranice použité pre vytvorenie masky. Samotnú masku je možné vidieť potom na obrázku 5.3.

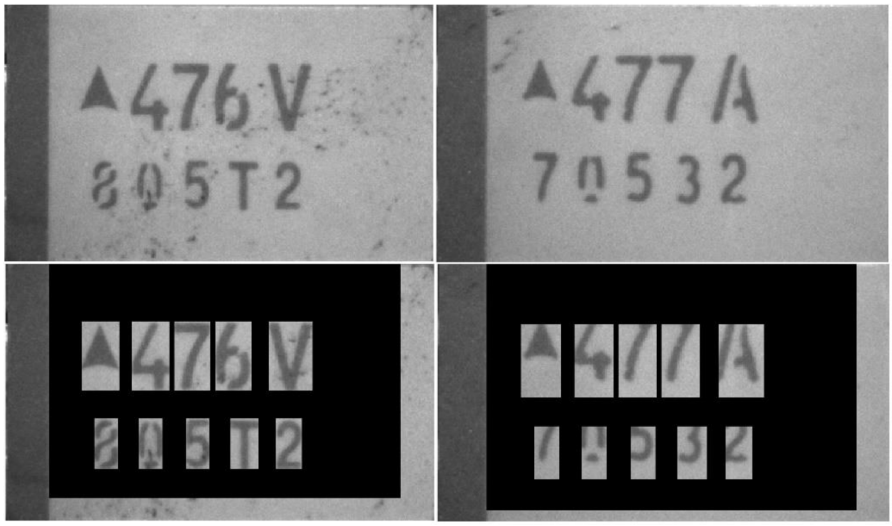


Obrázok 5.2 Identifikácia polohy znakov pre riadok 1

5.1.2 Aplikovanie masky a zhodnotenie

Maska je zakotvená k hornej hrane kondenzátoru a k hrane polaritného pruhu. Maska je binárna. Po umiestnení na obrázok, ak sa v maske nachádza hodnota 1, tak sa ponechá hodnota z obrázku, ak 0, zapíše sa 0. Maska je menšia než samotný kondenzátor, preto sa v jej okolí všetko nezačierni.

Podľa pokusov je možné vidieť, že text nemá pevnú polohu na kondenzátore, a môžu nastať malé posuny bez toho aby to bola chyba (usudzujúc podľa toho, že je na kondenzátore celý polaritný pruh). Pravdepodobne je táto malá odchýlka spôsobená malou variabilitou pri hľadaní okrajov kondenzátoru pri segmentácii. Kvôli tomuto faktu som sa rozhodol posunúť iným smerom ako pevná poloha znakov. Ukážka maskovania kondenzátorov je na obrázku 5.3.



Obrázok 5.3 Ukážka úspešného a neúspešného maskovania

5.2 Porovnávanie vzorov

Algoritmus vyhodnotenia popisu je založený na porovnávaní vzorov, z angl. template matching. Ide o metódu, ktorá je schopná vytvoriť mapu zhody hľadaného vzoru a vyšetrovaného obrázku. Font znakov laserovaných na kondenzátory, bol sprístupnený vo forme binárnych obrázkov. Tieto znaky sú spomínané vzory, ktoré budeme vyhľadávať v obraze. Aby algoritmus zistil, ktorý znak má hľadať, pozrie sa do ImageInfo.

5.2.1 ImageInfo

Každý obrázok obsahuje krátku časť textu, ktorá ohľadom neho dáva bližšie informácie. Tento text je sprístupnený, ak sa obrázkový súbor otvorí ako text, napr. v poznámkovom bloku. V MATLAB-e bola využitá funkcia „fileread“. V C# je potrebné obrázok previesť na pole bytov (Byte[]), pomocou File.ReadAllBytes(). Potom je pole prevedené pomocou Encoding.UTF8.GetString() na reťazec znakov (string). Príklad ImageInfo:

```
app:CVS810H-N|
appver:27.02.2019|mach:L423|view:1|UTC:20190409T000608248+0200|
ImageResult:1|ImageDebugResult:0|AlgResult:1|AlgType:TACT|
AlgSubType:1|iLFPosition:11|ImageResultIdent:EOH_ALL_OK|
AlgResultView:1|AlgResultIdent:EOH_ALL_OK|ResultText:Vše OK|
iiDllCfg:T1=l336D,T2=40501,T3=,Msk=BJ|
Batch:23281~A1440HGL~TJD336M1DSLRFJX~40501~D8MT
```

Obrázky boli využité pre vývoj iného algoritmu, a veľká časť ImageInfo je tvorená informáciami súvisiaceho s týmto vývojom. Výsledky tohto algoritmu nemusia byť správne, a aj keby, nie sú využité pri spracovaní. Čo sa využije z ImageInfo, sú v tomto zobrazení posledné dva riadky. Obsahujú informácie o várke, a tým pádom aj popise laserovanom na kondenzátor.

Prvou dôležitou informáciou je maska – „Msk=BJ“. Toto je jedným z parametrov, ktoré určujú popis kondenzátoru. Rôzne masky kondenzátorov môžu mať iné rozloženie popisu. Táto úloha sa zameriava iba na masku „BJ“, čo sú žlté kondenzátory.

Ďalším parametrom je tretie písmenko v tretej oddelenej skupine znakov po „Batch:“, toto je na konkrétnom prípade písmeno „D“, určuje typ.

Veľmi dôležité sú časti „T1=“ a „T2=“ kde sa určujú znaky v jednotlivých riadkoch popisu. Prvý znak po „T1=“ je kódovým znakom pre značku.

Z extrahovaných informácií je potom možné poskladať meno súboru, ktorý obsahuje font daného znaku, ako napríklad„BJ_D_1_A.png“, čo vyjadruje znak „A“ na druhom riadku (font značený 0 a 1 pre riadky), maska „BJ“, typ „D“. Znaky v prvom riadku potrebujú jeden znak navyše v názve súboru, v závislosti na polohe (prvý, posledný, stred). Tieto extra znaky sú vždy rovnaké, takže sú jednoducho pridané podľa poradia znaku v ImageInfo.

5.2.2 Popis algoritmu

V rámci konštruktoru triedy smdSample(štruktúra je vysvetlená v kapitole 6) sa pri vytvorení novej inštancie priamo načíta do premennej aj ImageInfo. Prvým krokom algoritmu je extrahovanie masky, typu, a obsahu jednotlivých riadkov z neho. V C# je toto jednoducho prevedený hľadaním správnych znakov pomocou metód (string).IndexOf(), ktorá nájde index hľadaného reťazca, a (string).Substring(), ktorá z nájdeného indexu a známej relatívnej polohy vytvorí správny podreťazec.

Aby sme mohli hľadať v obrázku binárne vzory, prevedieme na orezaný obrázok prahovanie. Využíva sa horného a dolného prahovanie, pre ohraničenie úzkeho jasového rozsahu. Prahy sú určené konštantným násobkom priemernej farby polaritného pruhu. Použitý je len na určenie farby, ale zo samotného obrázku je orezaný. Veľká časť z neho by logicky mohla byť naprahovaná kladne, a mohlo by to spôsobovať chyby. Príklad prahovaného vzorku a jedného hľadaného znaku z neho je viditeľný na obrázku 5.4.

Obrázok 5.4 Prahovaný vzorok a hľadaný znak

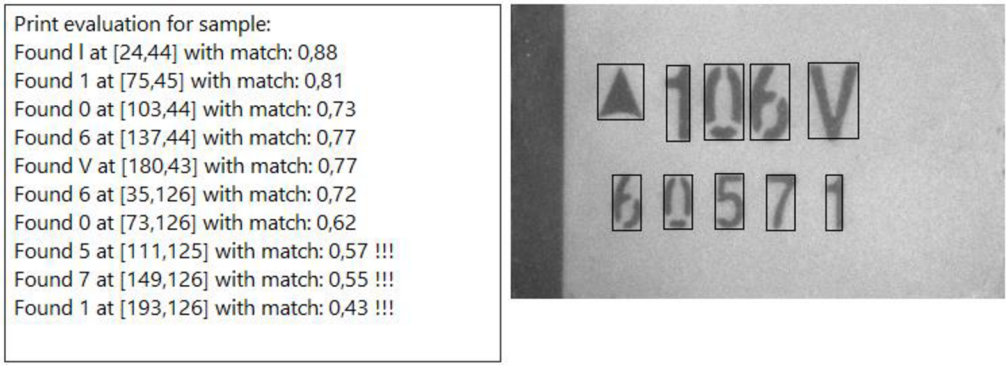
Fonty sú uložené vo výrazne vyššom rozlíšení, než v ktorom sa nachádzajú na kondenzátoroch. Znaky v prvom riadku sú veľké približne 60x40pix, a v druhom riadku 40x20pix. Fonty majú rozmery, v závislosti na písmene, cca 220x330pix a 120x240pix. Po načítaní správneho fontu je zmenšený konštantnou hodnotou, odvinutou od výšky písmen v pixeloch na obrázkoch, ktorá je malo premenlivá. Konkrétne sa jedná o konštantu 0,175. Upravený font je mierne väčší ako znak v prahovanom obrázku, a to preto, aby výsledné ohraničenie lepšie opísalo znak v pôvodnom obrázku.

Pre vytvorenie mapy zhody je využitá metóda MatchTemplate() triedy Mat, z knižnice OpenCVSharp. V zdrojovej matici hľadá vzor, a pomocou vybraného kritéria vytvorí mapu zhody, výpočtom kritéria v rôznych pozíciách. Ako kritérium sa vybrala normalizovaná suma štvorcov odchýlok. Počíta sa pomocou rovnice

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 * \sum_{x',y'} I(x+x',y+y')^2}}, \tag{5.1}$$

kde $T(x',y')$ vyjadruje vzor a iteráciu v rámci neho a $I(x + x', y + y')$ vyjadruje iteráciu na danej polohe v obrázku, v rozsahu vzoru.

Výhodou kritéria je, že výsledná hodnota je vďaka normalizácii v rozsahu 0-1. Jediný detail, lepšia zhoda odpovedá malej hodnote, keďže je založená na odchýlke. Pomocou metódy MinMaxLoc() je nájdená poloha minima, čiže najlepšej zhody. Pre intuitívnejšie zobrazenie je od výslednej hodnoty odčítaná jednotka, aby vyššia hodnota vyjadrila lepšiu zhodu. Hodnota zhody a poloha je vložená do reťazca, ktorý textom zobrazuje výsledok algoritmu. Pri zhode nižšej ako 0,6 je zobrazené varovanie vo forme troch výkričníkov. Do segmentovaného obrázku vzorku sa vykreslia ohraničenia znaku. Výstup je znázornený na obrázku 5.5.

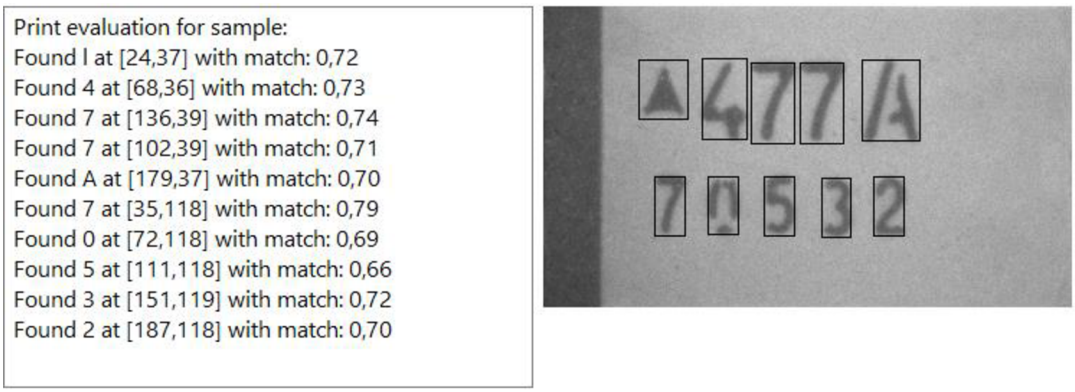


Obrázok 5.5 Výsledok spracovania popisu

5.2.3 Obmedzenia algoritmu

Výstup algoritmu je iba spojitá metrika, vyhodnocujúca zhodu so vzorom a neurčuje pevnú minimálnu povolenú zhodu, je skôr „orientačná“. Hranica 0.6 je práve iba varovanie, pre niektoré prípady je táto hodnota na oko v poriadku(napr. obrázok 5.5, znaky v druhom riadku), pre iné nastáva výrazná zmena tvaru už pri vyšších hodnotách(0 na obrázku 5.6).

Na obrázku 5.6 je možné vidieť aj druhý neošetrený problém – duplikáty znakov. Ak sa pozrieme na súradnicu x, tak znaky „7“ v prvom riadku sú v zlom poradí. Toto sa môže stať preto, lebo algoritmus hľadá prostú zhodu, a zoberie maximum. Dôvod prečo nájde obidve sedmičky, a nie dvakrát tú istú je ten, že pre predchádzanie tejto situácii bolo pridané po nájdení vzoru v prahovanom obrázku začiernenie jeho oblasti. Tento krok umožní nájsť druhú sedmičku, ktorá ma mierne menšiu zhodu, ak sa vyskytnú viacnásobné písmená. Vďaka veľkosti znakov by sa algoritmus nemal pomýliť aspoň v riadku znaku. Stále však môže zameniť poradie. Toto nemusí byť problém, pretože hodnota je spojená s polohou, avšak má to istý nádych nedokončenosti.

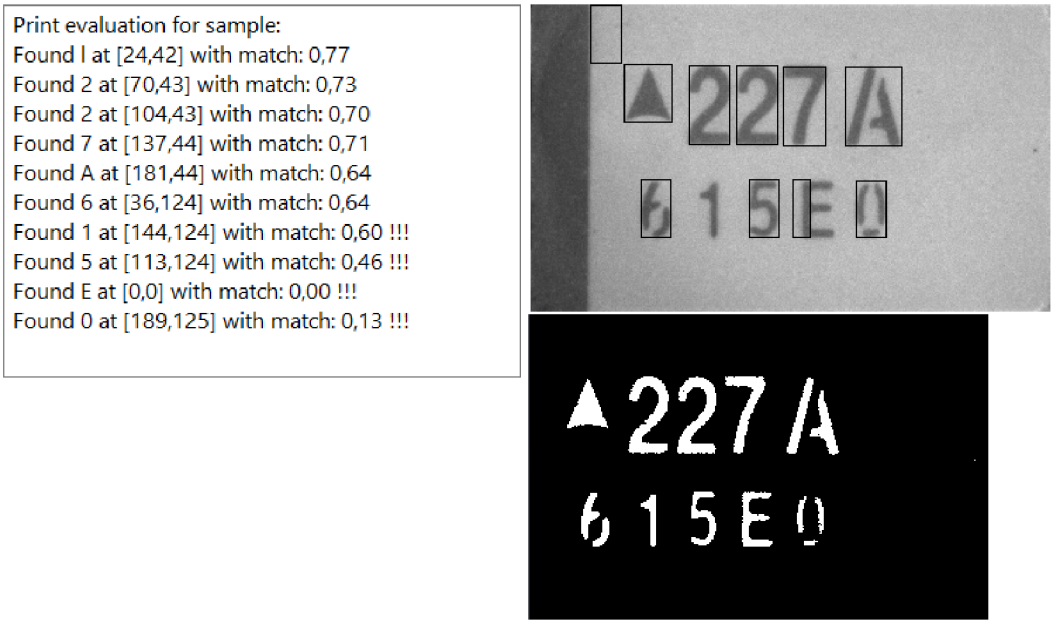


Obrázok 5.6 Vysoká zhoda „0“ napriek skreslenému tvaru

5.2.4 Vyhodnotenie úspešnosti

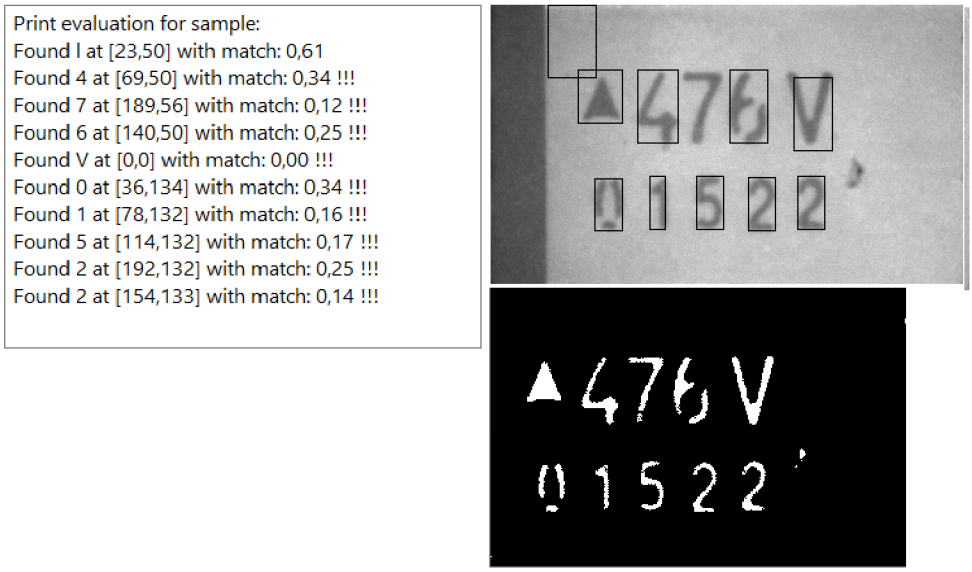
Ako aj s ostatnými algoritmami, vyhodnotenie prebehlo na testovacom datasete. Časová náročnosť sa pohybuje na použitom počítači v rozmedzí 15-35 milisekúnd, toto ale zahŕňa aj čas potrebný na segmentáciu kondenzátoru. Netrúfam si voľným okom hodnotiť presnosť textu, a z dôvodov už spomenutých v predchádzajúcej podkapitole, opomeniem výsledky metriky presnosti, a budem sa sústrediť iba na lokalizáciu textu.

Z 220 vyšetrovaných obrázkov sa text našiel nesprávne v 9 z nich. V ostatných prípadoch okno prekrývalo znaky správne, prípadne s drobnou chybou na okrajoch. V troch prípadoch sa našla najlepšia zhoda znaku „1“ ako časť iného znaku, napriek zdanlivo dobrému výsledku prahovania. Spôsobilo to jeho zakrytie a nesprávnu lokalizáciu oboch. Na príklade obrázku 5.7 sa takto znak „1“ našiel v znaku „E“.



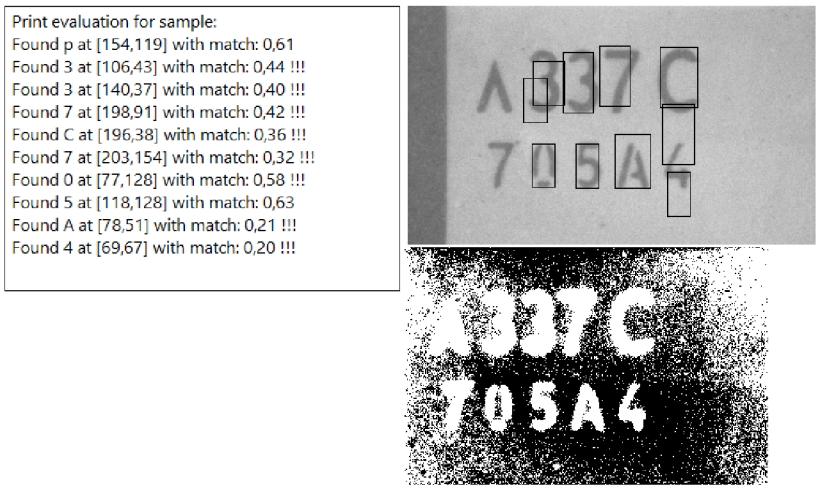
Obrázok 5.7 Chyba nájdenia znaku 1

V ďalších štyroch prípadoch testovania sa tiež našli znaky ako časti iných znakov, ale na rozdiel od predchádzajúcich príkladov predpokladám, že to bolo spôsobené zlým prahovaním. Na obrázku 5.8 sa zdá, že sa kvôli zlému prahovaniu našla časť znaku „V“ ako najlepšia zhoda znaku „7“. Po začiernení sedmičkou sa samozrejme už znak „V“ nenašiel.



Obrázok 5.8 Chyba nájdenia znaku 2

Ďalšia chyba bola nesprávna konštrukcia mena súboru fontu pre znak malého „e“. Pre malé písmená je nutné pridať za daný znak v mene súboru jeho veľkú verziu. Posledným problémom bol obrázok s mierne odlišnými svetlenými podmienkami a veľkým množstvom chyby v prahovanom obrázku (obrázok 5.9).

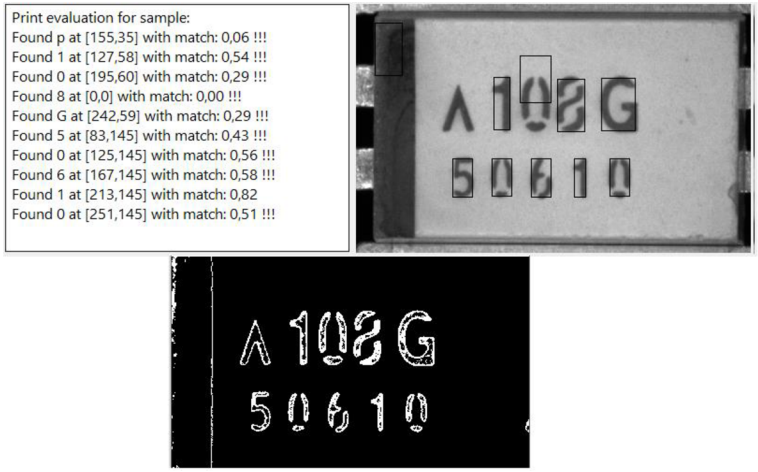


Obrázok 5.9 Chyba nájdenia znaku 3

5.2.5 Navrhované vylepšenia algoritmu

Čo sa týka fiktívnej budúcnosti, ak by sa pokračovalo vo vývoji, jednou z možností by bolo zlepšiť presnosť prahovaných obrázkov. V niektorých prípadoch ich nepresnosť spôsobuje nesprávne nachádzanie znakov alebo nepresné vyhodnotenie tvaru (napr. obrázok 5.10). Ak by som mal pokračovať, pokúsil by som sa vytvoriť prísnejšie kritérium prahu, prípadne využil aj hranových filtrov, a následne by sa mohli uzavreté plochy vyplniť doplna. Dôležité by bolo zachytiť presný okraj znakov.

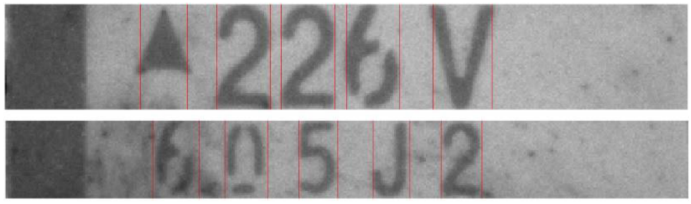
Algoritmus tiež nehodnotí, či je ucelený polaritný pruh. Toto by mohlo byť vyšetrované napríklad hľadaním výrazných lokálnych zmien, avšak možno iba do kladného smeru, aby sa neidentifikovala špina ako chýbajúca plocha. Kontrola polaritného pruhu by mohla umožniť aj vyhodnotenie nežiadúceho vertikálneho/horizontálneho posunu popisu.



Obrázok 5.10 Príklad zlého prahovania popisu.

5.3 Alternatívny algoritmus segmentácie popisu

Je tu tretí, alternatívny algoritmus, ktorý nebol implementovaný vo finálnej verzii. Je určený na dynamické segmentovanie jednotlivých znakov. Je založený na hľadaní hrany znakov na základe diferencie v amplitúdových projekciách. Jeho výstupom sú horné a dolné hrany riadkov, a bočné hrany jednotlivých znakov. Pre ukážku je príklad spracovania na veľmi špinavom kondenzátore na obrázku 5.11.



Obrázok 5.11 Ukážka výsledku neimplementovaného algoritmu.

6. KNIŽNICA DLL A GRAFICKÉ ROZHRANIE

V kapitole sú uvedené bližšie informácie, týkajúce sa prevedenia DLL knižnice a užívateľského rozhrania.

6.1 Knižnica smdClassifierLib

Algoritmy na vizuálne spracovanie kondenzátorov sú zabalené do knižnice DLL, s pracovným názvom smdClassifierLib. Knižnica je napísaná v jazyku C#. Už dlho pred vývojom knižnice sa predpokladalo, že bude napísaná v C++, z dôvodu pravdepodobného využitia známej knižnice OpenCV, na spracovanie obrazu. Užívateľské rozhranie sa však začalo vyvíjať vo Windows Forms, kvôli jednoduchosti. Windows Forms sa programuje v jazyku C#. Tu vzniká problém, pretože C# je manažovaným kódom, a C++ nie. V skratke, v C# nie je možné priamo zavolať funkcie z C++, ako je vo zvyku s bežným include. Je viac možností ako to vyriešiť. Osobne mi prišlo najjednoduchšie nájsť tzv. wrapper libraries, zabaľovacie knižnice, ktoré budú obsahovať potrebný kód a jednoducho sa pripoja k C# kódu, ako iné knižnice C#. Knižnica sa teda vytvorila v C#, napriek doposiaľ neexistujúcim skúsenostiam s týmto jazykom. Pre OpenCV sa našlo rovno viac možností: OpenCVSharp a EmguCV. Rozhodol som sa vyskúšať OpenCVSharp, a ostal som pri nej. Pre prácu s modelom YOLOv8 som použil knižnicu YoloDotNet.

6.1.1 Trieda smdSample

Prvou z dvoch tried v smdClassifierLib je smdSample. Prvotná predstava bola taká, že by v rámci jednej triedy existovali všetky dáta k spracovaniu jedného kondenzátoru, medzivýsledky aj vyhodnotené obrázky. Trieda by potom mohla mať potrebné metódy s algoritmami a inštancia triedy by sa postupne „obohacovala“ s reťazcom spracovania.

Základom je premenná baseImg, do ktorej sa vpíše zdrojový obrázok pri vytvorení inštancie triedy. Prvým krokom spracovania je vždy segmentácia kondenzátoru, prevedená metódou cropSample(). Jej výsledok je zapísaný do croppedImg. Ďalším logickým krokom je detekcia vád. K tomuto kroku viac pri druhej triede, dôležité je, že sa použije croppedImg a výsledok je zapísaný do detectionImg. Ďalej je možné vyhodnotiť popis. Metóda evaluatePrint() použije croppedImg aby vyhodnotila popis kondenzátoru, a výsledok zapíše do printEvalImg. Je možné povedať, že v tomto prevedení sú detectionImg a printEvalImg na tej istej hĺbke spracovania, ale takto je možné prehľadnejšie zobraziť výsledky jednotlivých algoritmov. A samozrejme vyhodnotenie popisu by mohlo byť podmienené výsledkom detekcie vád.

Trieda obsahuje ďalšie pomocné premenné či funkcie, ktoré umožňujú správnu funkciu algoritmov, alebo interakciu smerom von z triedy. Hlavný princíp funkcie je ale zachytený v predchádzajúcom odstavci. Jedna inštancia smdSample je teda, ako názov

naznačuje, jeden vzorok kondenzátoru, a obsahuje v sebe všetky kroky a medzivýsledky, patriace k tomuto vzorku.

6.1.2 Trieda yoloInstance

Druhou triedou v knižnici je yoloInstance. Je to menšia, pomocná trieda, vytvorená primárne nato, aby v sebe držala načítaný model YOLO. Z návrhového hľadiska by nedávalo zmysel aby sa model držal v jednom vzorku kondenzátoru, preto má vlastnú triedu.

Trieda obsahuje dve metódy, prvou je konštruktor, ktorý vytvorí inštanciu načítaním modelu zo súboru. Druhá metóda slúži na inferenciu. Jej vstupom je trieda smdSample, takto je umožnené aby si model zobral potrebné dáta vzorku a výsledky rovno vpísal do neho.

Na nešťastie sa nepodarilo sfunkčniť prácu s grafickou kartou v rámci knižnice YoloDotNet. Toto je možno ale výhodou, pretože mám silný pocit, že by kvôli rôznym potrebným súborom v počítači, a ich správnym verziám pre kompatibilitu, možno nikto nespustil inferenciu na inom stroji. Je nutné teda pre demonštráciu využiť procesorový formát .onnx, ktorý je ale pomalší.

6.2 Užívateľské rozhranie smdClassifierClient

Užívateľské rozhranie pre demonštráciu algoritmov v smdClassifierLib sa nazvalo veľmi zladene ako smdClassifierClient. Ako sa už spomenulo, je vytvorené pomocou Windows Forms. Konečná implementácia bola vytvorená na .NET 8. Vzhľad je možné vidieť na obrázku 6.1.

Obrázok 6.1 Ukážka vzhľadu smdClassifierClient

6.2.1 Ovládanie

Prvým krokom je výber priečinku pomocou tlačidla „Select folder“. Otvorí sa klasické dialógové okno Windows. Odporúča sa použiť priečinky „images“ z niektorej časti datasetu. Po výbere priečinku sa objaví zoznam obrázkov v ňom. Je možné kliknutím na ne ich načítať a zobraziť.

Po výbere obrázku je možné buď spracovať popis alebo povrchové vady. Na spracovanie povrchových vád je najprv potrebné načítať model pomocou „Load network“. V obidvoch prípadoch spracovania sa objaví grafické aj textové znázornenie výsledku. Tiež sa zobrazí čas spracovania.

7. ZÁVER

V práci sa podarilo vyvinúť algoritmy na spracovanie všetkých potrebných vlastností kondenzátoru. Čo sa týka ich presnosti, ako bolo vyhodnotené v jednotlivých kapitolách, nie všetky dosahujú vysokej spoľahlivosti. Pre všetky sa však našiel ďalší smer ako ich zlepšiť, a zahrnula sa tiež analýza chýb.

Detekcia povrchových vád bola silne poznačená chybami, alebo skôr nejasnosťami, primárne ohľadom značkovania datasetu pre učenie.

Algoritmus vyhodnocovania popisu nie je úplný, a aj v implementovaných častiach má svoje nedostatky.

Na záver bolo dôležité, aby sa v grafickom prostredí mohli vyvinuté algoritmy vhodne prezentovať.

Ak zanedbám slabšie výsledky navrhnutého systému spracovania, práca ma veľa naučila. Vyskúšal som si veľa činností, ktoré som doteraz nerobil. Veľmi veľa mi dalo, že som mohol pracovať na komplexnom probléme, ktorý som musel rozdeliť na menšie, a tieto postupne vylepšovať a rozvíjať pre dosiahnutie lepších výsledkov, alebo ich zahodiť a vybrať sa iným smerom. Zároveň sa snažiť celý tento proces precízne dokumentovať a zhodnotiť, následne prezentovať na primeranej úrovni ku kategórii práce.

LITERATURA

[1] ZSIDEK, Adrián. *Hyperspektrální analýza obrazu.* Brno, 2022. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/142237. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Karel Horák.

[2] JANÁKOVÁ I. Filtrace šumu a poruch [Internet]. [Brno]: Computer vision group, Department of Control and Instrumentation, Faculty of Electrotechnical Engineering and Communications, Brno University of Technology; [citované: 2.1.2023]. [50 str.] Dostupné z: http://vision.uamt.feec.vutbr.cz/ZVS/lectures/07_Filtrace_sumu_a_poruch.pdf. Czech.

[3] HORÁK K. Jasové transformace [Internet]. [Brno]: Computer vision group, Department of Control and Instrumentation, Faculty of Electrotechnical Engineering and Communications, Brno University of Technology; [citované: 2.1.2023]. [23 str.] Dostupné z: http://vision.uamt.feec.vutbr.cz/ZVS/lectures/05_Jasove_transformace.pdf. Czech.

[4] HORÁK K. Geometrické transformace [Internet]. [Brno]: Computer vision group, Department of Control and Instrumentation, Faculty of Electrotechnical Engineering and Communications, Brno University of Technology; [citované: 2.1.2023]. [16 str.] Dostupné z: http://vision.uamt.feec.vutbr.cz/ZVS/lectures/06_Geometricke_transformace.pdf. Czech.

[5] HORÁK K. Detekce hran a rohů [Internet]. [Brno]: Computer vision group, Department of Control and Instrumentation, Faculty of Electrotechnical Engineering and Communications, Brno University of Technology; [citované: 2.1.2023]. [29 str.] Dostupné z: http://vision.uamt.feec.vutbr.cz/ZVS/lectures/08_Detekce_hran_a_rohu.pdf. Czech.

[6] JANÁKOVÁ I. Segmentace [Internet]. [Brno]: Computer vision group, Department of Control and Instrumentation, Faculty of Electrotechnical Engineering and Communications, Brno University of Technology; [citované: 2.1.2023]. [31 str.] Dostupné z: http://vision.uamt.feec.vutbr.cz/POV/lectures/05_Segmentace.pdf. Czech.

[7] JANÁKOVÁ I. Detekce geometrických primitiv [Internet]. [Brno]: Computer vision group, Department of Control and Instrumentation, Faculty of Electrotechnical Engineering and Communications, Brno University of Technology; [citované: 2.1.2023]. [49 str.] Dostupné z: http://vision.uamt.feec.vutbr.cz/POV/lectures/06_Detekce%20geometrickych%20primitiv.pdf. Czech.

[8] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016. [citované: 4.8.2024]. [10 str.] Dostupné také z: https://arxiv.org/pdf/1506.02640

[9] YOLO Creator Joseph Redmon Stopped CV Research Due to Ethical Concerns. *Synced* [online]. 2020. Dostupné také z: https://syncedreview.com/2020/02/24/yolo-creator-says-he-stopped-cv-research-due-to-ethical-concerns/

[10] YOLOv8. *Roboflow* [online]. 2023 [cit. 2024-08-04]. Dostupné z: https://roboflow.com/model/yolov8

[11] YOLOv8. Ultralytics [online].Vytvorené 2023-11-12, Aktualizované 2024-07-04, [cit. 2024-08-04]. Dostupné z: https://docs.ultralytics.com/models/yolov8/

[12] Terven, J.; Córdova-Esparza, D.-M.; Romero-González, J.-A. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. Mach. Learn. Knowl. Extr. 2023, 5, 1680–1716. https://doi.org/10.3390/make5040083

[13] What is YOLOv8? The Ultimate Guide.[2024]. *Roboflow* [online]. 2023 [cit. 2024-08-04]. Dostupné z: https://blog.roboflow.com/whats-new-in-yolov8/

[14] HORÁK K. mAP for Object Detection [Internet]. Brno University of Technology / Czech Technical University in Prague; [citované: 4.8.2024]. [18 str.] Dostupné z: http://vision.uamt.feec.vutbr.cz/ROZ/lectures/MachineLearning_mAP.pdf . English.